Faire teilbare elektronische Geldsysteme mit Observern

Inauguraldissertation
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)

eingereicht beim
Fachbereich Mathematik und Informatik, Physik, Geographie
der Justus-Liebig-Universität Gießen

von
Patrick Märtens
geboren in Koblenz

Gießen, November 2014

Dekan:	Prof. Dr. Peter Jens Klar
1. Gutachter:	Prof. Dr. Albrecht Beutelspacher (Gießen)
2. Gutachter:	Prof. Dr. Jörg Schwenk (Bochum)
Datum der Disputation:	27. Mai 2015

ABSTRACT

Elektronische Geldsysteme stellen ein digitales Äquivalent zu herkömmlichen Bargeld dar. In ihrer einfachsten Form bestehen elektronische (Offline-) Geldsysteme aus drei Protokollen zum Abheben, Bezahlen und Einlösen elektronischer Münzen zwischen drei Parteien – der Bank, den Kunden und den Händlern. Die wichtigsten Sicherheitsanforderungen sind Unfälschbarkeit elektronischer Münzen, Identifikation von Double-Spendern und Anonymität der ehrlichen Kunden. Abhängig von ihrer Anwendung können elektronische Geldsysteme zusätzliche Eigenschaften wie Teilbarkeit, Fairness und Einsatz von Observern besitzen.

- Ein *teilbares* elektronisches Geldsystem ermöglicht einem Kunden in jeweils einer Transaktion eine Geldbörse mit dem Geldwert K abzuheben und mit einer Münze mit dem Wert k zu bezahlen. Typischerweise sind $K = 2^L$ und $k = 2^\ell$ für $0 \leq \ell \leq L$.
- In einem *fairen* elektronischen Geldsystemen kann eine *Trusted-Third-Party* (TTP) die Anonymität in Verdachtsfällen aufheben, um perfekte Verbrechen zu verhindern.
- Ein elektronisches Geldsystem *mit Observern* garantiert nicht nur das Aufspüren von Double-Spendern, sondern das unmittelbare Verhindern eines Double-Spendings.

Ziel der Dissertation ist die Konstruktion sicherer und effizienter fairer teilbarer elektronischer Geldsysteme mit Observern. Da die Literatur unterschiedliche Sicherheitsdefinitionen bereit stellt, werden zunächst neue Sicherheitsziele entwickelt, die die in der Literatur bestehenden umfassen und insbesondere mehr Schutz für Kunden vor einer korrupten Bank bieten.

Wir konstruieren ein teilbares elektronisches Geldsystem, das die neuen Sicherheitsanforderungen im Random-Oracle-Modell erfüllt und bei dem der Datentransfer aller Protokolle unabhängig vom Geldwert ist. Um ein möglichst effizientes System zu erhalten, stellen wir zudem ein neues Signaturverfahren mit effizienten Protokollen vor, das optimal an dieses angepasst ist. Zusätzlich modifizieren wir das Geldsystem zu einem zweiten teilbaren Geldsystem, das effizienter aber nur noch unter Verwendung eines sogenannten Type-3-Pairings sicher ist.

Dann werden beide Geldsysteme zu fairen Systemen erweitert, die lediglich zwei zusätzliche Protokolle besitzen, um einen gewissen Kunden bzw. eine bestimmte Münze zu verfolgen. Somit ist die TTP in der Lage, die Anonymität (von verdächtigen Auszahlungen und Kunden) aufzuheben, während sie nur am entsprechenden Protokoll beteiligt ist.

Des Weiteren modifizieren wir die Protokolle, damit Kunden unterschiedliche Geldbeträge abheben und Münzen mit beliebigem Geldwert zum Bezahlen verwenden können.

Schließlich beschreiben wir, wie die (fairen) teilbaren elektronischen Geldsysteme auf verschiedene Weisen zu Geldsystemen mit Observern erweitert werden können.

ENGLISH ABSTRACT

Electronic cash (e-cash) is the digital counterpart of conventional cash money in the electronic world. In its simplest form, an (offline) e-cash scheme consists of three protocols to withdraw, spend and deposit electronic coins amongst three parties, the bank, users and merchants. The most important security requirements are unforgeability of electronic coins, identification of double-spenders and anonymity of honest users. Depending on application, e-cash schemes may have some additional features such as divisibility, fairness and usage of observers.

- A *divisible* e-cash scheme allows a user to withdraw a wallet of monetary value K and to spend a coin of monetary value k in a single operation, respectively. Typically, we have $K = 2^L$ and $k = 2^\ell$ for $0 \leq \ell \leq L$.
- In a *fair* e-cash scheme a trusted third party is able to revoke the user's anonymity under suspicious activities to prevent perfect crime such as blackmailing or money laundering.
- An e-cash scheme with *observers* guarantees not only the detection of double-spenders afterwards, but also prior restraint of double-spending.

Aim of this thesis is the construction of secure and efficient fair divisible e-cash schemes with observers. Since publications provide different security definitions of e-cash schemes, we come up with new definitions which include the existing security requirements and which provide further security properties for honest users against a corrupt bank.

We design a divisible e-cash scheme in which the bandwidth of each protocol is independent of the monetary value while the system fulfills those new security definitions in the random oracle model. To gain a most efficient scheme, we also present a new pairing-based digital signature scheme with efficient protocols that was optimized for the e-cash scheme. Furthermore, we modify this scheme to get a second divisible e-cash scheme which is more efficient but only secure using a so called Type 3 pairing.

After that we extend both divisible e-cash schemes to fair ones which just have two additional protocols to trace a certain owner and a particular coin, respectively. So, the trusted third party is able to revoke the anonymity (of suspicious payments and users) while it is only involved in the corresponding protocol.

Furthermore, we modify the withdrawal, spend and deposit protocols so that the users are able to withdraw different monetary values $K' \leq K$ and to spend electronic coins of arbitrary monetary value $k \leq K'$ (not necessary a power of 2).

Finally, we describe different ways to extend our (fair) divisible e-cash schemes to schemes with observers.

INHALTSVERZEICHNIS

Abstract **iii**

English Abstract **v**

1 Einleitung **1**
1.1 Elektronische Geldsysteme: Stand der Forschung 2
1.2 Ziele und Aufbau der Arbeit 6

2 Kryptografische Grundlagen **11**
2.1 Verschiedene Notationen 11
2.1.1 Komplexitätstheorie und Protokolle 13
2.2 Ziele der Kryptografie 14
2.3 Hashfunktionen 16
2.3.1 Das Random-Oracle-Modell 17
2.4 Elliptische Kurven in der Kryptografie 18
2.4.1 Sicherheit und Effizienz 19
2.5 Bilineare Abbildungen 21
2.5.1 Verschiedene Pairing Typen 22
2.6 Kryptografische Probleme und Annahmen 24
2.6.1 Das Diskreter-Logarithmus-Problem 25
2.6.2 Das Darstellungsproblem 26
2.6.3 Die Diffie-Hellman-Probleme 27
2.6.4 Kryptografische Annahmen im Typ-2- und Typ-3-Pairing 31
2.7 Public-Key-Verschlüsselungsverfahren 33
2.7.1 Das ElGamal-Verschlüsselungsverfahren 36
2.7.2 Das lineare Verschlüsselungsverfahren 37
2.8 Digitale Signaturen 38
2.8.1 Die Schnorr-Signatur 42

2.8.2 Das Signaturverfahren von Boneh *et al.* (BBS) 43
2.9 Commitment-Schemata 44
2.9.1 Das Pedersen-Commitment-Schema 46
2.9.2 Das Polynom-Commitment-Schema von Kate *et al.* 47
2.10 Zero-Knowledge-Proofs-of-Knowledge 51
2.10.1 Sigma-Protokolle 54
2.10.1.1 Zero-Knowledge-Proofs-of-Knowledge über diskrete Logarithmen 56
2.10.1.2 Zero-Knowledge-Proof-of-Knowledge über ein SDH-Paar 58
2.10.2 Die Fiat-Shamir-Heuristik und nichtinteraktive Proofs-of-Knowledge 58
2.11 Beschränkte Akkumulatoren 59
2.11.1 Das Nguyen-Akkumulator-Schema 62
2.11.1.1 Die Erweiterung von Canard und Gouget 64
2.11.1.2 Zero-Knowledge-Beweise 64
2.12 Signaturverfahren mit effizienten Protokollen 65
2.12.1 Das erweiterte Signaturverfahren von Boneh *et al.* (BBS+) 68
2.12.1.1 Ein Commitment-Schema für das BBS+-Signaturverfahren 69
2.12.1.2 Das BBS+-Signaturverfahren 70
2.12.2 Das erweiterte spezielle Signaturverfahren von Au *et al.* (ESS+) . 73
2.12.2.1 Ein Commitment-Schema für das ESS+-Signaturverfahren 73
2.12.2.2 Das ESS+-Signaturverfahren 74

3 Ein neues Signaturverfahren mit effizienten Protokollen (BBS+A) 81
3.1 Ein Commitment-Schema für das BBS+A-Signaturverfahren 83
3.1.1 Sicherheitsanalyse des Commitment-Schemas 84
3.1.2 Bemerkungen zum Commitment-Schema 93
3.2 Das BBS+A-Signaturverfahren 94
3.2.1 Überblick über das BBS+A-Signaturverfahren 95
3.2.2 Konstruktion des BBS+A-Signaturverfahrens 105
3.2.3 Sicherheitsanalyse des BBS+A-Signaturverfahrens 114
3.2.4 Effizienz des BBS+A-Signaturverfahrens 125

4 Überblick elektronischer Geldsysteme 129
4.1 Elektronische Geldsysteme 129
4.2 Faire elektronische Geldsysteme 133
4.3 Elektronische Geldsysteme mit Observern 135
4.4 Sicherheitsziele (fairer) elektronischer Geldsysteme (mit Observern) . . . 136
4.5 Aktueller Forschungsstand 151
4.5.1 Analyse der Sicherheitsdefinitionen 153

5 Kompakte elektronische Geldsysteme 157
5.1 Das kompakte elektronische Geldsystem KG-I 159
5.1.1 Sicherheitsanalyse von KG-I 164
5.2 Das kompakte elektronische Geldsystem KG-II mit beliebigem Geldbörsenwert 165
5.2.1 Sicherheitsanalyse von KG-II 171

6 Teilbare elektronische Geldsysteme 173
6.1 Teilbare Münzen und Binär-Bäume 173
6.2 Das teilbare elektronische Geldsystem von Au *et al.* (ASM08) 174
6.3 Das teilbare elektronische Geldsystem von Canard und Gouget (CG10) . 176
6.3.1 Brechen der Unfälschbarkeit 177
6.4 Allgemeine Konstruktion der neuen teilbaren elektronischen Geldsysteme 179
6.5 Das teilbare elektronische Geldsystem TG-I 181
6.5.1 Sicherheitsanalyse von TG-I 190
6.6 Das teilbare elektronische Geldsystem TG-II 207
6.6.1 Sicherheitsanalyse von TG-II 211
6.7 Effizienzvergleich der teilbaren elektronischen Geldsysteme 221
6.8 Mögliche Varianten der teilbaren elektronischen Geldsysteme 224

7 Faire teilbare elektronische Geldsysteme 227
7.1 Allgemeine Konstruktion der fairen elektronischen Geldsysteme 228
7.2 Das faire teilbare elektronische Geldsystem FTG-0 229
7.2.1 Sicherheitsanalyse von FTG-0 232
7.3 Das faire teilbare elektronische Geldsystem FTG-I 238
7.3.1 Sicherheitsanalyse von FTG-I 245
7.4 Das faire teilbare elektronische Geldsystem FTG-II 255
7.4.1 Sicherheitsanalyse von FTG-II 263
7.5 Effizienzvergleich der fairen und teilbaren elektronischen Geldsysteme . . 276
7.6 Alternative Varianten der Münzverfolgung 279
7.6.1 Münzverfolgung mit der Methode von Au 279
7.6.2 Münzverfolgung mit der Methode von Camenisch *et al.* 280
7.7 Elektronische Geldsysteme mit erstattungsfähigen Geldbörsen 284
7.7.1 Unvereinbarkeit von Anonymität und Erstattungsfähigkeit 287

8 Elektronische Geldsysteme mit modifizierten Protokollen 289
8.1 Teilbare Münzen mit beliebigem Geldwert 289
8.1.1 Sicherheitsanalyse 294
8.2 Geldbörsen mit beliebigem Geldbetrag 296
8.2.1 Sicherheitsanalyse 299
8.3 Geldbörsen mit unterschiedlichen Geldbeträgen 303
8.3.1 Sicherheitsanalyse 308

9 Faire teilbare elektronische Geldsysteme mit Observern 309
9.1 Rückgabe der Observer an die Bank 310
9.2 Das faire teilbare elektronische Geldsystem mit Observern FTGO-I . . . 311
9.2.1 Sicherheitsanalyse von FTGO-I 321
9.3 Das faire teilbare elektronische Geldsystem mit Observern FTGvO-I . . . 334
9.3.1 Sicherheitsanalyse FTGvO-I 337
9.4 Das faire teilbare elektronische Geldsystem mit Observern FTGvmO-I . . 345
9.4.1 Sicherheitsanalyse von FTGvmO-I 349
9.5 Effizienzvergleich der teilbaren elektronischen Geldsysteme mit Observern 350

10 Zusammenfassung 353
10.1 Offene Fragestellungen 357

A Ablauf der Signatures-of-Knowledge 359
A.1 Die SoK des Issue-Protokolls 359
A.2 Die SoK des Prove-Protokolls 361
A.3 Die SoK des Prove-k-Protokolls 367
A.4 Die SoK des Prove-K-Protokolls 370
A.5 Die SoK des Bezahlprotokolls Spend von FTGO-I 373
A.6 Die zweite SoK des Bezahlprotokolls Spend von FTGO-I 379

Literaturverzeichnis 397

ABBILDUNGSVERZEICHNIS

2.1 Ein Beispiel für die Notation eines Protokolls 14
2.2 Ablauf eines Σ-Protokolls 55
2.3 Ablauf des Protokolls $PoK\left[(x) : X = g^x\right]$ 57
2.4 Signaturerstellungsprotokoll Issue der BBS+-Signatur 71
2.5 Signaturbeweisprotokoll Prove der BBS+-Signatur 72
2.6 Signaturerstellungsprotokoll Issue der ESS+-Signatur 75
2.7 Signaturbeweisprotokoll Prove der ESS+-Signatur 77
2.8 Signaturbeweisprotokoll Prove der ESS+-Signatur für einen Akkumulator 79

3.1 Signaturerstellungsprotokoll Issue der BBS+A-Signatur 107
3.2 Signaturbeweisprotokoll Prove der BBS+A-Signatur 109
3.3 Signaturbeweisprotokoll Prove-k der BBS+A-Signatur 111
3.4 Signaturbeweisprotokoll Prove*-k der BBS+A-Signatur 112
3.5 Signaturbeweisprotokoll Prove-K der BBS+A-Signatur 113

5.1 Abhebeprotokoll Withdraw von KG-I 161
5.2 Bezahlprotokoll Spend von KG-I 162
5.3 Einlöseprotokoll Deposit von KG-I 163
5.4 Abhebeprotokoll Withdraw von KG-II 168
5.5 Bezahlprotokoll Spend von KG-II 170
5.6 Einlöseprotokoll Deposit von KG-II 171

6.1 Darstellung eines Binär-Baums 174
6.2 Verwendung eines Binär-Baums 174
6.3 Berechnung des Binär-Baums in [ASM08] für $L = 2$ 175
6.4 Anwendung der Akkumulatoren in [CG10] für $L = 2$ 177
6.5 Angriff auf das Geldsystem in [CG10] für $L = 2$ 178
6.6 Berechnung des Binär-Baums für $L = 2$ 180

6.7 Berechnung der Seriennummern und Serienschlüssel für $\ell = 2$ 181
6.8 Kontoeröffnungsprotokoll Account von TG-I 183
6.9 Abhebeprotokoll Withdraw von TG-I 185
6.10 Bezahlprotokoll Spend von TG-I 187
6.11 Bezahlprotokoll Spend-K von TG-I 188
6.12 Einlöseprotokoll Deposit von TG-I 189
6.13 Bezahlprotokoll Spend von TG-II 209
6.14 Bezahlprotokoll Spend-K von TG-II 210

7.1 Zusammenhang der fairen teilbaren Geldsysteme 228
7.2 Kundenverfolgungsprotokoll UTrace von FTG-0 231
7.3 Münzverfolgungsprotokoll CTrace von FTG-0 232
7.4 Abhebeprotokoll Withdraw von FTG-I 241
7.5 Bezahlprotokoll Spend von FTG-I 242
7.6 Bezahlprotokoll Spend-K von FTG-I 243
7.7 Kundenverfolgungsprotokoll UTrace von FTG-I 244
7.8 Münzverfolgungsprotokoll CTrace von FTG-I 245
7.9 Abhebeprotokoll Withdraw von FTG-II 258
7.10 Bezahlprotokoll Spend von FTG-II 259
7.11 Bezahlprotokoll Spend-K von FTG-II 260
7.12 Kundenverfolgungsprotokoll UTrace von FTG-II 261
7.13 Münzverfolgungsprotokoll CTrace von FTG-II 262

8.1 Modifiziertes Bezahlprotokoll Spend von FTG-I 291
8.2 Modifiziertes Bezahlprotokoll Spend von FTG-II 293
8.3 Modifiziertes Abhebeprotokoll Withdraw von FTG-I 297
8.4 Modifiziertes Abhebeprotokoll Withdraw von FTG-I 306

9.1 Abhebeprotokoll Withdraw von FTGO-I 316
9.2 SoK des Bezahlprotokolls Spend von FTGO-I 318
9.3 Bezahlprotokoll Spend von FTGO-I 320
9.4 Abhebeprotokoll Withdraw von FTGvmO-I 347
9.5 Bezahlprotokoll Spend von FTGvmO-I 348

TABELLENVERZEICHNIS

2.1 Vom NIST empfohlene Schlüssellängen für äquivalente Sicherheitsniveaus 20
2.2 Relativer Berechnungsaufwand elliptischer Kurven 20
2.3 Effizienzvergleich verschiedener Pairing Typen 24

3.1 Effizienzvergleich der BBS+, ESS+ und BBS+A Signaturverfahren . . . 127

5.1 Öffentliche Parameter von KG-I 160
5.2 Öffentliche Parameter von KG-II 167

6.1 Öffentliche Parameter von TG-I 182
6.2 Öffentliche Parameter von TG-II 208
6.3 Unterschied TG-II im Vergleich zu TG-I 221
6.4 Effizienzvergleich zwischen [ASM08], [CG10] und TG-II 223

7.1 Öffentliche Parameter von FTG-0 230
7.2 Öffentliche Parameter von FTG-I 240
7.3 Öffentliche Parameter von FTG-II 257
7.4 Unterschied FTG-II im Vergleich zu FTG-I 276
7.5 Mehraufwand für faire teilbare Geldsysteme 277
7.6 Effizienzvergleich zwischen TG-I, TG-II, FTG-I und FTG-II 278

8.1 Effizienzvergleich der ursprünglichen und modifizierten Bezahlprotokolle . 294
8.2 Öffentliche Parameter von FTG-I bei dieser Modifikation 305

9.1 Mehraufwand für elektronische Geldsysteme mit Observern 351
9.2 Effizienzvergleich zwischen FTG-I und FTGmvO-I 352

10.1 Komplexitäten der teilbaren elektronischen Geldsysteme 356

KAPITEL 1

EINLEITUNG

Bequemes Einkaufen von zu Hause stellt durch die dynamische Entwicklung im Bereich des Online-Shoppings für viele eine mehr als lukrative Alternative zum herkömmlichen Einkauf in Geschäften dar. Die Zahl derjenigen, die die Möglichkeiten des virtuellen Shoppings nutzen, hat in den letzten Jahren eine rasante Zunahme erfahren. Nach den Presseinformationen des Bundesverbands Informationswirtschaft, Telekommunikation und neue Medien e. V. (BITCOM) haben im Jahr 2010 bereits 60 Prozent der über vierzehnjährigen Deutschen im Internet eingekauft [BIT10]. Damit hat sich die Quote innerhalb von fünf Jahren fast verdoppelt [BIT08] und im Vergleich zum Jahr 2003 beinahe verdreifacht [BIT07b]. Somit nutzen immer mehr Menschen die Vorteile des Online-Shoppings: „Es ist bequem, orts- und zeitunabhängig, man kann Preise vergleichen und die Rückgaberechte sind sehr verbraucherfreundlich“ [BIT11, Präsident des BITCOM]. Inzwischen kaufen 94 Prozent der über vierzehnjährigen Internetnutzer im Web ein, was 51 Millionen Bundesbürgern entspricht [BIT14]. Resultierend aus der wachsenden Popularität des Online-Shoppings steigt der Umsatz der Online-Händler, wie beispielsweise ebay, Amazon und Co stetig an. Die deutschen Konsumenten haben laut einer Studie der Gesellschaft für Konsumforschung (GfK) im Jahr 2008 für etwa 13,7 Milliarden Euro Waren und Dienstleistungen im Internet erworben, wodurch der Umsatz im Vergleich zum Vorjahr um 19 Prozent gestiegen ist [GFK09]. Die virtuellen Einkaufstouren werden auch immer häufiger mit internetfähigen Mobiltelefonen getätigt. So stieg die Zahl derjenigen Verbraucher, die ein Smartphone oder einen Tablet-Computer für Transaktionen nutzen, im Jahr 2014 auf 27 Prozent [BIT14]. Zu den im Internet gekauften Produkten zählt auch das Abrufen kostenpflichtiger, digitaler Inhalte, wie beispielsweise Musikdownloads oder Fahrkarten. Nach BITCOM-Studien hatte der digitale Musikmarkt 2006 mit über 26 Millionen Downloads ein Allzeithoch erreicht

[BIT07a] und Tickets für Flüge und Bahnfahrkarten waren im Jahr 2009 die beliebtesten Produkte des virtuellen Einkaufs [BIT09a].

Dennoch werden für den Bezahlvorgang weiterhin traditionelle Arten bevorzugt, die nicht primär für den Online-Handel konzipiert sind. Laut der BITCOM-Studie [BIT11] bezahlten 42 Prozent aller Internetnutzer mindestens einmal per Rechnung und rund 33 Prozent nutzten die Bezahlung per Vorkasse. Erst an dritter Stelle liegen mit 21 Prozent spezielle Internet-Bezahlverfahren wie PayPal oder Click&Buy, wobei sich in den letzten vier Jahren der Anteil beinahe verdoppelt hat (vgl. [BIT07b]). Es folgen Lastschrift (20 Prozent), Nachnahme (18 Prozent) und Kreditkarte (17 Prozent). Besonders wegen Sicherheitsbedenken und mangelnder Erfahrungen verzichtet jedoch fast jeder fünfte Internet-Nutzer auf das Online-Shopping und spezielle Bezahlsysteme [BIT09b].

Die herkömmlichen Zahlungsmethoden, wie Rechnung, Kreditkarte und Lastschrift, aber auch das Internet-Bezahlverfahren PayPal sind allerdings kaum eine geeignete Lösung für den Online-Zahlungsverkehr. Denn gerade beim Kauf kostengünstiger Waren, wie beispielsweise digitaler Inhalte, stehen die Transaktionsgebühren häufig in keinem Verhältnis zum Wert der gekauften Ware. Des Weiteren gewährleisten diese Bezahlsysteme keine sicheren und anonymen Zahlungen, wie es die Kunden mit Bargeld gewohnt sind. Einerseits muss jeder Käufer sensible private Daten wie Bankverbindung oder Kreditkartennummer preisgeben und andererseits sind die Einkäufe für das verwendete Kreditinstitut nachvollziehbar, was das Erstellen von Kundenprofilen ermöglicht. Vor allem durch den illegalen Handel mit persönlichen Daten der Kunden wird die Skepsis vieler Internetnutzer gegenüber momentan verwendeten Zahlungssystemen verstärkt.

Die Entwicklung eines einfachen, sicheren und anonymen Bezahlsystems verspricht zum einen eine hohe Akzeptanz bei den Anwendern und zum anderen, dass auch die Internetnutzer dieses System verwenden, deren Misstrauen gegenüber den oben genannten Zahlungsverfahren bisher zu groß war. Damit einhergehend kann durch ein solches Bezahlsystem der Umsatz der Online-Händler weiter gesteigert werden.

Bisher hat sich eine adäquate Bezahlmethode im Bereich des E-Commerce allerdings noch nicht etabliert. Eine Alternative stellen elektronische Geldsysteme dar, die sichere, anonyme und kostengünstige Transaktionen gewährleisten.

1.1 Elektronische Geldsysteme: Stand der Forschung

Als digitales Äquivalent zum herkömmlichen Bargeld wurde im Jahr 1988 von D. Chaum, A. Fiat und M. Noar [CFN89] erstmals ein *elektronisches Geldsystem* entwickelt. Ein solches System ermöglicht den *Kunden* sogenanntes *elektronisches Geld* bzw. *elektronische Münzen* bei einer *Bank* abzuheben und es zum Bezahlen bei *Händlern* zu verwenden. Diese können das elektronische Geld dann wiederum bei der Bank einlösen. Zentrale Sicherheitsanforderungen elektronischer Geldsysteme sind *Verifizierbarkeit* der Echtheit, *Fälschungssicherheit* und *Anonymität*, die im Wesentlichen durch eine *digitale Signatur* realisiert werden. Durch die Anonymität der Systeme ist gewährleistet, dass die

Zahlungsvorgänge eines Kunden von niemandem, auch nicht von der Bank, zurückverfolgt werden können. Dadurch stellt die Anonymität *das* wesentliche Sicherheitsmerkmal elektronischer Geldsysteme dar, da sie bei anderen Zahlungssystemen, wie beispielsweise Kreditkarte, Überweisung und Lastschrift, nicht vorhanden ist. Die durch die elektronischen Geldsysteme gewährleisteten anonymen Zahlungen sind aus Kundensicht eine positive und wünschenswerte Eigenschaft.

Die Konstruktion der ersten elektronischen Geldsysteme ist vor allem durch die Entwicklung einer *blinden Signatur* ermöglicht worden, die insbesondere auf D. Chaum [Cha83, Cha84] zurückzuführen ist. Bei einer blinden Signatur kennt der *Signierer* weder die zu signierende Nachricht noch die eigentliche Signatur. Das Konzept der blinden Signatur lässt sich am folgenden Beispiel gut erläutern: Eine Partei $\mathcal{A}$ möchte ein bestimmtes Dokument von einer Partei $\mathcal{B}$ blind signieren lassen. Dazu steckt $\mathcal{A}$ das Dokument zusammen mit einem Kohlepapier in einen Umschlag, auf dem anschließend von $\mathcal{B}$ unterschrieben wird. Somit ist $\mathcal{A}$ im Besitz einer Signatur auf ein gewünschtes Dokument, ohne dass $\mathcal{B}$ dieses je gesehen hat. Dadurch kann $\mathcal{B}$ das Dokument später auch nicht wiedererkennen. Auf elektronische Geldsysteme übertragen bedeutet dies, dass die Bank eine elektronische Münze blind signiert ohne diese zu kennen. Aufgrund dieser *Blindheit* wird die Bank eine abgehobene Münze beim späteren Einlösen nicht wiedererkennen können.

Da jede elektronische Münze lediglich ein digitaler Datensatz ist, den der Kunde beispielsweise auf seinem PC oder Smartphone abspeichert, resultiert im Gegensatz zu gewöhnlichem Bargeld allerdings das Problem, dass sie generell einfach dupliziert und mehrmals zum Bezahlen verwendet werden kann. Das mehrfache Ausgeben einer elektronischen Münze wird als *Double-Spending* bezeichnet. Diesem Problem kann auf verschiedene Weisen entgegen gewirkt werden, wobei elektronische Geldsysteme hauptsächlich in *Online-* und *Offline*-Systeme unterteilt werden. Generell muss die Bank bei beiden Systemen eine Datenbank verwalten, in der die bereits eingelösten Münzen abgespeichert werden.

Bei *elektronischen Online-Geldsystemen* ist die Bank während des Bezahlvorgangs mit dem Händler verbunden, um in Echtzeit zu verifizieren, ob die verwendete Münze bereits ausgegeben wurde. Da die Bank während aller Transaktionen online sein muss, können solche Systeme allerdings nicht als besonders effizient angesehen werden. Beispiele für Online-Geldsysteme sind [Cha83, CPS94, CPS96].

Bei *elektronischen Offline-Geldsystemen*, wie beispielsweise [CFN89] und [Bra93], entfällt die Überprüfung der Bank zum Bezahlzeitpunkt. Dadurch können diese Systeme effizienter realisiert werden und stellen gerade für die Bezahlung mit Kleinbeträgen eine geeignete Bezahlmethode dar. Der Nachteil ist allerdings, dass sich dadurch die mehrfache Ausgabe von Münzen nicht verhindern lässt. Aus diesem Grund setzten Offline-Systeme eine sogenannte *Double-Spending-Detection* ein, welche hauptsächlich durch *Geheimnisteilungsverfahren* (*Secret-Sharing*) [Sha79] realisiert wird. Dadurch ist die Anonymität beim einmaligen Bezahlen einer Münze gewährleistet, wohingegen beim zweiten Bezahlen mit derselben Münze die Bank die *Identität* (Kontonummer) des *Double-Spenders*

aufdecken kann. Dadurch wird ein Double-Spender *nach* seiner Tat identifiziert, während jeder ehrliche Kunde anonym bleibt. Im weiteren Verlauf dieser Arbeit bezeichnet, soweit nicht anders vermerkt, ein elektronisches Geldsystem stets ein elektronisches Offline-Geldsystem.

Grundsätzlich lässt sich die Problematik der Mehrfachausgabe in einem elektronischen Geldsystem durch rein kryptografische Mittel aber nicht verhindern. Eine Methode dieses Problem zu lösen, ist die sogenannte *Double-Spending-Prevention*. Durch die Integration einer zusätzlichen manipulationssicheren Hardware, die beim Bezahlvorgang involviert ist, wird eine Mehrfachausgabe unmittelbar verhindert. Eine solche Hardware wird als *Observer* bezeichnet und wurde erstmals 1992 von D. Chaum und T. Pedersen [CP93] in ein Geldsystem integriert. Elektronische Geldsysteme, die manipulationssichere Hardware zur Double-Spending-Prevention einsetzten, werden als *elektronische Geldsysteme mit Observern* (z.B. [Bra93, CP94, NS03, NS06, Sch09]) bezeichnet.

Das Sicherheitsziel Fälschungssicherheit garantiert einerseits, dass kein Kunde mehr Münzen ausgeben kann, als er bei der Bank abgehoben hat und andererseits, dass auch kein Händler mehr Münzen bei der Bank einlösen kann, als er eingenommen hat. Insbesondere hängt die Fälschungssicherheit des Geldsystems von der verwendeten Signatur ab. Damit das Fälschen von elektronischen Münzen verhindert werden kann, muss die eingesetzte Signatur *unfälschbar* sein.

Die weitere zentrale Sicherheitsanforderung Anonymität kann hauptsächlich auf zwei verschiedene Weisen realisiert werden. Durch die Verwendung einer blinden Signatur, wird die Anonymität bereits während des Abhebens der Münze realisiert. Elektronische Geldsysteme, die die Anonymität durch eine blinde Signatur gewährleisten, sind beispielsweise [CFN89, OO91, Bra94, CP94, FTY98, GT03]. Grundsätzlich gibt es aber noch einen weiteren Ansatz Anonymität zu erreichen, nämlich während des Bezahlvorgangs (vgl. [PW91]). Beispiele für solche Geldsysteme sind [STS99a, CHL05, ASM07, BCKL09, CG10]. Während das System [STS99a] nicht den Baustein der digitalen Signatur anwendet, wurde in [CHL05] erstmals ein *spezielles Signaturverfahren* in Kombination mit einem Beweisverfahren eingesetzt, welches von J. Camenisch und A. Lysyanskaya [CL02b, CL04] entwickelt wurde.

Im Gegensatz zu den zuvor vorhandenen Signaturverfahren kann der Benutzer bei diesem beweisen, im Besitz einer Signatur auf ein bestimmtes Dokument zu sein, ohne diese preisgeben zu müssen. Ein solcher Beweis wird als *Zero-Knowledge-Beweis* (siehe Abschnitt 2.10) bezeichnet, da keine weiteren Informationen preisgegeben werden. Die *CL-Signatur* [CL02b] dient als Vorlage weiterer spezieller Signaturverfahren, wie beispielsweise dem *BBS+*-Signaturverfahren [ASM06] oder dem *ESS+*-Signaturverfahren [AWSM07, ASM08].

Diese Signaturverfahren haben einen wesentlichen Vorteil gegenüber früheren, denn aus der Eigenschaft, während des Bezahlvorgangs beweisen zu können, im Besitz einer gültigen Signatur zu sein, resultiert, dass jeder Signaturbesitzer diese beliebig oft verwenden kann. Aufgrund der enormen Effizienz dieser Signaturverfahren werden diese in nahezu allen elektronischen Geldsystemen, die seit 2005 entwickelt werden, eingesetzt.

So dient die CL-Signatur als Grundbaustein des ersten *kompakten elektronischen Geldsystems*, das von J. Camenisch, S. Hohenberger und A. Lysyanskaya [CHL05] entwickelt wurde. Im Gegensatz zu vorherigen Geldsystemen ermöglichen kompakte Geldsysteme ein überaus effizientes Abheben einer *elektronischen Geldbörse* mit mehreren Münzen. Dadurch konnte in [CHL05] die Komplexität eines Abhebevorgangs von K Münzen von $O(\lambda \cdot K)$ auf $O(\lambda + \log(K))$ reduziert werden, wobei λ der Sicherheitsparameter ist. Zusätzlich wurden auch die Komplexität des Bezahlvorgangs und der benötigte Speicherplatz der elektronischen Münzen stark reduziert. Weitere kompakte elektronische Geldsysteme sind beispielsweise [Wei05, ASM07, AWSM07, CDG+09, BCKL09].

Durch die bereits erwähnten speziellen Signaturverfahren konnten neben den kompakten auch erstmals *teilbare elektronische Geldsysteme*, wie beispielsweise [CG07, ASM08, CG10, IL13] realisiert werden, welche die bereits erwähnten Sicherheitsanforderungen erfüllen. Bei einem teilbaren Geldsystem kann eine elektronische Geldbörse mit dem Wert K während des Bezahlvorgangs in eine Münze mit dem Wert k für $1 \leq k \leq K$ *geteilt* werden, um mit einem passenden Betrag bezahlen zu können. Dadurch wird einerseits die Zusatzeigenschaft *Wechselgeld* überflüssig und andererseits ein noch effizienterer Bezahlvorgang ermöglicht, da die Komplexität des Bezahlens mit ansteigendem Münzwert nur logarithmisch zunimmt, statt linear. Die Idee der *teilbaren Münze* entstand bereits Anfang der 90er Jahre [OO91, Oka95, CFT98], jedoch gewährleisten diese teilbaren Geldsysteme nicht die gewünschte Anonymität, da während des Bezahlvorgangs zusätzliche Informationen preisgegeben werden. Erst im Jahr 2007 konnte von S. Canard und A. Gouget [CG07] durch das in [CHL05] eingesetzte spezielle Signaturverfahren mit Zero-Knowledge-Beweisen ein teilbares elektronisches Geldsystem konstruiert werden, welches den geforderten Sicherheitsanforderungen gerecht wird.

Zusammenfassend werden die Sicherheitsanforderungen Verifizierbarkeit der Echtheit sowie Fälschungssicherheit durch den Einsatz eines digitalen Signaturverfahrens und Anonymität durch die zusätzliche Blindheit oder durch die Kombination mit Zero-Knowledge-Beweisen gewährleistet.

Der Großteil elektronischer Geldsysteme bietet *perfekte Anonymität* (z.B. [CFN89, Bra94]) oder *rechnerische Anonymität* (z.B. [CHL05, ASM08]). Perfekte Anonymität bedeutet, dass selbst ein rechnerisch *unbeschränkter Angreifer*, der über unendlich viel Zeit und Speicherplatz verfügt, niemals bezahlte Münzen dem jeweiligen ehrlichen Kunden zuordnen kann. Dagegen bedeutet rechnerische Anonymität, dass es für jeden in seiner Laufzeit und Speicherkapazität *beschränkten Angreifer* nahezu unmöglich ist, bezahlte Münzen dem jeweiligen ehrlichen Kunden zuzuordnen. Prinzipiell werden durch beide Arten allerdings kriminelle Aktivitäten wie Erpressung oder Geldwäsche begünstigt (vgl. [SN92, SPC95]), da ein Betrüger nicht identifiziert werden kann. Während bei herkömmlichen Bezahlmethoden beispielsweise Geldscheine anhand ihrer Seriennummer identifiziert und Transaktionen von Konto zu Konto verfolgt werden können, existieren solche Möglichkeiten bei elektronischen Geldsystemen mit perfekter oder rechnerischer Anonymität nicht.

Eine erforderliche Eigenschaft ist es also, die Anonymität in Verdachtsfällen aufheben

zu können, was als *aufhebbare Anonymität* bezeichnet wird. Um die kriminellen Möglichkeiten zu unterbinden, sind eine *Kundenverfolgung* und eine *Münzverfolgung* notwendig. Je nach Szenario kann mit diesen Verfahren der Eigentümer einer bestimmten Münze identifiziert oder der Bezahlort einzelner Münzen aufgedeckt und bestimmte Münzen gesperrt werden. Um ehrliche Kunden zu schützen darf es der Bank allerdings nicht möglich sein, diese Verfolgungstechniken alleine anwenden zu können. Aus diesem Grund wird zusätzlich eine vertrauenswürdige dritte Partei, die sogenannte *Trusted-Third-Party* (TTP), benötigt, die als *Deanonymisierer* fungiert und nur in Verdachtsfällen gemeinsam mit der Bank die jeweilige Verfolgungstechnik durchführt. Geldsysteme, die mithilfe eines Deanonymisierers Kunden- und Münzverfolgung gewährleisten, werden *faire elektronische Geldsysteme* (z.B. [BGK95, CMS96, CPS96, FTY96, FTY98, GT03, NS03, XY03, NS06, Sch09]) genannt.

1.2 Ziele und Aufbau der Arbeit

Der Themenbereich der elektronischen Geldsysteme ist im mathematischen Teilgebiet der Kryptografie angesiedelt, da durch kryptografische Methoden unter anderem bedeutende Ziele wie *Vertraulichkeit*, *Authentizität*, *Integrität* und *Verbindlichkeit* realisiert werden können (vgl. [BNS10]). Im folgenden Kapitel 2 werden zunächst die grundlegenden kryptografischen Verfahren und Protokolle beschrieben, die als Bausteine der in dieser Arbeit entwickelten teilbaren elektronischen Geldsysteme dienen. Ein zentrales Element elektronischer Geldsysteme sind digitale Signaturen, welche die oben genannten Ziele Authentizität, Integrität und Verbindlichkeit garantieren. Um die Konstruktion möglichst effizienter elektronischer Geldsysteme zu realisieren, wird in Kapitel 3 ein spezielles digitales Signaturverfahren entwickelt, welches anschließend bei den folgenden elektronischen Geldsystemen zum Einsatz kommt.

Ein grundlegendes Ziel dieser Arbeit ist die Konstruktion eines effizienten teilbaren elektronischen (Offline-) Geldsystems, das die folgenden Eigenschaften besitzt:

- Das Geldsystem gewährleistet die zentralen Sicherheitsziele Fälschungssicherheit sowie Anonymität und kein Kunde kann für ein Double-Spending bestraft werden, welches er nicht begangen hat.

- Das Geldsystem soll im Vergleich zu bisherigen mehr Sicherheit für Kunden garantieren:

  1. Auch nachdem ein Kunde ein Double-Spending begangen hat, sollen alle Sicherheitsziele, außer der Anonymität bzgl. der mehrfach ausgegebenen Münzen, weiterhin gewährleistet sein.

  2. Die Abhebeprotokolle sollen verifizierbar sein, damit eine betrügerische Bank keinem Kunden Geld abbuchen kann, welches er nicht abgehoben hat.

- Die Identifikation eines Double-Spenders ist ohne eine zusätzliche dritte Partei durchführbar.

- Das Geldsystem verfügt über effiziente Abhebe-, Bezahl- und Einlöseprotokolle. Idealerweise sind der Datentransfer und die Berechnungskomplexität unabhängig von der Größe der abgehobenen Geldbörse bzw. dem Wert der bezahlten Münze.

Um möglichst effiziente teilbare elektronische Geldsysteme zu konstruieren, wird in Kapitel 3 ein neues Signaturverfahren entwickelt, das speziell für die in dieser Arbeit vorgestellten elektronischen Geldsysteme konzipiert und somit effizienter als das einzig vergleichbare Signaturverfahren ist.

Da in der Literatur verschiedene Sicherheitsdefinitionen elektronischer Geldsysteme vorhanden sind, werden in Kapitel 4 zunächst elektronische Geldsysteme definiert und neue spielbasierende Definitionen entwickelt, die die in der Literatur bestehenden Sicherheitsziele umfassen.

Als Beispiel für eine weitere Einsatzmöglichkeit des in Kapitel 3 entwickelten Signaturverfahrens, wird in Kapitel 5 das kompakte elektronische Geldsystem von Au. *et al.* [ASM07] modifiziert und erweitert, um Kunden das Abheben eines beliebigen Geldbetrages zu ermöglichen, wobei die Komplexität des Bezahlvorgangs unabhängig vom Wert der abgehobenen Geldbörse ist. Bisher ist in der Literatur nach bestem Wissen kein kompaktes elektronisches Geldsystem mit diesen Eigenschaften zu finden.

In Kapitel 6 wird ein neues teilbares elektronisches Geldsystem konstruiert, das als Grundlage aller weiteren, in dieser Arbeit entwickelten Geldsysteme dient. Dieses Geldsystem ist nach bestem Wissen das erste teilbare elektronische Geldsystem, das die zentralen Sicherheitsziele gewährleistet und bei dem der Datentransfer der Abhebe- und Bezahlprotokolle unabhängig vom Wert der abgehobenen Geldbörse und der ausgegebenen Münze ist. Somit ist dieses Geldsystem effizienter als alle in der Literatur vorhandenen, vergleichbaren teilbaren elektronischen Geldsysteme mit denselben zentralen Sicherheitszielen. Dieses Geldsystem wird anschließend zu einem noch effizienteren teilbaren elektronischen Geldsystem modifiziert.

Des Weiteren wird in Kapitel 6 ein effizienter Angriff auf das teilbare elektronische Geldsystem [CG10] entwickelt, um elektronische Münzen zu fälschen.

Ein Hauptziel ist es dann, diese entwickelten teilbaren elektronischen Geldsysteme zu fairen Geldsystemen zu erweitern, so dass kriminelle Aktivitäten verhindert werden können. Dies wird in Kapitel 7 beschrieben, wobei die Integration des Deanonymisierers so effizient wie möglich erfolgt und folgende Bedingungen erfüllt sind:

- Das faire Geldsystem gewährleistet in Verdachtsfällen eine Kunden- und Münzverfolgung sowie das Sperren bestimmter Münzen durch den Deanonymisierer.

- Der Deanonymisierer ist offline. Das heißt, Kunden müssen sich nicht bei dem Deanonymisierer registrieren und die Abhebe-, Bezahl- und Einlöseprotokolle werden ohne Beteiligung des Deanonymisierers durchgeführt.

Insbesondere wird eine wesentlich effizientere Integration der Kunden- und Münzverfolgung entwickelt, als bspw. in [Au09, Abschnitt 5.6] vorgeschlagen wird. Weiter wird in diesem Kapitel bewiesen, dass jedes elektronische Geldsystem mit Erstattungsfähigkeit nicht anonym sein kann, wodurch die Aussagen von [LTW05] widerlegt werden.

Ein zusätzliches Ziel ist die Realisierung verschiedener Abhebe- und Bezahlprotokolle, um den Kunden flexible Möglichkeiten zu bieten, elektronisches Geld bei der Bank abzuheben und bei Händlern auszugeben. Bei den in Kapitel 6 und 7 entwickelten elektronischen Geldsystemen können die Kunden ausschließlich jeweils eine Geldbörse mit dem Wert $K = 2^L$ abheben und anschließend mit einer Münze mit dem Wert $k = 2^\ell$ für $0 \leq \ell \leq L$ bezahlen, wobei L ein festgelegter Wert ist. Aus diesem Grund werden in Kapitel 8 verschiedene Modifikationen der konstruierten teilbaren elektronischen Geldsysteme vorgestellt, so dass die folgenden Kriterien realisiert werden:

- Die Kunden können im (modifizierten) Abhebeprotokoll eine Geldbörse mit einem beliebigen Wert $K' \leq K$ abheben.
- Die Kunden können anschließend im (modifizierten) Bezahlprotokoll mit einer Münze mit einem beliebigen Wert $k \leq K'$ bezahlen.

Ein weiteres Hauptziel dieser Arbeit ist die Integration von Observern in die konstruierten elektronischen Geldsysteme, um eine mehrfache Ausgabe der Münzen unmittelbar zum Bezahlzeitpunkt zu verhindern. Wie die Integration der Fairness, soll auch diese Integration so effizient wie möglich erfolgen. Basierend auf unterschiedlichen Voraussetzungen, werden in Kapitel 9 drei verschiedene Varianten angegeben, wie die konstruierten elektronischen Geldsysteme effizient zu elektronischen Geldsystemen mit Observern erweitert werden können. Nach bestem Wissen sind bisher keine Veröffentlichungen bekannt, die sich mit einem Observereinsatz bei teilbaren Geldsystemen beschäftigen. Des Weiteren wird bewiesen, dass elektronische Geldsysteme mit Observern nicht anonym sein können, wenn die Observer von der Bank ausgehändigt und wieder an diese zurückgegeben werden.

Durch die flexiblen und effizienten Abhebe- und Bezahlprotokolle können auf die zusätzlichen Eigenschaften elektronischer Geldsysteme *Wechselgeld* und *Transferierbarkeit* nahezu verzichtet werden. Denn da die Kunden im modifizierten Bezahlprotokoll mit dem exakten Wert bezahlen können, benötigen die Händler kein Wechselgeld. Des Weiteren kann anstelle eines Münztransfers zwischen zwei Kunden ein Kreislauf von Bezahl-, Einlöse- und Abhebeprotokoll durchgeführt werden.

Auch eine optionale Münzverfolgung nach einem Double-Spending, wie etwa im kompakten Geldsystem von [CHL05], wird nicht benötigt, wenn ein Double-Spending als Verdachtsfall gilt und der Deanonymisierer folglich eine Münzverfolgung ausführen kann.

An dieser Stelle möchte ich mich bei Herrn Prof. Dr. Albrecht Beutelspacher für die Möglichkeit, diese Arbeit zu verfassen und für die ausgezeichnete Betreuung bedanken. Mein Dank gilt ebenso Herrn Prof. Dr. Jörg Schwenk für die Bereitschaft diese Arbeit zu begutachten. Des Weiteren möchte ich mich besonders bei meinem damaligen Dozenten Dr. Thomas Schwarzpaul für die wertvollen Anregungen und die Unterstützung bedanken. Ebenfalls danke ich meinen Kollegen Volker Greilich, Ferdinand Ihringer und Christian Reuter für die wertvollen Gespräche und Diskussionen. Weiter danke ich meiner Familie und meinen Freunden für ihre Aufmunterung, Unterstützung und das Korrekturlesen.

KAPITEL 2

KRYPTOGRAFISCHE GRUNDLAGEN

Zur Konstruktion eines fairen teilbaren elektronischen Geldsystems sind unterschiedliche komplexe Elemente erforderlich, um die vielseitigen Anforderungen wie Anonymität, Fälschungssicherheit, Double-Spending-Detection, Fairness und Teilbarkeit realisieren zu können. In diesem Kapitel werden alle kryptografischen Grundlagen und Bausteine vorgestellt, die zur Konstruktion der in Kapitel 6 bis 9 entwickelten elektronischen Geldsysteme verwendet werden.

Als Grundlage zum Thema Kryptografie sind die Bücher von A. Beutelspacher, J. Schwenk und K.-D. Wolfenstetter [BSW06] sowie von A. Beutelspacher, H. Neumann und T. Schwarzpaul [BNS10] zu empfehlen. Eine tiefgehende Behandlung der Theorie der Kryptografie ist in den Büchern von O. Goldreich [Gol01, Gol04] und weitere Informationen zu den Themen *elliptische Kurven* (Abschnitt 2.4) sowie *bilineare Abbildungen* (Abschnitt 2.5) sind im Buch von F. Blake, G. Seroussi und N. Smart [BSS05] sowie den Dissertationen von S. Hohenberger [Hoh06] und B. Lynn [Lyn07] zu finden. Als grundlegende Literatur für Algebra dienen [Kun91, Str98] und für Komplexitätstheorie [HU94, Rei99].

2.1 Verschiedene Notationen

In diesem Abschnitt werden zunächst verschiedene (algebraische) Notationen eingeführt, die im weiteren Verlauf dieser Arbeit verwendet werden.

Sei $\mathbb{G}$ eine Menge und $*$ eine Verknüpfung auf $\mathbb{G}$. D.h. $*$ ist eine Abbildung $* : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$.

Definition 2.1.1 (**Gruppe**). *Eine* Gruppe *besteht aus einer Menge* $\mathbb{G}$ *zusammen mit einer Verknüpfung* $*$*, so dass die folgenden Eigenschaften erfüllt sind:*

- (Assoziativität:) *Für alle Gruppenelemente* $g_1, g_2, g_3 \in \mathbb{G}$ *gilt:* $(g_1 * g_2) * g_3 = g_1 * (g_2 * g_3)$.
- *Es gibt ein* neutrales Element *in* $\mathbb{G}$*; d.h. ein Element* e*, so dass für alle Gruppenelemente* $g \in \mathbb{G}$ *gilt:* $e * g = g * e = g$.
- *Zu jedem Gruppenelement* $g \in \mathbb{G}$ *existiert ein* inverses Element $g^{-1} \in \mathbb{G}$*, so dass gilt:* $g * g^{-1} = g^{-1} * g = e$.

Bei einer multiplikativen Gruppe $\mathbb{G}$ bezeichnen wir das neutrale Element mit 1 und ein zu $g \in \mathbb{G}$ inverses Element mit g^{-1} oder $1/g$. Bei einer additiven Gruppe werden entsprechend die Bezeichnungen 0 und $-g$ verwendet.

Eine Gruppe wird häufig mit $(\mathbb{G}, *)$ oder einfach mit $\mathbb{G}$ bezeichnet. Eine Gruppe $\mathbb{G}$ heißt *abelsch* oder *kommutativ*, falls für alle Gruppenelemente $g_1, g_2 \in \mathbb{G}$ die Gleichung $g_1 * g_2 = g_2 * g_1$ erfüllt ist. Die Anzahl der Elemente von $\mathbb{G}$ wird mit $|\mathbb{G}|$ bezeichnet und heißt die *Ordnung* der Gruppe. Eine Gruppe $\mathbb{G}$ ist *endlich*, falls $|\mathbb{G}|$ endlich ist. In dieser Arbeit werden ausschließlich endliche, kommutative Gruppen betrachtet. Eine Gruppe $\mathbb{G}$ heißt *zyklisch*, falls es ein Element $g \in \mathbb{G}$ gibt, so dass die ganze Gruppe $\mathbb{G}$ von g *erzeugt* wird, also $\mathbb{G} = \{g^i \ : \ 0 \leq i < |\mathbb{G}|\}$ gilt, wobei g^i die i-fache Verknüpfung von g mit sich selbst bezeichnet. Ein solches Element g heißt *Generator* der Gruppe $\mathbb{G}$ und es wird die Notation $\mathbb{G} = \langle g \rangle$ verwendet, um zu kennzeichnen, dass g ein Generator von $\mathbb{G}$ ist. Ist $\mathbb{G}$ eine zyklische Gruppe der Ordnung p und p eine Primzahl, dann ist jedes Gruppenelement $g \in \mathbb{G} \setminus \{e\}$ ein Generator.

Die Menge der *natürlichen Zahlen* $\{1, 2, 3, \dots\}$ wird mit $\mathbb{N}$ und die Menge der *ganzen Zahlen* $\{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$ mit $\mathbb{Z}$ bezeichnet.

Mit $\mathbb{Z}_n$ wird die Menge $\{0, \dots, n-1\}$ und mit $\mathbb{Z}_n^*$ die Menge der positiven ganzen Zahlen, die kleiner als n und teilerfremd zu n sind, bezeichnet. Somit gelten $\mathbb{Z}_p = \{0, \dots, p-1\}$ und $\mathbb{Z}_p^* = \{1, \dots, p-1\}$ für jede Primzahl p. Für jede Zahl $n \in \mathbb{N}$ ist $(\mathbb{Z}_n, +)$ modulo n eine kommutative, zyklische, additive Gruppe der Ordnung n. Für eine Primzahl p ist $(\mathbb{Z}_p^*, \cdot)$ modulo p eine kommutative, zyklische, multiplikative Gruppe der Ordnung $p - 1$.

Da für eine Primzahl p die Gruppe $\mathbb{Z}_p$ sogar ein *Körper* ist, werden sämtliche Additionen und Multiplikationen von Zahlen in $\mathbb{Z}_p$ stets modulo p durchgeführt, auch wenn nicht an entsprechenden Stellen explizit „$\mathsf{mod}\ p$“ geschrieben wird.

Mit $g_1, \dots, g_L \in_{\mathcal{R}} \mathbb{G}$ wird die *zufällige* und *unabhängige* Wahl von Elementen $g_1, \dots, g_L$ gemäß der *Gleichverteilung* auf $\mathbb{G}$ beschrieben. Im Folgenden wird anstelle der Notation *zufällig und unabhängig gemäß der Gleichverteilung* vereinfachend lediglich die Formulierung *zufällig* verwendet.

Die Anzahl der Binärstellen einer natürlichen Zahl n wird mit $|n|$ bezeichnet.

Seien $s_1, s_2 \in \{0,1\}^*$ Bitstrings beliebiger Länge, dann bezeichnet $s_1 || s_2 \in \{0,1\}^*$ die *Konkatenation* von s_1 und s_2.

Für ein Tupel $T = (A, B, C, D, E)$ werden an einigen Stellen auch die vereinfachten Schreibweisen wie $T = (A, \cdots, E)$ oder $T = (\cdots, D, E)$ verwendet, wenn die nicht aufgelisteten Elemente irrelevant sind.

2.1.1 Komplexitätstheorie und Protokolle

Ein *Algorithmus* ist eine Berechnungsvorschrift, der eine variable *Eingabe* erhält und mit einer *Ausgabe* abbricht. Ein Algorithmus ist *deterministisch*, wenn zu jedem Zeitpunkt der nächste Berechnungsschritt eindeutig definiert ist. Dadurch gibt ein deterministischer Algorithmus immer dieselbe Ausgabe aus, wenn die Eingabe identisch ist. Mit $\mathcal{A}(\mathbb{G}', x) = y$ wird dargestellt, dass y die Ausgabe eines deterministischen Algorithmus $\mathcal{A}$ bei Eingabe von $(\mathbb{G}', x)$ ist, wobei wir mit $\mathbb{G}'$ die Beschreibung der nötigen Parameter einer Gruppe $\mathbb{G}$ bezeichnen.

Ein Algorithmus heißt *probabilistisch*, wenn für seine Durchführung Zufallseingaben benötigt werden oder nicht immer das korrekte Ergebnis ausgegeben wird. Ist $\mathcal{A}$ ein probabilistischer Algorithmus, dann wird seine Ausgabe bei der Eingabe von $(\mathbb{G}', x)$ und einer Zufallseingabe r mit $\mathcal{A}(\mathbb{G}', x; r) = y$ bezeichnet. In einigen Fällen wird im Folgenden anstelle der Notation $\mathcal{A}(\mathbb{G}', x; r) = y$ auch die Schreibweise $y \leftarrow \mathcal{A}(\mathbb{G}', x)$ verwendet, wenn bspw. die Zufallseingabe r irrelevant ist.

Um die Laufzeit von Algorithmen zu beschreiben, wird die übliche asymptotische Notation verwendet (vgl. auch [Cam98, Au09]). Der Ausdruck $f(n) = O(g(n))$ bedeutet, dass es eine positive Konstante c und eine positive Zahl n_0 gibt, so dass $0 \leq f(n) \leq c \cdot g(n)$ für alle $n \geq n_0$ gilt. Vereinfacht gesagt heißt dies, dass $g(n)$ eine obere Schranke von $f(n)$ ist. Falls $g(n)$ eine untere Schranke von $f(n)$ ist, also $g(n) = O(f(n))$ gilt, wird die Bezeichnung $f(n) = \Omega(g(n))$ verwendet. Falls $f(n) = O(g(n))$ und $f(n) = \Omega(g(n))$ gilt, schreiben wir $f(n) = \Theta(g(n))$.

Ein Algorithmus hat eine *polynomielle Laufzeit*, wenn die *worst-case-Laufzeit* $O(n^c)$ für eine Konstante c beträgt, wobei n die Eingabelänge ist. Ein solcher Algorithmus heißt *polynomiell beschränkt* oder auch *polynomiell*. Eine *voraussichtlich-polynomielle* Laufzeit ist eine Laufzeit, deren *Erwartungswert* polynomiell ist (vgl. [GMW91]). Ein *voraussichtlich-polynomieller* Algorithmus hat eine voraussichtlich-polynomielle Laufzeit.

Oftmals wird das Modell einer *Turing-Maschine* verwendet, um den Begriff Algorithmus zu präzisieren. Zwei Turing-Maschinen können ein *interaktives Protokoll* durchführen, um sich gegenseitig Nachrichten zu senden. Bei einem interaktiven Protokoll erhalten beide Parteien eine bestimmte Eingabe, führen gewisse Berechnungen durch und tauschen Nachrichten gemäß der Protokollvorschrift aus. Nach erfolgreichem Protokollablauf kann jede Partei eine Ausgabe erhalten. Führen zwei Turing-Maschinen $\mathcal{A}, \mathcal{B}$ ein Protokoll P aus, so bedeutet die Notation $a \leftarrow \mathcal{A}(x) \leftrightarrow \mathcal{B}(y) \rightarrow b$, dass x die Eingabe und a die Ausgabe von $\mathcal{A}$ und entsprechend y die Eingabe und b die Ausgabe von $\mathcal{B}$ ist. Die *Protokollansicht* der Turing-Maschine $\mathcal{A}$ wird mit $\mathsf{view}_{\mathcal{A}}^{\mathcal{B}}(\mathsf{P})$ bezeichnet und enthält sämtliche Informationen, die $\mathcal{A}$ während der Durchführung „gesehen“ hat. Dies beinhaltet die Eingabe x, die Ausgabe a, die von $\mathcal{A}$ gewählten Zufallszahlen und

berechneten Werte sowie die gesendeten und empfangenen Nachrichten.

In der folgenden Abbildung 2.1 wird ein Beispiel für die Notation eines Protokolls zwischen den beiden Parteien Alice $\mathcal{A}$ und Bob $\mathcal{B}$ gezeigt.

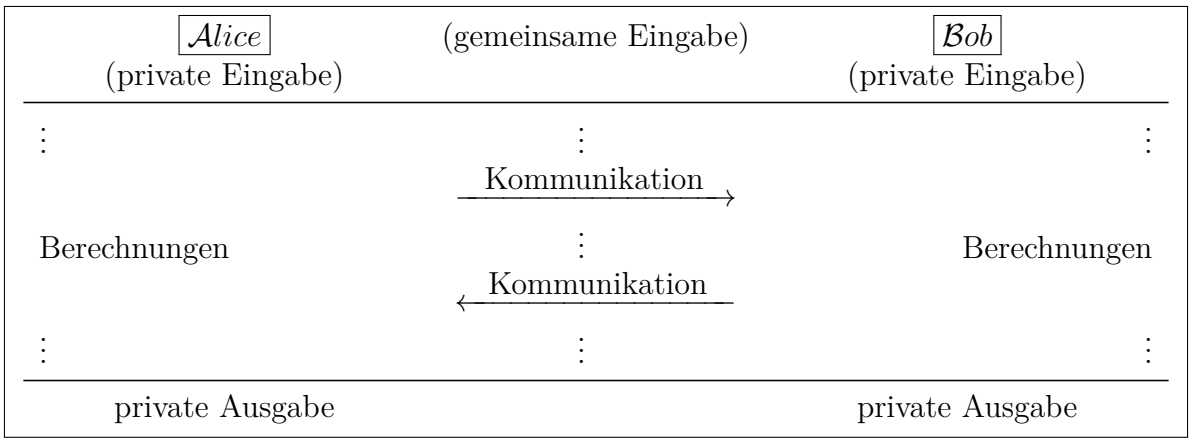

Abbildung 2.1: Ein Beispiel für die Notation eines Protokolls

2.2 Ziele der Kryptografie

Eines der primären und ältesten Ziele der Kryptografie ist die Gewährleistung von *vertraulicher Kommunikation* zwischen zwei Parteien über ein unsicheres Medium. Dies wird im Wesentlichen durch sogenannte *Verschlüsselungsverfahren* (siehe Abschnitt 2.7) realisiert, bei denen nur der Adressat einer verschlüsselten Nachricht in der Lage ist, diese zu lesen.

Weitere zentrale Ziele der Kryptografie sind *Integrität* und *Authentizität*, wobei zwischen *Teilnehmerauthentizität* und *Nachrichtenauthentizität* unterschieden wird. Teilnehmerauthentizität liegt vor, wenn ein Teilnehmer seine Identität zweifelsfrei nachweisen kann und wird bspw. durch *digitale Signaturen* (siehe Abschnitt 2.8) oder *Zero-Knowledge-Beweise* (siehe Abschnitt 2.10) realisiert (vgl. [FS89]). Nachrichtenauthentizität ist gegeben, wenn sich der Empfänger zweifelsfrei vom Ursprung einer Nachricht überzeugen kann. Die Integrität garantiert, dass Nachrichten nicht unbemerkt von einer dritten Partei manipuliert werden können. Diese beiden Ziele können ebenfalls durch digitale Signaturen, oftmals in Kombination mit Hashfunktionen (vgl. Abschnitt 2.3), gewährleistet werden.

Durch digitale Signaturen wird ebenfalls eine *Verbindlichkeit* realisiert, die ein weiteres Ziel der Kryptografie darstellt. Verbindlichkeit ist vorhanden, wenn der Empfänger einer Nachricht einer dritten Partei nachweisen kann, dass die Nachricht tatsächlich von einem bestimmten Sender stammt.

Ein weiteres Ziel der Kryptografie, das insbesondere für elektronische Geldsysteme eine zentrale Rolle spielt, ist *Anonymität*. Denn bei den in dieser Arbeit entwickelten teilbaren

elektronischen Geldsysteme ist es entscheidend, dass die Identität eines Kunden während des Bezahlvorgangs verborgen bleibt. Dies wird durch spezielle Signaturverfahren mit effizienten Protokollen (siehe Abschnitt 2.12) in Kombination mit Zero-Knowledge-Beweisen (siehe Abschnitt 2.10) realisiert.

Eine notwendige Bedingung für die Konstruktion vieler kryptografischer Anwendungen, wie bspw. eines Verschlüsselungs- oder Authentifikationsverfahrens, ist die Existenz eines *geheimen Schlüssels*. Je nachdem, ob zwei Parteien dazu einen gemeinsamen Schlüssel verwenden oder nicht, wird zwischen *symmetrischer* und *asymmetrischer Kryptografie*, die auch als *Public-Key-Kryptografie* bezeichnet wird, unterschieden. Im Gegensatz zur symmetrischen Kryptografie verfügt jeder Teilnehmer bei der Public-Key-Kryptografie über ein *Schlüsselpaar* $(\mathsf{pk}, \mathsf{sk})$, bestehend aus einem *privaten Schlüssel* sk, der vom Teilnehmer geheim gehalten und somit auch als *geheimer Schlüssel* bezeichnet wird, und einem dazugehörigen *öffentlichen Schlüssel* pk, der für alle anderen Teilnehmer öffentlich zugänglich gemacht wird. Eine der wichtigsten Eigenschaften der asymmetrischen Kryptografie ist, dass niemand in der Lage ist, aus einem öffentlichen Schlüssel den entsprechenden privaten Schlüssel zu berechnen, was als *Public-Key-Eigenschaft* bezeichnet wird (vgl. [BNS10]).

Damit kryptografische Verfahren ein bestimmtes Sicherheitsniveau garantieren, wird ein bestimmter *Sicherheitsparameter* $\lambda \in \mathbb{N}$ verwendet. Der Sicherheitsparameter wird häufig auch in unärer Schreibweise 1^λ angegeben und alle polynomiellen Algorithmen in dieser Arbeit erhalten 1^λ als Eingabe.

Die Sicherheit asymmetrischer Verfahren basiert auf bestimmten kryptografischen Problemen, wie beispielsweise dem *Faktorisierungsproblem* [RSA78], dem *Diskreter-Logarithmus-Problem* oder verschiedenen *Diffie-Hellman-Problemen* (siehe Abschnitt 2.6) und den Annahmen, dass diese Probleme schwer zu lösen sind, d.h. in polynomieller Laufzeit nur mit *vernachlässigbarer* Wahrscheinlichkeit gelöst werden können.

Definition 2.2.1 (**Vernachlässigbare Funktion**). *Eine Funktion* $\nu\colon \mathbb{N} \to \mathbb{R}$ *heißt* vernachlässigbar*, falls es für jede Zahl* $c \in \mathbb{N}$ *ein* $\lambda_0 \in \mathbb{N}$ *gibt, so dass für alle* $\lambda \geq \lambda_0$ *gilt:*

$$\nu(\lambda) < \lambda^{-c}.$$

Der Ausdruck „es gibt ein $\lambda_0 \in \mathbb{N}$, so dass für alle $\lambda \geq \lambda_0$“ wird zur Simplifizierung zukünftig durch die Formulierung „für alle hinreichend großen $\lambda \in \mathbb{N}$“ ersetzt (vgl. auch [Gol01]).

In Abschnitt 2.6 werden kryptografische Probleme und Annahmen behandelt, die für die beweisbare Sicherheit kryptografischer Anwendungen von zentraler Bedeutung sind. Anschließend werden in den folgenden Abschnitten dieses Kapitels die kryptografischen Bausteine beschrieben, die zur Realisierung der genannten Ziele und zur Konstruktion der in Kapitel 5 bis 9 entwickelten elektronischen Geldsysteme benötigt werden.

2.3 Hashfunktionen

Eine *Hashfunktion* ist eine Kompressionsfunktion, die eine Zeichenkette beliebiger Länge auf eine Zeichenkette fester Länge, den sogenannten *Hashwert*, abbildet.

Bevor wir eine formale Definition angeben, beschreiben wir die Vorstellung einer Hashfunktion. Vereinfacht formuliert ist es bei einer Hashfunktion $\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^n$ mit $n \in \mathbb{N}$ einfach, bei Eingabe eines Urbilds $m \in \{0,1\}^*$ das Bild $\mathsf{H}(m) \in \{0,1\}^n$ zu berechnen. Damit Hashfunktionen für kryptografische Verfahren geeignet sind, sollten weitere Eigenschaften erfüllt sein. So sollte das Invertieren, also bei Eingabe eines zufällig gewählten Bilds $y \in_{\mathcal{R}} \{0,1\}^n$ ein Urbild $m \in \{0,1\}^*$ mit $\mathsf{H}(m) = y$ zu finden, schwierig sein (Einweg-Eigenschaft). Weiter nennt man eine Hashfunktion schwach kollisionsresistent, wenn es schwierig ist, zu einem zufällig gewählten Urbild $m \in_{\mathcal{R}} \{0,1\}^*$ ein weiteres Urbild $m' \in \{0,1\}^*$ mit $\mathsf{H}(m) = \mathsf{H}(m')$ zu finden (vgl. [NY89]). Ist es sogar schwierig, zwei verschiedene Urbilder $m, m' \in \{0,1\}^*$ mit $\mathsf{H}(m) = \mathsf{H}(m')$ zu finden, handelt es sich um eine stark kollisionsresistente Hashfunktion (vgl. [Dam88]).

Die folgende formale Definition orientiert sich an [Gol04].

Definition 2.3.1 (**Hashfunktion**). *Sei $\ell\colon \mathbb{N} \to \mathbb{N}$. Eine Familie von Funktionen $F_{\mathsf{H}} := \left\{\mathsf{H}_s\colon \{0,1\}^* \to \{0,1\}^{\ell(|s|)}\right\}_{s\in\{0,1\}^*}$ heißt* Familie von Hashfunktionen*, wenn es einen probabilistischen polynomiellen Algorithmus I gibt, so dass die folgenden Eigenschaften gelten:*

1. *Für ein Polynom poly, alle hinreichend großen $\lambda \in \mathbb{N}$ und jedes $s \leftarrow I\left(1^\lambda\right)$ gilt $\lambda \leq \mathsf{poly}(|s|)$. Zudem kann λ in polynomieller Laufzeit aus s berechnet werden.*

2. *Es gibt einen (deterministischen) polynomiellen Algorithmus, der bei Eingabe von $s \in \{0,1\}^*$ und eines Urbilds $m \in \{0,1\}^*$ das Bild $\mathsf{H}_s(m) \in \{0,1\}^{\ell(|s|)}$ ausgibt.*

Eine Familie von Hashfunktionen F_{H} heißt

- Familie von Einweg-Hashfunktionen*, falls es für jeden polynomiellen Algorithmus $\mathcal{A}$ eine vernachlässigbare Funktion ν_1 gibt, so dass für alle hinreichend großen $\lambda \in \mathbb{N}$ gilt:*

$$\Pr\left[s \leftarrow I\left(1^\lambda\right);\ y \in_{\mathcal{R}} \{0,1\}^{\ell(|s|)};\ m \leftarrow \mathcal{A}(s,y)\ :\ \mathsf{H}_s(m) = y\right] \leq \nu_1(\lambda);$$

- schwach kollisionsresistent*, falls es für jeden polynomiellen Algorithmus $\mathcal{A}$ eine vernachlässigbare Funktion ν_2 gibt, so dass für alle hinreichend großen $\lambda \in \mathbb{N}$ gilt:*

$$\Pr\left[s \leftarrow I\left(1^\lambda\right);\ m \in_{\mathcal{R}} \{0,1\}^*;\ m' \leftarrow \mathcal{A}(s,m)\ :\ m \neq m' \ \wedge\ \mathsf{H}_s(m) = \mathsf{H}_s(m')\right] \leq \nu_2(\lambda);$$

- stark kollisionsresistent*, falls es für jeden polynomiellen Algorithmus $\mathcal{A}$ eine vernachlässigbare Funktion ν_3 gibt, so dass für alle hinreichend großen $\lambda \in \mathbb{N}$ gilt:*

$$\Pr\left[s \leftarrow I\left(1^{\lambda}\right);\ (m, m') \leftarrow \mathcal{A}(s)\ :\ m \neq m'\ \wedge\ \mathsf{H}_s(m) = \mathsf{H}_s(m')\right] \leq \nu_3(\lambda);$$

- Familie von kryptografischen Hashfunktionen*, wenn F_{H} eine Familie von stark kollisionsresistenten Einweg-Hashfunktionen ist.*

Wir nennen eine Hashfunktion $\mathsf{H}_s \in F_{\mathsf{H}}$ kryptografische Hashfunktion*, wenn F_{H} eine Familie von kryptografischen Hashfunktionen ist und* $\mathsf{H}_s \in_{\mathcal{R}} F_{\mathsf{H}}$ *zufällig aus dieser Familie gewählt ist.*

Im weiteren Verlauf dieser Arbeit werden Hashfunktionen vereinfachend mit H statt H_s bezeichnet.

Hashfunktionen werden in der Kryptografie beispielsweise für *Hash-and-Sign-Verfahren* (vgl. [BNS10]) und bestimmte *Zero-Knowledge-Beweise* (vgl. Unterabschnitt 2.10.2) verwendet. Des Weiteren werden im teilbaren elektronischen Geldsystem [ASM08] und auch für die in Kapitel 6 bis 9 entwickelten teilbaren elektronischen Geldsysteme kryptografische Hashfunktionen für die Konstruktion der elektronischen Münzen eingesetzt.

2.3.1 Das Random-Oracle-Modell

Das *Random-Oracle-Modell* wurde 1986 von A. Fiat und A. Shamir [FS86] eingeführt und 1993 von M. Bellare und P. Rogaway [BR93] formalisiert und vorgeschlagen, um die Sicherheit kryptografischer Verfahren, bei denen Hashfunktionen eingesetzt werden, zu analysieren. Insbesondere wird die Sicherheit der in Kapitel 6 bis 9 beschriebenen teilbaren elektronischen Geldsysteme im Random-Oracle-Modell bewiesen.

In diesem Modell wird jede verwendete Hashfunktion H durch ein *Random-Oracle* $\mathcal{O}^{\mathsf{H}}\colon \{0,1\}^* \to \{0,1\}^{\infty}$ ersetzt, auf welches von allen Parteien inklusive dem Angreifer zugegriffen werden kann. Der Grundgedanke dieses Modells ist, dass sich das Random-Oracle wie eine *ideale* Hashfunktion verhält, deren Ausgabewerte unabhängig und gleichverteilt sind. Dieser Gedanke geht auf A. Fiat und A. Shamir [FS86] und die sogenannte Fiat-Shamir-Heuristik (vgl. Unterabschnitt 2.10.2) zurück. Das *Random-Oracle-Paradigma* suggeriert, dass ein praktisches Protokoll P durch die Entwicklung und Prüfung eines Protokolls $\mathsf{P}^{\mathcal{O}}$ im Random-Oracle-Modell und anschließender Ersetzung des Random-Oracles $\mathcal{O}^{\mathsf{H}}$ durch eine geeignete Hashfunktion H erzeugt wird. R. Canetti, O. Goldreich und S. Halevi [CGH98, CGH04] haben allerdings Beispiele für kryptografische Verfahren präsentiert, die im Random-Oracle-Modell beweisbar sicher sind, aber aus *jeder Implementierung* mit realen Hashfunktionen ein unsicheres Verfahren resultiert. Da es sich dabei allerdings um „künstliche“ Beispiele handelt und kein praktisches System so konstruiert werden würde, ist das Random-Oracle-Modell in der Kryptografie weithin akzeptiert, um die Sicherheit kryptografischer Verfahren zu beweisen.

2.4 Elliptische Kurven in der Kryptografie

Elliptische Kurven sind in der Kryptografie von großer Bedeutung. Zum einen erreichen sie im Vergleich zu anderen Gruppen, wie beispielsweise $\mathbb{Z}_p^*$, dasselbe Sicherheitsniveau (siehe Diskreter-Logarithmus-Problem in Abschnitt 2.6) mit einer deutlich kürzeren Schlüssellänge, was der Anzahl der Binärstellen von p entspricht. Zum anderen können durch elliptische Kurven sogenannte *bilineare Abbildungen* realisiert werden, die in Abschnitt 2.5 behandelt werden.

Eine elliptische Kurve kann über einem beliebigen Körper $\mathbb{K}$ definiert werden. Für $a, b \in \mathbb{K}$ ist die Menge aller Punkte $(x, y) \in \mathbb{K}^2$, die die Gleichung $y^2 = x^3 + ax + b$ erfüllen, eine elliptische Kurve E. In der Kryptografie sind allerdings nur bestimmte elliptische Kurven über dem Körper $\mathbb{Z}_p$ für eine Primzahl $p > 3$ von Interesse.

Definition 2.4.1 (Elliptische Kurve über $\mathbb{Z}_p$). *Sei $p > 3$ eine Primzahl. Für $a, b \in \mathbb{Z}_p$ mit $4a^3 + 27b^2 \neq 0 \mod p$ ist die Menge*

$$E(\mathbb{Z}_p) := \left\{(x, y) \in \mathbb{Z}_p^2 \ : \ y^2 = x^3 + ax + b\right\} \cup \{O\}$$

eine elliptische Kurve über $\mathbb{Z}_p$. Dabei ist O ein zusätzlicher „uneigentlicher" Punkt, der auch Punkt im Unendlichen *genannt wird.*

Wir definieren auf $E(\mathbb{Z}_p)$ eine Verknüpfung $+$ wie folgt: Für zwei Punkte $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2) \in E(\mathbb{Z}_p)$ gelten folgende Additionsgesetze:

1. *$P_1 + O = P_1$ und $O + P_2 = P_2$;*
2. *falls $x_1 = x_2$ und $y_1 = -y_2$ ist $P_1 + P_2 = O$;*
3. *andernfalls ist*

$$m := \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} & \text{falls } P_1 \neq P_2, \\ \dfrac{3x_1^2 + a}{2y_1} & \text{falls } P_1 = P_2 \end{cases}$$

und $P_1 + P_2 = (x_3, y_3)$ mit $x_3 = m^2 - x_1 - x_2$ und $y_3 = m(x_1 - x_3) - y_1$.

Bemerkung 2.4.1. *Die Bedingung $p > 3$ wird gefordert, da die oben definierte Addition für $p = 2$ und $p = 3$ nicht wohldefiniert ist. Der Wert $\Delta := 4a^3 + 27b^2 \mod p$ wird als* Diskriminante *bezeichnet. Die Bedingung $\Delta \neq 0$ in Definition 2.4.1 garantiert, dass die so definierten elliptischen Kurven automatisch* nichtsingulär *sind.*

Satz 2.4.1. *Eine gemäß Definition 2.4.1 definierte elliptische Kurve $E(\mathbb{Z}_p)$ für eine Primzahl $p > 3$ ist bzgl. der Operation $+$ eine endliche, kommutative Gruppe mit neutralem Element O. Das zu zu einem Element $P = (x, y) \in E(\mathbb{Z}_p)$ inverse Element ist $-P = (x, -y)$.*

Beweis. Siehe z.B. [HPS08, Kapitel 5]. □

Die Anzahl der Elemente der Gruppe $E(\mathbb{Z}_p)$ kann im Allgemeinen nicht ohne spezielle Algorithmen berechnet werden. Nach dem Satz von Hasse-Weil kann $|E(\mathbb{Z}_p)|$ jedoch wie folgt abgeschätzt werden:

$$|E(\mathbb{Z}_p)| = p + 1 - t \text{ mit } t \in \mathbb{Z} \text{ und } -2\sqrt{p} \leq t \leq 2\sqrt{p}.$$

Im Jahr 2005 wurde von P. Barreto und M. Naehrig [BN06] eine Klasse elliptischer Kurven entwickelt, die sich besonders gut zur Implementierung bilinearer Abbildungen eignen. Diese *BN-Kurven* werden durch eine ganze Zahl $z \in \mathbb{Z}$ parametrisiert, so dass die beiden Zahlen $p(z) = 36z^4 + 36z^3 + 24z^2 + 6z + 1$ und $n(z) = 36z^4 + 36z^3 + 18z^2 + 6z + 1$ Primzahlen sind. In [BN06] wurde gezeigt, dass es für jede solche Zahl z eine elliptische Kurve $E\colon y^2 = x^3 + b$ über $\mathbb{Z}_p$ gibt, wobei $p = p(z)$ und die Anzahl der Punkte $|E(\mathbb{Z}_p)| = n(z)$ ist. Ebenfalls ist in [BN06] ein Algorithmus angegeben, der bei Eingabe der *ungefähren* Gruppenordnung die Parameter (b, n, p, y) ausgibt, so dass die elliptische Kurve $y^2 = x^3 + b$ über $\mathbb{Z}_p$ die Ordnung n hat und $g = (1, y)$ ein Generator der Gruppe $E(\mathbb{Z}_p)$ ist.

Obwohl eine elliptische Kurve über einem endlichen Körper $E(\mathbb{Z}_p)$ eine additive Gruppe ist, wird im weiteren Verlauf dieser Arbeit die Standardnotation ([BLS01]) eingehalten und $E(\mathbb{Z}_p)$ als multiplikative Gruppe geschrieben. Denn bei einer bilinearen Abbildung (vgl. Abschnitt 2.5) werden zwei Elemente elliptischer Kurven auf ein Element einer multiplikativen Gruppe abgebildet. Durch die in derselben Notation geschriebenen Gruppen werden die in dieser Arbeit behandelten kryptografischen Verfahren verständlicher (vgl. auch [Hoh06]). Des Weiteren ist es übersichtlicher, alle kryptografischen Probleme, Annahmen und Bausteine nur für *eine* Gruppe zu formulieren.

2.4.1 Sicherheit und Effizienz

Die enorme Effizienz elliptischer Kurven stellt einen Hauptgrund für deren große Bedeutung in der Kryptografie dar. So macht die US-National-Security-Agency (NSA) die folgende Aussage auf ihrer Homepage [NSA09]:

> „Elliptic Curve Cryptography provides greater security and more efficient performance than the first generation public key techniques (RSA and Diffie-Hellman) now in use.“

Bereits im Jahr 1985 wurden elliptische Kurven von V. Miller [Mil85] vorgeschlagen, um die Effizienz kryptografischer Protokolle zu verbessern und 1998 wurde der *Elliptic-Curve-Digital-Signature-Algorithm (ECDSA)* (siehe z.B. [JMV01]) als US-Standard angenommen. Denn anders als in anderen, für kryptografische Protokolle typisch verwendete Gruppen (wie bspw. $\mathbb{Z}_p^*$ mit Primzahl p), sind für elliptische Kurven lediglich generische Algorithmen und keine speziellen Algorithmen bekannt, um die in Abschnitt 2.6 beschriebenen kryptografischen Probleme, wie das Diskreter-Logarithmus-Problem oder

verschiedene Diffie-Hellman-Probleme, zu lösen. Aus diesem Grund gewährleisten elliptische Kurven dasselbe Sicherheitsniveau wie andere Gruppen mit einer deutlich kürzeren Schlüssellänge. In der folgenden Tabelle 2.1 (entnommen aus [NSA09]) werden die vom *National-Institute-of-Standards-and-Technology (NIST)* empfohlenen Schlüssellängen (in Bits) für übliche kryptografische Algorithmen, wie *DES* und *AES*, sowie für *RSA*, *Diffie-Hellman* (siehe z.B. [BNS10]) und elliptische Kurven aufgelistet, die benötigt werden, um eine äquivalente Sicherheit zu bieten.

Symmetrisch	**RSA und Diffie-Hellman**	**Elliptische Kurven**
80	1.024	160
112	2.048	224
128	3.072	256
192	7.680	384
256	15.360	521

Tabelle 2.1: Vom NIST empfohlene Schlüssellängen für äquivalente Sicherheitsniveaus

Neben den im Vergleich kürzeren Schlüssellängen bei gleichem Sicherheitsniveau besitzen elliptische Kurven den weiteren Vorteil, dass die arithmetischen Gruppenoperationen, wie beispielsweise modulare Multiplikationen und modulare Exponentiationen, aufgrund der kürzeren Bitdarstellung effizienter ausgeführt werden. Tabelle 2.2 (ebenfalls aus [NSA09] entnommen) zeigt das Verhältnis des geschätzten Aufwandes für Berechnungen in Diffie-Hellman-Gruppen zum Aufwand für Berechnungen in elliptischen Kurven für die in Tabelle 2.1 beschriebenen Schlüssellängen. Diesen Schätzungen liegt zugrunde, dass das Multiplizieren zweier n-Bit Zahlen modulo einer n-Bit Primzahl etwa n^2 Operationen und das Invertieren einer n-Bit Zahl modulo einer n-Bit Primzahl etwa acht Multiplikationen benötigt (siehe [NSA09]).

Sicherheitsniveau	**Verhältnis von DH-Aufwand : EC-Aufwand**
80	3:1
112	6:1
128	10:1
192	32:1
256	64:1

Tabelle 2.2: Relativer Berechnungsaufwand elliptischer Kurven

Für eine n-Bit Primzahl p besteht jedes Element $P = (x, y)$ einer elliptischen Kurve aus $2n$ Bits. Es ist allerdings möglich, dass jeder Punkt $P \in E(\mathbb{Z}_p)$ *komprimiert* werden

kann. Denn für jede x-Koordinate $x \in \mathbb{Z}_p$ gibt es die beiden möglichen y-Koordinaten $y_{1,2} = \pm\sqrt{x^3 + ax + b} \in \mathbb{Z}_p$. Um einen Punkt (x, y) zu komprimieren, werden lediglich die x-Koordinate $x \in \mathbb{Z}_p$ und ein weiteres Extra-Bit $b \in \{0, 1\}$ für die y-Koordinate benötigt, damit zwischen beiden Wurzeln $y = \pm\sqrt{x^3 + ax + b}$ unterschieden werden kann, bspw.

$$b = \begin{cases} 0 & \text{falls } 0 \leq y < \frac{1}{2}p, \\ 1 & \text{falls } \frac{1}{2}p < y < p \end{cases}$$

(vgl. [SB04, HPS08, CHKM09]). Dadurch kann jedes Element $P \in E(\mathbb{Z}_p)$ durch $n + 1$ Bits eindeutig dargestellt und auch wieder zum Punkt $P = (x, y)$ *dekomprimiert* werden.

2.5 Bilineare Abbildungen

Ein elementarer Baustein aktueller kompakter und teilbarer elektronischer Geldsysteme, wie beispielsweise [ASM07, AWSM07, ASM08, CG10], sind *bilineare Abbildungen*, die auch als *Pairings* bezeichnet werden. Bei einer bilinearen Abbildung $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ wird ein Paar $(g, h) \in \mathbb{G}_1 \times \mathbb{G}_2$ auf ein Element $\hat{e}(g, h) \in \mathbb{G}_T$ abgebildet, wobei alle drei Gruppen $\mathbb{G}_1, \mathbb{G}_2$ und $\mathbb{G}_T$ dieselbe Primzahlordnung p besitzen. Dabei ist es auch möglich, dass $\mathbb{G}_1 = \mathbb{G}_2$ ist.

Definition 2.5.1 (**Bilineare Abbildung**). *Seien $\mathbb{G}_1, \mathbb{G}_2$ und $\mathbb{G}_T$ Gruppen mit derselben Primzahlordnung p. Eine Abbildung $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ heißt* bilineare Abbildung *oder* Pairing*, wenn für alle Gruppenelemente $g_1, g_2 \in \mathbb{G}_1$ und $h_1, h_2 \in \mathbb{G}_2$ gilt:*

$$\hat{e}(g_1 \cdot g_2, h_1) = \hat{e}(g_1, h_1) \cdot \hat{e}(g_2, h_1) \quad \textit{und} \quad \hat{e}(g_1, h_1 \cdot h_2) = \hat{e}(g_1, h_1) \cdot \hat{e}(g_1, h_2).$$

Bilineare Abbildungen ermöglichen unter anderem die Realisierung effizienter Akkumulatoren und digitaler Signaturen (siehe Abschnitt 2.11 und 2.12), welche als Grundlage der in Kapitel 6 bis 9 vorgestellten teilbaren elektronischen Geldsysteme dienen. Des Weiteren wird die Münzverfolgung der fairen teilbaren elektronischen Geldsysteme in Abschnitt 7.3 und 7.4 durch bilineare Abbildungen realisiert.

Definition 2.5.2 (**Familien von bilinearen Abbildungen**). *Sei $\mathsf{Setup}_{\mathsf{Bil}}$ ein probabilistischer polynomieller Algorithmus, der bei Eingabe eines Sicherheitsparameters 1^λ die Systemparameter $\mathsf{sp}_{\mathsf{Bil}} := (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, h)$ ausgibt, wobei p eine zufällig gewählte Primzahl mit $p = \Theta\left(2^\lambda\right)$, $\mathbb{G}_1'$ die Beschreibung einer Gruppe $\mathbb{G}_1$, $\mathbb{G}_2'$ die Beschreibung einer Gruppe $\mathbb{G}_2$ sowie $\mathbb{G}_T'$ die Beschreibung einer Gruppe $\mathbb{G}_T$ ist und die folgenden Eigenschaften erfüllt sind:*

1. *$\mathbb{G}_1, \mathbb{G}_2$ und $\mathbb{G}_T$ sind zyklische, multiplikative Gruppen der Ordnung p.*
2. *Es gilt: $\mathbb{G}_1 = \langle g \rangle$ und $\mathbb{G}_2 = \langle h \rangle$, wobei im Fall $\mathbb{G}_1 = \mathbb{G}_2$ für die Generatoren $g = h$ gilt.*

3. *Für alle* $u_0 \in \mathbb{G}_1, v_0 \in \mathbb{G}_2$ *und* $a, b \in \mathbb{Z}_p$ *gilt:* $\hat{e}\left(u_0^a, v_0^b\right) = \hat{e}\left(u_0, v_0\right)^{ab}$.

4. *Es gilt:* $\hat{e}\left(g, h\right) \neq 1$.

Man sagt, der Algorithmus $\mathsf{Setup}_{\mathsf{Bil}}$ *beschreibt eine* Familie von bilinearen Abbildungen *oder eine* Familie von Pairings.

Die entscheidende Eigenschaft einer Familie von bilinearen Abbildungen ist ihre *Bilinearität* (Eigenschaft 3 in Definition 2.5.2). Insbesondere gelten auch für alle Elemente $u_0 \in \mathbb{G}_1, v_0 \in \mathbb{G}_2$ und für jede Zahl $a \in \mathbb{Z}_p$ die Gleichungen $\hat{e}\left(u_0, v_0^a\right) = \hat{e}\left(u_0^a, v_0\right)$ sowie $\hat{e}\left(u_0^{1/a}, v_0^a\right) = \hat{e}\left(u_0, v_0\right)$. Besonders diese Eigenschaft wurde bei der Konstruktion der Akkumulatoren und digitalen Signaturen, die im Abschnitt 2.11 bzw. 2.12 beschrieben sind, angewandt. Beispiele für Familien von Pairings, die in der Kryptografie häufig eingesetzt werden, sind das *Weil-Pairing* [BF01, BLS01] und das *Tate-Pairing* (z.B. [CHKM09]).

Es sei darauf hingewiesen, dass im gesamten weiteren Verlauf dieser Arbeit die Elemente $g, g_0, g_1, \ldots$ sowie $u_0, u_1, u_2, \ldots$ stets Generatoren der Gruppe $\mathbb{G}_1$ und $h, h_0, h_1, \ldots$ Generatoren der Gruppe $\mathbb{G}_2$ sind.

In dieser Arbeit werden Pairings hauptsächlich als *Black-Boxes* behandelt, ohne genauer auf die jeweiligen Konstruktionen einzugehen (vgl. [BSS05, Kapitel X]. Detaillierte Berechnungen einzelner Pairings sind beispielsweise in [BSS05, Lyn07, HPS08, CM09, CHKM09] zu finden. Es ist aber dennoch möglich, verschiedene Typen von Pairings zu unterscheiden. Im folgenden Unterabschnitt wird eine solche Klassifikation vorgenommen.

2.5.1 Verschiedene Pairing Typen

Im Folgenden wird durch $\mathsf{Setup}_{\mathsf{Bil}}$ stets eine Familie von Pairings beschrieben, wobei wie in Definition 2.5.2 $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, h) \leftarrow \mathsf{Setup}_{\mathsf{Bil}}(1^\lambda)$ ist. Da bei einer bilinearen Abbildung $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ die Gruppen $\mathbb{G}_1$ und $\mathbb{G}_2$ zyklisch sind und dieselbe Primzahlordnung p haben, sind beide Gruppen isomorph. Es ist aber nicht immer einfach, einen Isomorphismus zwischen diesen Gruppen anzugeben. Aus diesem Grund nahmen S. Galbraith, K. Paterson und N. Smart [GPS06, GPS08] eine Einteilung in unterschiedliche *Pairing Typen* vor:

Typ-1: Für alle $\lambda \in \mathbb{N}$ und alle $\mathsf{sp}_{\mathsf{Bil}} \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right)$ gilt $\mathbb{G}_1 = \mathbb{G}_2$ mit $g = h$.

Typ-2: Für alle $\lambda \in \mathbb{N}$ und alle $\mathsf{sp}_{\mathsf{Bil}} \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right)$ gilt $\mathbb{G}_1 \neq \mathbb{G}_2$. Zudem gibt es einen deterministischen polynomiellen Algorithmus, der bei Eingabe der Systemparameter $\mathsf{sp}_{\mathsf{Bil}}$ und eines Elements $v_0 \in \mathbb{G}_2$ mit $v_0 = h^a$ für $a \in \mathbb{Z}_p$ das entsprechende Element $u_0 \in \mathbb{G}_1$ mit $u_0 = g^a$ ausgibt.

Vereinfachend sagen wir im Folgenden, dass bei einem Typ-2-Pairing ein effizient berechenbarer Isomorphismus $\psi\colon \mathbb{G}_2 \to \mathbb{G}_1$ mit $g = \psi(h)$ existiert.

Typ-3: Für alle $\lambda \in \mathbb{N}$ und alle $\mathsf{sp}_{\mathsf{Bil}} \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^{\lambda}\right)$ gilt $\mathbb{G}_1 \neq \mathbb{G}_2$. Zudem gibt es für jeden polynomiellen Algorithmus $\mathcal{A}$ eine vernachlässigbare Funktion ν, so dass für alle hinreichend großen $\lambda \in \mathbb{N}$ gilt:

$$\Pr\Big[\mathsf{sp}_{\mathsf{Bil}} \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^{\lambda}\right);\ a, b \in_{\mathcal{R}} \mathbb{Z}_p;\ (u_0, v_0) \leftarrow \mathcal{A}\left(\mathsf{sp}_{\mathsf{Bil}}, g^a, h^b\right) : u_0 = g^b \vee v_0 = h^a\Big] \leq \nu(\lambda).$$

Vereinfachend sagen wir im Folgenden, dass bei einem Typ-3-Pairing keine effizient berechenbaren Isomorphismen zwischen den beiden Gruppen $\mathbb{G}_1$ und $\mathbb{G}_2$ bekannt sind.

Der Fall, dass effizient berechenbare Isomorphismen $\psi\colon \mathbb{G}_2 \to \mathbb{G}_1$ sowie $\psi^{-1}\colon \mathbb{G}_1 \to \mathbb{G}_2$ bekannt sind und $\mathbb{G}_1 \neq \mathbb{G}_2$ ist, kann als Typ-1-Pairing angesehen werden (vgl. [GPS06, Hoh06]).

Beispiele für Familien von Pairings, von denen angenommen wird, dass sie zum Typ-3 gehören, sind das Tate-Pairing, das *ate-Pairing* [HSV06] und das *R-ate-Pairing* [LLP09].

Das Typ-1-Pairing wird auch als *symmetrisches* und die Typen 2 bzw. 3 als *asymmetrisches* Pairing bezeichnet. Obwohl das Typ-3-Pairing hinsichtlich des Berechnungsaufwandes der Grundoperationen (vgl. Tabelle 2.3) der effizienteste Pairing Typ ist (vgl. [GPS06, GPS08, CM09, CHKM09]), wird der Typ-2 in der Literatur am häufigsten verwendet (z.B. [BF01, BLS01, BB04, BBS04, ASM06, Lyn07]), wobei der Typ-1 als *Spezialfall* des Typ-2-Pairings interpretiert werden kann, indem der Isomorphismus ψ als Identität id interpretiert wird. Der Grund für den häufigen Einsatz des Typ-2-Pairings beruht auf der (falschen) Annahme, dass der effizient berechenbare Isomorphismus $\psi\colon \mathbb{G}_2 \to \mathbb{G}_1$ für die entsprechenden Anwendungen notwendig sei. Entweder würde der Isomorphismus für das eigentliche Protokoll oder nur für den Sicherheitsbeweis benötigt und somit wurde davon ausgegangen, dass viele Anwendungen nur durch ein Typ-2-Pairing und nicht durch ein Typ-3-Pairing realisiert werden können. Im Jahr 2009 konnten S. Chatterjee und A. Menezes [CM09] jedoch zeigen, dass viele kryptografische Verfahren vom Typ-2-Pairing in das Typ-3-Pairing transferiert werden können. In Unterabschnitt 2.6.4 wird das in [CM09] beschriebene Vorgehen genauer erläutert und in Kapitel 3 angewandt, um ein digitales Signaturverfahren (vgl. Abschnitt 2.8) zu konstruieren, welches in allen Pairing Typen sicher ist.

In einigen kryptografischen Protokollen wird ein weiterer Pairing Typ-4 verwendet (vgl. [CCS07, GPS08, CHM10]). Bei einem Typ-4-Pairing ist, wie beim Typ-2-Pairing, ein effizient berechenbarer Isomorphismus $\psi\colon \mathbb{G}_2 \to \mathbb{G}_1$ bekannt und es ist zusätzlich wie bei einem Typ-3-Pairing möglich, dass Hashfunktionen (vgl. Abschnitt 2.3) in die Gruppe $\mathbb{G}_2$ abbilden. Da wir letztere Eigenschaft allerdings in dieser Arbeit nicht benötigen, betrachten wir das Typ-4-Pairing einfach als Sonderfall des Typ-2-Pairings.

Damit für einen bestimmten Sicherheitsparameter $\lambda \in \mathbb{N}$ das gewünschte Sicherheitsniveau erreicht wird, sollten die Gruppen $\mathbb{G}_1, \mathbb{G}_2$ und $\mathbb{G}_T$ die Ordnung p besitzen, wobei p eine (2λ)-Bit Primzahl ist (vgl. Tabelle 2.1 aus Unterabschnitt 2.4.1). Um die in dieser

Arbeit entwickelten kryptografischen Systeme mit anderen aus der Literatur zu vergleichen, wird ein Sicherheitsparameter von $\lambda = 128$ gewählt, was aktuell und auch zukünftig eine hohe Sicherheit garantiert. Entsprechend hat die Primzahlordnung p der Gruppen $\mathbb{G}_1, \mathbb{G}_2$ und $\mathbb{G}_T$ eine Länge von 256 Bit.

In der folgenden Tabelle 2.3 werden die Bitlängen der Gruppenelemente und der erwartete Aufwand (hinsichtlich Multiplikationen in $\mathbb{Z}_p$) der Grundoperationen des Typ-2-, Typ-3- und Typ-4-Pairings aufgelistet (siehe [CHKM09, CHM10]).

	Typ-2	**Typ-3**	**Typ-4**
Bitlängen der Elemente in $\mathbb{G}_1$	257	257	257
Bitlängen der Elemente in $\mathbb{G}_2$	770	513	770
Bitlängen der Elemente in $\mathbb{G}_T$	1.024	1.024	1.024
Komprimieren der Elemente in $\mathbb{G}_1$	gratis	gratis	gratis
Komprimieren der Elemente in $\mathbb{G}_2$	gratis	gratis	gratis
Dekomprimieren der Elemente in $\mathbb{G}_1$	315 m	315 m	315 m
Dekomprimieren der Elemente in $\mathbb{G}_2$	989 m	674 m	989 m
Multiplizieren in $\mathbb{G}_1$	11 m	11 m	11 m
Quadrieren in $\mathbb{G}_1$	7 m	7 m	7 m
Multiplizieren in $\mathbb{G}_2$	41 m	30 m	41 m
Quadrieren in $\mathbb{G}_2$	24 m	17 m	24 m
Potenzieren in $\mathbb{G}_1$	1.533 m	1.533 m	1.533 m
Potenzieren in $\mathbb{G}_2$	4.585 m	3.052 m	4.585 m
Potenzieren mit festgelegter Basis in $\mathbb{G}_1$	718 m	718 m	718 m
Potenzieren mit festgelegter Basis in $\mathbb{G}_2$	2.624 m	1.906 m	2.624 m
$\mathsf{H}\colon \{0,1\}^* \to \mathbb{G}_1$	315 m	315 m	315 m
$\mathsf{H}\colon \{0,1\}^* \to \mathbb{G}_2$	–	3.726 m	4.041 m
Pairing	15.175 m	15.175 m	15.175 m
Prüfen, ob Element von $\mathbb{G}_1$	gratis	gratis	gratis
Prüfen, ob Element von $\mathbb{G}_2$	23.775 m	3.052 m	3.052 m

Tabelle 2.3: Effizienzvergleich verschiedener Pairing Typen

2.6 Kryptografische Probleme und Annahmen

Das Konzept der asymmetrischen Kryptografie, welche auch als Public-Key-Kryptografie bezeichnet wird, wurde 1976 von W. Diffie und M. Hellman [DH76] vorgeschlagen

und war der richtungsweisende Wegbereiter für neue kryptografische Anwendungen, wie *Public-Key-Verschlüsselungsverfahren* und *digitale Signaturen*, die in den Abschnitten 2.7 bzw. 2.8 behandelt werden.

Zunächst benötigen wir die folgende Definition zur Erzeugung von Familien von Gruppen, wobei wir uns in der gesamten Arbeit auf Gruppen mit Primzahlordnung beschränken.

Definition 2.6.1 (**Familien von Gruppen**). *Sei* $\mathsf{Setup}_\mathsf{G}$ *ein probabilistischer polynomieller Algorithmus, der bei Eingabe eines Sicherheitsparameters* 1^λ *die Systemparameter* $\mathsf{sp}_\mathsf{G} := (p, \mathbb{G}', g)$ *ausgibt, wobei* p *eine zufällig gewählte Primzahl mit* $p = \Theta\left(2^\lambda\right)$, $\mathbb{G}'$ *die Beschreibung einer zyklischen Gruppe* $\mathbb{G}$ *der Ordnung* p *und* $g \in_\mathcal{R} \mathbb{G}$ *ein zufällig gewählter Generator ist. Es ist auch möglich, dass* $\mathsf{Setup}_\mathsf{G}$ *neben einem Sicherheitsparameter* 1^λ *bereits eine Primzahl* $p = \Theta\left(2^\lambda\right)$ *als Eingabe erhält und die Systemparameter* $\mathsf{sp}_\mathsf{G} = (p, \mathbb{G}', g)$ *ausgibt.*

Im Folgenden werden die kryptografischen Probleme und Annahmen beschrieben, auf denen die Sicherheit der teilbaren elektronischen Geldsysteme in Kapitel 6 bis 9 beruht.

2.6.1 Das Diskreter-Logarithmus-Problem

Das Diskreter-Logarithmus-Problem ist neben dem Faktorisierungsproblem eines der wichtigsten Probleme, um die Sicherheit kryptografischer Algorithmen zu beweisen. Alle weiteren Probleme dieses Abschnitts beruhen auf dem Diskreter-Logarithmus-Problem. Das bedeutet, falls das Diskreter-Logarithmus-Problem in einer Familie von Gruppen in polynomieller Laufzeit mit nicht vernachlässigbarer Wahrscheinlichkeit gelöst werden kann, können auch alle anderen hier vorgestellten Probleme in dieser Familie in polynomieller Laufzeit mit nicht vernachlässigbarer Wahrscheinlichkeit gelöst werden.

Definition 2.6.2 (**Diskrete Logarithmusfunktion**). *Für eine zyklische Gruppe* $\mathbb{G}$ *der Ordnung* $n \in \mathbb{N}$ *und einen Generator* $g \in \mathbb{G}$ *heißt die Abbildung* $\exp_g \colon \mathbb{N} \to \mathbb{G}$ *mit* $a \mapsto g^a$ diskrete Exponentialfunktion *zur Basis* g. *Für ein Gruppenelement* $g_1 \in \mathbb{G}$ *ist der* diskrete Logarithmus $\log_g(g_1)$ *von* g_1 *zur Basis* g *die eindeutig bestimmte Zahl* $x \in \mathbb{Z}_n$ *mit* $g_1 = g^x$. *Somit ist die* diskrete Logarithmusfunktion *die Umkehrfunktion der diskreten Exponentialfunktion.*

Während die diskrete Exponentialfunktion mit dem *Square-and-Multiply-Algorithmus* in polynomieller Laufzeit berechnet werden kann, sind bis heute keine polynomiellen Algorithmen zum Berechnen diskreter Logarithmen bekannt. Daher wird vermutet, dass es sich bei der diskreten Exponentialfunktion um eine sogenannte *Einweg*-Funktion handelt, die leicht zu berechnen aber schwer zu invertieren ist. Dies führt zur folgenden Annahme.

Definition 2.6.3 (**Diskreter-Logarithmus (DLog)**). *Sei* $\mathsf{sp}_{\mathsf{G}} = (p, \mathbb{G}', g)$ *die Ausgabe von* $\mathsf{Setup}_{\mathsf{G}}$. *Das* Diskreter-Logarithmus-Problem *lautet: Gib bei Eingabe von* $(p, \mathbb{G}', g, g^x)$ *mit* $x \in_{\mathcal{R}} \mathbb{Z}_p$ *die Zahl* x *aus.*

Wir sagen, dass die Diskreter-Logarithmus-Annahme *für* $\mathsf{Setup}_{\mathsf{G}}$ *gilt, falls es für jeden polynomiellen Algorithmus* $\mathcal{A}$ *eine vernachlässigbare Funktion* ν *gibt, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\Pr\left[(p, \mathbb{G}', g) \leftarrow \mathsf{Setup}_{\mathsf{G}}\left(1^{\lambda}\right);\ x \in_{\mathcal{R}} \mathbb{Z}_p;\ x' \leftarrow \mathcal{A}\left(p, \mathbb{G}', g, g^x\right)\ :\ x' = x\right] \leq \nu(\lambda).$$

Typische Wahlen, für Familien von Gruppen, von denen vermutet wird, dass die Diskreter-Logarithmus-Annahme gilt, sind die Familie der multiplikativen Gruppen $\mathbb{Z}_p^*$ der Primzahlordnung p, die Familie der eindeutig bestimmten zyklischen Untergruppen $\mathbb{G}_q$ der Primzahlordnung q von $\mathbb{Z}_p^*$ für $q \mid (p-1)$ oder die Familie der elliptischen Kurven $E(\mathbb{Z}_p)$ mit Primzahl $p > 3$.

2.6.2 Das Darstellungsproblem

Das *Darstellungsproblem in Gruppen mit Primzahlordnung* wurde 1993 von S. Brands [Bra93] vorgestellt, um ein effizientes elektronisches Geldsystem zu konstruieren, welches im Vergleich zu [CFN89] ohne die eher ineffiziente *Cut-and-Choose-Methode* auskommt. Das Darstellungsproblem ist eng mit dem Diskreter-Logarithmus-Problem verwandt.

Definition 2.6.4 (**Generatortupel und Darstellung**). *Für eine zyklische Gruppe* $\mathbb{G}$ *der Primzahlordnung* p *und eine ganze Zahl* $L \geq 2$ *heißt ein Tupel* $(g_1, \ldots, g_L) \in \mathbb{G}^L$ Generatortupel (der Länge L)*, falls die Elemente* $g_1, \ldots, g_L$ *paarweise verschiedene Generatoren der Gruppe* $\mathbb{G}$ *sind.*

Ein Tupel $(g_1, \ldots, g_L) \in_{\mathcal{R}} \mathbb{G}^L$ *heißt* zufälliges Generatortupel*, falls* $(g_1, \ldots, g_L)$ *ein Generatortupel ist und die Elemente* $g_1, \ldots, g_L \in_{\mathcal{R}} \mathbb{G}$ *zufällig gewählt sind.*

Für ein beliebiges Gruppenelement $A \in \mathbb{G}$ *heißt ein Tupel* $(a_1, \ldots, a_L) \in \mathbb{Z}_p^L$ Darstellung *von* A *bzgl. eines Generatortupels* $(g_1, \ldots, g_L)$*, falls* $A = \prod_{i=1}^{L} g_i^{a_i}$ *gilt.*

Im Folgenden wird für ein Produkt $A = \prod_{i=1}^{L} g_i^{a_i}$ auch die Schreibweise $A = g_1^{a_1} \cdots g_L^{a_L}$ verwendet. Analog wird für eine Summe $a = \sum_{i=1}^{L} a_i$ auch die Notation $a = a_1 + \cdots + a_L$ benutzt.

In [Bra93] wurde gezeigt, dass es zu jedem Element $A \in \mathbb{G}$ einer Gruppe $\mathbb{G}$ der Ordnung p genau p^{L-1} Darstellungen bzgl. eines Generatortupels der Länge L gibt. Für das neutrale Element 1 wird die Darstellung $(0, \ldots, 0) \in \mathbb{Z}_p^L$ bzgl. eines Generatortupels der Länge L als *triviale Darstellung* bezeichnet.

Definition 2.6.5 (**Darstellungsproblem in Gruppen mit Primzahlordnung**). *Sei* $\mathsf{sp}_{\mathsf{G}} = (p, \mathbb{G}', g)$ *die Ausgabe von* $\mathsf{Setup}_{\mathsf{G}}$*,* $(g_1, \ldots, g_L) \in_{\mathcal{R}} \mathbb{G}^L$ *ein zufälliges Generatortupel und* $A \in_{\mathcal{R}} \mathbb{G}$ *ein zufälliges Element. Das Darstellungsproblem lautet: Gib bei Eingabe von* $(p, \mathbb{G}', g_1, \ldots, g_L, A)$ *eine Darstellung* $(a_1 \ldots, a_L) \in \mathbb{Z}_p^L$ *von* A *bzgl.* $(g_1, \ldots, g_L)$ *aus.*

Für Familien von Gruppen mit Primzahlordnung wurde in [Bra93] die Äquivalenz von Diskreter-Logarithmus-Problem und Darstellungsproblem gezeigt. Insbesondere sind für $L = 1$ beide Probleme identisch, weshalb in der Definition 2.6.4 $L \geq 2$ gefordert wird. Aus der Äquivalenz von Diskreter-Logarithmus-Problem und Darstellungsproblem in Familien von Gruppen mit Primzahlordnung folgt das folgende Korollar.

Korollar 2.6.1. *Unter der Diskreter-Logarithmus-Annahme gibt es für jeden polynomiellen Algorithmus $\mathcal{A}$ eine vernachlässigbare Funktion ν, so dass für alle hinreichend großen $\lambda \in \mathbb{N}$ und $L \geq 2$ gilt:*

$$\Pr\Big[(p, \mathbb{G}', g) \leftarrow \mathsf{Setup}_{\mathsf{G}}\left(1^{\lambda}\right);\ (g_1, \ldots, g_L) \in_{\mathcal{R}} \mathbb{G}^L; \\ (A, (a_1, \ldots, a_L), (b_1, \ldots, b_L)) \leftarrow \mathcal{A}(p, \mathbb{G}', g_1, \ldots, g_L)\ : \\ (a_1, \ldots, a_L) \neq (b_1, \ldots, b_L)\ \wedge\ A = g_1^{a_1} \cdots g_L^{a_L} = g_1^{b_1} \cdots g_L^{b_L}\Big] \leq \nu(\lambda).$$

Beweis. Siehe [Bra93]. □

2.6.3 Die Diffie-Hellman-Probleme

In ihrem Artikel haben W. Diffie und M. Hellman [DH76] ein Verfahren vorgestellt, das zwei Parteien die Erzeugung eines gemeinsamen Geheimnisses ermöglicht, obwohl beide nur über einen unsicheren Kanal miteinander kommunizieren. Eine Grundlage für diese sogenannte *Diffie-Hellman-Schlüsselvereinbarung* ist das *Computational-Diffie-Hellman-Problem*.

Definition 2.6.6 (**Computational-Diffie-Hellman (CDH)**). *Sei $\mathsf{sp}_{\mathsf{G}} = (p, \mathbb{G}', g)$ die Ausgabe von $\mathsf{Setup}_{\mathsf{G}}$. Das* Computational-Diffie-Hellman-Problem *lautet: Gib bei Eingabe von $\left(p, \mathbb{G}', g, g^a, g^b\right)$ mit $a, b \in_{\mathcal{R}} \mathbb{Z}_p$ das Element g^{ab} aus.*

Wir sagen, dass die Computational-Diffie-Hellman-Annahme *für $\mathsf{Setup}_{\mathsf{G}}$ gilt, falls es für jeden polynomiellen Algorithmus $\mathcal{A}$ eine vernachlässigbare Funktion ν gibt, so dass für alle hinreichend großen $\lambda \in \mathbb{N}$ gilt:*

$$\Pr\left[(p, \mathbb{G}', g) \leftarrow \mathsf{Setup}_{\mathsf{G}}\left(1^{\lambda}\right); a, b \in_{\mathcal{R}} \mathbb{Z}_p; A \leftarrow \mathcal{A}\left(p, \mathbb{G}', g, g^a, g^b\right)\ :\ A = g^{ab}\right] \leq \nu(\lambda).$$

Falls das Diskreter-Logarithmus-Problem in einer Familie von Gruppen in polynomieller Laufzeit mit nicht vernachlässigbarer Wahrscheinlichkeit gelöst werden kann, kann auch das Computational-Diffie-Hellman-Problem in dieser Familie in polynomieller Laufzeit mit nicht vernachlässigbarer Wahrscheinlichkeit gelöst werden. Ob die Umkehrung auch gilt und somit beide Probleme äquivalent sind, ist allerdings noch offen. Für bestimmte Familien von Gruppen haben U. Maurer und S. Wolf [Mau94, MW99] sowie A. Joux und K. Nguyen [JN01] jedoch gezeigt, dass beide Probleme unter gewissen Voraussetzungen äquivalent sind.

Eine noch stärkere Annahme als die Computational-Diffie-Hellman-Annahme ist die *Decisional-Diffie-Hellman-Annahme* ([Bon98]). In dieser wird angenommen, dass kein

polynomieller Algorithmus $\mathcal{A}$ das Element g^{ab} von einem zufällig gewählten Gruppenelement unterscheiden kann.

Definition 2.6.7 (Decisional-Diffie-Hellman (DDH)). *Sei* $\mathsf{sp}_\mathsf{G} = (p, \mathbb{G}', g)$ *die Ausgabe von* $\mathsf{Setup}_\mathsf{G}$. *Das* Decisional-Diffie-Hellman-Problem *lautet: Entscheide bei Eingabe von* $\left(p, \mathbb{G}', g, g^a, g^b, g^c\right)$ *mit* $a, b \in_\mathcal{R} \mathbb{Z}_p$ *und* $c \in \mathbb{Z}_p$, *ob* $g^c = g^{ab}$ *ist, d.h. gib* 1 *aus, falls* $g^c = g^{ab}$ *ist und andernfalls* 0.

Wir sagen, dass die Decisional-Diffie-Hellman-Annahme *für* $\mathsf{Setup}_\mathsf{G}$ *gilt, falls es für jeden polynomiellen Algorithmus* $\mathcal{A}$ *eine vernachlässigbare Funktion* ν *gibt, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\begin{aligned}\Big|&\Pr\left[(p, \mathbb{G}', g) \leftarrow \mathsf{Setup}_\mathsf{G}\left(1^\lambda\right); a, b \in_\mathcal{R} \mathbb{Z}_p \,:\, 1 \leftarrow \mathcal{A}\left(p, \mathbb{G}', g, g^a, g^b, g^{ab}\right)\right] \\ &- \Pr\left[(p, \mathbb{G}', g) \leftarrow \mathsf{Setup}_\mathsf{G}\left(1^\lambda\right); a, b, c \in_\mathcal{R} \mathbb{Z}_p \,:\, 1 \leftarrow \mathcal{A}\left(p, \mathbb{G}', g, g^a, g^b, g^c\right)\right]\Big| \leq \nu(\lambda).\end{aligned}$$

In Familien von Gruppen, in denen das CDH-Problem in polynomieller Laufzeit mit nicht vernachlässigbarer Wahrscheinlichkeit gelöst werden kann, kann auch das DDH-Problem in polynomieller Laufzeit mit nicht vernachlässigbarer Wahrscheinlichkeit gelöst werden. Die Umkehrung gilt jedoch nicht. Obwohl davon ausgegangen wird, dass in der Familie der Gruppen $\mathbb{Z}_p^*$ mit Primzahl p die CDH-Annahme gilt, gilt die DDH-Annahme in dieser Familie nicht. Denn anhand des Legrende-Symbols kann in der Familie der Gruppen $\mathbb{Z}_p^*$ in polynomieller Laufzeit entschieden werden, ob ein Gruppenelement ein quadratischer Rest bzw. Nichtrest ist. Weiter wurden von A. Joux und K. Nguyen [JN01] erstmals bilineare Abbildungen folgendermaßen dazu genutzt, das DDH-Problem effizient zu lösen, obwohl angenommen wird, dass die CDH-Annahme gilt. Sei $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$ ein Typ-1-Pairing (siehe Unterabschnitt 2.5.1), dann kann das DDH-Problem in der Gruppe $\mathbb{G}_1 = \langle g \rangle$ durch Überprüfung der Gleichung

$$\hat{e}\left(g^a, g^b\right) \stackrel{?}{=} \hat{e}\left(g, g^c\right)$$

leicht gelöst werden. Gruppen mit dieser Eigenschaft werden *Gap-Diffie-Hellman-Gruppen* (GDH-Gruppen) genannt (vgl. [BLS01]). Aber auch im Typ-2-Pairing $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ mit einem effizient berechenbaren Isomorphismus $\psi\colon \mathbb{G}_2 \to \mathbb{G}_1$ kann das DDH-Problem in der Gruppe $\mathbb{G}_2$ einfach gelöst werden. Sei h ein Generator der Gruppe $\mathbb{G}_2$, dann kann durch Überprüfung der Gleichung

$$\hat{e}\left(\psi\left(h^a\right), h^b\right) \stackrel{?}{=} \hat{e}\left(\psi\left(h\right), h^c\right) \tag{2.1}$$

leicht entschieden werden, ob $h^c = h^{ab}$ ist, oder nicht. Weitere Familien von Gruppen, in denen das DDH-Problem in polynomieller Laufzeit mit nicht vernachlässigbarer Wahrscheinlichkeit gelöst werden kann, werden in [Ver01] und [GR04] beschrieben.

Während bei einem Typ-2-Pairing $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ mit einem effizient berechenbaren Isomorphismus $\psi\colon \mathbb{G}_2 \to \mathbb{G}_1$ das DDH-Problem in der Gruppe $\mathbb{G}_2$ leicht gelöst

werden kann, wird angenommen, dass dies für die Gruppe $\mathbb{G}_1$ nicht zutrifft. Dies ist die sogenannte *External-Decisional-Diffie-Hellman-Annahme* ([BBS04, ACM05, BGMM05, CHL06b, ASM07]).

Definition 2.6.8 (**External-Decisional-Diffie-Hellman (XDH)**). *Sei* $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, h)$ *die Ausgabe von* $\mathsf{Setup}_{\mathsf{Bil}}$. *Das* External-Decisional-Diffie-Hellman-Problem *lautet: Entscheide bei Eingabe von* $\left(\mathsf{sp}_{\mathsf{Bil}}, g^a, g^b, g^c\right)$ *mit* $a, b \in_{\mathcal{R}} \mathbb{Z}_p$ *und* $c \in \mathbb{Z}_p$, *ob* $g^c = g^{ab}$ *ist, d.h. gib* 1 *aus, falls* $g^c = g^{ab}$ *ist und andernfalls* 0.

Wir sagen, dass die External-Decisional-Diffie-Hellman-Annahme *für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gilt, falls es für jeden polynomiellen Algorithmus* $\mathcal{A}$ *eine vernachlässigbare Funktion* ν *gibt, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\Big|\Pr\left[\mathsf{sp}_{\mathsf{Bil}} \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right); a, b \in_{\mathcal{R}} \mathbb{Z}_p \ : \ 1 \leftarrow \mathcal{A}\left(\mathsf{sp}_{\mathsf{Bil}}, g^a, g^b, g^{ab}\right)\right] - \Pr\left[\mathsf{sp}_{\mathsf{Bil}} \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right); a, b, c \in_{\mathcal{R}} \mathbb{Z}_p \ : \ 1 \leftarrow \mathcal{A}\left(\mathsf{sp}_{\mathsf{Bil}}, g^a, g^b, g^c\right)\right]\Big| \leq \nu(\lambda).$$

Die XDH-Annahme impliziert, dass kein effizient berechenbarer Isomorphismus $\psi' \colon \mathbb{G}_1 \to \mathbb{G}_2$ mit $h = \psi'(g)$ bekannt ist, da ansonsten das XDH-Problem analog zur Gleichung 2.1 durch Überprüfung der Gleichung

$$\hat{e}\left(g^a, \psi'\left(g^b\right)\right) \stackrel{?}{=} \hat{e}\left(g^c, \psi'(g)\right)$$

einfach gelöst werden könnte.

Da bei einem Typ-3-Pairing $\hat{e} \colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ keine effizient berechenbaren Isomorphismen zwischen den Gruppen $\mathbb{G}_1$ und $\mathbb{G}_2$ bekannt sind, wird vermutet, dass das DDH-Problem in beiden Gruppen schwer zu lösen ist, was als *Symmetric-External-Decisional-Diffie-Hellman-Annahme* ([BGMM05, ACHM05, AWSM07]) bezeichnet wird.

Definition 2.6.9 (**Symmetric-External-Decisional-Diffie-Hellman (SXDH)**). *Sei* $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, h)$ *die Ausgabe von* $\mathsf{Setup}_{\mathsf{Bil}}$. *Das* Symmetric-External-Decisional-Diffie-Hellman-Problem *lautet: Entscheide bei Eingabe von* $\left(\mathsf{sp}_{\mathsf{Bil}}, g^a, g^b, g^c, h^d, h^e, h^f\right)$ *mit* $a, b, d, e \in_{\mathcal{R}} \mathbb{Z}_p$ *und* $c, f \in \mathbb{Z}_p$, *ob* $g^c = g^{ab}$ *oder* $h^f = h^{de}$ *ist, d.h. gib* 1 *aus, falls* $g^c = g^{ab}$ *oder* $h^f = h^{de}$ *ist und andernfalls* 0.

Wir sagen, dass die Symmetric-External-Decisional-Diffie-Hellman-Annahme *für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gilt, falls es für jeden polynomiellen Algorithmus* $\mathcal{A}$ *eine vernachlässigbare Funktion* ν *gibt, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\Big|\Pr\left[\mathsf{sp}_{\mathsf{Bil}} \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right); a, b, d, e \in_{\mathcal{R}} \mathbb{Z}_p \ : \ 1 \leftarrow \mathcal{A}\left(\mathsf{sp}_{\mathsf{Bil}}, g^a, g^b, g^{ab}, h^d, h^e, h^{de}\right)\right] - \Pr\left[\mathsf{sp}_{\mathsf{Bil}} \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right); a, \ldots, f \in_{\mathcal{R}} \mathbb{Z}_p \ : \ 1 \leftarrow \mathcal{A}\left(\mathsf{sp}_{\mathsf{Bil}}, g^a, g^b, g^c, h^d, h^e, h^f\right)\right]\Big| \leq \nu(\lambda).$$

Die SXDH-Annahme ist eine starke Annahme, da sie nur gilt, wenn kein effizient berechenbarer Isomorphismus zwischen den Gruppen $\mathbb{G}_1$ und $\mathbb{G}_2$ bekannt ist. Insbesondere impliziert die SXDH-Annahme die XDH-Annahme. Damit das SXDH-Problem in polynomieller Laufzeit nur mit vernachlässigbarer Wahrscheinlichkeit gelöst werden kann, muss also ein Typ-3-Pairing (siehe Unterabschnitt 2.5.1) verwendet werden.

Eine weitere, zur DDH-Annahme ähnliche Annahme, ist die sogenannte *Decisional-Bilinear-Diffie-Hellman-Annahme* ([BF03, Gal05, CM09]), die im Folgenden für ein Typ-2-Pairing definiert wird.

Definition 2.6.10 (**Decisional-Bilinear-Diffie-Hellman-Typ-2 (DBDH2)**). *Seien* $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, h)$ *die Ausgabe von* $\mathsf{Setup}_{\mathsf{Bil}}$ *und* $\psi\colon \mathbb{G}_2 \to \mathbb{G}_1$ *ein effizient berechenbarer Isomorphismus mit* $g = \psi(h)$. *Das* Decisional-Bilinear-Diffie-Hellman-Typ-2-Problem *lautet: Entscheide bei Eingabe von* $\left(\mathsf{sp}_{\mathsf{Bil}}, \psi, g^a, h^b, h^c, \hat{e}\,(g,h)^d\right)$ *mit* $a, b, c \in_{\mathcal{R}} \mathbb{Z}_p$ *und* $d \in \mathbb{Z}_p$, *ob* $\hat{e}\,(g,h)^d = \hat{e}\,(g,h)^{abc}$ *ist, d.h. gib* 1 *aus, falls* $\hat{e}\,(g,h)^d = \hat{e}\,(g,h)^{abc}$ *ist und andernfalls* 0.

Wir sagen, dass die Decisional-Bilinear-Diffie-Hellman-Typ-2-Annahme *für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gilt, falls es für jeden polynomiellen Algorithmus* $\mathcal{A}$ *eine vernachlässigbare Funktion* ν *gibt, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\begin{aligned}\Big|\Pr\Big[\mathsf{sp}_{\mathsf{Bil}} \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right); a,b,c \in_{\mathcal{R}} \mathbb{Z}_p \;:\; 1 \leftarrow \mathcal{A}\left(\mathsf{sp}_{\mathsf{Bil}}, \psi, g^a, h^b, h^c, \hat{e}\,(g,h)^{abc}\right)\Big] \\ - \Pr\Big[\mathsf{sp}_{\mathsf{Bil}} \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right); a,b,c,d \in_{\mathcal{R}} \mathbb{Z}_p \;:\; 1 \leftarrow \mathcal{A}\left(\mathsf{sp}_{\mathsf{Bil}}, \psi, g^a, h^b, h^c, \hat{e}\,(g,h)^{d}\right)\Big]\Big| \\ \leq \nu(\lambda)\,.\end{aligned}$$

Auch die *Decisional-Linear-Diffie-Hellman-Annahme* ([BBS04]) ist eine zur DDH-Annahme ähnliche Annahme.

Definition 2.6.11 (**Decisional-Linear-Diffie-Hellman (DLDH)**). *Seien* $\mathsf{sp}_{\mathsf{G}} = (p, \mathbb{G}', g)$ *die Ausgabe von* $\mathsf{Setup}_{\mathsf{G}}$ *und* $g_1, g_2, g_3 \in_{\mathcal{R}} \mathbb{G}$ *drei zufällig gewählte Generatoren. Das* Decisional-Linear-Diffie-Hellman-Problem *lautet: Entscheide bei Eingabe von* $\left(p, \mathbb{G}', g_1, g_2, g_3, g_1^a, g_2^b, g_3^c\right)$ *mit* $a, b \in_{\mathcal{R}} \mathbb{Z}_p$ *und* $c \in \mathbb{Z}_p$, *ob* $g_3^c = g_3^{a+b}$ *ist, d.h. gib* 1 *aus, falls* $g_3^c = g_3^{a+b}$ *ist und andernfalls* 0.

Wir sagen, dass die Decisional-Linear-Diffie-Hellman-Annahme *für* $\mathsf{Setup}_{\mathsf{G}}$ *gilt, falls es für jeden polynomiellen Algorithmus* $\mathcal{A}$ *eine vernachlässigbare Funktion* ν *gibt, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\begin{aligned}\Big|\Pr\Big[(p,\mathbb{G}',g) \leftarrow \mathsf{Setup}_{\mathsf{G}}\left(1^\lambda\right); a,b \in_{\mathcal{R}} \mathbb{Z}_p \;:\; 1 \leftarrow \mathcal{A}\left(p, \mathbb{G}', g_1, g_2, g_3, g_1^a, g_2^b, g_3^{a+b}\right)\Big] \\ - \Pr\Big[(p,\mathbb{G}',g) \leftarrow \mathsf{Setup}_{\mathsf{G}}\left(1^\lambda\right); a,b,c \in_{\mathcal{R}} \mathbb{Z}_p \;:\; 1 \leftarrow \mathcal{A}\left(p, \mathbb{G}', g_1, g_2, g_3, g_1^a, g_2^b, g_3^{c}\right)\Big]\Big| \\ \leq \nu(\lambda)\,.\end{aligned}$$

Es ist leicht zu zeigen, dass das DDH-Problem in einer Familie von Gruppen in polynomieller Laufzeit mit nicht vernachlässigbarer Wahrscheinlichkeit gelöst werden kann,

falls das DLDH-Problem in dieser Familie in polynomieller Laufzeit mit nicht vernachlässigbarer Wahrscheinlichkeit gelöst werden kann. Es wird jedoch vermutet, dass die Umkehrung nicht stimmt (siehe [BBS04]). Somit wird angenommen, dass die DLDH-Annahme auch in Familien von Gruppen gilt, in denen die DDH-Annahme nicht gilt.

Ein weiteres wichtiges Problem ist das sogenannte *q-Strong-Diffie-Hellman-Problem* ([BB04, BBS04, ASM06, BB08]). Dieses ist formal nur für ein Typ-2-Pairing (und damit auch für ein Typ-1-Pairing, vgl. Unterabschnitt 2.5.1) definiert.

Definition 2.6.12 (**q-Strong-Diffie-Hellman-Typ-2 (q-SDH2)**). *Seien* $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, h)$ *die Ausgabe von* $\mathsf{Setup}_{\mathsf{Bil}}$ *und* $\psi\colon \mathbb{G}_2 \to \mathbb{G}_1$ *ein effizient berechenbarer Isomorphismus mit* $g = \psi(h)$. *Das* q-Strong-Diffie-Hellman-Typ-2-Problem *lautet: Gib bei Eingabe von* $\left(\mathsf{sp}_{\mathsf{Bil}}, \psi, h^x, h^{x^2}, \ldots, h^{x^q}\right)$ *mit* $x \in_{\mathcal{R}} \mathbb{Z}_p^*, q \in \mathbb{N}$ *ein Paar* $\left(g^{\frac{1}{x+c}}, c\right) \in \mathbb{G}_1 \times \mathbb{Z}_p$ *aus.*

Wir sagen, dass die q-Strong-Diffie-Hellman-Typ-2-Annahme *für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gilt, falls es für jeden polynomiellen Algorithmus* $\mathcal{A}$ *eine vernachlässigbare Funktion* ν *gibt, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\Pr\Big[\mathsf{sp}_{\mathsf{Bil}} \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right);\ x \in_{\mathcal{R}} \mathbb{Z}_p^*;\ (A, c) \leftarrow \mathcal{A}\left(\mathsf{sp}_{\mathsf{Bil}}, \psi, h^x, h^{x^2}, \ldots, h^{x^q}\right) : c \in \mathbb{Z}_p \ \wedge\ A = g^{\frac{1}{x+c}}\Big] \leq \nu(\lambda)\,.$$

Im folgenden Unterabschnitt 2.6.4 wird ein analoges Problem für das Typ-3-Pairing definiert und die Äquivalenz beider Probleme gezeigt. So kann gezeigt werden, dass das in Kapitel 3 entwickelte Signaturverfahren in allen Pairing Typen sicher ist.

2.6.4 Kryptografische Annahmen im Typ-2- und Typ-3-Pairing

Viele kryptografische Protokolle arbeiten auf einem Typ-2-Pairing mit einem effizient berechenbaren Isomorphismus $\psi\colon \mathbb{G}_2 \to \mathbb{G}_1$, wobei ψ entweder für das eigentliche Protokoll (z.B. [BGLS03, BS04]) oder nur für den Sicherheitsbeweis (z.B. [BLS04, BB04, BBS04]) benötigt wird. S. Chatterjee und A. Menezes [CM09] haben allerdings gezeigt, dass der Isomorphismus in den meisten Fällen doch nicht für den Sicherheitsbeweis benötigt wird. Somit können viele Protokolle im Typ-2-Pairing, deren Sicherheit auf der Existenz eines effizient berechenbaren Isomorphismus ψ beruht, in das Typ-3-Pairing transformiert werden. Aus diesem Grund ist das Typ-2-Pairing bloß eine ineffiziente Variante des Typ-3-Pairings (vgl. [GPS08, CHKM09, CM09, CHM10]).

S. Chatterjee und A. Menezes [CM09] haben ausführlich beschrieben, wie ein Protokoll vom Typ-2-Pairing in ein Protokoll im Typ-3-Pairing mit gleicher Sicherheit transformiert werden kann. Dazu wird als erstes ein *natürliches Gegenstück* eines kryptografischen Problems des Typ-2-Pairings benötigt. Seien $\hat{e}_2\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ ein Typ-2-Pairing und $\mathcal{P} - 2$ ein kryptografisches Problem im Typ-2-Pairing. Analoges gilt für $\hat{e}_3\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ und $\mathcal{P} - 3$ im Typ-3-Pairing. Die Grundidee ist, dass für ein gegebenes Element $h^a \in \mathbb{G}_2$ der Instanz des Problems $\mathcal{P} - 2$ auch indirekt das Element

$g^a \in \mathbb{G}_1$ gegeben ist, da dieses durch $g^a = \psi(h^a)$ berechnet werden kann. Da dies im Typ-3-Pairing nicht der Fall ist, muss bei der Instanz der Problems $\mathcal{P} - 3$ neben dem Element $h^a \in \mathbb{G}_2$ auch das Element $g^a \in \mathbb{G}_1$ gegeben sein. In [CM09] wurde die Äquivalenz beider Probleme am Beispiel des DBDH-Problems gezeigt und dieses auch für das Typ-3-Pairing definiert.

Definition 2.6.13 (**Decisional-Bilinear-Diffie-Hellman (DBDH)**). *Sei* $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \hat{e}, g, h)$ *die Ausgabe von* $\mathsf{Setup}_{\mathsf{Bil}}$. *Das* Decisional-Bilinear-Diffie-Hellman-Problem *lautet: Entscheide bei Eingabe von* $\left(\mathsf{sp}_{\mathsf{Bil}}, g^a, g^b, g^c, h^b, h^c, \hat{e}(g,h)^d\right)$ *mit* $a, b, c \in_{\mathcal{R}} \mathbb{Z}_p$ *und* $d \in \mathbb{Z}_p$, *ob* $\hat{e}(g,h)^d = \hat{e}(g,h)^{abc}$ *ist, d.h. gib* 1 *aus, falls* $\hat{e}(g,h)^d = \hat{e}(g,h)^{abc}$ *ist und andernfalls* 0.

Wir sagen, dass die Decisional-Bilinear-Diffie-Hellman-Annahme *für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gilt, falls es für jeden polynomiellen Algorithmus* $\mathcal{A}$ *eine vernachlässigbare Funktion* ν *gibt, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\begin{aligned}\Big|\Pr\left[\mathsf{sp}_{\mathsf{Bil}} \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right); a, b, c \in_{\mathcal{R}} \mathbb{Z}_p \;:\; 1 \leftarrow \mathcal{A}\left(\mathsf{sp}_{\mathsf{Bil}}, g^a, g^b, g^c, h^b, h^c, \hat{e}(g,h)^{abc}\right)\right] \\ - \Pr\left[\mathsf{sp}_{\mathsf{Bil}} \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right); a, b, c, d \in_{\mathcal{R}} \mathbb{Z}_p \;:\; 1 \leftarrow \mathcal{A}\left(\mathsf{sp}_{\mathsf{Bil}}, g^a, g^b, g^c, h^b, h^c, \hat{e}(g,h)^d\right)\right]\Big| \\ \leq \nu(\lambda).\end{aligned}$$

Entsprechend kann das q-SDH-Problem für das Typ-3-Pairing definiert werden (vgl. [CM09]).

Definition 2.6.14 (q-**Strong-Diffie-Hellman (q-SDH)**). *Sei* $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \hat{e}, g, h)$ *die Ausgabe von* $\mathsf{Setup}_{\mathsf{Bil}}$. *Das* q-Strong-Diffie-Hellman-Problem *lautet: Gib bei Eingabe von* $\left(\mathsf{sp}_{\mathsf{Bil}}, g^x, g^{x^2}, \ldots, g^{x^q}, h^x, h^{x^2}, \ldots, h^{x^q}\right)$ *mit* $x \in_{\mathcal{R}} \mathbb{Z}_p^*, q \in \mathbb{N}$ *ein Paar* $\left(g^{\frac{1}{x+c}}, c\right) \in \mathbb{G}_1 \times \mathbb{Z}_p$ *aus.*

Wir sagen, dass die q-Strong-Diffie-Hellman-Annahme *für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gilt, falls es für jeden polynomiellen Algorithmus* $\mathcal{A}$ *eine vernachlässigbare Funktion* ν *gibt, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\begin{aligned}\Pr\Big[\mathsf{sp}_{\mathsf{Bil}} \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right);\; x \in_{\mathcal{R}} \mathbb{Z}_p^*;\; (A, c) \leftarrow \mathcal{A}\left(\mathsf{sp}_{\mathsf{Bil}}, g^x, \ldots, g^{x^q}, h^x, \ldots, h^{x^q}\right) : \\ c \in \mathbb{Z}_p \;\wedge\; A = g^{\frac{1}{x+c}}\Big] \leq \nu(\lambda).\end{aligned}$$

Ein zum q-SDH-Problem ähnliches Problem ist das sogenannte *q-polynomial-Diffie-Hellman-Problem* ([KZG10a, KZG10b]).

Definition 2.6.15 (q-**polynomial-Diffie-Hellman (q-polyDH)**). *Sei* $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \hat{e}, g, h)$ *die Ausgabe von* $\mathsf{Setup}_{\mathsf{Bil}}$. *Das* q-polynomial-Diffie-Hellman-Problem *lautet: Gib bei Eingabe von* $\left(\mathsf{sp}_{\mathsf{Bil}}, g^\alpha, g^{\alpha^2}, \ldots, g^{\alpha^q}, h^\alpha, h^{\alpha^2}, \ldots, h^{\alpha^q}\right)$ *mit* $\alpha \in_{\mathcal{R}} \mathbb{Z}_p^*, q \in \mathbb{N}$ *ein Paar* $\left(g^{\phi(\alpha)}, \phi(\mathsf{x})\right) \in \mathbb{G}_1 \times \mathbb{Z}_p[\mathsf{x}]$ *mit* $2^\lambda > \deg(\phi) > q$ *aus.*

Wir sagen, dass die q-polynomial-Diffie-Hellman-Annahme *für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gilt, falls es für jeden polynomiellen Algorithmus* $\mathcal{A}$ *eine vernachlässigbare Funktion* ν *gibt, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *und für* $p \geq 2^{2\lambda}$ *gilt:*

$$\Pr\Big[\mathsf{sp}_{\mathsf{Bil}} \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^{\lambda}\right);\ \alpha \in_{\mathcal{R}} \mathbb{Z}_p^*;\ (A, \phi(\mathsf{x})) \leftarrow \mathcal{A}\left(\mathsf{sp}_{\mathsf{Bil}}, g^{\alpha}, \ldots, g^{\alpha^q}, h^{\alpha}, \ldots, h^{\alpha^q}\right) : \\ \phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}] \ \wedge\ 2^{\lambda} > \deg(\phi) > q \ \wedge\ A = g^{\phi(\alpha)}\Big] \leq \nu(\lambda).$$

Im Folgenden werden alle Instanzen eines Problems immer für ein Typ-3-Pairing bzgl. der Systemparameter $\mathsf{sp}_{\mathsf{Bil}}$ aus Definition 2.5.2 angegeben, unabhängig davon, um welchen Pairing Typen es sich tatsächlich handelt. Falls es sich um ein Typ-2-Pairing handelt, gilt $g = \psi(h)$, wobei $\psi\colon \mathbb{G}_2 \to \mathbb{G}_1$ der effizient berechenbare Isomorphismus von $\mathbb{G}_2$ nach $\mathbb{G}_1$ ist. Für das Typ-1-Pairing ist der Isomorphismus $\psi = id$ die Identität und somit gilt $g = h$. Ist beispielsweise eine Instanz $(p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_1, g_2, h)$ gegeben, gilt $\mathbb{G}_1 = \langle g \rangle$ sowie $\mathbb{G}_2 = \langle h \rangle$ und $g_1, g_2 \in \mathbb{G}_1$. Falls es sich um ein Typ-2-Pairing handelt, gilt $g = \psi(h)$ und im Typ-1-Pairing entsprechend $g = h$.

An einigen Stellen wird bei den Systemparameter anstelle des Generators g der Generator u_0 verwendet, wobei die obigen Eigenschaften analog für u_0 gelten.

2.7 Public-Key-Verschlüsselungsverfahren

Möchten zwei Parteien *vertraulich* miteinander kommunizieren, können diese ein sogenanntes *Verschlüsselungsverfahren* verwenden. Bei einem solchen Verfahren *verschlüsselt* der *Sender* eine Nachricht, damit diese nur vom gewünschten *Empfänger* wieder *entschlüsselt* werden kann. Grundvoraussetzung eines sicheren Verschlüsselungsverfahrens ist die Existenz eines geheimen Schlüssels. Während bei symmetrischen Verfahren Sender und Empfänger denselben Schlüssel verwenden, wird bei der asymmetrischen Kryptografie lediglich das Schlüsselpaar des Empfängers benötigt, da zum Ver- und Entschlüsseln zwei verschiedene Schlüssel verwendet werden. Dieses Konzept der Public-Key-Verschlüsselung wurde, wie die Public-Key-Kryptografie, 1976 von W. Diffie und M. Hellman [DH76] eingeführt. Der Grundgedanke eines Public-Key-Verschlüsselungsverfahrens ist, dass *nur* der Empfänger einer Nachricht in der Lage sein muss (und auch nur er in der Lage sein darf), diese zu entschlüsseln und somit *nur* der Empfänger ein Geheimnis benötigt. Zum Verschlüsseln einer Nachricht verwendet der Sender den öffentlichen Schlüssel des Empfängers und sendet ihm den daraus resultierenden *Geheimtext*. Nun setzt der Empfänger seinen privaten Schlüssel ein, um wieder die ursprüngliche Nachricht, die auch *Klartext* genannt wird, zu erhalten. Die bekanntesten Public-Key-Verschlüsselungsverfahren sind das RSA- [RSA78] und das ElGamal-Verschlüsselungsverfahren [ElG85a, ElG85b], welches in Unterabschnitt 2.7.1 behandelt wird. Durch eine Erweiterung der ElGamal-Verschlüsselung entsteht das lineare Verschlüsselungsverfahren [BBS04], das in Unterabschnitt 2.7.2 beschrieben ist.

Auch für faire elektronische Geldsysteme sind Public-Key-Verschlüsselungsverfahren von Bedeutung. Denn diese sind eine Grundvoraussetzung dafür, die Anonymität in

Verdachtsfällen aufheben zu können, so dass eine Kunden- und Münzverfolgung realisiert werden kann.

Definition 2.7.1 (**Public-Key-Verschlüsselungsverfahren**). *Ein Tripel* (GenEnc, Enc, Dec) *polynomieller Algorithmen heißt* Public-Key-Verschlüsselungsverfahren, *falls die folgenden Eigenschaften erfüllt sind:*

1. *Der* Schlüsselgenerierungsalgorithmus GenEnc *ist ein probabilistischer Algorithmus, der bei Eingabe des Sicherheitsparameters* 1^λ *einen privaten Schlüssel* sk *und einen öffentlichen Schlüssel* pk *erzeugt, durch den die Menge aller Nachrichten* $\mathbb{M}_{\mathsf{Enc}}$ *festgelegt ist. Die Ausgabe von* GenEnc *ist das Schlüsselpaar* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenEnc}\left(1^\lambda\right)$.

2. *Der* Verschlüsselungsalgorithmus Enc *ist ein probabilistischer Algorithmus, der bei Eingabe eines öffentlichen Schlüssels* pk *und einer Nachricht* $m \in \mathbb{M}_{\mathsf{Enc}}$ *einen Geheimtext* $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$ *berechnet und ausgibt.*

3. *Der* Entschlüsselungsalgorithmus Dec *ist ein deterministischer Algorithmus, der bei Eingabe eines öffentlichen Schlüssels* pk, *eines privaten Schlüssels* sk *und eines Geheimtextes* c *die Nachricht* $\mathsf{Dec}(\mathsf{pk}, \mathsf{sk}, c) = m$ *berechnet und ausgibt.*

4. *Für jedes Schlüsselpaar* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenEnc}\left(1^\lambda\right)$ *und jede Nachricht* $m \in \mathbb{M}_{\mathsf{Enc}}$ *gilt:*

$$\mathsf{Dec}(\mathsf{pk}, \mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m.$$

Im weiteren Verlauf dieser Arbeit wird an einigen Stellen zur Vereinfachung auf die explizite Nennung der Eingabe der Schlüssel pk und sk verzichtet und ein Geheimtext $\mathsf{Enc}(\mathsf{pk}, m)$ verkürzend mit $\mathsf{Enc}(m)$ und eine Nachricht $\mathsf{Dec}(\mathsf{pk}, \mathsf{sk}, c)$ mit $\mathsf{Dec}(c)$ bezeichnet.

Die Sicherheit von Public-Key-Verschlüsselungsverfahren kann grundsätzlich durch zwei unterschiedliche Konzepte definiert werden, nämlich *polynomielle Ununterscheidbarkeit* und *semantische Sicherheit*, die beide Anfang der 80er Jahre von S. Goldwasser und S. Micali [GM82, GM84] entwickelt wurden. Vereinfachend formuliert bedeutet polynomielle Ununterscheidbarkeit, dass kein polynomieller Algorithmus einen Geheimtext $\mathsf{Enc}(m_b)$ mit $b \in_{\mathcal{R}} \{0, 1\}$ der entsprechenden Nachricht m_0 bzw. m_1 zuordnen kann. Semantische Sicherheit meint simplifiziert, dass das, was ein polynomieller Algorithmus mit gegebenen Geheimtext $\mathsf{Enc}(m)$ über die Nachricht m berechnen kann, der Algorithmus auch ohne den Geheimtext berechnen kann, und stellt somit das Public-Key-Analogon zur *perfekten Sicherheit* von C. Shannon [Sha49] dar.

Grundsätzlich müssen bei der Sicherheitsanalyse eines Public-Key-Verschlüsselungsverfahrens unterschiedliche Angreifer betrachtet werden, weshalb im Wesentlichen die folgenden Angriffstypen unterschieden werden:

- *Angriff mit gewählten Klartexten* (chosen plaintext attack (CPA)): Der Angreifer $\mathcal{A}$ kann sich Klartexte seiner Wahl verschlüsseln lassen (vgl. [DH76]).
- *Angriff mit gewählten Geheimtexten* (chosen ciphertext attack (CCA)): Der Angreifer $\mathcal{A}$ kann sich zusätzlich Geheimtexte entschlüsseln lassen (vgl. [NY90]).

Im Jahr 1988 haben S. Micali, C. Rackoff und B. Sloan [MRS88] die Äquivalenz von *polynomieller Ununterscheidbarkeit unter einem Angriff mit gewählten Klartexten* und *semantischer Sicherheit* bewiesen. Die polynomielle Ununterscheidbarkeit unter einem Angriff mit gewählten Klartexten eines Public-Key-Verschlüsselungsverfahrens wird durch das folgende Spiel modelliert:

Spiel 2.7.1 (**Polynomielle Ununterscheidbarkeit**). *Das Spiel wird zwischen einem polynomiellen Angreifer $\mathcal{A}$ und einem Verschlüsselungs-Orakel $\mathcal{O}_{\mathsf{Enc}}$ durchgeführt.*

Setup: *Das Orakel $\mathcal{O}_{\mathsf{Enc}}$ führt bei Eingabe des Sicherheitsparameters 1^λ den Algorithmus* GenEnc *aus und erhält ein Schlüsselpaar* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenEnc}\left(1^\lambda\right)$. *Anschließend sendet $\mathcal{O}_{\mathsf{Enc}}$ den öffentlichen Schlüssel* pk *an den Angreifer $\mathcal{A}$.*

Anfrage: *Der Angreifer $\mathcal{A}$ wählt zwei Nachrichten $m_0, m_1 \in \mathbb{M}_{\mathsf{Enc}}$ seiner Wahl und sendet diese an das Orakel $\mathcal{O}_{\mathsf{Enc}}$.*

Challenge: *Das Orakel $\mathcal{O}_{\mathsf{Enc}}$ wählt zufällig ein Bit $b \in_{\mathcal{R}} \{0,1\}$ und verschlüsselt die entsprechende Nachricht zu*

$$c \leftarrow \begin{cases} \mathsf{Enc}\left(\mathsf{pk}, m_0\right) & \textit{falls } b = 0, \\ \mathsf{Enc}\left(\mathsf{pk}, m_1\right) & \textit{falls } b = 1. \end{cases}$$

Dann sendet das Orakel $\mathcal{O}_{\mathsf{Enc}}$ den Geheimtext c an den Angreifer $\mathcal{A}$.

Ausgabe: *Der Angreifer $\mathcal{A}$ gibt ein Bit $b' \in \{0,1\}$ aus und gewinnt, falls $b' = b$ ist.*

Es ist zu beachten, dass der Angreifer $\mathcal{A}$ zu jedem Zeitpunkt beliebige Klartexte $m \in \mathbb{M}_{\mathsf{Enc}}$ seiner Wahl verschlüsseln kann, da $\mathcal{A}$ dazu nur den öffentlichen Schlüssel pk benötigt.

Definition 2.7.2 (**Sicherheit eines Verschlüsselungsverfahrens**). *Ein Public-Key-Verschlüsselungsverfahren* $(\mathsf{GenEnc}, \mathsf{Enc}, \mathsf{Dec})$ *mit Sicherheitsparameter $\lambda \in \mathbb{N}$ heißt* polynomiell ununterscheidbar unter einem Angriff mit gewählten Klartexten *(oder auch* semantisch sicher*), falls es für jeden polynomiellen Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion ν gibt, so dass für alle hinreichend großen $\lambda \in \mathbb{N}$ gilt:*

$$\Pr\left[\mathcal{A} \textit{ gewinnt Spiel 2.7.1}\right] \leq \frac{1}{2} + \nu(\lambda).$$

Ein Public-Key-Verschlüsselungsverfahren ist also polynomiell ununterscheidbar unter einem Angriff mit gewählten Klartexten bzw. semantisch sicher, falls jeder polynomielle Angreifer $\mathcal{A}$ nur mit einer vernachlässigbar besseren Wahrscheinlichkeit entscheiden kann, welche Nachricht verschlüsselt wurde, als zu raten. Aus diesem Grund sind deterministische Verschlüsselungsverfahren, wie beispielsweise die RSA-Verschlüsselung [RSA78], niemals semantisch sicher, weshalb sich die obige Definition 2.7.1 ausschließlich auf probabilistische Verschlüsselungsalgorithmen bezieht.

Bei einem Angriff mit gewählten Geheimtexten kann sich der Angreifer $\mathcal{A}$ vor der Wahl der beiden Nachrichten m_0, m_1 beliebige Geheimtexte vom Verschlüsselungs-Orakel $\mathcal{O}_{\mathsf{Enc}}$ entschlüsseln lassen. Ein noch stärkerer Angriff ist der *adaptive Angriff mit gewählten Geheimtexten* (chosen ciphertext attack 2 (CCA2)), bei dem sich der Angreifer zusätzlich vor der Ausgabe des Bits b' beliebige Geheimtexte, außer seiner empfangenen Challenge $c \leftarrow \mathsf{Enc}(m_b)$ vom Orakel $\mathcal{O}_{\mathsf{Enc}}$ entschlüsseln lassen kann (vgl. [DDN91]).

Äquivalent zum obigen Spiel 2.7.1 ist, dass der Angreifer $\mathcal{A}$ in der Challenge-Phase beide Geheimtexte $c_b \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$ und $c_{\bar{b}} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_{\bar{b}})$ für $b \in_{\mathcal{R}} \{0,1\}$ und $\bar{b} = 1 - b$ erhält, und entscheiden muss, welcher Geheimtext zur Nachricht m_0 bzw. m_1 gehört (vgl. [TY98, HTY99]).

Die semantische Sicherheit ist zwar der schwächste der drei Sicherheitsbegriffe, aber für die entwickelten (fairen) teilbaren elektronischen Geldsysteme in Kapitel 6 bis 9 ausreichend, um die Anonymität der Kunden zu beweisen.

In den folgenden beiden Unterabschnitten 2.7.1 und 2.7.2 werden zwei semantisch sichere Public-Key-Verschlüsselungsverfahren vorgestellt, die in den fairen teilbaren elektronischen Geldsystemen in Kapitel 7 eingesetzt werden, um eine Kunden- und Münzverfolgung zu garantieren.

2.7.1 Das ElGamal-Verschlüsselungsverfahren

Das *ElGamal-Verschlüsselungsverfahren* ist eine Erweiterung der Diffie-Hellman-Schlüsselvereinbarung [DH76] und wurde 1984 von T. ElGamal [ElG85a, ElG85b] entwickelt. Das ursprünglich von T. ElGamal vorgestellte Verschlüsselungsverfahren arbeitet in der Familie der zyklischen Gruppen $\mathbb{Z}_p^*$, kann aber in jeder Familie von zyklischen Gruppen mit Primzahlordnung eingesetzt werden, in der die Decisional-Diffie-Hellman-Annahme gilt.

GenEnc führt bei Eingabe des Sicherheitsparameters 1^λ den Algorithmus $\mathsf{Setup}_{\mathsf{G}}$ aus und erhält die Systemparameter $\mathsf{sp}_{\mathsf{G}} = (p, \mathbb{G}', g) \leftarrow \mathsf{Setup}_{\mathsf{G}}\left(1^\lambda\right)$. Dann wählt der Algorithmus zufällig eine Zahl $x \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Element $X = g^x$. Die Ausgabe ist das Schlüsselpaar

$$(\mathsf{pk}, \mathsf{sk}) = ((p, \mathbb{G}', g, X), x) \leftarrow \mathsf{GenEnc}\left(1^\lambda\right),$$

wobei für die Menge aller Nachrichten $\mathbb{M}_{\mathsf{Enc}} = \mathbb{G}$ gilt.

Enc verschlüsselt bei Eingabe des öffentlichen Schlüssels pk eine Nachricht $m \in \mathbb{G}$, indem der Algorithmus zufällig eine Zahl $r \in_{\mathcal{R}} \mathbb{Z}_p$ wählt und den Geheimtext

$$c = (c_1, c_2) = (m \cdot X^r, g^r) \leftarrow \mathsf{Enc}\,(\mathsf{pk}, m)$$

berechnet und ausgibt.

Dec entschlüsselt bei Eingabe des Schlüsselpaares $(\mathsf{pk}, \mathsf{sk})$ einen Geheimtext $c \in \mathbb{G}^2$, indem der Algorithmus den Klartext

$$\mathsf{Dec}\,(\mathsf{pk}, \mathsf{sk}, c) = c_1 \cdot c_2^{-x} = m$$

berechnet und ausgibt.

Das ursprünglich in der Familie von zyklischen Gruppen $\mathbb{Z}_p^*$ arbeitende ElGamal-Verschlüsselungsverfahren ist nicht semantisch sicher, da die quadratische Rest- bzw. Nichtresteigenschaft eines Geheimtextes mit der des entsprechenden Klartextes in Verbindung gebracht werden kann (vgl. [Mao03]). Insbesondere gilt in der Familie der Gruppen $\mathbb{Z}_p^*$ die Decisional-Diffie-Hellman-Annahme nicht (vgl. Unterabschnitt 2.6.3). Allerdings haben 1998 Y. Tsiounis und M. Yung [TY98] sowie D. Boneh [Bon98] gezeigt, dass das ElGamal-Verschlüsselungsverfahren für $\mathsf{Setup}_\mathsf{G}$ semantisch sicher ist, wenn die DDH-Annahme für $\mathsf{Setup}_\mathsf{G}$ gilt. In [TY98] wurde zwar der Beweis (für die *Äquivalenz* von DDH-Problem und semantischer Sicherheit der ElGamal-Verschlüsselung) formal ausschließlich für die Familie der eindeutig bestimmten zyklischen Untergruppen $\mathbb{G}_q$ der Primzahlordnung q von $\mathbb{Z}_p^*$ durchgeführt, dieser lässt sich aber analog auf jede Familie von zyklischen Gruppen mit Primzahlordnung übertragen.

Satz 2.7.1. *Das ElGamal-Verschlüsselungsverfahren* $(\mathsf{GenEnc}, \mathsf{Enc}, \mathsf{Dec})$ *ist semantisch sicher, falls die DDH-Annahme für* $\mathsf{Setup}_\mathsf{G}$ *gilt.*

Beweis. Siehe [TY98, Bon98]. □

2.7.2 Das lineare Verschlüsselungsverfahren

Das *lineare Verschlüsselungsverfahren* ist eine natürliche Erweiterung des ElGamal-Verschlüsselungsverfahrens und wurde 2004 von D. Boneh, X. Boyen und H. Shacham [BBS04] vorgestellt. Die Sicherheit des Verfahrens basiert auf der zur Decisional-Diffie-Hellman-Annahme ähnlichen Decisional-Linear-Diffie-Hellman-Annahme. Da davon ausgegangen wird, dass die DLDH-Annahme sogar in Familien von Gruppen gilt, in denen das DDH-Problem effizient gelöst werden kann (vgl. Unterabschnitt 2.6.3), ist das lineare Verschlüsselungsverfahren für eine Gruppe $\mathbb{G}_2$ geeignet, wobei $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ eine bilineare Abbildung vom Typ-1 bzw. Typ-2 ist (vgl. Unterabschnitt 2.5.1).

GenEnc führt bei Eingabe des Sicherheitsparameters 1^λ den Algorithmus $\mathsf{Setup_G}$ aus und erhält die Systemparameter $\mathsf{sp_G} = (p, \mathbb{G}', g) \leftarrow \mathsf{Setup_G}\left(1^\lambda\right)$. Dann wählt der Algorithmus zufällig zwei Zahlen $x, y \in_\mathcal{R} \mathbb{Z}_p^*$ und berechnet die Elemente $X = g^{\frac{1}{x}}$ sowie $Y = g^{\frac{1}{y}}$, so dass gilt: $X^x = Y^y = g$. Die Ausgabe ist das Schlüsselpaar

$$(\mathsf{pk}, \mathsf{sk}) = ((p, \mathbb{G}', g, X, Y), (x, y)) \leftarrow \mathsf{GenEnc}\left(1^\lambda\right),$$

wobei für die Menge aller Nachrichten $\mathbb{M}_{\mathsf{Enc}} = \mathbb{G}$ gilt.

Enc verschlüsselt bei Eingabe des öffentlichen Schlüssels pk eine Nachricht $m \in \mathbb{G}$, indem der Algorithmus zufällig zwei Zahlen $a, b \in_\mathcal{R} \mathbb{Z}_p$ wählt und den Geheimtext

$$c = (c_1, c_2, c_3) = \left(X^a, Y^b, m \cdot g^{a+b}\right) \leftarrow \mathsf{Enc}\left(\mathsf{pk}, m\right)$$

berechnet und ausgibt.

Dec entschlüsselt bei Eingabe des Schlüsselpaares $(\mathsf{pk}, \mathsf{sk})$ einen Geheimtext $c \in \mathbb{G}^3$, indem der Algorithmus den Klartext

$$\mathsf{Dec}\left(\mathsf{pk}, \mathsf{sk}, c\right) = c_3 \cdot c_1^{-x} \cdot c_2^{-y} = m$$

berechnet und ausgibt.

Satz 2.7.2. *Das lineare Verschlüsselungsverfahren* $(\mathsf{GenEnc}, \mathsf{Enc}, \mathsf{Dec})$ *ist semantisch sicher, falls die DLDH-Annahme für* $\mathsf{Setup_G}$ *gilt.*

Beweis. Siehe [BBS04]. □

2.8 Digitale Signaturen

Neben der Vertraulichkeit von Nachrichten, welches durch Verschlüsselungsverfahren realisiert werden kann, weitere zentrale Ziele der Kryptografie sind die *Integrität* und *Authentizität* einer Nachricht. Möchte der Empfänger also sicherstellen, dass eine Nachricht unverändert ist und tatsächlich vom erwarteten Sender stammt, können entweder *Message-Authentication-Codes* (MACs) (vgl. [BNS10]) oder *digitale Signaturen* eingesetzt werden. Während MACs nur dem Empfänger die Möglichkeit zur Überprüfung gewährleisten, kann eine digitale Signatur auf eine Nachricht auch von anderen Teilnehmern verifiziert werden, woraus eine *Verbindlichkeit* resultiert.

Digitale Signaturverfahren stellen somit neben den im vorherigen Abschnitt 2.7 vorgestellten Public-Key-Verschlüsselungsverfahren weitere wichtige kryptografische Anwendungen dar, die von W. Diffie und M. Hellman [DH76] erfunden und ebenfalls erst durch ihre vorgeschlagene asymmetrische Kryptografie ermöglicht wurden. Als digitales Äquivalent zur herkömmlichen Unterschrift kann ein *Signierer* eine beliebige Nachricht

signieren, so dass die daraus resultierende *Signatur* von jeder Partei überprüft werden kann. Dazu wird zur Signaturerstellung der private Schlüssel des Signierers und zur Verifikation der dazugehörige öffentliche Schlüssel verwendet. Somit werden bei einer digitalen Signatur die Schlüssel im Vergleich zur Verschlüsselung in umgekehrter Reihenfolge angewandt, was besonders bei der RSA-Signatur sowie der RSA-Verschlüsselung [RSA78] deutlich wird.

Da bei einem elektronischen Geldsystem die elektronischen Münzen von der Bank signiert werden und jeder Kunde und Händler deren Gültigkeit verifizieren kann, stellen digitale Signaturverfahren *den* zentralen Baustein nahezu aller elektronischer Geldsysteme dar.

Definition 2.8.1 (**Digitales Signaturverfahren**). *Ein Tripel* (GenSign, Sign, VerifySign) *polynomieller Algorithmen heißt* digitales Signaturverfahren*, falls die folgenden Eigenschaften erfüllt sind:*

1. *Der* Schlüsselgenerierungsalgorithmus GenSign *ist ein probabilistischer Algorithmus, der bei Eingabe des Sicherheitsparameters* 1^λ *einen privaten Schlüssel* sk *und einen öffentlichen Schlüssel* pk *erzeugt, durch den die Menge aller Nachrichten* $\mathbb{M}_{\mathsf{Sign}}$ *festgelegt ist. Die Ausgabe von* GenSign *ist das Schlüsselpaar* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenSign}\left(1^\lambda\right)$.

2. *Der* Signaturalgorithmus Sign *ist ein probabilistischer Algorithmus, der bei Eingabe eines öffentlichen Schlüssels* pk*, eines privaten Schlüssels* sk *und einer Nachricht* $m \in \mathbb{M}_{\mathsf{Sign}}$ *eine Signatur* $\sigma \leftarrow \mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, m)$ *erstellt und ausgibt.*

3. *Der* Verifikationsalgorithmus VerifySign *ist ein deterministischer Algorithmus, der bei Eingabe eines öffentlichen Schlüssels* pk*, einer Nachricht* m *und einer Signatur* σ *die korrekte Berechnung von* σ *verifiziert. Nach erfolgreicher Verifikation wird* 1 *und andernfalls* 0 *ausgegeben.*

4. *Für jedes Schlüsselpaar* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenSign}\left(1^\lambda\right)$ *und jede Nachricht* $m \in \mathbb{M}_{\mathsf{Sign}}$ *gilt:*
$$\mathsf{VerifySign}(\mathsf{pk}, m, \mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, m)) = 1.$$

Eine Signatur σ *auf eine Nachricht* $m \in \mathbb{M}_{\mathsf{Sign}}$ *heißt* gültig*, falls* $\mathsf{VerifySign}(\mathsf{pk}, m, \sigma) = 1$ *gilt.*

Analog zum vorherigen Abschnitt 2.7 werden auch hier an einigen Stellen die vereinfachten Schreibweisen $\mathsf{Sign}(m)$ und $\mathsf{VerifySign}(m, \sigma)$ verwendet.

Wie bei den Public-Key-Verschlüsselungsverfahren müssen auch für die Sicherheitsanalyse der digitalen Signaturverfahren verschiedene Angreifer betrachtet werden (vgl. [GMR84, GMR88]):

- *Angriff ohne bekannte Signaturen:* Der Angreifer $\mathcal{A}$ kennt nur den öffentlichen Schlüssel pk des Signierers.

- *Angriff mit bekannten Signaturen:* Der Angreifer $\mathcal{A}$ kennt den öffentlichen Schlüssel pk des Signierers und mehrere Nachrichten-Signatur-Paare $(m_1, \sigma_1), \ldots, (m_q, \sigma_q)$, wobei $\mathcal{A}$ keinen Einfluss auf die Wahl der Nachrichten $m_1, \ldots, m_q \in \mathbb{M}_{\mathsf{Sign}}$ hat.

- *Angriff mit gewählten Nachrichten:* Der Angreifer $\mathcal{A}$ kennt den öffentlichen Schlüssel pk des Signierers und erhält gültige Signaturen $\sigma_1, \ldots, \sigma_q$ auf von $\mathcal{A}$ gewählte Nachrichten $m_1, \ldots, m_q \in \mathbb{M}_{\mathsf{Sign}}$, wobei zuerst alle q Nachrichten gewählt werden, bevor der Angreifer $\mathcal{A}$ die Signaturen erhält.

- *Adaptiver Angriff mit gewählten Nachrichten*: Der Angreifer $\mathcal{A}$ kennt den öffentlichen Schlüssel pk des Signierers und erhält gültige Signaturen $\sigma_1, \ldots, \sigma_q$ auf von $\mathcal{A}$ gewählte Nachrichten $m_1, \ldots, m_q \in \mathbb{M}_{\mathsf{Sign}}$, wobei die Nachrichten in Abhängigkeit der erhaltenen Signaturen gewählt werden können.

Neben unterschiedlichen Angriffen müssen für die Analyse der Sicherheit eines digitalen Signaturverfahrens allerdings auch die möglichen Erfolge des Angreifers berücksichtigt werden, weshalb die folgenden vier Erfolgsstufen unterschieden werden (vgl. [GMR84, GMR88]):

- *Existentielle Fälschbarkeit:* Der Angreifer $\mathcal{A}$ kann zu einer Nachricht $m \in \mathbb{M}_{\mathsf{Sign}}$, die nicht notwendig von ihm gewählt wurde, eine gültige Signatur σ fälschen.

- *Selektive Fälschbarkeit:* Der Angreifer $\mathcal{A}$ kann zu einer von ihm gewählten Nachricht $m \in \mathbb{M}_{\mathsf{Sign}}$ eine gültige Signatur σ fälschen.

- *Universelle Fälschbarkeit:* Der Angreifer $\mathcal{A}$ kann zu jeder beliebigen Nachricht $m \in \mathbb{M}_{\mathsf{Sign}}$ eine gültige Signatur σ fälschen, obwohl er den privaten Signaturschlüssel sk nicht kennt.

- *Total-Break:* Der Angreifer $\mathcal{A}$ kann den privaten Signaturschlüssel sk berechnen.

Ein digitales Signaturverfahren erfüllt den höchsten Sicherheitsanspruch, wenn es für jeden polynomiellen Angreifer $\mathcal{A}$ unter einem adaptiven Angriff mit gewählten Nachrichten nicht existentiell fälschbar ist, weshalb dies der Standard-Begriff für die Sicherheit digitaler Signaturen ist [GMR88]. Das erste digitale Signaturverfahren mit diesem Sicherheitsniveau wurde 1988 von S. Goldwasser, S. Micali und R. Rivest [GMR88] entwickelt. Abhängig von den Möglichkeiten des Angreifers $\mathcal{A}$ wird weiter zwischen *existentieller Fälschbarkeit* und *starker existentieller Fälschbarkeit* unterschieden (vgl. [ADR02]).

- *Existentielle Fälschbarkeit unter einem adaptiven Angriff mit gewählten Nachrichten:* Der Angreifer $\mathcal{A}$ kann zu einer Nachricht $m \in \mathbb{M}_{\mathsf{Sign}}$, die nicht notwendig von ihm gewählt wurde, eine gültige Signatur σ erstellen, wobei $m \notin \{m_1, \ldots, m_q\}$ ist.

- *Starke existentielle Fälschbarkeit unter einem adaptiven Angriff mit gewählten Nachrichten:* Der Angreifer $\mathcal{A}$ kann zu einer Nachricht $m \in \mathbb{M}_{\mathsf{Sign}}$, die nicht notwendig von ihm gewählt wurde, eine gültige Signatur σ erstellen, wobei $(m, \sigma) \notin \{(m_1, \sigma_1), \ldots, (m_q, \sigma_q)\}$ ist.

Die (starke) existentielle Unfälschbarkeit eines digitalen Signaturverfahrens wird durch das folgende Spiel modelliert:

Spiel 2.8.1 (**(Starke) Existentielle Unfälschbarkeit**). *Das Spiel wird zwischen einem polynomiellen Angreifer $\mathcal{A}$ und einem Signatur-Orakel $\mathcal{O}_{\mathsf{Sign}}$ durchgeführt.*

Setup: *Das Orakel $\mathcal{O}_{\mathsf{Sign}}$ führt bei Eingabe des Sicherheitsparameters 1^λ den Algorithmus $\mathsf{GenSign}$ aus und erhält ein Schlüsselpaar $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenSign}\left(1^\lambda\right)$. Anschließend sendet $\mathcal{O}_{\mathsf{Sign}}$ den öffentlichen Schlüssel pk an den Angreifer $\mathcal{A}$.*

Anfragen: *Der Angreifer $\mathcal{A}$ stellt q adaptive Anfragen an das Orakel $\mathcal{O}_{\mathsf{Sign}}$, wobei $\mathcal{A}$ die Nachrichten $m_1, \ldots, m_q \in \mathbb{M}_{\mathsf{Sign}}$ abhängig von vorangegangenen Ergebnissen wählen kann. Das Orakel $\mathcal{O}_{\mathsf{Sign}}$ beantwortet jede Anfrage $1 \leq i \leq q$ mit einer gültigen Signatur $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, m_i)$. Sei $Q := \{(m_1, \sigma_1), \ldots, (m_q, \sigma_q)\}$ die Menge der Nachrichten-Signatur-Paare.*

Ausgabe: *Der Angreifer $\mathcal{A}$ gibt ein Nachrichten-Signatur-Paar $(m^\star, \sigma^\star)$ aus und gewinnt, falls $\mathsf{VerifySign}(\mathsf{pk}, m^\star, \sigma^\star) = 1$ gilt und die folgende Bedingung erfüllt ist:*

- existentielle Unfälschbarkeit: $(m^\star, \cdot) \notin Q$ *bzw.*
- starke existentielle Unfälschbarkeit: $(m^\star, \sigma^\star) \notin Q$.

Definition 2.8.2 (**Sicherheit eines digitalen Signaturverfahrens**). *Ein digitales Signaturverfahren* $(\mathsf{GenSign}, \mathsf{Sign}, \mathsf{VerifySign})$ *mit Sicherheitsparameter $\lambda \in \mathbb{N}$ heißt* existentiell unfälschbar *bzw.* stark existentiell unfälschbar *unter einem adaptiven Angriff mit gewählten Nachrichten, falls es für jeden polynomiellen Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion ν gibt, so dass für alle hinreichend großen $\lambda \in \mathbb{N}$ gilt:*

$$\Pr[\mathcal{A} \text{ gewinnt Spiel 2.8.1}] \leq \nu(\lambda).$$

Falls $\mathcal{A}$ das Spiel 2.8.1 gewinnt, werden das Nachrichten-Signatur-Paar $(m^\star, \sigma^\star)$ bzw. die Signatur $\sigma^\star$ auch als Fälschung *oder* gefälscht *bezeichnet.*

Ein digitales Signaturverfahren ist also existentiell unfälschbar unter einem adaptiven Angriff mit gewählten Nachrichten, falls jeder polynomielle Angreifer $\mathcal{A}$ nur mit vernachlässigbarer Wahrscheinlichkeit eine Signatur $\sigma^\star$ auf eine neue Nachricht $m^\star \notin \{m_1, \ldots, m_q\}$ fälschen kann, obwohl er sich beliebige Nachrichten $m_1, \ldots, m_q \in \mathbb{M}_{\mathsf{Sign}}$ seiner Wahl signieren lassen durfte. Kann jeder polynomielle Angreifer $\mathcal{A}$ nur mit vernachlässigbarer Wahrscheinlichkeit eine Signatur $\sigma^\star$ auf eine beliebige Nachricht $m^\star$ fälschen, obwohl er sich vorher beliebige Nachrichten $m_1, \ldots, m_q \in \mathbb{M}_{\mathsf{Sign}}$ seiner Wahl

signieren lassen durfte, ist das digitale Signaturverfahren stark existentiell unfälschbar unter einem adaptiven Angriff mit gewählten Nachrichten.

Wie bereits erwähnt, bietet ein unter einem adaptiven Angriff mit gewählten Nachrichten (stark) existentiell unfälschbares Signaturverfahren das höchste Maß an Sicherheit. Dieses wird bei einem elektronischen Geldsystem auch benötigt, da sich jeder Kunde beliebige Nachrichten von der Bank signieren lässt und nicht in der Lage sein darf, eine Signatur auf eine Nachricht zu fälschen.

Im folgenden Unterabschnitt 2.8.1 wird die häufig verwendete *Schnorr-Signatur* beschrieben.

Anschließend wird in Unterabschnitt 2.8.2 ein digitales Signaturverfahren vorgestellt, welches als Grundlage des in Unterabschnitt 2.12.1 behandelten *Signaturverfahrens mit effizienten Protokollen* dient.

2.8.1 Die Schnorr-Signatur

Im Jahr 1989 wurde von C. Schnorr ein digitales Signaturverfahren entwickelt, das eine Variante des ElGamal-Signaturverfahrens [ElG85b] darstellt und speziell für Smart-Cards entwickelt wurde. Das ursprünglich von C. Schnorr entwickelte Signaturverfahren, welches häufig als Schnorr-Signatur [Sch91] bezeichnet wird, arbeitet auf der eindeutig bestimmten zyklischen Untergruppe $\mathbb{G}_q$ der Ordnung q von $\mathbb{Z}_p^*$ für Primzahlen p und q mit $q \mid (p-1)$, kann aber auch auf jeder zyklischen Gruppe mit Primzahlordnung eingesetzt werden, in der die Diskreter-Logarithmus-Annahme gilt.

GenSign führt bei Eingabe des Sicherheitsparameters 1^λ den Algorithmus $\mathsf{Setup_G}$ aus und erhält die Systemparameter $\mathsf{sp_G} = (p, \mathbb{G}', g) \leftarrow \mathsf{Setup_G}\left(1^\lambda\right)$. Dann wählt der Algorithmus zufällig eine Zahl $x \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Element $X = g^x$. Anschließend wählt der Algorithmus eine kryptografische Hashfunktion $\mathsf{H}\colon \{0,1\}^* \to \mathbb{Z}_p$. Die Ausgabe ist das Schlüsselpaar

$$(\mathsf{pk}, \mathsf{sk}) = ((p, \mathbb{G}', g, X, \mathsf{H}), x) \leftarrow \mathsf{GenSign}\left(1^\lambda\right),$$

wobei für die Menge aller Nachrichten $\mathbb{M}_{\mathsf{Sign}} = \{0,1\}^*$ gilt.

Sign signiert bei Eingabe des Schlüsselpaares $(\mathsf{pk}, \mathsf{sk})$ eine Nachricht $m \in \{0,1\}^*$, indem der Algorithmus zufällig eine Zahl $\rho \in_{\mathcal{R}} \mathbb{Z}_p$ wählt und das Element $a = g^\rho \in \mathbb{G}$, den Hashwert $c = \mathsf{H}(a||m) \in \mathbb{Z}_p$ und die Zahl $z = \rho + cx \mod p$ berechnet. Die Ausgabe ist die Signatur

$$\sigma = (c, z) \leftarrow \mathsf{Sign}\,(\mathsf{pk}, \mathsf{sk}, m)\,.$$

VerifySign verifiziert bei Eingabe des öffentlichen Schlüssels pk eine Signatur $\sigma = (c, z)$ auf eine Nachricht m, indem der Algorithmus das Element $\tilde{a} = g^z X^{-c}$ berechnet und die Gleichung $c \stackrel{?}{=} \mathsf{H}\,(\tilde{a}||m)$ überprüft. Entsprechend ist die Ausgabe $\mathsf{VerifySign}\,(\mathsf{pk}, m, \sigma) = \{1, 0\}$.

D. Pointcheval und J. Stern [PS96, PS00] konnten zeigen, dass die Schnorr-Signatur im Random-Oracle-Modell (siehe Unterabschnitt 2.3.1) sicher ist.

Satz 2.8.1. *Das Schnorr-Signaturverfahren* $(\mathsf{GenSign}, \mathsf{Sign}, \mathsf{VerifySign})$ *ist im Random-Oracle-Modell existentiell unfälschbar unter einem adaptiven Angriff mit gewählten Nachrichten, falls die DLog-Annahme für* $\mathsf{Setup}_{\mathsf{G}}$ *gilt.*

Beweis. Siehe [Sch91, PS96, PS00]. □

2.8.2 Das Signaturverfahren von Boneh *et al.* (BBS)

In diesem Unterabschnitt wird das digitale Signaturverfahren von D. Boneh, X. Boyen und H. Shacham [BBS04] beschrieben, das auf einem Typ-2-Pairing $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ mit einem effizient berechenbaren Isomorphismus $\psi\colon \mathbb{G}_2 \to \mathbb{G}_1$ (und damit auch auf einem Typ-1-Pairing, vgl. Unterabschnitt 2.5.1) arbeitet und dessen Sicherheit auf der q-SDH-Annahme basiert. Der Isomorphismus ψ wird jedoch nur für den Sicherheitsbeweis in [BBS04] und nicht für das eigentliche Verfahren selbst benötigt. Im Jahr 2009 haben S. Chatterjee und A. Menezes [CM09] allerdings gezeigt, dass ein leicht verändertes Verfahren auch im Typ-3-Pairing sicher ist, obwohl es keinen effizient berechenbaren Isomorphismus zwischen den Gruppen $\mathbb{G}_1$ und $\mathbb{G}_2$ gibt (vgl. Unterabschnitt 2.6.4).

Bei dem Signaturverfahren, welches im Folgenden auch mit BBS bezeichnet wird, handelt es sich um ein sogenanntes *Gruppen-Signaturverfahren*, das den Mitgliedern einer Gruppe gewährleistet, sich anonym als Gruppenmitglied auszuweisen. Aus diesem Grund wählen die Teilnehmer keine Nachricht, sondern erhalten eine Signatur auf einen öffentlichen Generator. Im Folgenden wird nicht das vollständige Gruppen-Signaturverfahren, sondern lediglich der für das erweiterte Verfahren (BBS+) in Unterabschnitt 2.12.1 relevante Teil vorgestellt.

GenSign führt bei Eingabe des Sicherheitsparameters 1^λ den Algorithmus $\mathsf{Setup}_{\mathsf{Bil}}$ aus und erhält die Systemparameter $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, h) \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right)$, wobei $\mathbb{G}_1 = \langle g \rangle, \mathbb{G}_2 = \langle h \rangle$ und $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ eine bilineare Abbildung ist (siehe Abschnitt 2.5). Dann wählt der Algorithmus zufällig eine Zahl $x \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet das Element $X = h^x$. Die Ausgabe ist das Schlüsselpaar

$$(\mathsf{pk}, \mathsf{sk}) = \left((p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, h, X), x\right) \leftarrow \mathsf{GenSign}\left(1^\lambda\right),$$

wobei für die Menge aller Nachrichten $\mathbb{M}_{\mathsf{Sign}} = g$ gilt.

Sign signiert bei Eingabe des Schlüsselpaares $(\mathsf{pk}, \mathsf{sk})$ den Generator $g \in \mathbb{G}_1$, indem der Algorithmus zufällig eine Zahl $e \in_{\mathcal{R}} \mathbb{Z}_p$ wählt und das Element $\Sigma = g^{\frac{1}{x+e}}$ berechnet. Die Ausgabe ist die Signatur

$$\sigma = (\Sigma, e) \leftarrow \mathsf{Sign}\left(\mathsf{pk}, \mathsf{sk}, g\right).$$

VerifySign verifiziert bei Eingabe des öffentlichen Schlüssels pk eine Signatur $\sigma = (\Sigma, e)$ auf die Nachricht g, indem der Algorithmus die Gleichung

$$\hat{e}\left(\Sigma, Xh^{e}\right) \stackrel{?}{=} \hat{e}\left(g, h\right)$$

überprüft. Entsprechend ist die Ausgabe $\mathsf{VerifySign}\left(\mathsf{pk}, m, \sigma\right) = \{1, 0\}$.

In [BBS04] wurde jedoch auch eine Möglichkeit vorgestellt, wie ein Gruppenmitglied beweisen kann, im Besitz einer gültigen Signatur σ zu sein, ohne diese preisgeben zu müssen. Ein solcher Beweis wird als *Zero-Knowledge-Proof-of-Knowledge* (siehe Abschnitt 2.10) bezeichnet und die detaillierte Ausführung ist in Unterunterabschnitt 2.10.1.2 beschrieben.

2.9 Commitment-Schemata

Commitment-Schemata sind grundlegende Bestandteile vieler kryptografischer Protokolle, die es einer Partei ermöglichen, sich an eine Nachricht zu binden, während sie diese geheim hält. Somit stellen diese ein digitales Äquivalent zu einem versiegeltem Umschlag dar. Commitment-Schemata wurden 1982 von M. Blum [Blu83] eingeführt, um das Problem *Münzwurf am Telefon* zu lösen und werden beispielsweise für Identifikationsverfahren [FS86], Mehrparteien-Protokolle [GMW87] und in Verbindung mit vielen interaktiven Zero-Knowledge-Beweissystemen (z.B. [Dam90, GMW91, Dam99, CV05], vgl. Abschnitt 2.10) verwendet. In dieser Arbeit werden Commitment-Schemata zur Konstruktion der in Abschnitt 2.12 und Kapitel 3 beschriebenen digitalen Signaturverfahren mit speziellen Protokollen benötigt.

Vereinfacht formuliert ist ein Commitment-Schema ein effizientes Protokoll mit zwei Phasen zwischen zwei Parteien, dem *Sender* und dem *Receiver*. In der *Commit-Phase bindet* sich der Sender an eine Nachricht m, wobei er m geheim hält. Dazu berechnet der Sender ein *Commitment* bzgl. m und sendet dieses an den Receiver. In der *Reveal-Phase* wird m vom Sender offen gelegt, was auch als *öffnen* bezeichnet wird, so dass der Receiver die korrekte Berechnung des Commitments verifizieren kann.

Da in dieser Arbeit keine *interaktiven* Commitment-Schemata benötigt werden, wird nur ein *nichtinteraktives* Commitment-Schema definiert:

Definition 2.9.1 (**Commitment-Schema**). *Ein Tripel* $(\mathsf{GenCom}, \mathsf{Com}, \mathsf{VerifyCom})$ *polynomieller Algorithmen heißt* Commitment-Schema*, falls die folgenden Eigenschaften erfüllt sind:*

1. *Der* Schlüsselgenerierungsalgorithmus GenCom *ist ein probabilistischer Algorithmus, der bei Eingabe des Sicherheitsparameters* 1^{λ} *einen öffentlichen Schlüssel* pk *erzeugt, durch den die Menge aller Nachrichten* $\mathbb{M}_{\mathsf{Com}}$ *sowie die Menge aller Zufallswerte* $\mathbb{R}$ *festgelegt sind. Die Ausgabe von* GenCom *ist das Schlüsselpaar* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenCom}\left(1^{\lambda}\right)$*, wobei* sk *möglicherweise einige* Trapdoor-Informationen *enthält.*

2. *Der* Bindungsalgorithmus Com *ist ein probabilistischer Algorithmus, der bei Eingabe des öffentlichen Schlüssels* pk*, einer Nachricht* $m \in \mathbb{M}_{\mathsf{Com}}$ *und eines Zufallswertes* $r \in_{\mathcal{R}} \mathbb{R}$ *ein Commitment* $\mathsf{Com}(\mathsf{pk}, m; r) = C$ *berechnet und ausgibt.*

3. *Der* Verifikationsalgorithmus $\mathsf{VerifyCom}$ *ist ein deterministischer Algorithmus, der bei Eingabe des öffentlichen Schlüssels* pk*, eines Commitments* C*, einer Nachricht* m *und eines Zufallswertes* r *die korrekte Berechnung von* C *verifiziert. Nach erfolgreicher Verifikation wird* 1 *und andernfalls* 0 *ausgegeben.*

4. *Für jedes Schlüsselpaar* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenCom}\left(1^{\lambda}\right)$*, jede Nachricht* $m \in \mathbb{M}_{\mathsf{Com}}$ *und jeden Zufallswert* $r \in \mathbb{R}$ *gilt:*

$$\mathsf{VerifyCom}(\mathsf{pk}, \mathsf{Com}(\mathsf{pk}, m; r), m, r) = 1.$$

Wie in den vorherigen Abschnitten wird im weiteren Verlauf dieser Arbeit ein Commitment $\mathsf{Com}(\mathsf{pk}, m; r)$ an einigen Stellen vereinfachend auch mit $\mathsf{Com}(\mathsf{pk}, m)$, $\mathsf{Com}(m; r)$ oder $\mathsf{Com}(m)$ bezeichnet, wenn der Zufallswert $r \in_{\mathcal{R}} \mathbb{R}$ irrelevant ist, der in der Literatur auch häufig als *Decommitment*, *Zeuge* oder als *Opening* bezeichnet wird.

Damit ein Commitment-Schema $(\mathsf{GenCom}, \mathsf{Com}, \mathsf{VerifyCom})$ für kryptografische Protokolle verwendet werden kann, sollte es *verbergend* (hiding) und *bindend* (binding) sein. Informell gesprochen bedeutet verbergend, dass kein Receiver während der Commit-Phase Informationen über die Nachricht m erhält und bindend, dass kein Sender ein Commitment während der Reveal-Phase auf unterschiedliche Weisen öffnen kann. Je nach Verwendungszweck kann ein Commitment-Schema dabei entweder *perfekt verbergend und rechnerisch bindend* oder *rechnerisch verbergend und perfekt bindend* sein (vgl. [DN02]). In dieser Arbeit werden nur perfekt verbergende Commitment-Schemata benötigt.

Definition 2.9.2 (**Sicherheit eines Commitment-Schemas**)**.** *Ein Commitment-Schema* $(\mathsf{GenCom}, \mathsf{Com}, \mathsf{VerifyCom})$ *heißt* perfekt verbergend*, falls für jeden (nicht notwendig polynomiellen) Receiver* $\mathcal{R}$ *gilt:*

$$\Pr\Big[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenCom}\left(1^{\lambda}\right);\ m_0, m_1 \in \mathbb{M}_{\mathsf{Com}};\ b \in_{\mathcal{R}} \{0,1\};$$
$$C \leftarrow \mathsf{Com}(\mathsf{pk}, m_b);\ b' \leftarrow \mathcal{R}(\mathsf{pk}, m_0, m_1, C)\ :\ b' = b\Big] = \frac{1}{2}.$$

Ein Commitment-Schema $(\mathsf{GenCom}, \mathsf{Com}, \mathsf{VerifyCom})$ *mit Sicherheitsparameter* $\lambda \in \mathbb{N}$ *heißt* rechnerisch bindend*, falls es für jeden polynomiellen Sender* $\mathcal{S}$ *eine vernachlässigbare Funktion* ν *gibt, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\Pr\Big[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenCom}\left(1^{\lambda}\right);\ (C, (m_0, r_0), (m_1, r_1)) \leftarrow \mathcal{S}(\mathsf{pk})\ :\ m_0 \neq m_1 \wedge$$
$$\mathsf{VerifyCom}(\mathsf{pk}, C, m_0, r_0) = \mathsf{VerifyCom}(\mathsf{pk}, C, m_1, r_1) = 1\Big] \leq \nu(\lambda).$$

Ist $(\mathsf{GenCom}, \mathsf{Com}, \mathsf{VerifyCom})$ *ein perfekt verbergendes Commitment-Schema, so heißt das Commitment* $\mathsf{Com}(\mathsf{pk}, m; r) = C$ perfekt verbergend.

Ein Commitment-Schema ($\mathsf{GenCom}, \mathsf{Com}, \mathsf{VerifyCom}$) ist also perfekt verbergend, wenn selbst ein rechnerisch unbeschränkter Receiver bei Eingabe des öffentlichen Schlüssels pk, zweier Nachrichten $m_0, m_1 \in \mathbb{M}_{\mathsf{Com}}$ und eines Commitments $\mathsf{Com}(m_b)$ mit $b \in_{\mathcal{R}} \{0,1\}$ nur raten kann, welcher der beiden Nachrichten m_0, m_1 im Commitment $\mathsf{Com}(m_b)$ gebunden ist.

Im Folgenden wird ein perfekt verbergendes Commitment-Schema vorgestellt, das elementar für die Konstruktion der digitalen Signaturverfahren mit effizienten Protokollen in Abschnitt 2.12 sowie in Kapitel 3 ist. Anschließend wird ein Commitment-Schema für Polynome beschrieben, das für das entwickelte Commitment-Schema für das in Kapitel 3 beschriebene Signaturverfahren mit effizienten Protokollen benötigt wird.

2.9.1 Das Pedersen-Commitment-Schema

Ein häufig eingesetztes, perfekt verbergendes Commitment-Schema ist das *Pedersen-Commitment-Schema*, welches 1991 von T. Pedersen [Ped91] beschrieben wurde und auf der Diskreter-Logarithmus-Annahme basiert. Der Algorithmus Com des Pedersen-Commitment-Schemas wird zukünftig auch mit PedCom bezeichnet. Das ursprünglich vorgestellte Pedersen-Commitment-Schema arbeitet zwar auf der eindeutig bestimmten zyklischen Untergruppe $\mathbb{G}_q$ der Ordnung q von $\mathbb{Z}_p^*$ für zwei Primzahlen p und q mit $q \mid (p-1)$, kann aber auf jeder anderen zyklischen Gruppe mit Primzahlordnung eingesetzt werden, in der die Diskreter-Logarithmus-Annahme gilt.

GenCom führt bei Eingabe des Sicherheitsparameters 1^λ den Algorithmus $\mathsf{Setup_G}$ aus und erhält die Systemparameter $\mathsf{sp_G} = (p, \mathbb{G}', g) \leftarrow \mathsf{Setup_G}\left(1^\lambda\right)$. Dann wählt der Algorithmus zufällig zwei Generatoren $g_0, g_1 \in_{\mathcal{R}} \mathbb{G}$. Die Ausgabe ist der öffentliche Schlüssel

$$\mathsf{pk} = (p, \mathbb{G}', g_0, g_1) \leftarrow \mathsf{GenCom}\left(1^\lambda\right),$$

wobei für die Menge aller Nachrichten und Zufallswerte $\mathbb{M}_{\mathsf{Com}} = \mathbb{R} = \mathbb{Z}_p$ gilt.

PedCom bindet sich bei Eingabe des öffentlichen Schlüssels pk an eine Nachricht $m \in \mathbb{Z}_p$, indem der Algorithmus zufällig eine Zahl $r \in_{\mathcal{R}} \mathbb{Z}_p$ wählt und das Pedersen-Commitment

$$C = g_0^r g_1^m \leftarrow \mathsf{PedCom}(\mathsf{pk}, m)$$

berechnet und ausgibt.

VerifyCom verifiziert bei Eingabe des öffentlichen Schlüssels pk, einer Nachricht $m \in \mathbb{Z}_p$ und einer Zufallszahl $r \in \mathbb{Z}_p$ die korrekte Berechnung des Commitments $C \in \mathbb{G}$, indem der Algorithmus die Gleichung

$$C \stackrel{?}{=} g_0^r g_1^m$$

überprüft. Entsprechend ist die Ausgabe $\mathsf{VerifyCom}(\mathsf{pk}, C, m, r) = \{1, 0\}$.

Das Pedersen-Commitment-Schema ist perfekt verbergend und rechnerisch bindend, falls für $\mathsf{Setup}_{\mathsf{G}}$ die Diskreter-Logarithmus-Annahme für $\mathsf{Setup}_{\mathsf{G}}$ gilt (vgl. [Ped91]). Denn durch die Zufallszahl $r \in_{\mathcal{R}} \mathbb{Z}_p$ wird die Nachricht $m \in \mathbb{Z}_p$ perfekt verborgen, da es für jedes Commitment $C \in \mathbb{G}$ und jede Nachricht $m \in \mathbb{Z}_p$ genau eine Zahl $r \in \mathbb{Z}_p$ mit $C = g_0^r g_1^m$ gibt. Seien dazu $\delta \in \mathbb{Z}_p^*$ mit $g_1 = g_0^\delta$ der diskrete Logarithmus von g_1 zur Basis g_0 und $C = g_0^c$ für eine Zahl $c \in \mathbb{Z}_p$, dann ist $r = c - \delta m \mod p$ somit eindeutig festgelegt. Die Eigenschaft rechnerisch bindend folgt, da es aufgrund des Darstellungsproblems nicht effizient möglich ist, zu einem Commitment zwei verschiedene Darstellungen $(r, m) \in \mathbb{Z}_p^2$ und $(r', m') \in \mathbb{Z}_p^2$ bzgl. des Generatortupels (g_0, g_1) anzugeben.

Das Pedersen-Commitment-Schema kann leicht erweitert werden, um ein Commitment bzgl. einer Nachricht $m = (m_1, \ldots, m_L) \in \mathbb{Z}_p^L$ zu berechnen, wobei $(m_1, \ldots, m_L)$ auch als *Nachrichtenblock* bezeichnet wird. Dazu gibt der Schlüsselgenerierungsalgorithmus $\mathsf{GenCom}\left(1^\lambda, L\right)$ bei zusätzlicher Eingabe der Zahl $L \in \mathbb{N}$ neben einer Primzahl p und einer Gruppe $\mathbb{G}$ ein zufälliges Generatortupel $(g_0, \ldots, g_L) \in_{\mathcal{R}} \mathbb{G}^{L+1}$ aus, wobei für die Menge aller Nachrichten $\mathbb{M}_{\mathsf{Com}} = \mathbb{Z}_p^L$ gilt. Der Bindungsalgorithmus PedCom berechnet entsprechend das Pedersen-Commitment

$$C = g_0^r g_1^{m_1} \cdots g_L^{m_L} \leftarrow \mathsf{PedCom}\left(\mathsf{pk}, m\right).$$

Dieses erweiterte Pedersen-Commitment-Schema ist ebenfalls perfekt verbergend und rechnerisch bindend unter der Diskreter-Logarithmus-Annahme (vgl. [CKOS01]) und bildet die Grundlage für die in Abschnitt 2.12 thematisierten digitalen Signaturverfahren mit effizienten Protokollen. Insgesamt gilt der folgende Satz.

Satz 2.9.1. *Die beiden oben beschriebenen Varianten des Pedersen-Commitment-Schemas* $(\mathsf{GenCom}, \mathsf{PedCom}, \mathsf{VerifyCom})$ *sind perfekt verbergend und rechnerisch bindend, falls die DLog-Annahme für* $\mathsf{Setup}_{\mathsf{G}}$ *gilt.*

Beweis. Siehe [Ped91, CKOS01]. □

2.9.2 Das Polynom-Commitment-Schema von Kate *et al.*

A. Kate, G. Zaverucha und I. Goldberg [KZG10a, KZG10b] haben ein Commitment-Schema für Polynome $\phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ mit $\deg(\phi) \leq q$ entwickelt, dessen Sicherheit auf der q-SDH-Annahme basiert und im Folgenden mit *Polynom-Commitment-Schema* bezeichnet wird. In diesem Unterabschnitt wird nur die deterministische Variante vorgestellt, welche nicht verbergend ist, da diese für das Commitment-Schema in Kapitel 3 verwendet wird. Die probabilistische Variante aus [KZG10a] ist zwar perfekt verbergend, kann aber nicht für das in Kapitel 3 entwickelte Commitment-Schema verwendet werden.

Formal ist das Polynom-Commitment-Schema zwar nur im Typ-1-Pairing beschrieben, kann aber auch im Typ-2 und 3 angewandt werden (vgl. [KZG10b]). Insbesondere können die Algorithmen $\mathsf{GenCom}, \mathsf{Com}$ und $\mathsf{VerifyCom}$ auch auf einer Gruppe $\mathbb{G}$ arbeiten, ohne dass ein Pairing benötigt wird. Es werden allerdings zwei weitere Algorithmen

$\mathsf{EvalWit}$ und $\mathsf{VerifyEval}$, mit den folgenden Eigenschaften beschrieben, wobei der Schlüsselgenerierungsalgorithmus GenCom zusätzlich den maximalen Grad $K \in \mathbb{N}$ als Eingabe erhält und durch den öffentlichen Schlüssel pk auch eine Menge aller Indizes $\mathbb{I}$ festgelegt ist.

1. Der *Zeugenberechnungsalgorithmus* $\mathsf{EvalWit}$ ist ein deterministischer Algorithmus, der bei Eingabe des öffentlichen Schlüssels pk, eines Polynoms $\phi(\mathsf{x}) \in \mathbb{M}_{\mathsf{Com}}$ und eines Index $i \in \mathbb{I}$ das Tripel $\mathsf{EvalWit}(\mathsf{pk}, \phi(\mathsf{x}), i) = (i, \phi(i), W_i)$ berechnet und ausgibt, wobei W_i der Zeuge für die Berechnung $\phi(i)$ von $\phi(\mathsf{x})$ an der Stelle i ist.

2. Der *Berechnungsverifikationsalgorithmus* $\mathsf{VerifyEval}$ ist ein deterministischer Algorithmus, der bei Eingabe des öffentlichen Schlüssels pk, eines Commitments C, eines Index i, eines Wertes $\phi(i)$ und eines Zeugen W_i die korrekte Berechnung $\phi(i)$ verifiziert. Nach erfolgreicher Verifikation wird 1 und andernfalls 0 ausgegeben.

3. *(Durchführbarkeit:)* Für jedes Schlüsselpaar $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenCom}\left(1^\lambda, K\right)$, jedes Polynom $\phi(\mathsf{x}) \in \mathbb{M}_{\mathsf{Com}}$ und jeden Index $i \in \mathbb{I}$ gilt:

$$\mathsf{VerifyEval}(\mathsf{pk}, \mathsf{Com}(\mathsf{pk}, \phi(\mathsf{x})), \mathsf{EvalWit}(\mathsf{pk}, \phi(\mathsf{x}), i)) = 1.$$

Durch die neuen Algorithmen muss auch die Sicherheitsdefinition erweitert werden. Somit wird zusätzlich neben der Gebundenheit an die Nachricht $m = \phi(\mathsf{x}) \in \mathbb{M}_{\mathsf{Com}}$, was der Sicherheitseigenschaft rechnerisch bindend aus Definition 2.9.2 entspricht und in [KZG10a] als *polynomial binding* bezeichnet wird, eine Gebundenheit an die Berechnung $\phi(i)$ gefordert, was im Folgenden mit *Berechnungsgebundenheit* bzw. *berechnungsbindend* bezeichnet wird (und in [KZG10a] als *evaluation binding* definiert ist).

Definition 2.9.3 (**Berechnungsgebundenheit**)**.** *Ein Commitment-Schema* (GenCom, $\mathsf{Com}, \mathsf{VerifyCom}, \mathsf{EvalWit}, \mathsf{VerifyEval}$) *mit Sicherheitsparameter* $\lambda \in \mathbb{N}$ *heißt* berechnungsbindend*, falls es für jeden polynomiellen Sender* $\mathcal{S}$ *eine vernachlässigbare Funktion* ν *gibt, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\Pr\Big[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenCom}\left(1^\lambda, K\right);\ \left(C, (i, \phi(i), W_i), \left(i, \phi(i)', W_i'\right)\right) \leftarrow \mathcal{S}(\mathsf{pk})\ :\ \phi(i) \neq \phi(i)'$$
$$\wedge\ \mathsf{VerifyEval}(\mathsf{pk}, C, i, \phi(i), W_i) = \mathsf{VerifyEval}\left(\mathsf{pk}, C, i, \phi(i)', W_i'\right) = 1\Big] \leq \nu(\lambda).$$

Des Weiteren soll es nicht möglich sein, ein Commitment C für ein Polynom $\phi(\mathsf{x})$ mit $\deg(\phi) > K$ zu berechnen. Diese Eigenschaft wird in [KZG10a] als *strong correctness* bezeichnet. In dieser Arbeit wird dies allerdings, in Anlehnung an den *beschränkten Akkumulator* (siehe Abschnitt 2.11), ebenfalls als *Beschränktheit* benannt und ist in [KZG10b] folgendermaßen als Spiel definiert:

Spiel 2.9.1 (**Beschränktheit**)**.** *Das Spiel wird zwischen einem polynomiellen Angreifer* $\mathcal{A}$ *und einem Challenger* $\mathcal{C}$ *durchgeführt. Sei* $K' \in \mathbb{N}$ *polynomiell beschränkt mit* $K' > K$*.*

Setup: *Der Challenger $\mathcal{C}$ führt bei Eingabe des Sicherheitsparameters 1^λ den Algorithmus* GenCom *aus und erhält das Schlüsselpaar* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenCom}\left(1^\lambda, K\right)$ *des Commitment-Schemas. Anschließend sendet $\mathcal{C}$ den öffentlichen Schlüssel* pk *an den Angreifer $\mathcal{A}$.*

Commit: *Der Angreifer $\mathcal{A}$ sendet ein Commitment C sowie eine Zahl $K' > K$ an $\mathcal{C}$.*

Challenge: *Der Challenger wählt zufällig $K'+1$ paarweise verschiedene Indizes $i^{(1)}, \ldots, i^{(K'+1)} \in_{\mathcal{R}} \mathbb{I}$ und sendet diese an $\mathcal{A}$.*

Ausgabe: *Der Angreifer $\mathcal{A}$ gibt $K'+1$ Tripel $\left(i^{(j)}, \phi\left(i^{(j)}\right), W_{i^{(j)}}\right)$ aus und gewinnt das Spiel, falls*

1. $\mathsf{VerifyEval}\left(\mathsf{pk}, C, i^{(j)}, \phi\left(i^{(j)}\right), W_{i^{(j)}}\right) = 1$ *für alle $1 \leq j \leq K'+1$ gilt,*
2. *die Interpolation der $K'+1$ Werte $\phi\left(i^{(1)}\right), \ldots, \phi\left(i^{(K'+1)}\right)$ ein Polynom vom Grad K' erzeugt und*
3. *die Interpolation (in den Exponenten) der $K'+1$ Zeugen $W_{i^{(1)}}, \ldots, W_{i^{(K'+1)}}$ ein Polynom vom Grad $K'-1$ erzeugt.*

Definition 2.9.4 (Beschränktheit). *Ein Commitment-Schema* $(\mathsf{GenCom}, \mathsf{Com}, \mathsf{VerifyCom}, \mathsf{PolyWit}, \mathsf{VerifyEval})$ *mit Sicherheitsparameter $\lambda \in \mathbb{N}$ heißt* beschränkt*, falls es für jeden polynomiellen Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion ν gibt, so dass für alle hinreichend großen $\lambda \in \mathbb{N}$ gilt:*

$$\Pr\left[\mathcal{A} \text{ gewinnt Spiel 2.9.1}\right] \leq \nu(\lambda).$$

Das Polynom-Commitment-Schema ist nach Definition von [KZG10a] dann verborgen, wenn der Empfänger $\mathcal{R}$ bei Eingabe eines Commitments $\mathsf{Com}(\mathsf{pk}, \phi(\mathsf{x}))$ und $k \leq \deg(\phi)$ Tripel $\left(i^{(j)}, \phi\left(i^{(j)}\right), W_{i^{(j)}}\right)$ mit $\mathsf{VerifyEval}\left(\mathsf{pk}, \mathsf{Com}(\mathsf{pk}, \phi(\mathsf{x})), i^{(j)}, \phi\left(i^{(j)}\right), W_{i^{(j)}}\right) = 1$ für $1 \leq j \leq k$ die Berechnung $\phi(i^\star)$ für einen Index $i^\star \notin \{i^{(1)}, \ldots, i^{(k)}\}$ nur mit vernachlässigbarer Wahrscheinlichkeit bestimmen kann. Da wir dieses Schema für das in Kapitel 3 entwickelte Signaturverfahren benötigen und dort nur Zero-Knowledge-Beweise (siehe Abschnitt 2.10) über die korrekte Berechnung von $\phi(i)$ verwendet werden, ist diese Definition für uns allerdings irrelevant.

Nun wird die Konstruktion des Polynom-Commitment-Schemas aus [KZG10a] beschrieben, wobei wir den Algorithmus Com auch mit $\mathsf{PolyCom}$ bezeichnen.

GenCom führt bei Eingabe des Sicherheitsparameters 1^λ sowie des maximalen Grades $K \in \mathbb{N}$ den Algorithmus $\mathsf{Setup}_{\mathsf{Bil}}$ aus und erhält die Systemparameter $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, h) \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right)$, wobei $\mathbb{G}_1 = \langle u_0 \rangle, \mathbb{G}_2 = \langle h \rangle$ und $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ eine bilineare Abbildung ist. Dann wählt der Algorithmus zufällig eine Zahl $\alpha \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Elemente $h_1 = h^\alpha$ sowie $u_i = u_0^{\alpha^i}$ für $i = 1, \ldots, K$. Die Ausgabe ist der öffentliche Schlüssel

$$\mathsf{pk} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, \ldots, u_K, h, h_1) \leftarrow \mathsf{GenCom}\left(1^\lambda, K\right),$$

wobei für die Menge aller Nachrichten $\mathbb{M}_{\mathsf{Com}} = \{\phi(\mathsf{x}) : \phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}], \deg(\phi) \leq K\}$, für die Menge aller Zufallswerte $\mathbb{R} = \emptyset$ und für die Menge aller Indizes $\mathbb{I} = \mathbb{Z}_p$ gilt.

PolyCom bindet sich bei Eingabe des öffentlichen Schlüssels pk an ein Polynom $\phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ mit $\phi(\mathsf{x}) = \sum_{j=0}^{K} \phi_j \mathsf{x}^j$, indem der Algorithmus das Polynom-Commitment

$$C = \mathsf{PolyCom}(\mathsf{pk}, \phi(\mathsf{x})) = u_0^{\phi(\alpha)} = \prod_{j=0}^{K} u_j^{\phi_j}$$

berechnet und ausgibt. Somit kann das Commitment C mit den öffentlichen Generatoren $u_0, \ldots, u_K$ berechnet werden, ohne dass die Kenntnis der geheimen Zahl α erforderlich ist.

VerifyCom verifiziert bei Eingabe des öffentlichen Schlüssels pk und eines Polynoms $\phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ die korrekte Berechnung des Commitments $C \in \mathbb{G}_1$, indem der Algorithmus die Gleichung

$$C \stackrel{?}{=} u_0^{\phi(\alpha)}$$

überprüft. Entsprechend ist die Ausgabe $\mathsf{VerifyCom}(\mathsf{pk}, C, \phi(\mathsf{x})) = \{1, 0\}$.

EvalWit berechnet bei Eingabe des öffentlichen Schlüssels pk, eines Polynoms $\phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ mit $\deg(\phi) \leq K$ und einer Zahl $i \in \mathbb{Z}_p$ den Wert $\phi(i) \in \mathbb{Z}_p$ und den Zeugen $W_i \in \mathbb{G}_1$ für die Berechnung $\phi(i)$, indem der Algorithmus das Polynom

$$\phi_i(\mathsf{x}) = \frac{\phi(\mathsf{x}) - \phi(i)}{\mathsf{x} - i} \in \mathbb{Z}_p[\mathsf{x}] \text{ und den Zeugen } W_i = u_0^{\phi_i(\alpha)}$$

berechnet, wobei der Zeuge W_i analog zum Commitment mit den öffentlichen Generatoren $u_0, \ldots, u_{K-1}$ berechnet wird. Beachte, dass das Polynom $\phi(\mathsf{x}) - \phi(i)$ ohne Rest durch das Polynom $\mathsf{x} - i$ teilbar ist (vgl. [KZG10a]). Die Ausgabe ist $\mathsf{EvalWit}(\mathsf{pk}, \phi(\mathsf{x}), i) = (i, \phi(i), W_i)$.

VerifyEval verifiziert bei Eingabe des öffentlichen Schlüssels pk, eines Commitments $C \in \mathbb{G}_1$, einer Zahl $i \in \mathbb{Z}_p$ und eines Zeugen $W_i \in \mathbb{G}_1$ die korrekte Berechnung $\phi(i) \in \mathbb{Z}_p$, indem der Algorithmus die Gleichung

$$\hat{e}(C, h) \stackrel{?}{=} \hat{e}\left(W_i, h_1 h^{-i}\right) \hat{e}\left(u_0^{\phi(i)}, h\right)$$

überprüft. Entsprechend ist die Ausgabe $\mathsf{VerifyEval}(\mathsf{pk}, C, i, \phi(i), W_i) = \{1, 0\}$.

Das Schema ist durchführbar, da gilt:

$$\begin{aligned}\hat{e}(C, h) &= \hat{e}\left(u_0^{\phi(\alpha)}, h\right) = \hat{e}\left(u_0^{\phi_i(\alpha)\cdot(\alpha-i)+\phi(i)}, h\right) = \hat{e}\left(u_0^{\phi_i(\alpha)}, h^{(\alpha-i)}\right) \hat{e}\left(u_0^{\phi(i)}, h\right) \\ &= \hat{e}\left(W_i, h_1 h^{-i}\right) \hat{e}\left(u_0^{\phi(i)}, h\right).\end{aligned}$$

Satz 2.9.2. *Das Polynom-Commitment-Schema* (GenCom, PolyCom, VerifyCom, EvalWit, VerifyEval) *ist rechnerisch bindend, berechnungsbindend und beschränkt, falls die q-SDH-Annahme und die q-polyDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gelten.*

Beweis. Siehe [KZG10a, KZG10b]. □

2.10 Zero-Knowledge-Proofs-of-Knowledge

Wichtige und häufig verwendete kryptografische Bausteine sind die sogenannten *Zero-Knowledge-Proofs-of-Knowledge*. Sie werden insbesondere für die in den folgenden Abschnitten 2.11 und 2.12 vorgestellten beschränkten Akkumulatoren und digitalen Signaturverfahren mit effizienten Protokollen benötigt, die als Grundbausteine für die elektronischen Geldsysteme dienen.

Vereinfacht formuliert ist ein Zero-Knowledge-Proof-of-Knowledge (ZKPoK) ein Beweis, der nichts außer der Gültigkeit einer Behauptung preisgibt. Das Konzept der Zero-Knowledge-Beweise wurde erstmals von S. Goldwasser, S. Micali und C. Rackoff [GMR85, GMR89] vorgestellt und beruht auf den im selben Artikel beschriebenen *interaktiven Beweissystemen*. Ein solcher interaktiver Beweis wird von einem *Prover* $\mathcal{P}$ (dem Beweisführenden) und einem *Verifier* $\mathcal{V}$ (dem Verifizierenden) durchgeführt, wobei der Prover $\mathcal{P}$ den Verifier $\mathcal{V}$ von der Gültigkeit einer Behauptung überzeugen möchte. Die Kommunikation zwischen $\mathcal{P}$ und $\mathcal{V}$ findet dabei interaktiv statt, was bedeutet, dass sich beide Parteien abwechselnd Nachrichten senden und $\mathcal{V}$ am Ende des Protokolls akzeptiert oder ablehnt, wobei die Ausgabe 1 als accept und 0 als reject interpretiert wird.

Im Folgenden wird mit $\mathcal{P}^*$ ein *betrügerischer Prover* bezeichnet, der versucht $\mathcal{V}$ von einer falschen Behauptung zu überzeugen. Ein *betrügerischer Verifier*, der sich nicht notwendig gemäß Protokoll verhält, wird entsprechend mit $\mathcal{V}^*$ bezeichnet.

Definition 2.10.1 (**Interaktives Beweissystem**). *Gegeben sei eine Sprache* $L \subseteq \{0, 1\}^*$. *Ein Paar* $(\mathcal{P}, \mathcal{V})$ *interaktiver Turing-Maschinen heißt* interaktives Beweissystem *für die Sprache* L, *falls* $\mathcal{V}$ *polynomiell ist und die folgenden Eigenschaften erfüllt sind:*

1. Durchführbarkeit *(Completeness): Für alle* $x \in L$ *mit* $|x| = \lambda$ *gibt es eine vernachlässigbare Funktion* ν, *so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\Pr\Big[\mathcal{P}(x) \leftrightarrow \mathcal{V}(x) \to 1\Big] \geq 1 - \nu(\lambda).$$

2. Korrektheit *(Soundness): Für alle* $x^* \notin L$ *mit* $|x^*| = \lambda$ *gibt es eine vernachlässigbare Funktion* μ, *so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\Pr\Big[\mathcal{P}^*(x^*) \leftrightarrow \mathcal{V}(x^*) \to 1\Big] \leq \mu(\lambda).$$

Die Durchführbarkeit eines interaktiven Beweises gewährleistet also, dass jeder *ehrliche* Prover $\mathcal{P}$ den Verifier $\mathcal{V}$, außer mit vernachlässigbarer Wahrscheinlichkeit, von der Gültigkeit seiner Behauptung überzeugen kann. Dagegen garantiert die Korrektheit, dass jeder unbeschränkte, betrügerische Prover $\mathcal{P}^*$ den Verifier $\mathcal{V}$ nur mit vernachlässigbarer Wahrscheinlichkeit von der Gültigkeit einer falschen Behauptung überzeugen kann.

Möchte der Prover $\mathcal{P}$ dem Verifier $\mathcal{V}$ allerdings nicht nur beweisen, dass eine Behauptung wahr ist ($x \in L$), sondern auch, dass er *etwas weiß* (z.B. geheime Informationen), wird ein interaktiver *Proof-of-Knowledge* durchgeführt. Um darzustellen, was es bedeutet, dass $\mathcal{P}$ etwas weiß, wird folgende Definition benötigt:

Definition 2.10.2. *Gegeben sei eine binäre Relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$. Dann ist*

- *w ein* Zeuge *für x, falls gilt: $(x,w) \in R$;*
- *$L_R := \{x : \exists w \text{ mit } (x,w) \in R\}$ eine* Sprache*;*
- *R eine* polynomiell beschränkte *Relation, falls es ein Polynom* poly *gibt, so dass für alle $(x,w) \in R$ gilt: $|w| \leq \mathsf{poly}(|x|)$;*
- *R eine $\mathcal{NP}$-Relation, falls R polynomiell beschränkt ist und es einen polynomiellen Algorithmus gibt, der bei Eingabe von (x,w) entscheidet, ob $(x,w) \in R$ gilt.*

Die Formulierung *$\mathcal{P}$ weiß etwas*, bedeutet nun, dass $\mathcal{P}$ für x einen Zeugen w kennt, so dass für eine bestimmte Relation R gilt: $(x,w) \in R$. Diese Eigenschaft wird dadurch modelliert, dass es einen *Knowledge-Extractor* $\mathcal{E}$ gibt, der bei Eingabe von x den Prover $\mathcal{P}$ als *Blackbox* verwendet und einen Zeugen w mit $(x,w) \in R$ extrahiert. Das heißt, eine Maschine weiß etwas, wenn sie von einen anderen Maschine dazu verwendet werden kann, die relevanten Informationen effizient zu berechnen.

Der Begriff Proof-of-Knowledge (PoK) wurde erstmals von S. Goldwasser, S. Micali und C. Rackoff [GMR85] vorgestellt und anschließend von weiteren Autoren formal definiert (z.B. [BG92]).

Definition 2.10.3 (Proof-of-Knowledge). *Gegeben seien eine polynomiell beschränkte Relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$, die zugehörige Sprache L_R und eine Funktion $\kappa\colon \{0,1\}^* \to [0,1]$. Ein Paar $(\mathcal{P},\mathcal{V})$ interaktiver Funktionen heißt interaktives* Proof-of-Knowledge*-System für die Relation R, falls $\mathcal{V}$ in probabilistischer, polynomieller Zeit berechnet werden kann und die folgenden Eigenschaften erfüllt sind:*

1. Durchführbarkeit *(Completeness): Für alle $(x,w) \in R$ gilt:*

$$\Pr\Big[\mathcal{P}(x,w) \leftrightarrow \mathcal{V}(x) \to 1\Big] = 1.$$

2. Beweiskraft *(Validity, Knowledge Soundness): Sei $\epsilon(x)$ die Wahrscheinlichkeit, mit der $\mathcal{V}$ akzeptiert, wobei x seine Eingabe ist. Es gibt eine probabilistische Orakel-Maschine $\mathcal{E}$, die den Prover $\mathcal{P}^*$ als Blackbox verwendet und eine Konstante $c > 0$, so dass für jede interaktive Funktion $\mathcal{P}^*$ und alle $x \in L_R$ gilt:*

Falls $\epsilon(x) > \kappa(x)$ *ist, dann gibt* $\mathcal{E}$ *einen korrekten Zeugen* w *mit* $(x, w) \in R$ *in erwarteter Laufzeit begrenzt durch*

$$\frac{|x|^c}{\epsilon(x) - \kappa(x)}$$

aus. Die Orakel-Maschine $\mathcal{E}$ *heißt* universeller Knowledge-Extractor *und* κ *heißt die* Knowledge Error Funktion.

Durch die Durchführbarkeit eines interaktiven Proof-of-Knowledge-Systems $(\mathcal{P}, \mathcal{V})$ ist also gewährleistet, dass jeder ehrliche Prover $\mathcal{P}$ den Verifier $\mathcal{V}$ immer von der Gültigkeit seiner Behauptung überzeugen kann. Intuitiv kann die Knowledge Error Funktion $\kappa(x)$ als die Wahrscheinlichkeit betrachtet werden, dass $\mathcal{V}$ akzeptiert, obwohl $\mathcal{P}^*$ keinen Zeugen w für x kennt. Vereinfacht formuliert garantiert die Beweiskraft, dass $\mathcal{V}$ nur akzeptiert, falls $\mathcal{P}^*$ einen Zeugen kennt, das heißt, falls $\mathcal{P}^*$ von einem Extractor $\mathcal{E}$ verwendet werden kann, um einen Zeugen zu berechnen.

Eine äquivalente Formulierung zur Beweiskraft ist, dass $\mathcal{E}$ eine probabilistische voraussichtlich-polynomielle Orakel-Maschine ist, die mit Wahrscheinlichkeit $\epsilon(x) - \kappa(x)$ einen Zeugen w mit $(x, w) \in R$ extrahiert (vgl. [BG92]).

Wenn ein interaktives Beweissystem bzw. ein interaktives Proof-of-Knowledge-System $(\mathcal{P}, \mathcal{V})$ die zusätzliche Eigenschaft besitzt, dass jeder Verifier $\mathcal{V}^*$ durch die Interaktion mit $\mathcal{P}$ keine weiteren Informationen erhält, außer dass die Behauptung $x \in L$ bzw. $(x, w) \in R$ wahr ist, spricht man von *Zero-Knowledge* (ZK). Das bedeutet, dass jeder Verifier $\mathcal{V}^*$ nach der Interaktion mit $\mathcal{P}$ nicht mehr Wissen besitzt, als er vor der Interaktion besaß, außer dass die Behauptung wahr ist. Mathematisch wird diese *Zero-Knowledge-Eigenschaft* dadurch modelliert, dass der Beweis von einem polynomiellen *Simulator* $\mathcal{SIM}$ simuliert werden kann, ohne dass $\mathcal{SIM}$ die Behauptung beweisen kann. Je nach Simulation wird zwischen verschiedenen Arten von Zero-Knowledge unterschieden. Zunächst wird die folgende Definition benötigt ([GM84, GMR85]).

Definition 2.10.4 (Ununterscheidbarkeit). *Gegeben seien eine Sprache* $L \subseteq \{0,1\}^*$ *und zwei Mengen von Zufallsvariablen* $A = \{A(x)\}_{x \in L}$ *sowie* $B = \{B(x)\}_{x \in L}$. *Die Mengen* A *und* B *heißen*

- perfekt ununterscheidbar*, falls für alle* $x \in L$ *die Zufallsvariablen* $A(x)$ *und* $B(x)$ *identisch verteilt sind;*
- rechnerisch ununterscheidbar*, falls es keinen polynomiellen Algorithmus gibt, der die beiden Mengen unterscheiden kann. Das heißt, für jeden probabilistischen, polynomiellen Algorithmus* $\mathcal{A}$ *und alle* $x \in L$ *mit* $|x| = \lambda$ *gibt es eine vernachlässigbare Funktion* ν*, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\left|\Pr\left[1 \leftarrow \mathcal{A}(x, A(x))\right] - \Pr\left[1 \leftarrow \mathcal{A}(x, B(x))\right]\right| \leq \nu(\lambda).$$

Mit diesen zwei Typen der Ununterscheidbarkeit können verschiedene Arten von Zero-Knowledge definiert werden. Die folgende Definition orientiert sich an [GMR89] und gilt für alle interaktiven Protokolle $(\mathcal{P}, \mathcal{V})$.

Definition 2.10.5 (**Zero-Knowledge**). *Gegeben seien eine Sprache $L \subseteq \{0,1\}^*$ und ein interaktives Protokoll $(\mathcal{P}, \mathcal{V})$, wobei $\mathsf{view}_{\mathcal{V}}^{\mathcal{P}}(x)$ die* Protokollansicht *von $\mathcal{V}$ bei einer Interaktion mit $\mathcal{P}$ mit gemeinsamer Eingabe x bezeichnet. Das Protokoll $(\mathcal{P}, \mathcal{V})$ heißt*

- perfekt bzw. rechnerisch zero-knowledge*, wenn es für alle probabilistischen, polynomiellen interaktiven Turing-Maschinen $\mathcal{V}^*$ einen probabilistischen voraussichtlich-polynomiellen Algorithmus $\mathcal{SIM}$ gibt, so dass für alle $x \in L$ die Protokollansicht $\mathsf{view}_{\mathcal{V}^*}^{\mathcal{P}}(x)$ und die Ausgabe des Algorithmus $\mathcal{SIM}(x)$ perfekt bzw. rechnerisch ununterscheidbar sind;*

- perfekt bzw. rechnerisch honest-verifier zero-knowledge*, wenn es einen probabilistischen voraussichtlich-polynomiellen Algorithmus $\mathcal{SIM}$ gibt, so dass für alle $x \in L$ die Protokollansicht $\mathsf{view}_{\mathcal{V}}^{\mathcal{P}}(x)$ und die Ausgabe des Algorithmus $\mathcal{SIM}(x)$ perfekt bzw. rechnerisch ununterscheidbar sind.*

Die *Honest-Verifier-Zero-Knowledge-Eigenschaft* ist zwar schwächer, aber für reale Anwendungen und Sicherheitsbeweise dennoch von großer Bedeutung, da dadurch effiziente interaktive 3-Schritte-Honest-Verifier-Zero-Knowledge-Proofs-of-Knowledge (siehe Unterabschnitt 2.10.1) konstruiert werden können. Diese können dann im *Random-Oracle-Modell* (vgl. Unterabschnitt 2.3.1) mit Hilfe der *Fiat-Shamir-Heuristik* (vgl. Unterabschnitt 2.10.2) in *nichtinteraktive* Beweise (vgl. Unterabschnitt 2.10.2) transformiert werden.

2.10.1 Sigma-Protokolle

In diesem Unterabschnitt wird eine besondere Klasse von interaktiven Zero-Knowledge-Proofs-of-Knowledge, die sogenannten *Sigma-Protokolle* oder auch *Σ-Protokolle*, behandelt, die auf I. Damgård [Dam90] zurückzuführen sind. Weiter formalisiert und untersucht wurden diese Protokolle insbesondere durch R. Cramer und I. Damgård in [CDS94, Cra96, CDM00].

Ein Σ-Protokoll ist ein interaktives Protokoll $(\mathcal{P}, \mathcal{V})$, welches aus drei Schritten besteht, wobei $\mathcal{P}$ für eine bestimmte Relation R in zero-knowledge beweist, dass $(x, w) \in R$ gilt. Der Ablauf eines Σ-Protokolls ist schemenhaft in folgender Abbildung 2.2 dargestellt, wobei x die gemeinsame Eingabe von $\mathcal{P}$ und $\mathcal{V}$ und w die private Eingabe von $\mathcal{P}$ ist. Nach erfolgreicher Verifikation gibt $\mathcal{V}$ die Zahl 1 und andernfalls die Zahl 0 aus.

Im ersten Schritt berechnet der Prover $\mathcal{P}$ ein *Commitment* a und sendet dieses an den Verifier $\mathcal{V}$. Dieser wählt im zweiten Schritt zufällig eine *Challenge* c und sendet sie an $\mathcal{P}$. Auf diese Challenge antwortet $\mathcal{P}$ im dritten Schritt mit einer *Response* z. Anschließend verifiziert $\mathcal{V}$ das Tupel (x, a, c, z) und akzeptiert den Beweis oder lehnt ihn ab.

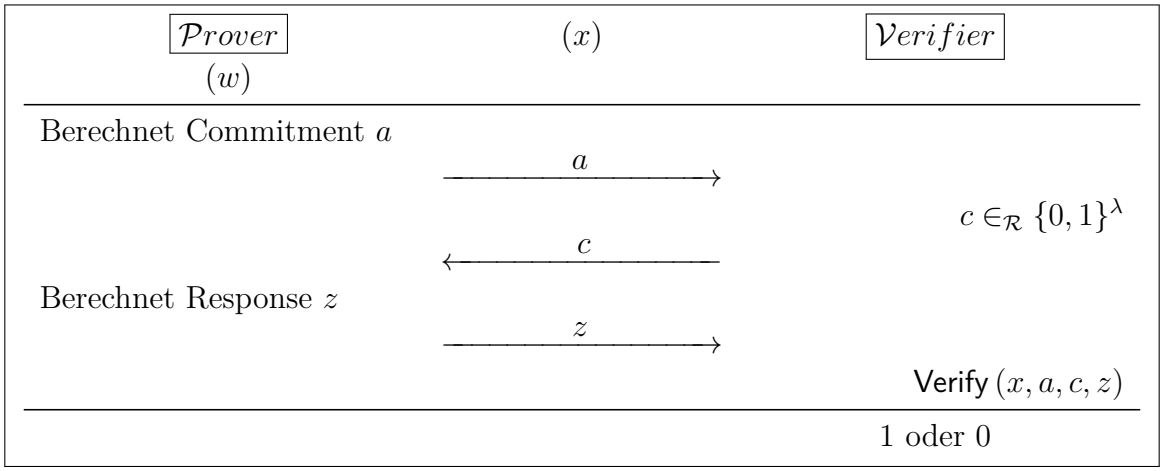

Abbildung 2.2: Ablauf eines Σ-Protokolls

Definition 2.10.6 (Σ**-Protokolle**). *Gegeben seien eine $\mathcal{NP}$-Relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$, die Sprache L_R und ein interaktives Protokoll $(\mathcal{P}, \mathcal{V})$, wobei $\mathcal{P}$ und $\mathcal{V}$ probabilistische polynomielle Turing-Maschinen sind. Dann heißt $(\mathcal{P}, \mathcal{V})$ ein* Σ-Protokoll *für die Relation R, falls die folgenden Eigenschaften erfüllt sind:*

1. 3-Schritte-Protokoll: *Das Protokoll $(\mathcal{P}, \mathcal{V})$ besteht aus drei Schritten und läuft wie in Abbildung 2.2 ab, wobei das Tripel $(a, c, z) \leftarrow \mathsf{view}_{\mathcal{V}(x)}^{\mathcal{P}(x,w)}$ die* Protokollansicht *zwischen $\mathcal{V}$ und $\mathcal{P}$ beschreibt.*

2. Durchführbarkeit *(Completeness): Für alle $(x, w) \in R$ gilt:*

$$\Pr\Big[\mathcal{P}(x, w) \leftrightarrow \mathcal{V}(x) \to 1\Big] = 1.$$

3. Spezielle Korrektheit *(Special Soundness): Es gibt einen probabilistischen polynomiellen Algorithmus $\mathcal{E}$, so dass für alle $x \in L_R$ und alle Paare von Protokollansichten, $(a, c, z), (a, c', z') \leftarrow \mathsf{view}_{\mathcal{V}(x)}^{\mathcal{P}(x,\cdot)}$ mit $c \neq c'$ gilt:*

$$\Pr\Big[w \leftarrow \mathcal{E}\Big(x, (a, c, z), (a, c', z')\Big) \ : \ (x, w) \in R\Big] = 1.$$

4. Special-Honest-Verifier-Zero-Knowledge*: Es gibt einen probabilistischen polynomiellen Algorithmus $\mathcal{SIM}$, so dass für alle $x \in L_R$*

$$\Pr\Big[(a', c, z') \leftarrow \mathcal{SIM}(x, c) \ : \ \mathsf{Verify}(x, a', c, z') = 1\Big] = 1$$

gilt und die Protokollansicht (a, c, z) und die Ausgabe des Algorithmus $\mathcal{SIM}(x, c)$ identisch verteilt sind.

Die spezielle Korrektheit bedeutet somit, dass es einen polynomiellen Knowledge-Extractor $\mathcal{E}$ gibt, der bei Eingabe eines beliebigen $x \in L_R$ und zwei beliebiger Protokollansichten $(a, c, z), (a, c', z')$ mit $c \neq c'$ einen Zeugen w mit $(x, w) \in R$ berechnet. Die Special-Honest-Verifier-Zero-Knowledge-Eigenschaft garantiert, dass jede Protokollansicht (a, c, z) zwischen einem ehrlichen Prover $\mathcal{P}$ und einem ehrlichen Verifier $\mathcal{V}$ mit gemeinsamer Eingabe x von einem polynomiellen Simulator $\mathcal{SIM}$ bei Eingabe von (x, c) perfekt simuliert werden kann. Die leicht abgeänderte Eigenschaft, bei der $\mathcal{SIM}$ nur x als Eingabe erhält und eine korrekte Protokollansicht (a', c', z') perfekt simulieren kann, wird als Honest-Verifier-Zero-Knowledge-Eigenschaft bezeichnet (vgl. [CDM00]).

Jedes Σ-Protokoll für eine Relation R mit einer Challenge-Länge $\lambda \in \mathbb{N}$ ist ein interaktives Beweissystem für eine Sprache L_R und ein interaktives Proof-of-Knowledge-System für die Relation R mit Knowledge-Error $2^{-\lambda}$ (siehe [CDM00, Dam11]). Insgesamt ist die Special-Honest-Verifier-Zero-Knowledge-Eigenschaft zwar schwächer als die Zero-Knowledge-Eigenschaft, aber sie bleibt sogar bei paralleler Ausführung eines Σ-Protokolls, im Gegensatz zur Zero-Knowledge-Eigenschaft, erhalten. Somit sind alle Eigenschaften eines Σ-Protokolls invariant unter parallelen Ausführungen (siehe [Dam11]). Des Weiteren kann jedes Σ-Protokoll mit Challenge-Länge $\lambda \in \mathbb{N}$ durch die Verwendung eines Commitment-Schemas in ein perfektes 4-Schritte-Zero-Knowledge-Proof-of-Knowledge-System mit Knowledge-Error $2^{-\lambda}$ transformiert werden (siehe [CDM00]).

In den folgenden beiden Abschnitten werden zentrale Σ-Protokolle vorgestellt, die als Grundlagen für die in Kapitel 6 bis 9 entwickelten teilbaren elektronischen Geldsysteme dienen.

2.10.1.1 Zero-Knowledge-Proofs-of-Knowledge über diskrete Logarithmen

Ein einfacher und häufig verwendeter Zero-Knowledge-Proof-of-Knowledge ist ein Beweis über die Kenntnis eines bestimmten diskreten Logarithmus. Sei dazu $\mathbb{G} = \langle g \rangle$ eine zyklische Gruppe der Primzahlordnung $p = \Theta\left(2^{\lambda}\right)$ mit Generator g. Sei weiter $X = g^x$ für eine zufällige Zahl $x \in_{\mathcal{R}} \mathbb{Z}_p$. Möchte nun ein Prover $\mathcal{P}$ einen Verifier $\mathcal{V}$ davon überzeugen, dass $\mathcal{P}$ die Zahl x kennt, wird der folgende Proof-of-Knowledge durchgeführt, wobei die übliche Notation aus [CS97, Cam98] verwendet wird:

$$PoK\left[(x) : X = g^x\right].$$

Die in den runden Klammern stehende Zahl $x \in \mathbb{Z}_p$ ist die geheime Zahl, deren Kenntnis vom Prover $\mathcal{P}$ bewiesen wird und die die Gleichung $X = g^x$ erfüllt. Die Primzahl p, die Gruppe $\mathbb{G}$, der Generator $g \in \mathbb{G}$ sowie das Element X sind dabei dem Prover $\mathcal{P}$ sowie dem Verifier $\mathcal{V}$ bekannt. Im Detail wird dieser Proof-of-Knowledge folgendermaßen durchgeführt und ist in Abbildung 2.3 dargestellt:

Commitment: Der Prover $\mathcal{P}$ wählt zufällig eine Zahl $\rho \in_{\mathcal{R}} \mathbb{Z}_p$, berechnet das Commitment $a = g^{\rho}$ und sendet a an $\mathcal{V}$.

Challenge: Der Verifier $\mathcal{V}$ wählt zufällig eine Challenge $c \in_{\mathcal{R}} \mathbb{Z}_p$ und sendet c an $\mathcal{P}$.

Response: Der Prover $\mathcal{P}$ berechnet die Response $z = \rho + cx \mod p$ und sendet z an $\mathcal{V}$.

Verify: Der Verifier $\mathcal{V}$ verifiziert die Gleichung $g^z \stackrel{?}{=} aX^c$.

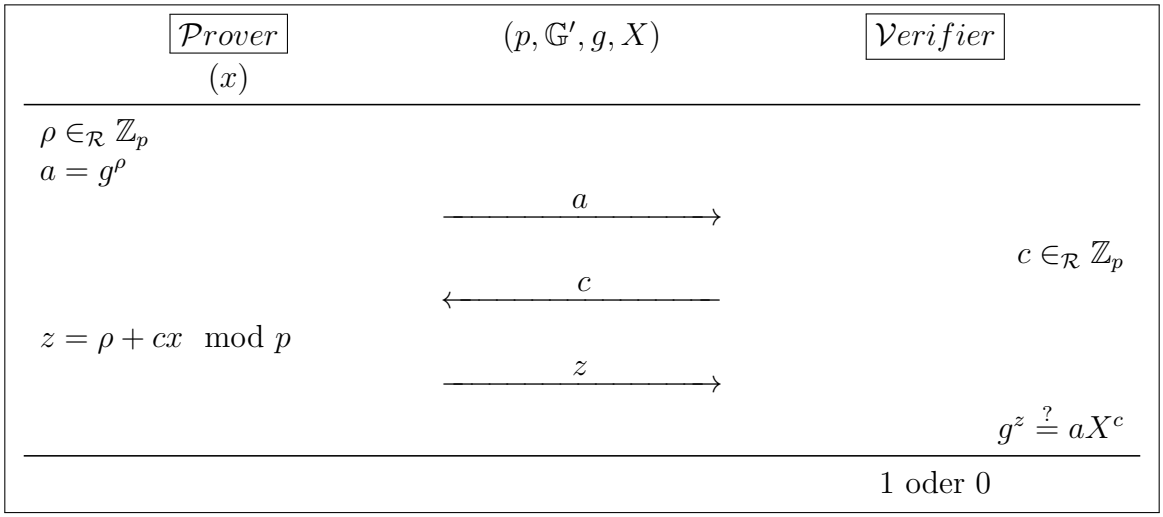

Abbildung 2.3: Ablauf des Protokolls $PoK\,[(x) : X = g^x]$

Um zu zeigen, dass es sich bei diesem Protokoll um ein Σ-Protokoll gemäß Definition 2.10.6 handelt, müssen die folgenden Eigenschaften gezeigt werden:

Durchführbarkeit: Verhält sich der Prover $\mathcal{P}$ wie beschrieben, gilt

$$g^z = g^{\rho+cx} = g^\rho\,(g^x)^c = aX^c.$$

Spezielle Korrektheit: Der Knowledge-Extractor $\mathcal{E}$ erhält die Parameter $(p, \mathbb{G}', g, X)$, das Commitment a und zwei Challenge-Response-Paare (c, z) sowie (c', z') mit $c \neq c'$ und $g^z = aX^c$ sowie $g^{z'} = aX^{c'}$. Daraus folgt $a = g^z X^{-c} = g^{z'} X^{-c'}$ und weiter

$$X = g^{\frac{z-z'}{c-c'}}.$$

Der Knowledge-Extractor $\mathcal{E}$ berechnet nun die Zahl $x = \frac{z-z'}{c-c'} \mod p$ mit $X = g^x$. Somit hat $\mathcal{E}$ die Zahl x aus dem Proof-of-Knowledge *extrahiert*.

Special-Honest-Verifier-Zero-Knowledge: Der Simulator $\mathcal{SIM}$ erhält die Parameter $(p, \mathbb{G}', g, X)$ sowie die Challenge c. Dann wählt $\mathcal{SIM}$ zufällig die Response $z' \in_{\mathcal{R}} \mathbb{Z}_p$ und definiert das Commitment $a' := g^{z'} X^c$. Somit erfüllt das Tripel (a', c, z') trivialer Weise die Verifikation und da $z' \in_{\mathcal{R}} \mathbb{Z}_p$ ist, folgt $a' \in_{\mathcal{R}} \mathbb{G}$. Damit sind (a, c, z) und (a', c, z') identisch verteilt.

Dieser Proof-of-Knowledge entspricht dabei genau dem *Schnorr-Identifikationsverfahren* [Sch91] (vgl. auch [Cam98, Kapitel 2]).

Weitere Beispiele sind ein Proof-of-Knowledge über die Kenntnis einer Darstellung eines Elements $A \in \mathbb{G}$ bzgl. eines Generatortupels $(g_1, g_2) \in \mathbb{G}^2$

$$PoK\left[(a_1, a_2) : A = g_1^{a_1} g_2^{a_2}\right]$$

oder ein Proof-of-Knowledge über eine bestimmte Beziehung eines diskreten Logarithmus

$$PoK\left[(a_1, a_2) : A = g_1^{a_1} \ \wedge \ B = g_1^{a_2} g_2^{a_1}\right].$$

Die Durchführung sowie der Beweis, dass es sich um ein Σ-Protokoll handelt, erfolgen jeweils analog zum obigen Beispiel.

2.10.1.2 Zero-Knowledge-Proof-of-Knowledge über ein SDH-Paar

Ein weiterer zentraler Zero-Knowledge-Proof-of-Knowledge ist ein Beweis über ein SDH-Paar $(\Sigma, e) \in \mathbb{G}_1 \times \mathbb{Z}_p$ mit $g = \Sigma^{x+e}$ (vgl. Definition 2.6.14 und Unterabschnitt 2.8.2). Seien dazu $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, h)$ mit $\mathbb{G}_1 = \langle g \rangle$ und $\mathbb{G}_2 = \langle h \rangle$ die Parameter einer bilinearen Abbildung $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Seien weiter $g_0, g_1 \in_{\mathcal{R}} \mathbb{G}_1$ zufällige Generatoren und $X = h^x$ für eine zufällige Zahl $x \in_{\mathcal{R}} \mathbb{Z}_p^*$. Jedes SDH-Paar (Σ, e) erfüllt die Gleichung $\hat{e}\left(\Sigma, X h^e\right) = \hat{e}\left(g, h\right)$. Möchte nun ein Prover $\mathcal{P}$ einen Verifier $\mathcal{V}$ davon überzeugen, dass $\mathcal{P}$ ein SDH-Paar (Σ, e) kennt, wird der folgende Proof-of-Knowledge durchgeführt, wobei $B_1 = g_0^{b_1} g_1^{b_2}, B_2 = \Sigma g_1^{b_1}, \beta_1 = b_1 e$ und $\beta_2 = b_2 e$ für zwei Zufallszahlen $b_1, b_2 \in_{\mathcal{R}} \mathbb{Z}_p$ ist (vgl. [BBS04, ASM06] und Anhang A):

$$PoK\Bigg[(b_1, b_2, \beta_1, \beta_2, e) : B_1 = g_0^{b_1} g_1^{b_2} \ \wedge \ 1 = B_1^e g_0^{-\beta_1} g_1^{-\beta_2} \ \wedge \ \frac{\hat{e}\left(B_2, X\right)}{\hat{e}(g, h)} = \hat{e}\left(g_1, X\right)^{b_1} \hat{e}\left(g_1, h\right)^{\beta_1} \hat{e}\left(B_2, h\right)^{-e}\Bigg].$$

2.10.2 Die Fiat-Shamir-Heuristik und nichtinteraktive Proofs-of-Knowledge

Im Jahr 1986 haben A. Fiat und A. Shamir [FS86] ein Verfahren vorgeschlagen, mit dem jedes interaktive Σ-Protokoll in ein *nichtinteraktives* Protokoll transformiert werden kann. Dazu wird die Challenge c nicht wie bei einem Σ-Protokoll zufällig vom Verifier $\mathcal{V}$ gewählt, sondern durch eine geeignete Hashfunktion $\mathsf{H}\colon \{0,1\}^* \to \mathbb{Z}_p$ generiert. Die nichtinteraktive Variante des in Unterunterabschnitt 2.10.1.1 vorgestellten Proof-of-Knowledge wird folgendermaßen generiert:

Commitment: Der Prover $\mathcal{P}$ wählt zufällig eine Zahl $\rho \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Commitment $a = g^{\rho}$.

Challenge: Der Prover $\mathcal{P}$ berechnet die Challenge $c = \mathsf{H}(a)$.

Response: Der Prover $\mathcal{P}$ berechnet die Response $z = \rho + cx \mod p$ und sendet das Paar (c, z) an $\mathcal{V}$.

Verify: Der Verifier $\mathcal{V}$ berechnet das Commitment $\tilde{a} = g^z X^{-c}$ und verifiziert die Gleichung $c \stackrel{?}{=} \mathsf{H}(\tilde{a})$.

Fließt neben dem Commitment a eine beliebige Nachricht $m \in \{0,1\}^*$ mit in die Hashfunktion H zur Berechnung der Challenge $c = \mathsf{H}(a||m)$ ein, handelt es sich um eine Signatur auf die Nachricht m bzgl. des öffentlichen Schlüssels X. Aus diesem Grund werden die nichtinteraktiven Varianten eines Proof-of-Knowledge auch als Signature-of-Knowledge (SoK) bezeichnet (vgl. [CS97, Cam98]). Für eine Signature-of-Knowledge auf eine Nachricht $m \in \{0,1\}^*$ wird die Notation

$$SoK\left[(x) : X = g^x\right](m)$$

verwendet. Diese Signature-of-Knowledge über die Kenntnis eines diskreten Logarithmus entspricht genau der *Schnorr-Signatur* aus Unterabschnitt 2.8.1.

Im Folgenden fließen in jeder Signature-of-Knowledge die Elemente, über die eine Relation bewiesen wird, zur Berechnung der Challenge mit in die Hashfunktion ein (siehe auch Anhang A). Somit wird bei obiger Signature-of-Knowledge

$$SoK\left[(x) : X = g^x\right](m)$$

die Challenge c durch $c = \mathsf{H}\left(X||a||m\right)$ berechnet.

Des Weiteren wird im Folgenden ein Protokoll stets direkt abgebrochen, falls eine Signature-of-Knowledge (bzw. ein Proof-of-Knowledge) nicht erfolgreich verifiziert wird.

2.11 Beschränkte Akkumulatoren

Ein wichtiger Baustein für aktuelle teilbare elektronische Geldsysteme, wie beispielsweise [ASM08] und [CG10], sind sogenannte *beschränkte Akkumulatoren* [AWSM07]. Akkumulatoren wurden 1993 von J. Benaloh und M. de Mare [BM94] eingeführt und erlauben die Aggregation einer Menge von Werten $x_1, \ldots, x_k$ zu einem einzigen Element V, dem *Akkumulator*. Nach der Akkumulierung kann zu jedem Wert $x_i \in \{x_1, \ldots, x_k\}$ ein *Zeuge* $\hat{W}_i$ angegeben werden, der bestätigt, dass der Wert x_i in V akkumuliert ist. In [BM94] werden zur Konstruktion eines Akkumulators Einweg-Hashfunktionen H mit $\mathsf{H}\colon \mathbb{Y} \times \mathbb{X} \to \mathbb{Y}$ vorgeschlagen. Bei einem gegebenen Initialwert $y \in \mathbb{Y}$ des Akkumulators werden die Werte $x_1, \ldots, x_k \in \mathbb{X}$ durch die Berechnung $V = \mathsf{H}\left(\cdots \mathsf{H}\left(\mathsf{H}\left(y, x_1\right), x_2\right), \cdots, x_k\right) \in \mathbb{Y}$ akkumuliert. Weiter forderten die Autoren die Eigenschaft, dass ein Akkumulator-Schema *quasi kommutativ* ist, das heißt, dass die Gleichung $\mathsf{H}\left(\mathsf{H}\left(y, x_1\right), x_2\right) = \mathsf{H}\left(\mathsf{H}\left(y, x_2\right), x_1\right)$ für alle $y \in \mathbb{Y}$ und $x_1, x_2 \in \mathbb{X}$ gilt.

Da es in der Literatur verschiedene Definitionen für Akkumulatoren gibt, ist die folgende an [CH10] angelehnt.

Definition 2.11.1 (**Akkumulator-Schema**). *Ein Tupel* $(\mathsf{GenAcc}, \mathsf{Acc}, \mathsf{Wit}, \mathsf{VerifyAcc})$ *polynomieller Algorithmen heißt* Akkumulator-Schema*, falls die folgenden Eigenschaften erfüllt sind:*

1. *Der* Schlüsselgenerierungsalgorithmus GenAcc *ist ein probabilistischer Algorithmus, der bei Eingabe des Sicherheitsparameters* 1^λ *und einer oberen Schranke* $K \in \mathbb{N}$ *einen öffentlichen Schlüssel* pk *erzeugt, durch den die Menge aller Werte* $\mathbb{X}$ *festgelegt ist. Die Ausgabe von* GenAcc *ist das Schlüsselpaar* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenAcc}\left(1^\lambda, K\right)$*, wobei* sk *möglicherweise einige Trapdoor-Informationen enthält.*

2. *Der* Akkumulierungsalgorithmus Acc *ist ein deterministischer Algorithmus, der bei Eingabe eines öffentlichen Schlüssels* pk *und einer Wertemenge* $X = \{x_1, \ldots, x_k\} \subset \mathbb{X}$ *mit* $k \leq K$ *einen Akkumulator* $\mathsf{Acc}(\mathsf{pk}, X) = V$ *berechnet und ausgibt.*

3. *Der* Zeugenberechnungsalgorithmus Wit *ist ein deterministischer Algorithmus, der bei Eingabe eines öffentlichen Schlüssels* pk*, einer Wertemenge* X *und eines Wertes* $x_i \in X$ *einen Zeugen* $\mathsf{Wit}(\mathsf{pk}, X, x_i) = \hat{W}_i$ *für den Wert* x_i *berechnet und ausgibt.*

4. *Der* Verifikationsalgorithmus $\mathsf{VerifyAcc}$ *ist ein deterministischer Algorithmus, der bei Eingabe eines öffentlichen Schlüssels* pk*, eines Akkumulators* V*, eines Wertes* x_i *und eines Zeugen* $\hat{W}_i$ *verifiziert, ob der Wert* x_i *im Akkumulator* V *akkumuliert ist. Nach erfolgreicher Verifikation wird* 1 *und andernfalls* 0 *ausgegeben.*

5. *Für alle öffentlichen Schlüssel* $\mathsf{pk} \leftarrow \mathsf{GenAcc}\left(1^\lambda, K\right)$*, alle Wertemengen* $X = \{x_1, \ldots, x_k\} \subset \mathbb{X}$ *mit* $k \leq K$ *und alle Werte* $x_i \in X$ *gilt:*

$$\mathsf{VerifyAcc}\left(\mathsf{pk}, \mathsf{Acc}\left(\mathsf{pk}, X\right), x_i, \mathsf{Wit}\left(\mathsf{pk}, X, x_i\right)\right) = 1.$$

Ein Akkumulator-Schema $(\mathsf{GenAcc}, \mathsf{Acc}, \mathsf{Wit}, \mathsf{VerifyAcc})$ *mit Sicherheitsparameter* $\lambda \in \mathbb{N}$ *und oberer Schranke* $K \geq 2$ *heißt* quasi kommutativ*, falls für alle öffentlichen Schlüssel* $\mathsf{pk} \leftarrow \mathsf{GenAcc}\left(1^\lambda, K\right)$*, alle Werte* $x_1, x_2 \in \mathbb{X}$ *und den Akkumulierungsalgorithmus* Acc *gilt:*

$$\mathsf{Acc}\left(\mathsf{pk}, \{x_1, x_2\}\right) = \mathsf{Acc}\left(\mathsf{pk}, \{x_2, x_1\}\right).$$

Im Folgenden bezeichnet ein Akkumulator-Schema $(\mathsf{GenAcc}, \mathsf{Acc}, \mathsf{Wit}, \mathsf{VerifyAcc})$ stets ein quasi kommutatives Akkumulator-Schema.

Wie in den vorherigen Abschnitten wird auch hier im weiteren Verlauf dieser Arbeit an einigen Stellen vereinfachend auf die explizite Nennung der Eingabe des öffentlichen Schlüssels pk verzichtet und entsprechend die Bezeichnungen $\mathsf{Acc}(X), \mathsf{Wit}(X, x_i)$ sowie $\mathsf{VerifyAcc}\left(\mathsf{Acc}(X), x_i, \mathsf{Wit}(X, x_i)\right)$ verwendet. Ist $\mathsf{Acc}(X)$ ein Akkumulator und $x_i \in X$, so werden die Formulierungen „der Wert x_i ist im bzw. befindet sich im Akkumulator V“ gebraucht.

Als Sicherheitseigenschaft von Akkumulatoren ist in [BM94] die Einweg-Eigenschaft definiert. Auf die obige Definition 2.11.1 bezogen, bedeutet dies, dass kein polynomieller Algorithmus bei Eingabe einer Wertemenge X, des Akkumulators $\mathsf{Acc}(X)$ und

eines weiteren Wertes $x^\star \notin X$ einen passenden Zeugen $\hat{W}^\star$ für $x^\star$ berechnen kann, so dass $\mathsf{VerifyAcc}\left(\mathsf{Acc}\left(X\right), x^\star, \hat{W}^\star\right) = 1$ ist. Ebenfalls wird in diesem Artikel informell eine stärkere Sicherheitseigenschaft betrachtet, die als *starke Einweg-Eigenschaft* bezeichnet wird (vgl. [BP97]). Dies bedeutet, dass kein polynomieller Algorithmus bei Eingabe einer Wertemenge X und des Akkumulators $\mathsf{Acc}\left(X\right)$ ein Paar $\left(x^\star, \hat{W}^\star\right)$ mit $x^\star \notin X$ erzeugen kann, so dass $\mathsf{VerifyAcc}\left(\mathsf{Acc}\left(X\right), x^\star, \hat{W}^\star\right) = 1$ gilt. Da der Algorithmus den Wert $x^\star$ wählen kann, ist dies zwar eine stärkere Sicherheitseigenschaft, aber für viele Akkumulator-Schemata immer noch nicht ausreichend. N. Barić und B. Pfitzmann [BP97] fordern eine noch stärke Eigenschaft, die als *kollisionsfrei* bezeichnet wird. Hier wählt der Algorithmus auch die Wertemenge $X \subset \mathbb{X}$. Au *et al.* [ATSM09, Au09] verlangen weiter, dass der Wert $x^\star$ sogar ein Element eines anderen Wertebereichs $\mathbb{X}^\star$ anstelle von $\mathbb{X}$ sein darf. Formal ist die Sicherheit von Akkumulator-Schemata folgendermaßen definiert:

Definition 2.11.2 (**Sicherheit eines Akkumulator-Schemas**)**.** *Ein Akkumulator-Schema* $(\mathsf{GenAcc}, \mathsf{Acc}, \mathsf{Wit}, \mathsf{VerifyAcc})$ *mit Sicherheitsparameter* $\lambda \in \mathbb{N}$ *und einer in* λ *polynomiellen oberen Schranke* K *heißt* kollisionsfrei*, falls es für jeden polynomiellen Algorithmus* $\mathcal{A}$ *eine vernachlässigbare Funktion* ν *gibt, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\Pr\Big[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenAcc}\left(1^\lambda, K\right);\ \left(X, x^\star, \hat{W}^\star\right) \leftarrow \mathcal{A}\left(\mathsf{pk}\right)\ :\ X \subset \mathbb{X}^\star\ \wedge\ |X| \leq K\ \wedge\ x^\star \in \mathbb{X}^\star \setminus X\ \wedge\ \mathsf{VerifyAcc}\left(\mathsf{pk}, \mathsf{Acc}\left(\mathsf{pk}, X\right), x^\star, \hat{W}^\star\right) = 1\Big] \leq \nu\left(\lambda\right).$$

Zur Konstruktion elektronischer Geldsysteme benötigt ein kollisionsfreies Akkumulator-Schema aber noch eine weitere Eigenschaft. Um zu verhindern, dass ein Kunde mit mehr Münzen bezahlen kann, als er abgehoben hat, muss das Akkumulator-Schema *beschränkt* sein (vgl. [AWSM07]). Dies bedeutet, dass kein polynomieller Algorithmus mehr Werte als die obere Schranke akkumulieren kann.

Definition 2.11.3 (**Beschränkte Akkumulatoren**)**.** *Ein kollisionsfreies Akkumulator-Schema* $(\mathsf{GenAcc}, \mathsf{Acc}, \mathsf{Wit}, \mathsf{VerifyAcc})$ *mit Sicherheitsparameter* $\lambda \in \mathbb{N}$ *und einer in* λ *polynomiellen oberen Schranke* K *heißt* beschränkt*, falls es für jeden polynomiellen Algorithmus* $\mathcal{A}$ *eine vernachlässigbare Funktion* ν *gibt, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\Pr\Big[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenAcc}\left(1^\lambda, K\right);\ \left(x_1, \hat{W}_1, \ldots, x_{K+1}, \hat{W}_{K+1}, V\right) \leftarrow \mathcal{A}\left(\mathsf{pk}\right)\ :\ x_i \in \mathbb{X}^\star\ \wedge\ \mathsf{VerifyAcc}\left(\mathsf{pk}, V, x_i, \hat{W}_i\right) = 1\ \forall\ i \in \{1, \ldots, K+1\}\Big] \leq \nu\left(\lambda\right).$$

Ist $(\mathsf{GenAcc}, \mathsf{Acc}, \mathsf{Wit}, \mathsf{VerifyAcc})$ *ein beschränktes Akkumulator-Schema, dann heißt der Akkumulator* $\mathsf{Acc}\left(\mathsf{pk}, X\right) = V$ beschränkt*.*

In [AWSM07] wurde der Begriff beschränkter Akkumulator erstmals definiert und festgestellt, dass das von L. Nguyen [Ngu05] entwickelte Akkumulator-Schema diese Eigenschaft besitzt.

2.11.1 Das Nguyen-Akkumulator-Schema

In diesem Unterabschnitt wird der für die teilbaren elektronischen Geldsysteme in Kapitel 6 bis 9 als Grundlage dienende Akkumulator beschrieben, der 2005 von L. Nguyen [Ngu05] entwickelt wurde und dessen Kollisionsfreiheit auf der q-Strong-Diffie-Hellman-Annahme basiert (siehe [Ngu05]). In diesem Artikel wurde der Akkumulator zwar in einem Typ-1- bzw. Typ-2-Pairing beschrieben, aber das Verfahren kann analog in ein Typ-3-Pairing übertragen werden (vgl. [AWSM07, Au09]).

GenAcc führt bei Eingabe des Sicherheitsparameters 1^λ sowie einer oberen Schranke $K \in \mathbb{N}$ den Algorithmus $\mathsf{Setup}_{\mathsf{Bil}}$ aus und erhält die Systemparameter $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, h) \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right)$, wobei $\mathbb{G}_1 = \langle u_0 \rangle, \mathbb{G}_2 = \langle h \rangle$ und $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ eine bilineare Abbildung ist. Dann wählt der Algorithmus zufällig eine Zahl $\alpha \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Elemente $h_1 = h^\alpha$ sowie $u_i = u_0^{\alpha^i}$ für $i = 1, \ldots, K$. Die Ausgabe ist der öffentliche Schlüssel

$$\mathsf{pk} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, \ldots, u_K, h, h_1) \leftarrow \mathsf{GenAcc}\left(1^\lambda, K\right),$$

wobei für die Menge aller Werte $\mathbb{X} = \mathbb{Z}_p \setminus \{-\alpha\}$ gilt.

Acc berechnet bei Eingabe des öffentlichen Schlüssels pk und einer Wertemenge $X = \{x_1, \ldots, x_K\} \subset \mathbb{Z}_p \setminus \{-\alpha\}$ den Akkumulator

$$V = \mathsf{Acc}\left(\mathsf{pk}, X\right) = u_0^{\prod_{j=1}^{K}(\alpha + x_j)} = u_0^{\sum_{j=0}^{K}\phi_j \alpha^j} = \prod_{j=0}^{K} u_j^{\phi_j}.$$

Dabei sind die Zahlen $\phi_0, \ldots, \phi_K \in \mathbb{Z}_p$ die Koeffizienten des Polynoms $\phi(\mathsf{x}) = \prod_{j=1}^{K}(\mathsf{x} + x_j) = \sum_{j=0}^{K}\phi_j \mathsf{x}^j \in \mathbb{Z}_p[\mathsf{x}]$ mit $\deg(\phi) = K$. Daher kann jeder Akkumulator mit den öffentlichen Generatoren $u_0, \ldots, u_K$ berechnet werden, ohne dass die Kenntnis der geheimen Zahl α erforderlich ist.

Wit berechnet bei Eingabe des öffentlichen Schlüssels pk, einer Wertemenge $X = \{x_1, \ldots, x_K\} \subset \mathbb{Z}_p \setminus \{-\alpha\}$ und eines Wertes $x_i \in X$ den Zeugen

$$\hat{W}_i = \mathsf{Wit}\left(\mathsf{pk}, X, x_i\right) = \mathsf{Acc}\left(\mathsf{pk}, X \setminus \{x_i\}\right) = u_0^{\prod_{j=1, j\neq i}^{K}(\alpha + x_j)}.$$

Somit erfüllt jedes korrekte *Werte-Zeugen-Paar* $\left(x_i, \hat{W}_i\right)$ die Gleichung $\hat{W}_i^{\alpha + x_i} = V$ und somit $\hat{e}\left(\hat{W}_i, h_1 h^{x_i}\right) = \hat{e}\left(V, h\right)$.

VerifyAcc verifiziert bei Eingabe des öffentlichen Schlüssels pk, eines Akkumulators $V \in \mathbb{G}_1$, eines Wertes $x_i \in \mathbb{Z}_p \setminus \{-\alpha\}$ und eines Zeugen $\hat{W}_i \in \mathbb{G}_1$, ob der Wert x_i in V akkumuliert ist, indem der Algorithmus die Gleichung

$$\hat{e}\left(\hat{W}_i, h_1 h^{x_i}\right) \stackrel{?}{=} \hat{e}\left(V, h\right)$$

überprüft. Entsprechend ist die Ausgabe $\mathsf{VerifyAcc}\left(\mathsf{pk}, V, x_i, \hat{W}_i\right) = \{1, 0\}$.

Satz 2.11.1. *Das Nguyen-Akkumulator-Schema* $(\mathsf{GenAcc}, \mathsf{Acc}, \mathsf{Wit}, \mathsf{VerifyAcc})$ *ist kollisionsfrei und beschränkt, falls die q-SDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gilt.*

Beweis. Siehe [Ngu05, AWSM07, Au09]. □

Es sei darauf hingewiesen, dass es in der Literatur allerdings keinen formalen Beweis für die Beschränktheit des hier vorgestellten Akkumulator-Schemas gibt, sondern in [AWSM07, Au09] lediglich die „Feststellung“ gemacht wurde, dass das Nguyen-Akkumulator-Schema beschränkt sei. Für die in Kapitel 6 bis 9 entwickelten elektronischen Geldsysteme werden die Beschränktheit und Kollisionsfreiheit des Nguyen-Akkumulator-Schemas allerdings nicht benötigt, da die Beschränktheit und die Gebundenheit des Polynom-Commitment-Schemas für die Sicherheitsbeweise ausreichend sind (vgl. auch Kapitel 3).

Bemerkung 2.11.1. *Das hier vorgestellte Akkumulator-Schema von Nguyen ist* dynamisch. Dynamische Akkumulatoren *wurden 2002 von J. Camenisch und A. Lysyanskaya [CL02a] entwickelt und erlauben ein dynamisches* Hinzufügen *bzw.* Löschen *eines Wertes und die entsprechende Aktualisierung des Akkumulators sowie der Zeugen. Während beim Nguyen-Akkumulator-Schema ein gegebener Zeuge ohne den privaten Schlüssel* α *aktualisiert werden kann, wird dieser allerdings zur Aktualisierung eines gegebenen Akkumulators benötigt:*

Hinzufügen: Gegeben seien eine Wertemenge $X \subset \mathbb{Z}_p \setminus \{-\alpha\}$, der Akkumulator $V = \mathsf{Acc}(X)$ und ein Wert $x' \notin X$. Dann werden der Akkumulator V und der Zeuge $\hat{W}_i$ für einen Wert $x_i \in X$ folgendermaßen aktualisiert:

$$V' = \mathsf{Acc}(X \cup \{x'\}) = V^{\alpha+x'} \text{ und } \hat{W}_i' = V\hat{W}_i^{x'-x_i}.$$

Löschen: Gegeben seien eine Wertemenge $X \subset \mathbb{Z}_p \setminus \{-\alpha\}$, der Akkumulator $V = \mathsf{Acc}(X)$ und ein Wert $x' \in X$. Dann werden der Akkumulator V und der Zeuge $\hat{W}_i$ für einen Wert $x_i \in X \setminus \{x'\}$ folgendermaßen aktualisiert:

$$V' = \mathsf{Acc}(X \setminus \{x'\}) = V^{\frac{1}{\alpha+x'}} \text{ und } \hat{W}_i' = \left(\frac{\hat{W}_i}{V'}\right)^{\frac{1}{x'-x_i}}.$$

Da es zur Gewährleistung der Fälschungssicherheit der teilbaren elektronischen Geldsysteme in Kapitel 6 bis 9 aber notwendig ist, dass die Kunden und Händler den privaten Schlüssel α nicht kennen (vgl. auch [ASM08]), ist die dynamische Eigenschaft des Akkumulator-Schemas für diese Arbeit irrelevant.

2.11.1.1 Die Erweiterung von Canard und Gouget

S. Canard und A. Gouget [CG10] haben das Akkumulator-Schema weiterentwickelt, so dass ein Zeuge $\hat{W}_I$ für mehrere akkumulierte Werte $x_1, \ldots, x_k \in X$ berechnet werden kann. Dazu bedarf es einiger Veränderungen zum vorgestellten Akkumulator-Schema, die im Folgenden beschrieben sind.

Gen berechnet zusätzlich die Elemente $h_i = h^{\alpha^i}$ für $i = 2, \ldots, K$. Die Ausgabe ist der öffentliche Schlüssel $\mathsf{pk} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, \ldots, u_K, h, h_1, \ldots, h_K)$.

Wit: Sei $I \subseteq \{1, \ldots, K\}$ eine Teilmenge mit $k \leq K$ Elementen und $X_I := \{x_i \mid i \in I\}$ eine Wertemenge. Der Zeugenberechnungsalgorithmus Wit erhält als Eingabe neben dem öffentlichen Schlüssel pk und einer Wertemenge $X = \{x_1, \ldots, x_K\} \subset \mathbb{Z}_p \backslash \{-\alpha\}$ eine weitere Wertemenge $X_I \subseteq X$ und berechnet den Zeugen

$$\hat{W}_I = \mathsf{Wit}\left(\mathsf{pk}, X, X_I\right) = \mathsf{Acc}\left(\mathsf{pk}, X \setminus X_I\right) = u_0^{\prod_{j=1, j \notin I}^{K} (\alpha + x_j)}.$$

Da das Akkumulator-Schema quasi kommutativ ist, bezeichne o.B.d.A. I die Menge $\{1, \ldots, k\}$ und X_I die Wertemenge $\{x_1, \ldots, x_k\}$ mit $k \leq K$. Dann ist $\hat{W}_I = u_0^{\prod_{j=k+1}^{K} (\alpha + x_j)}$. Somit erfüllt jedes korrekte *Wertemengen-Zeugen-Paar* $\left(X_I, \hat{W}_I\right)$ die Gleichung $\hat{W}_I^{(\alpha+x_1)\cdots(\alpha+x_k)} = V$.

Verify verifiziert bei Eingabe des öffentlichen Schlüssels pk, eines Akkumulators $V \in \mathbb{G}_1$, einer Wertemenge $X_I \subset \mathbb{Z}_p \setminus \{-\alpha\}$ und eines Zeugen $\hat{W}_I \in \mathbb{G}_1$, ob die Wertemenge X_I in V akkumuliert ist, indem der Algorithmus die Gleichung

$$\hat{e}\left(\hat{W}_I, h^{\prod_{j=1}^{k} (\alpha + x_j)}\right) \stackrel{?}{=} \hat{e}\left(V, h\right)$$

überprüft. Dabei kann das Element $h^{\prod_{j=1}^{k} (\alpha + x_j)}$ analog zu V mit den öffentlichen Generatoren $h, h_1, \ldots, h_K$ berechnet werden.

2.11.1.2 Zero-Knowledge-Beweise

Es existieren ebenfalls auch Zero-Knowledge-Proofs-of-Knowledge für den Nguyen-Akkumulator, so dass der Akkumulator V, der akkumulierte Wert x_i oder der Zeuge $\hat{W}_i$ nicht preisgegeben werden müssen (vgl. auch Unterunterabschnitt 2.10.1.2). Die folgenden Zero-Knowledge-Proofs-of-Knowledge stammen aus [Ngu05], [AWSM07] und [CG10], wobei beim ersten lediglich das Werte-Zeugen-Paar $\left(x_i, \hat{W}_i\right)$ und beim zweiten zusätzlich der Akkumulator V perfekt verborgen ist. Seien $g, g_1 \in_{\mathcal{R}} \mathbb{G}_1$ zufällige Generatoren der Gruppe $\mathbb{G}_1$.

Ein geheimer Wert in einem Akkumulator: Möchte der Prover $\mathcal{P}$ beweisen, dass sich ein geheimer Wert x_i in einem Akkumulator V befindet, wählt $\mathcal{P}$ zufällig zwei Zahlen $a_1, a_2 \in_{\mathcal{R}} \mathbb{Z}_p$, berechnet die Elemente $A_1 = g^{a_1} g_1^{a_2}$ und $A_2 = \hat{W}_i g_1^{a_1}$ und führt mit $\mathcal{V}$ den folgenden Proof-of-Knowledge aus, wobei $\delta_1 = a_1 x_i$ und $\delta_2 = a_2 x_i$ ist:

$$PoK\Big[(a_1, a_2, \delta_1, \delta_2, x_i) : A_1 = g^{a_1} g_1^{a_2} \ \wedge \ 1 = A_1^{x_i} g^{-\delta_1} g_1^{-\delta_2} \ \wedge \\ \hat{e}\,(V, h)\,\hat{e}\,(A_2, h_1)^{-1} = \hat{e}\,(A_2, h)^{x_i}\,\hat{e}\,(g_1, h)^{-\delta_1}\,\hat{e}\,(g_1, h_1)^{-a_1}\Big].$$

Dieser Proof-of-Knowledge bildet die Grundlage für das Bezahlprotokoll des kompakten elektronischen Geldsystems in Abschnitt 5.1 und entspricht dem Proof-of-Knowledge über ein SDH-Paar aus Unterunterabschnitt 2.10.1.2.

Ein geheimer Wert in einem geheimen Akkumulator: Möchte der Prover $\mathcal{P}$ beweisen, dass sich ein geheimer Wert x_i in einem geheimen Akkumulator V befindet, wählt $\mathcal{P}$ zufällig drei Zahlen $a_1, a_2, a_3 \in_{\mathcal{R}} \mathbb{Z}_p$, berechnet die Elemente $A_1 = g^{a_1} g_1^{a_2}$, $A_2 = \hat{W}_i g_1^{a_1}$ und $A_3 = V g^{a_3}$ und führt mit $\mathcal{V}$ den folgenden Proof-of-Knowledge aus, wobei $\delta_1 = a_1 x_i$ und $\delta_2 = a_2 x_i$ ist:

$$PoK\Big[(a_1, a_2, a_3, \delta_1, \delta_2, x_i) : A_1 = g^{a_1} g_1^{a_2} \ \wedge \ 1 = A_1^{x_i} g^{-\delta_1} g_1^{-\delta_2} \ \wedge \\ \hat{e}\,(A_3, h)\,\hat{e}\,(A_2, h_1)^{-1} = \hat{e}\,(A_2, h)^{x_i}\,\hat{e}\,(g, h)^{a_3}\,\hat{e}\,(g_1, h)^{-\delta_1}\,\hat{e}\,(g_1, h_1)^{-a_1}\Big].$$

Dieser Proof-of-Knowledge bildet die Grundlage für das Signaturbeweisprotokoll Prove des BBS+A-Signaturverfahrens in Abschnitt 3.2 sowie für das Bezahlprotokoll des kompakten elektronischen Geldsystems in Abschnitt 5.2.

Mehrere Werte in einem geheimen Akkumulator: Möchte der Prover $\mathcal{P}$ beweisen, dass sich die Werte $x_1, \ldots, x_k$ für $I = \{1, \ldots, k\}$ und $k \leq K$ in einem geheimen Akkumulator V befinden, wählt $\mathcal{P}$ zufällig zwei Zahlen $a_1, a_3 \in_{\mathcal{R}} \mathbb{Z}_p$, berechnet die Elemente $A_2 = \hat{W}_I g_1^{a_1}$ und $A_3 = V g^{a_3}$ und führt mit $\mathcal{V}$ den folgenden Proof-of-Knowledge aus, wobei $h_I = h^{\prod_{j=1}^{k} (\alpha + x_j)}$ ist:

$$PoK\Big[(a_1, a_3) : \hat{e}\,(A_3, h)\,\hat{e}\,(A_2, h_I)^{-1} = \hat{e}\,(g, h)^{a_3}\,\hat{e}\,(g_1, h_I)^{-a_1}\Big].$$

Dieser Proof-of-Knowledge bildet die Grundlage für das Signaturbeweisprotokoll Prove-k des BBS+A-Signaturverfahrens in Abschnitt 3.2 sowie für die Bezahlprotokolle der teilbaren elektronischen Geldsysteme in Kapitel 6 bis 9.

2.12 Signaturverfahren mit effizienten Protokollen

Wie bereits erwähnt, sind digitale Signaturverfahren *der* kryptografische Baustein für elektronische Geldsysteme. Um eine anonyme Abwicklung der Zahlungen der Kunden

zu garantieren, müssen Signaturverfahren allerdings erweitert werden. So setzten die ersten elektronischen Geldsysteme [CFN89, CP94, Bra94, FTY98] die von D. Chaum [Cha83, Cha84] entwickelten *blinden Signaturverfahren* ein. Im Jahr 2005 wurde von J. Camenisch, S. Hohenberger und A. Lysyanskaya [CHL05] jedoch erstmals ein elektronisches Geldsystem entwickelt, welches die Anonymität der Kunden durch ein *Signaturverfahren mit effizienten Protokollen*, dessen Entwicklung ebenfalls auf J. Camenisch und A. Lysyanskaya [CL02b, CL04] zurückgeht, gewährleistet.

Vereinfacht formuliert ist ein Signaturverfahren mit effizienten Protokollen ein digitales Signaturverfahren (vgl. Abschnitt 2.8), das zusätzlich zwei Protokolle Issue und Prove bereitstellt, um (1) eine Signatur auf ein Commitment zu erhalten und um (2) in zero-knowledge zu beweisen, im Besitz einer gültigen Signatur auf ein bestimmtes Commitment zu sein.

Das Protokoll Issue wird interaktiv zwischen einem *User* $\mathcal{U}$ und einem *Signer* $\mathcal{S}$ ausgeführt, damit $\mathcal{U}$ von $\mathcal{S}$ eine Signatur σ auf eine Nachricht $m \in \mathbb{M}_{\mathsf{Sign}}$ erhält, wobei m durch ein Commitment verborgen wird. Anschließend kann der User $\mathcal{U}$ durch das Prove-Protokoll, das ein Zero-Knowledge-Proof-of-Knowledge ist, einem *Verifier* $\mathcal{V}$ beweisen, im Besitz einer gültigen Signatur auf eine Nachricht zu sein. Das Protokoll Prove kann dabei interaktiv oder nichtinteraktiv durchgeführt werden, wobei die in dieser Arbeit beschriebenen Protokolle in der nichtinteraktiven Version vorgestellt werden, da die daraus konstruierten Bezahlprotokolle der elektronischen Geldsysteme ebenfalls nichtinteraktiv ausgeführt werden müssen (vgl. auch Bemerkung 3.2.1 in Unterabschnitt 3.2.2).

Definition 2.12.1 (Digitales Signaturverfahren mit effizienten Protokollen). *Ein Tupel* $(\mathsf{GenSign}, \mathsf{Sign}, \mathsf{VerifySign}, \mathsf{Issue}, \mathsf{Prove})$ *polynomieller Algorithmen bzw. Protokolle heißt* digitales Signaturverfahren mit effizienten Protokollen*, falls* $(\mathsf{GenSign}, \mathsf{Sign}, \mathsf{VerifySign})$ *ein digitales Signaturverfahren ist und die folgenden Eigenschaften erfüllt sind:*

1. *Der* Schlüsselgenerierungsalgorithmus $\mathsf{GenSign}$ *ist ein probabilistischer Algorithmus, der bei Eingabe des Sicherheitsparameters* 1^λ *einen privaten Schlüssel* sk *und einen öffentlichen Schlüssel* pk *erzeugt, durch den die Menge aller Nachrichten* $\mathbb{M}_{\mathsf{Sign}}$ *und ein Commitment-Schema* $(\mathsf{GenCom}, \mathsf{Com}, \mathsf{VerifyCom})$ *mit* $\mathbb{M}_{\mathsf{Com}} = \mathbb{M}_{\mathsf{Sign}}$ *festgelegt sind. Die Ausgabe von* $\mathsf{GenSign}$ *ist das Schlüsselpaar* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenSign}\left(1^\lambda\right)$.

2. *Das* Signaturerstellungsprotokoll Issue *ist ein interaktives Protokoll*

$$\{\sigma, \perp\} \leftarrow \mathcal{U}\left(\mathsf{pk}, C, m, r\right) \leftrightarrow \mathcal{S}\left(\mathsf{pk}, \mathsf{sk}, C\right)$$

zwischen einem User $\mathcal{U}$ *und einem Signer* $\mathcal{S}$*. Die gemeinsame Eingabe enthält den öffentlichen Schlüssel* pk *und ein Commitment* $C = \mathsf{Com}\left(\mathsf{pk}, m; r\right)$*. Die private Eingabe von* $\mathcal{U}$ *enthält eine Nachricht* $m \in \mathbb{M}_{\mathsf{Sign}}$ *sowie einen Zufallswert* r *und die private Eingabe von* $\mathcal{S}$ *ist der private Signaturschlüssel* sk*. Die Ausgabe von* $\mathcal{U}$ *ist entweder eine Signatur* σ *oder* $\perp$*.*

3. *Das* Signaturbeweisprotokoll Prove *ist ein Protokoll*

$$\mathcal{U}(\mathsf{pk}, D, m, r', \sigma) \leftrightarrow \mathcal{V}(\mathsf{pk}, D) \rightarrow \{1, 0\}$$

zwischen einem User $\mathcal{U}$ und einem Verifier $\mathcal{V}$. Die gemeinsame Eingabe enthält den öffentlichen Schlüssel pk *und ein Commitment $D = \mathsf{Com}(\mathsf{pk}, m; r')$. Die private Eingabe von $\mathcal{U}$ enthält eine Nachricht m, einen Zufallswert r' und eine Signatur σ. Nach erfolgreicher Verifikation gibt $\mathcal{V}$ die Zahl 1 und andernfalls die Zahl 0 aus.*

4. *Verhalten sich $\mathcal{U}, \mathcal{S}$ und $\mathcal{V}$ jeweils gemäß Protokoll, dann gilt für jedes Schlüsselpaar $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenSign}\left(1^\lambda\right)$, jede Nachricht $m \in \mathbb{M}_{\mathsf{Sign}}$ und jede Signatur σ mit $\sigma \leftarrow \mathcal{U}(\mathsf{pk}, \mathsf{Com}(\mathsf{pk}, m; r), m, r) \leftrightarrow \mathcal{S}(\mathsf{pk}, \mathsf{sk}, \mathsf{Com}(\mathsf{pk}, m; r))$:*

$$\mathcal{U}(\mathsf{pk}, \mathsf{Com}(\mathsf{pk}, m; r'), m, r', \sigma) \leftrightarrow \mathcal{V}(\mathsf{pk}, \mathsf{Com}(\mathsf{pk}, m; r')) \rightarrow 1.$$

Generell können bei einem Signaturverfahren mit effizienten Protokollen dieselben Angreifer und Erfolgsstufen wie in Abschnitt 2.8 für die Sicherheitsanalyse betrachtet werden. Durch die beiden Protokolle Issue und Prove muss die Definition der Sicherheit allerdings modifiziert werden. Neben der (starken) Unfälschbarkeit müssen weiter die beiden Protokolle Issue und Prove jeweils aus Sicht aller Parteien sicher sein. Vereinfacht bedeutet dies, dass (1) während des Protokolls Issue keine Informationen über die Nachricht (*User-Privacy*) und (2) während des Protokolls Prove keine Informationen über das Nachrichten-Signatur-Paar preisgegeben werden (*Zero-Knowledge*). Weiter darf ein User $\mathcal{U}$ durch die Ausführung der Protokolle Issue und Prove keinen Vorteil für eine (starke) existentielle Fälschung erhalten. Das heißt, (3) aus Sicht des Signers $\mathcal{S}$ ist es wichtig, dass ein User $\mathcal{U}$ durch die Durchführung des Protokolls Issue nicht mehr Informationen erhält, als durch die Befragung eines Signatur-Orakels $\mathcal{O}_{\mathsf{Sign}}$ (*Signer-Privacy*). Dies wird dadurch modelliert, dass selbst ein betrügerischer User $\mathcal{U}^*$ nicht unterscheiden kann, ob er das Protokoll Issue mit dem echten Signer $\mathcal{S}$ oder einem Simulator $\mathcal{SIM}$ durchführt, der den privaten Signaturschlüssel sk nicht kennt, aber Zugriff auf ein Signatur-Orakel $\mathcal{O}_{\mathsf{Sign}}$ hat. Schließlich (4) kann jeder User $\mathcal{U}$ nur dann einen Verifier $\mathcal{V}$ im Beweisprotokoll Prove überzeugen, wenn $\mathcal{U}$ das Nachrichten-Signatur-Paar kennt (*Proof-of-Knowledge*).

Definition 2.12.2 (Sicherheit eines digitalen Signaturverfahrens mit effizienten Protokollen). *Ein digitales Signaturverfahren mit effizienten Protokollen* (GenSign, Sign, VerifySign, Issue, Prove) *mit Sicherheitsparameter $\lambda \in \mathbb{N}$ heißt* sicher*, falls das digitale Signaturverfahren* (GenSign, Sign, VerifySign) *(stark) existentiell unfälschbar unter einem adaptiven Angriff mit gewählten Nachrichten ist, das Signaturbeweisprotokoll* Prove *ein Zero-Knowledge-Proof-of-Knowledge-Protokoll ist und für das Signaturerstellungsprotokoll* Issue *die folgenden Eigenschaften erfüllt sind:*

Signer-Privacy: *Für jeden polynomiellen User $\mathcal{U}^*$ gibt es einen probabilistischen voraussichtlich-polynomiellen Algorithmus $\mathcal{SIM}$ und eine vernachlässigbare Funktion*

ν, so dass für alle Nachrichten $m \in \mathbb{M}_{\mathsf{Sign}}$ und alle hinreichend großen $\lambda \in \mathbb{N}$ gilt:

$$\left| \begin{array}{r} \Pr\left[(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{GenSign}\left(1^\lambda\right);\ 1 \leftarrow \mathcal{U}^*\left(\mathsf{pk}, C, m, r\right) \leftrightarrow \mathcal{S}\left(\mathsf{pk},\mathsf{sk}, C\right)\right] \\ -\Pr\left[(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{GenSign}\left(1^\lambda\right);\ 1 \leftarrow \mathcal{U}^*\left(\mathsf{pk}, C, m, r\right) \leftrightarrow \mathcal{SIM}^{\mathcal{O}_{\mathsf{Sign}}}\left(\mathsf{pk}, C\right)\right] \end{array} \right| \leq \nu\left(\lambda\right),$$

wobei $\mathcal{SIM}$ Zugriff auf ein Signatur-Orakel $\mathcal{O}_{\mathsf{Sign}}$ hat, das auf jede Nachricht $m \in \mathbb{M}_{\mathsf{Sign}}$ mit einer gültigen Signatur $\sigma \leftarrow \mathsf{Sign}\left(\mathsf{pk},\mathsf{sk},m\right)$ antwortet.

User-Privacy: *Für jeden polynomiellen Signer $\mathcal{S}^*$ gibt es einen probabilistischen voraussichtlich-polynomiellen Algorithmus $\mathcal{SIM}$ und eine vernachlässigbare Funktion μ, so dass für alle Nachrichten $m \in \mathbb{M}_{\mathsf{Sign}}$ und alle hinreichend großen $\lambda \in \mathbb{N}$ gilt:*

$$\left| \begin{array}{l} \Pr\left[(\mathsf{pk},\mathsf{sk}) \leftarrow \mathcal{S}^*\left(1^\lambda\right);\ \mathcal{U}\left(\mathsf{pk}, C, m, r\right) \leftrightarrow \mathcal{S}^*\left(\mathsf{pk},\mathsf{sk}, C\right) \rightarrow 1\right] \\ -\Pr\left[(\mathsf{pk},\mathsf{sk}) \leftarrow \mathcal{S}^*\left(1^\lambda\right);\ \mathcal{SIM}\left(\mathsf{pk}, C\right) \leftrightarrow \mathcal{S}^*\left(\mathsf{pk},\mathsf{sk}, C\right) \rightarrow 1\right] \end{array} \right| \leq \mu\left(\lambda\right).$$

J. Camenisch und A. Lysyanskaya haben zwar in [CL02a] erstmals digitale Signaturverfahren mit effizienten Protokollen konstruiert, allerdings gibt es in [CL02a] keine Sicherheitsdefinition für das Issue-Protokoll. Im Beweis zu Lemma 7 in [CL02a] über die Sicherheit des Issue-Protokolls wird allerdings genau die oben definierte Signer-Privacy bewiesen (wobei in [CL02a] die Rede von einem Extractor $\mathcal{E}$ statt eines Simulators $\mathcal{SIM}$ ist). Durch die Signer-Privacy wird modelliert, dass im Signaturerstellungsprotokoll Issue keine Informationen üben den geheimen Signaturschlüssel sk preisgegeben werden. Denn da der polynomielle Simulator $\mathcal{SIM}$ die Rolle des Signers $\mathcal{S}$ simulieren kann, erhält selbst ein betrügerischer User $\mathcal{U}^*$ nur die Informationen vom Signer $\mathcal{S}$, die $\mathcal{U}^*$ auch vom Simulator $\mathcal{SIM}$ erhält. Weil der Simulator $\mathcal{SIM}$ den privaten Schlüssel sk nicht kennt, kann also auch kein betrügerischer User $\mathcal{U}^*$ Informationen über sk von $\mathcal{SIM}$ erhalten. Letztendlich erlangt somit kein User weitere Informationen durch die Durchführung des Issue-Protokolls, als durch die Befragung des Signatur-Orakels $\mathcal{O}_{\mathsf{Sign}}$.

Da der in der Signer-Privacy definierte Simulator $\mathcal{SIM}$ keine *Trapdoor-Informationen* besitzt, ist die Sicherheitsdefinition 2.12.2 stärker als die in [Au09, Definition 3.1] definierte Sicherheit, bei der der Simulator den öffentlichen Schlüssel selbst erstellt bzw. simuliert (vgl. Unterabschnitt 3.2.3 in [Au09]).

2.12.1 Das erweiterte Signaturverfahren von Boneh *et al.* (BBS+)

In diesem Unterabschnitt wird eine Erweiterung des Signaturverfahrens von D. Boneh, X. Boyen und H. Shacham [BBS04] (siehe Unterabschnitt 2.8.2) zu einem digitalen Signaturverfahren (BBS+) mit effizienten Protokollen vorgestellt, so dass ein Nachrichtenblock $(m_1, \ldots, m_L) \in \mathbb{Z}_p^L$ signiert werden kann. Die wesentlichen Bausteine dieser Erweiterung sind das-Pedersen-Commitment-Schema (vgl. Unterabschnitt 2.9.1) und Zero-Knowledge-Proofs-of-Knowledge (vgl. Unterabschnitt 2.10.1).

Die Idee der BBS+-Signatur stammt von J. Camenisch und A. Lysyanskaya [CL04], die kurz auf die Konstruktionen des Signaturverfahrens eingegangen sind und einen Vergleich zu deren in [CL04] entwickelten Signaturverfahren mit effizienten Protokollen durchgeführt haben. Grundsätzlich kann die BBS+-Signatur als SDH-Variante der CL-Signatur [CL04] betrachtet werden, weshalb L. Nguyen [Ngu06] diese in seinem Coupon-System mit CL-SDH-Signatur bezeichnet. Dort wurde das BBS+-Signaturverfahren verwendet, wobei lediglich das Signieren einer Nachricht $m \in \mathbb{Z}_p$ benötigt wurde. Eine ausführliche Konstruktion der BBS+-Signatur inklusive Sicherheitsanalyse folgte von M. Au, W. Susilo und Y. Mu [ASM06]. Die Autoren haben gezeigt, dass ein polynomieller Angreifer $\mathcal{A}$, der die BBS+-Signatur nach maximal q Orakel-Anfragen mit einer Wahrscheinlichkeit ϵ stark existentiell fälscht, von einem polynomiellen Angreifer $\mathcal{B}$ dazu verwendet werden kann, das q-SDH-Problem mit einer Wahrscheinlichkeit von mindestens ϵ/q zu lösen. S. Schäge [Sch11] konnte sogar beweisen, dass die Reduktion *tight* ist, d.h. der Angreifer $\mathcal{B}$ kann $\mathcal{A}$ dazu verwenden, das q-SDH-Problem mit fast derselben Wahrscheinlichkeit ϵ zu lösen. Wie der Vorgänger (BBS) arbeitet auch das ursprünglich vorgestellte BBS+-Signaturverfahren auf einem Typ-2-Pairing $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ und die Sicherheit basiert ebenfalls auf der q-SDH-Annahme. Allerdings ist das Verfahren auch in einem Typ-3-Pairing sicher (vgl. Unterabschnitt 2.6.4 und 3.2.3).

2.12.1.1 Ein Commitment-Schema für das BBS+-Signaturverfahren

Damit im Signaturerstellungsprotokoll Issue eine Nachricht $m = (m_1, \ldots, m_L) \in \mathbb{Z}_p^L$ perfekt verborgen ist, wird das in Unterabschnitt 2.9.1 beschriebene, perfekt verbergende Pedersen-Commitment-Schema verwendet.

GenCom führt bei Eingabe des Sicherheitsparameters 1^λ den Algorithmus $\mathsf{Setup_G}$ aus und erhält die Systemparameter $\mathsf{sp_G} = (p, \mathbb{G}', g) \leftarrow \mathsf{Setup_G}\left(1^\lambda\right)$. Dann wählt der Algorithmus zufällig ein Generatortupel $(g_0, \ldots, g_L) \in_{\mathcal{R}} \mathbb{G}^{L+1}$. Die Ausgabe ist der öffentliche Schlüssel

$$\mathsf{pk} = (p, \mathbb{G}', g_0, \ldots, g_L) \leftarrow \mathsf{GenCom}\left(1^\lambda, L\right),$$

wobei für die Menge aller Nachrichten $\mathbb{M}_{\mathsf{Com}} = \mathbb{Z}_p^L$ und für die Menge aller Zufallswerte $\mathbb{R} = \mathbb{Z}_p$ gilt.

PedCom bindet sich bei Eingabe des öffentlichen Schlüssels pk an eine Nachricht $m = (m_1, \ldots, m_L) \in \mathbb{Z}_p^L$, indem der Algorithmus zufällig eine Zahl $r \in_{\mathcal{R}} \mathbb{Z}_p$ wählt und das Pedersen-Commitment

$$C = g_0^r g_1^{m_1} \cdots g_L^{m_L} \leftarrow \mathsf{PedCom}\left(\mathsf{pk}, m\right)$$

berechnet und ausgibt.

VerifyCom verifiziert bei Eingabe des öffentlichen Schlüssels pk, einer Nachricht $m = (m_1, \ldots, m_L) \in \mathbb{Z}_p^L$ und einer Zufallszahl $r \in \mathbb{Z}_p$ die korrekte Berechnung des Commitments $C \in \mathbb{G}$, indem der Algorithmus die Gleichung

$$C \stackrel{?}{=} g_0^r g_1^{m_1} \cdots g_L^{m_L}$$

überprüft. Entsprechend ist die Ausgabe $\mathsf{VerifyCom}\,(\mathsf{pk}, C, m, r) = \{1, 0\}$.

2.12.1.2 Das BBS+-Signaturverfahren

Im Folgenden wird das BBS+-Signaturverfahren detailliert beschrieben:

GenSign führt bei Eingabe des Sicherheitsparameters 1^λ und einer Zahl $L \in \mathbb{N}$ den Algorithmus $\mathsf{Setup_{Bil}}$ aus und erhält die Systemparameter $\mathsf{sp_{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, h) \leftarrow \mathsf{Setup_{Bil}}\left(1^\lambda\right)$, wobei $\mathbb{G}_1 = \langle g \rangle, \mathbb{G}_2 = \langle h \rangle$ und $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ eine bilineare Abbildung ist. Dann wählt der Algorithmus zufällig ein Generatortupel $(g_0, \ldots, g_L) \in_{\mathcal{R}} \mathbb{G}_1^{L+1}$ sowie eine Zahl $x \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet das Element $X = h^x$. Die Ausgabe ist das Schlüsselpaar

$$(\mathsf{pk}, \mathsf{sk}) = ((p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_0, \ldots, g_L, h, X), x) \leftarrow \mathsf{GenSign}\left(1^\lambda, L\right),$$

wobei für die Menge aller Nachrichten $\mathbb{M}_{\mathsf{Sign}} = \mathbb{Z}_p^L$ gilt.

Sign signiert bei Eingabe des Schlüsselpaares $(\mathsf{pk}, \mathsf{sk})$ eine Nachricht $m = (m_1, \ldots, m_L) \in \mathbb{Z}_p^L$, indem der Algorithmus zufällig zwei Zahlen $e, s \in_{\mathcal{R}} \mathbb{Z}_p$ wählt und das Element

$$\Sigma = (g g_0^s g_1^{m_1} \cdots g_L^{m_L})^{\frac{1}{x+e}}$$

berechnet. Die Ausgabe ist die Signatur

$$\sigma = (\Sigma, e, s) \leftarrow \mathsf{Sign}\,(\mathsf{pk}, \mathsf{sk}, m)\,.$$

VerifySign verifiziert bei Eingabe des öffentlichen Schlüssels pk eine Signatur $\sigma = (\Sigma, e, s)$ auf eine Nachricht $m = (m_1, \ldots, m_L) \in \mathbb{Z}_p^L$, indem der Algorithmus die Gleichung

$$\hat{e}\,(\Sigma, X h^e) \stackrel{?}{=} \hat{e}\,(g g_0^s g_1^{m_1} \cdots g_L^{m_L}, h)$$

überprüft. Entsprechend ist die Ausgabe $\mathsf{VerifySign}\,(\mathsf{pk}, m, \sigma) = \{1, 0\}$.

Issue: Damit ein User $\mathcal{U}$ eine Signatur σ auf eine Nachricht $m = (m_1, \ldots, m_L) \in \mathbb{Z}_p^L$ bzgl. des öffentlichen Schlüssels pk erhält und der Signer $\mathcal{S}$ keinerlei Informationen über die Nachricht erlangt, berechnet $\mathcal{U}$ das perfekt verbergende Pedersen-Commitment

$$C = g_0^{s'} g_1^{m_1} \cdots g_L^{m_L} \leftarrow \mathsf{PedCom}\,(\mathsf{pk}, m)$$

für eine Zufallszahl $s' \in_{\mathcal{R}} \mathbb{Z}_p$. Anschließend führen $\mathcal{U}$ und $\mathcal{S}$ das interaktive Signaturerstellungsprotokoll Issue durch, welches in Abbildung 2.4 dargestellt ist.

Schritt 1: Der User $\mathcal{U}$ und der Signer $\mathcal{S}$ führen den folgenden Proof-of-Knowledge

$$PoK\left[(s', m_1, \ldots, m_L) : C = g_0^{s'} g_1^{m_1} \cdots g_L^{m_L}\right]$$

durch, wodurch $\mathcal{U}$ beweist, dass er eine Darstellung von C bzgl. des Generatortupels $(g_0, \ldots, g_L)$ kennt. Dieser Proof-of-Knowledge kann dabei entweder interaktiv als 4-Schritte-Zero-Knowledge-Proof-of-Knowledge oder nichtinteraktiv im Random-Oracle-Modell als Signature-of-Knowledge ausgeführt werden (vgl. Abschnitt 2.10).

Schritt 2: Der Signer $\mathcal{S}$ verifiziert PoK, wählt zufällig zwei Zahlen $e, s'' \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Element

$$\Sigma = \left(g g_0^{s''} C\right)^{\frac{1}{x+e}} = (g g_0^{s} g_1^{m_1} \cdots g_L^{m_L})^{\frac{1}{x+e}}$$

für $s = s' + s'' \mod p$. Dann sendet $\mathcal{S}$ das Tripel (Σ, e, s'') an $\mathcal{U}$.

Schritt 3: Der User $\mathcal{U}$ berechnet $s = s' + s'' \mod p$, verifiziert die Signatur durch Überprüfung der Gleichung $\hat{e}(\Sigma, X h^e) \stackrel{?}{=} \hat{e}\left(g g_0^{s''} C, h\right)$ und speichert die Signatur $\sigma = (\Sigma, e, s)$ ab.

$\boxed{\mathcal{U}ser}$	(pk, C)	$\boxed{\mathcal{S}igner}$
$(m_1, \ldots, m_L, s')$		(sk)
	$PoK\left[(s', m_1, \ldots, m_L) : C = g_0^{s'} g_1^{m_1} \cdots g_L^{m_L}\right]$	
		Verifiziere PoK
		$e, s'' \in_{\mathcal{R}} \mathbb{Z}_p$
		$\Sigma = \left(g g_0^{s''} C\right)^{\frac{1}{x+e}}$
	Σ, e, s'' $\longleftarrow$	
$s = s' + s'' \mod p$		
$\hat{e}(\Sigma, X h^e) \stackrel{?}{=} \hat{e}\left(g g_0^{s''} C, h\right)$		
$\sigma = (\Sigma, e, s)$ oder $\perp$		

Abbildung 2.4: Signaturerstellungsprotokoll Issue der BBS+-Signatur

Prove: Damit ein User $\mathcal{U}$ einem Verifier $\mathcal{V}$ beweisen kann, im Besitz einer gültigen Signatur $\sigma = (\Sigma, e, s)$ bzgl. pk auf eine Nachricht $m = (m_1, \ldots, m_L) \in \mathbb{Z}_p^L$ zu sein und der Verifier $\mathcal{V}$ keinerlei Informationen über die Nachricht und die Signatur erhält, führen $\mathcal{U}$ und $\mathcal{V}$ das nichtinteraktive Signaturbeweisprotokoll Prove durch, welches ein Zero-Knowledge-Proof-of-Knowledge-Protokoll und in Abbildung 2.5 dargestellt ist.

Zur Vereinfachung nehmen wir an, dass der User $\mathcal{U}$ beweisen möchte, dass er für das perfekt verbergende Pedersen-Commitment $D_m = g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L}$ mit $r_m \in_{\mathcal{R}} \mathbb{Z}_p$ eine gültige Signatur $\sigma = (\Sigma, e, s)$ mit $\hat{e}(\Sigma, Xh^e) = \hat{e}(gg_0^s g_1^{m_1} \cdots g_L^{m_L}, h)$ kennt.

Schritt 1: Der User $\mathcal{U}$ wählt zufällig die beiden Zahlen $b_1, b_2 \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Elemente $B_1 = g_0^{b_1} g_1^{b_2}$ sowie $B_2 = \Sigma g_1^{b_1}$ um das Element Σ perfekt zu verbergen. Anschließend sendet $\mathcal{U}$ die Elemente B_1, B_2 an $\mathcal{V}$ und erstellt die Signature-of-Knowledge SoK, wobei $\beta_1 = b_1 e$ und $\beta_2 = b_2 e$ ist:

$$SoK\Bigg[(b_1, b_2, \beta_1, \beta_2, e, s, m_1, \ldots, m_L, r_m) : B_1 = g_0^{b_1} g_1^{b_2} \ \wedge \ 1 = B_1^e g_0^{-\beta_1} g_1^{-\beta_2} \ \wedge$$

$$D_m = g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L} \ \wedge \ \frac{\hat{e}(B_2, X)}{\hat{e}(g, h)} = \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(g_0^s g_1^{m_1+\beta_1} g_2^{m_2} \cdots g_L^{m_L} B_2^{-e}, h\right)\Bigg].$$

Durch SoK beweist $\mathcal{U}$ in zero-knowledge, dass er im Besitz einer gültigen Signatur $\sigma = (\Sigma, e, s)$ auf eine Nachricht $m = (m_1, \ldots, m_L)$ ist.

Schritt 2: Der Verifier $\mathcal{V}$ verifiziert SoK.

$\boxed{\mathcal{U}ser}$	(pk, D_m)	$\boxed{\mathcal{V}erifier}$
$(m_1, \ldots, m_L, r_m, \Sigma, e, s)$		
$b_1, b_2 \in_{\mathcal{R}} \mathbb{Z}_p$ $B_1 = g_0^{b_1} g_1^{b_2}$ $B_2 = \Sigma g_1^{b_1}$		
	$\xrightarrow{\qquad B_1, B_2 \qquad}$	
		$SoK\Big[(b_1, b_2, \beta_1, \beta_2, e, s, m_1, \ldots, m_L, r_m) : B_1 = g_0^{b_1} g_1^{b_2} \ \wedge \ 1 = B_1^e g_0^{-\beta_1} g_1^{-\beta_2} \ \wedge$ $D_m = g^{r_m} g_1^{m_1} \cdots g_L^{m_L} \ \wedge \ \frac{\hat{e}(B_2, X)}{\hat{e}(g, h)} = \hat{e}\left(g_1^{b_1}, X\right)\left(g_0^s g_1^{m_1+\beta_1} g_2^{m_2} \cdots g_L^{m_L} B_2^{-e}, h\right)\Big]$ Verifiziere SoK
		1 oder 0

Abbildung 2.5: Signaturbeweisprotokoll Prove der BBS+-Signatur

Da das BBS+-Signaturverfahren (GenSign, Sign, VerifySign) stark existentiell unfälschbar unter einem adaptiven Angriff mit gewählten Nachrichten ist, die Nachricht $m = (m_1, \ldots, m_L) \in \mathbb{Z}_p^L$ im Signaturerstellungsprotokoll Issue durch die Zufallszahl $s' \in_{\mathcal{R}} \mathbb{Z}_p$ perfekt verborgen ist und das Signaturbeweisprotokoll Prove ein Zero-Knowledge-Proof-of-Knowledge-Protokoll ist, gilt folgender Satz:

Satz 2.12.1. *Das BBS+-Signaturverfahren mit effizienten Protokollen* (GenSign, Sign, VerifySign, Issue, Prove) *ist sicher, falls die q-SDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gilt.*

Beweis. Siehe [ASM06, Au09, Sch11] und Abschnitt 3.2. □

2.12.2 Das erweiterte spezielle Signaturverfahren von Au *et al.* (ESS+)

Für ihr kompaktes elektronisches Geldsystem mit Akkumulatoren haben M. Au, Q. Wu, W. Susilo und Y. Mu [AWSM07] ein spezielles Signaturverfahren entwickelt, welches eine Signatur auf ein beliebiges Gruppenelement $M \in \mathbb{G}_1$ ermöglicht und dessen Sicherheit auf der SXDH-Annahme basiert, wodurch es nur in einem Typ-3-Pairing sicher ist (vgl. Unterabschnitt 2.5.1 und 2.6.3). Dieses Signaturverfahren wurde im selben Artikel zum *ESS*-Signaturverfahren erweitert, um ein Gruppenelement $M \in \mathbb{G}_1$ und einen Nachrichtenblock $(m_1, \ldots, m_L) \in \mathbb{Z}_p^L$ zu signieren. Weiter wurden auch Protokolle zur Signaturerstellung und zum Signaturbeweis entwickelt. Im Signaturerstellungsprotokoll wurde allerdings die Nachricht M (dies war im elektronischen Geldsystem in [AWSM07] ein Akkumulator) im Klartext an den Signer $\mathcal{S}$ gesendet und nur der Nachrichtenblock $(m_1, \ldots, m_L)$ durch das Pedersen-Commitment-Schema perfekt verborgen. Im kompakten elektronischen Geldsystem [AWSM07] wurde somit der Akkumulator V während des Abhebeprotokolls an die Bank $\mathcal{B}$ gesendet. Um die Anonymität der Kunden auch in einem teilbaren elektronischen Geldsystem zu gewährleisten, haben M. Au, W. Susilo und Y. Mu [ASM08] ihr entwickeltes ESS-Signaturverfahren zur *ESS+*-Signatur erweitert, indem ein perfekt verbergendes Commitment-Schema für eine Nachricht $m = (M, m_1, \ldots, m_L) \in \mathbb{G} \times \mathbb{Z}_p^L$ entwickelt wurde. Dieses wird nun vorgestellt.

2.12.2.1 Ein Commitment-Schema für das ESS+-Signaturverfahren

Damit im Signaturerstellungsprotokoll Issue eine Nachricht $m = (M, m_1, \ldots, m_L) \in \mathbb{G}_1 \times \mathbb{Z}_p^L$ perfekt verborgen ist, wird das folgende, perfekt verbergende Commitment-Schema verwendet.

GenCom führt bei Eingabe des Sicherheitsparameters 1^λ den Algorithmus $\mathsf{Setup_G}$ aus und erhält die Systemparameter $\mathsf{sp_G} = (p, \mathbb{G}', g) \leftarrow \mathsf{Setup_G}\left(1^\lambda\right)$. Dann wählt der Algorithmus zufällig ein Generatortupel $(g_a, g_b, g_0, \ldots, g_L) \in_{\mathcal{R}} \mathbb{G}^{L+3}$. Die Ausgabe ist der öffentliche Schlüssel

$$\mathsf{pk} = (p, \mathbb{G}', g_a, g_b, g_0, \ldots, g_L) \leftarrow \mathsf{GenCom}\left(1^\lambda, L\right),$$

wobei für die Menge aller Nachrichten $\mathbb{M}_{\mathsf{Com}} = \mathbb{G} \times \mathbb{Z}_p^L$ und für die Menge aller Zufallswerte $\mathbb{R} = \mathbb{Z}_p^2$ gilt.

Com bindet sich bei Eingabe des öffentlichen Schlüssels pk an eine Nachricht $m = (M, m_1, \ldots, m_L) \in \mathbb{G} \times \mathbb{Z}_p^L$, indem der Algorithmus zufällig zwei Zahlen $r_1, r_2 \in_{\mathcal{R}} \mathbb{Z}_p$ wählt und das Commitment

$$C = (C_M, C_m) = (M g_a^{r_1}, g_b^{r_1} g_0^{r_2} g_1^{m_1} \cdots g_L^{m_L}) \leftarrow \mathsf{Com}\,(\mathsf{pk}, m)$$

berechnet und ausgibt.

VerifyCom verifiziert bei Eingabe des öffentlichen Schlüssels pk, einer Nachricht $m = (M, m_1, \ldots, m_L) \in \mathbb{G} \times \mathbb{Z}_p^L$ und eines Zufallswertes $r = (r_1, r_2) \in \mathbb{Z}_p^2$ die korrekte Berechnung des Commitments $C \in \mathbb{G}^2$, indem der Algorithmus die Gleichungen

$$C_M \stackrel{?}{=} M g_a^{r_1} \text{ und } C_m \stackrel{?}{=} g_b^{r_1} g_0^{r_2} g_1^{m_1} \cdots g_L^{m_L}$$

überprüft. Entsprechend ist die Ausgabe $\mathsf{VerifyCom}(\mathsf{pk}, C, m, r) = \{1, 0\}$.

Durch die Zufallszahl $r_1 \in_{\mathcal{R}} \mathbb{Z}_p$ ist das Element $M \in \mathbb{G}$ und durch die Zufallszahl $r_2 \in_{\mathcal{R}} \mathbb{Z}_p$ der Block $(m_1, \ldots, m_L) \in \mathbb{Z}_p^L$ perfekt verborgen. Es ist zu beachten, dass das Element C_M alleine nicht bindend ist, da C_M für ein beliebiges $r_1' \in \mathbb{Z}_p$ mit $M' = C_1 g_a^{-r_1'}$ geöffnet werden kann. Allerdings ist das Element C_m analog zum Pedersen-Commitment-Schema unter der Diskreter-Logarithmus-Annahme rechnerisch bindend, woraus die rechnerische Gebundenheit des hier vorgestellten Commitment-Schemas folgt (vgl. [Au09]).

2.12.2.2 Das ESS+-Signaturverfahren

Im Folgenden wird das ESS+-Signaturverfahren detailliert beschrieben:

GenSign führt bei Eingabe des Sicherheitsparameters 1^λ und einer Zahl $L \in \mathbb{N}$ den Algorithmus $\mathsf{Setup_{Bil}}$ aus und erhält die Systemparameter $\mathsf{sp_{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, h) \leftarrow \mathsf{Setup_{Bil}}\left(1^\lambda\right)$, wobei $\mathbb{G}_1 = \langle g \rangle, \mathbb{G}_2 = \langle h \rangle$ und $\hat{e} \colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ eine bilineare Abbildung vom Typ-3 ist. Dann wählt der Algorithmus zufällig ein Generatortupel $(g_a, g_b, g_0, \ldots, g_L) \in_{\mathcal{R}} \mathbb{G}_1^{L+3}$, einen Generator $h_1 \in_{\mathcal{R}} \mathbb{G}_2$ sowie zwei Zahlen $x, y \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Elemente $X = h^x, Y = g^y$ sowie $Z = \hat{e}(Y, h)$. Die Ausgabe ist das Schlüsselpaar

$$(\mathsf{pk}, \mathsf{sk}) = ((p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_a, g_b, g_0, \ldots, g_L, h, h_1, X, Z), (x, Y)) \leftarrow \mathsf{GenSign}\left(1^\lambda, L\right),$$

wobei für die Menge aller Nachrichten $\mathbb{M}_{\mathsf{Sign}} = \mathbb{G}_1 \times \mathbb{Z}_p^L$ gilt.

Sign signiert bei Eingabe des Schlüsselpaares $(\mathsf{pk}, \mathsf{sk})$ eine Nachricht $m = (M, m_1, \ldots, m_L) \in \mathbb{G}_1 \times \mathbb{Z}_p^L$, indem der Algorithmus zufällig drei Zahlen $e, s, t \in_{\mathcal{R}} \mathbb{Z}_p$ wählt und die Elemente

$$\Sigma_1 = Y\left(M g_a^t\right)^e, \Sigma_2 = \left(g g_b^t g_0^s g_1^{m_1} \cdots g_L^{m_L}\right)^{\frac{1}{x+e}} \text{ sowie } \Sigma_3 = h^e$$

berechnet. Die Ausgabe ist die Signatur

$$\sigma = (\Sigma_1, \Sigma_2, \Sigma_3, s, t) \leftarrow \mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, m).$$

VerifySign verifiziert bei Eingabe des öffentlichen Schlüssels pk eine Signatur $\sigma = (\Sigma_1, \Sigma_2, \Sigma_3, s, t)$ auf eine Nachricht $m = (M, m_1, \ldots, m_L) \in \mathbb{G}_1 \times \mathbb{Z}_p^L$, indem der Algorithmus die Gleichungen

$$\hat{e}\left(\Sigma_1, h\right) \stackrel{?}{=} Z \cdot \hat{e}\left(Mg_a^t, \Sigma_3\right) \text{ und } \hat{e}\left(\Sigma_2, X\Sigma_3\right) \stackrel{?}{=} \hat{e}\left(gg_b^t g_0^s g_1^{m_1} \cdots g_L^{m_L}, h\right)$$

überprüft. Entsprechend ist die Ausgabe $\mathsf{VerifySign}(\mathsf{pk}, m, \sigma) = \{1, 0\}$.

Issue: Damit ein User $\mathcal{U}$ eine Signatur σ auf eine Nachricht $m = (M, m_1, \ldots, m_L) \in \mathbb{G}_1 \times \mathbb{Z}_p^L$ bzgl. des öffentlichen Schlüssels pk erhält und der Signer $\mathcal{S}$ keinerlei Informationen über die Nachricht erlangt, berechnet $\mathcal{U}$ das perfekt verbergende Commitment

$$(C_M, C_m) = \left(Mg_a^t, g_b^t g_0^{s'} g_1^{m_1} \cdots g_L^{m_L}\right) \leftarrow \mathsf{Com}\left(\mathsf{pk}, m\right)$$

für zwei Zufallszahlen $s', t \in_{\mathcal{R}} \mathbb{Z}_p$. Anschließend führen $\mathcal{U}$ und $\mathcal{S}$ das interaktive Signaturerstellungsprotokoll Issue durch, welches in folgender Abbildung 2.6 dargestellt ist.

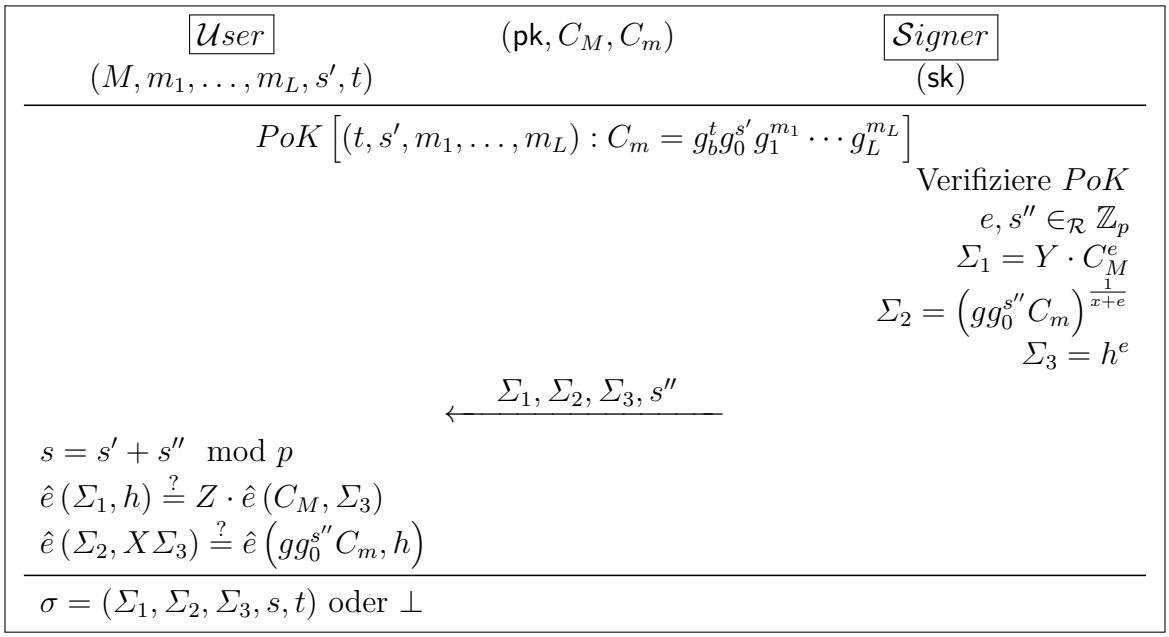

Abbildung 2.6: Signaturerstellungsprotokoll Issue der ESS+-Signatur

Schritt 1: Der User $\mathcal{U}$ und der Signer $\mathcal{S}$ führen den folgenden Proof-of-Knowledge

$$PoK\left[(t, s', m_1, \ldots, m_L) : C_m = g_b^t g_0^{s'} g_1^{m_1} \cdots g_L^{m_L}\right]$$

durch, wodurch $\mathcal{U}$ beweist, dass er eine Darstellung von C_m bzgl. des Generatortupels $(g_b, g_0, \ldots, g_L)$ kennt. Dieser Proof-of-Knowledge kann dabei

entweder interaktiv als 4-Schritte-Zero-Knowledge-Proof-of-Knowledge oder nichtinteraktiv im Random-Oracle-Modell als Signature-of-Knowledge ausgeführt werden (vgl. Abschnitt 2.10).

Schritt 2: Der Signer $\mathcal{S}$ verifiziert PoK, wählt zufällig zwei Zahlen $e, s'' \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Elemente

$$\Sigma_1 = Y \cdot C_M^e = Y\left(Mg_a^t\right)^e,\ \Sigma_2 = \left(gg_0^{s''}C_m\right)^{\frac{1}{x+e}} = \left(gg_b^t g_0^s g_1^{m_1}\cdots g_L^{m_L}\right)^{\frac{1}{x+e}}$$

für $s = s' + s'' \mod p$ sowie

$$\Sigma_3 = h^e.$$

Dann sendet $\mathcal{S}$ das Tupel $(\Sigma_1, \Sigma_2, \Sigma_3, s'')$ an $\mathcal{U}$.

Schritt 3: Der User $\mathcal{U}$ berechnet $s = s' + s'' \mod p$, verifiziert die Signatur durch Überprüfung der Gleichungen $\hat{e}(\Sigma_1, h) \stackrel{?}{=} Z \cdot \hat{e}(C_M, \Sigma_3)$ sowie $\hat{e}(\Sigma_2, X\Sigma_3) \stackrel{?}{=} \hat{e}\left(gg_0^{s''}C_m, h\right)$ und speichert die Signatur $\sigma = (\Sigma_1, \Sigma_2, \Sigma_3, s, t)$ ab.

Prove: Damit ein User $\mathcal{U}$ einem Verifier $\mathcal{V}$ beweisen kann, im Besitz einer gültigen Signatur $\sigma = (\Sigma_1, \Sigma_2, \Sigma_3, s, t)$ bzgl. pk auf eine Nachricht $m = (M, m_1, \ldots, m_L) \in \mathbb{G}_1 \times \mathbb{Z}_p^L$ zu sein und der Verifier $\mathcal{V}$ keinerlei Informationen über die Nachricht und die Signatur erhält, führen $\mathcal{U}$ und $\mathcal{V}$ das nichtinteraktive Signaturbeweisprotokoll Prove durch, welches ein Zero-Knowledge-Proof-of-Knowledge-Protokoll und in Abbildung 2.7 dargestellt ist.

Zur Vereinfachung nehmen wir an, dass der User $\mathcal{U}$ beweisen möchte, dass er für das perfekt verbergende Commitment $(D_M, D_m) = (Mg_a^{r_M}, g_b^{r_M} g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L})$ mit $r_M, r_m \in_{\mathcal{R}} \mathbb{Z}_p$ eine gültige Signatur $\sigma = (\Sigma_1, \Sigma_2, \Sigma_3, s, t)$ mit $\hat{e}(\Sigma_1, h) = Z \cdot \hat{e}(Mg_a^t, \Sigma_3)$ und $\hat{e}(\Sigma_2, X\Sigma_3) = \hat{e}(gg_b^t g_0^s g_1^{m_1} \cdots g_L^{m_L}, h)$ kennt.

Schritt 1: Der User $\mathcal{U}$ wählt zufällig vier Zahlen $b_1, b_2, b_3, b_4 \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Elemente $B_1 = \Sigma_1 g_1^{b_1}, B_2 = \Sigma_2 g_1^{b_2}, B_3 = \Sigma_3 h_1^{b_3}$ sowie $B_4 = g_0^{b_3} g_1^{b_4}$, um die Elemente Σ_1, Σ_2 und Σ_3 perfekt zu verbergen. Anschließend sendet $\mathcal{U}$ die Elemente B_1, B_2, B_3, B_4 an $\mathcal{V}$ und erstellt die Signature-of-Knowledge SoK, wobei $\beta_{i,j} = b_i b_j, \beta_{iM} = b_i r_M$ und $\beta_{it} = b_i t$ für $i, j = 1, \ldots, 4$ ist:

$$\begin{aligned} SoK\Big[&(r_M, r_m, b_1, \ldots, b_4, \beta_{2,3}, \beta_{2,4}, \beta_{3M}, \beta_{4M}, \beta_{3t}, \beta_{4t}, s, t, m_1, \ldots, m_L) : \\ &B_4 = g_0^{b_3} g_1^{b_4} \ \wedge\ 1 = B_4^{b_2} g_0^{-\beta_{2,3}} g_1^{-\beta_{2,4}} \ \wedge\ 1 = B_4^{r_M} g_0^{-\beta_{3M}} g_1^{-\beta_{4M}} \ \wedge \\ &1 = B_4^t g_0^{-\beta_{3t}} g_1^{-\beta_{4t}} \ \wedge\ D_m = g_b^{r_M} g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L} \ \wedge \\ &Z\hat{e}(D_M, B_3)\,\hat{e}(B_1, h)^{-1} = \hat{e}\left(g_1^{-b_1}, h\right)\hat{e}\left(g_a^{\beta_{3t}-\beta_{3M}} D_M^{b_3}, h_1\right)\hat{e}\left(g_a^{r_M - t}, B_3\right) \ \wedge \\ &\hat{e}(B_2, XB_3)\,\hat{e}(g, h)^{-1} = \hat{e}\left(g_b^t g_0^s g_1^{m_1} \cdots g_L^{m_L}, h\right)\hat{e}\left(g_1^{-\beta_{2,3}} B_2^{b_3}, h_1\right)\hat{e}\left(g_1^{b_2}, XB_3\right)\Big]. \end{aligned}$$

Durch SoK beweist $\mathcal{U}$ in zero-knowledge, dass er im Besitz einer gültigen Signatur $\sigma = (\Sigma_1, \Sigma_2, \Sigma_3, s,\ t)$ auf eine Nachricht $m = (M, m_1, \ldots, m_L) \in \mathbb{G}_1 \times \mathbb{Z}_p^L$ ist.

Schritt 2: Der Verifier $\mathcal{V}$ verifiziert SoK.

$\boxed{\mathcal{U}ser}$	(pk, D_M, D_m)	$\boxed{\mathcal{V}erifier}$
$(M, m_1, \ldots, m_L, r_M, r_m,$ $\Sigma_1, \Sigma_2, \Sigma_3, s, t)$		
$b_1, b_2, b_3, b_4 \in_{\mathcal{R}} \mathbb{Z}_p$		
$B_1 = \Sigma_1 g_1^{b_1}$		
$B_2 = \Sigma_2 g_1^{b_2}$		
$B_3 = \Sigma_3 h_1^{b_3}$		
$B_4 = g_0^{b_3} g_1^{b_4}$		
	$\xrightarrow{B_1, B_2, B_3, B_4}$	
$SoK\Big[(r_M, r_m, b_1, \ldots, b_4, \beta_{2,3}, \beta_{2,4}, \beta_{3M}, \beta_{4M}, \beta_{3t}, \beta_{4t}, s, t, m_1, \ldots, m_L) :$		
$B_4 = g_0^{b_3} g_1^{b_4} \ \wedge\ 1 = B_4^{b_2} g_0^{-\beta_{2,3}} g_1^{-\beta_{2,4}} \ \wedge\ 1 = B_4^{r_M} g_0^{-\beta_{3M}} g_1^{-\beta_{4M}} \ \wedge$		
$1 = B_4^t g_0^{-\beta_{3t}} g_1^{-\beta_{4t}} \ \wedge\ D_m = g_b^{r_M} g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L} \ \wedge$		
$Z\hat{e}\,(D_M, B_3)\,\hat{e}\,(B_1, h)^{-1} = \hat{e}\left(g_1^{-b_1}, h\right)\hat{e}\left(g_a^{\beta_{3t}-\beta_{3M}} D_M^{b_3}, h_1\right)\hat{e}\,(g_a^{r_M - t}, B_3) \ \wedge$		
$\hat{e}\,(B_2, XB_3)\,\hat{e}\,(g, h)^{-1} = \hat{e}\,(g_b^t g_0^s g_1^{m_1} \cdots g_L^{m_L}, h)\,\hat{e}\left(g_1^{-\beta_{2,3}} B_2^{b_3}, h_1\right)\hat{e}\left(g_1^{b_2}, XB_3\right)\Big]$		
Verifiziere SoK		
1 oder 0		

Abbildung 2.7: Signaturbeweisprotokoll Prove der ESS+-Signatur

M. Au, W. Susilo und Y. Mu haben in [ASM08] die Sicherheit des ESS+-Signaturverfahrens unter der Annahme bewiesen, dass das ESS-Signaturverfahren [AWSM07] sicher ist. Da die Sicherheit nicht auf eine Standard-Annahme reduziert werden konnte, wurde die ESS-Signatur als *AWSM-Annahme* (siehe [ASM08]) umformuliert und die Sicherheit auf das *AWSM-Problem* reduziert. Da in [AWSM07] bewiesen wurde, dass die ESS-Signatur unter der SXDH-Annahme existentiell unfälschbar unter einem adaptiven Angriff mit gewählten Nachrichten ist und somit auch die AWSM-Annahme gilt, folgt der folgende Satz.

Satz 2.12.2. *Das ESS+-Signaturverfahren mit effizienten Protokollen* (GenSign, Sign, VerifySign, Issue, Prove) *ist sicher, falls die AWSM-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gilt.*

Beweis. Siehe [AWSM07, ASM08, Au09]. □

Um das ESS+-Signaturverfahren mit dem in Kapitel 3 entwickelten Signaturverfahren vergleichen zu können (siehe Tabelle 3.1), ersetzten wir das beliebige Element $M \in$

$\mathbb{G}_1$ durch den Nguyen-Akkumulator $V \in \mathbb{G}_1$ (vgl. Unterabschnitt 2.11.1). Insbesondere war dies auch die Intention von [AWSM07, ASM08], da dort ebenfalls der Nguyen-Akkumulator V signiert wurde.

Dazu wählt der Schlüsselgenerierungsalgorithmus $\mathsf{GenSign}$ zusätzlich, wie beim Akkumulator-Schema, zufällig die Generatoren $u_0 \in_{\mathcal{R}} \mathbb{G}_1, v_0 \in_{\mathcal{R}} \mathbb{G}_2$ sowie die Zahl $\alpha \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Elemente $u_i = u_0^{\alpha^i}$ sowie $v_i = v_0^{\alpha^i}$ für $1 \leq i \leq K$. Die Ausgabe des Schlüsselgenerierungsalgorithmus $\mathsf{GenSign}$ bei zusätzlicher Eingabe der oberen Schranke $K \in \mathbb{N}$ ist das Schlüsselpaar

$$(\mathsf{pk}, \mathsf{sk}) = ((p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_a, g_b, g_0, \ldots, g_L, u_0, \ldots, u_K, h, h_1, v_0, \ldots, v_K, X, Z), \\ (x, Y)) \leftarrow \mathsf{GenSign}\left(1^\lambda, L, K\right).$$

Des Weiteren muss das Signaturbeweisprotokoll Prove modifiziert werden:

Zur Vereinfachung nehmen wir an, dass der User $\mathcal{U}$ beweisen möchte, dass er für die perfekt verbergenden Commitments $(A_3, D_m) = (V g_a^{a_3}, g_b^{r_m} g_1^{m_1} \cdots g_L^{m_L})$ und $D_n = g_0^{n_i} g_1^{r_n}$ mit $a_3, r_m, r_n \in_{\mathcal{R}} \mathbb{Z}_p$ eine gültige Signatur $\sigma = (\Sigma_1, \Sigma_2, \Sigma_3, s, t)$ mit $n_i \in V$ und $\hat{e}(\Sigma_1, h) = Z \cdot \hat{e}(V g_a^t, \Sigma_3)$ sowie $\hat{e}(\Sigma_2, X\Sigma_3) = \hat{e}(g g_b^t g_0^s g_1^{m_1} \cdots g_L^{m_L}, h)$ kennt. Dadurch ist das folgende Proof-of-Knowledge-Protokoll eine Kombination der Beweise für Akkumulator-Schemata und der ESS+-Signatur.

Schritt 1: Der User $\mathcal{U}$ wählt, wie bei der ESS+-Signatur, zufällig vier Zahlen $b_1, b_2, b_3, b_4 \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Elemente $B_1 = \Sigma_1 g_1^{b_1}, B_2 = \Sigma_2 g_1^{b_2}, B_3 = \Sigma_3 h_1^{b_3}$ sowie $B_4 = g_0^{b_3} g_1^{b_4}$. Weiter wählt $\mathcal{U}$, analog zum Akkumulator-Schema, zufällig die beiden Zahlen $a_1, a_2 \in_{\mathcal{R}} \mathbb{Z}_p$, berechnet den Zeugen $\hat{W}_i = u_0^{\prod_{j=1, j\neq i}^{K} (\alpha + n_j)}$ mit $\hat{W}_i^{\alpha + n_i} = V$ und die Elemente $A_1 = g^{a_1} g_1^{a_2}$ sowie $A_2 = \hat{W}_i g_1^{a_1}$. Dadurch sind die Elemente $\Sigma_1, \Sigma_2, \Sigma_3$ und $\hat{W}_i$ perfekt verborgen. Anschließend sendet $\mathcal{U}$ die Elemente $A_1, A_2, B_1, B_2, B_3, B_4$ an $\mathcal{V}$ und erstellt die Signature-of-Knowledge SoK, wobei $\delta_1 = a_1 n_i, \delta_2 = a_2 n_1, \beta_{i,j} = b_i b_j, \beta_{it} = b_i t$ und $\gamma_{i,j} = a_i b_j$ für $i, j = 1, \ldots, 4$ ist:

$$SoK\Big[(r_m, r_n, a_1, a_2, a_3, \delta_1, \delta_2, b_1, \ldots, b_4, \beta_{2,3}, \beta_{2,4}, \beta_{3t}, \beta_{4t}, \gamma_{3,3}, \gamma_{3,4}, s, t, m_1, \ldots, m_L, \\ n_i) : A_1 = g^{a_1} g_1^{a_2} \ \wedge\ 1 = A_1^{n_i} g^{-\delta_1} g_1^{-\delta_2} \ \wedge\ B_4 = g_0^{b_3} g_1^{b_4} \ \wedge\ 1 = B_4^{b_2} g_0^{-\beta_{2,3}} g_1^{-\beta_{2,4}} \ \wedge \\ 1 = B_4^{a_3} g_0^{-\gamma_{3,3}} g_1^{-\gamma_{3,4}} \ \wedge\ 1 = B_4^t g_0^{-\beta_{3t}} g_1^{-\beta_{4t}} \ \wedge\ D_m = g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L} \ \wedge\ D_n = g_0^{n_i} g_1^{r_n} \\ \wedge\ \hat{e}(A_3, v_0)\, \hat{e}(A_2, v_1)^{-1} = \hat{e}\left(g_a^{a_3} g_1^{-\delta_1} A_2^{n_i}, v_0\right) \hat{e}\left(g_1^{-a_1}, v_1\right) \ \wedge \\ Z \hat{e}(A_3, B_3)\, \hat{e}(B_1, h)^{-1} = \hat{e}\left(g_1^{-b_1}, h\right) \hat{e}\left(g_a^{\beta_{3t} - \gamma_{3,3}} A_3^{b_3}, h_1\right) \hat{e}\left(g_a^{a_3 - t}, B_3\right) \ \wedge \\ \hat{e}(B_2, XB_3)\, \hat{e}(g, h)^{-1} = \hat{e}\left(g_b^t g_0^s g_1^{m_1} \cdots g_L^{m_L}, h\right) \hat{e}\left(g_1^{-\beta_{2,3}} B_2^{b_3}, h_1\right) \hat{e}\left(g_1^{b_2}, XB_3\right)\Big].$$

Durch SoK beweist $\mathcal{U}$ in zero-knowledge, dass er im Besitz einer gültigen Signatur $\sigma = (\Sigma_1, \Sigma_2, \Sigma_3, s, t)$ auf eine Nachricht $m = (V, m_1, \ldots, m_L) \in \mathbb{G}_1 \times \mathbb{Z}_p^L$ ist und der Wert n_i in V akkumuliert ist.

Schritt 2: Der Verifier $\mathcal{V}$ verifiziert SoK.

$\boxed{\mathcal{U}ser}$	$(\mathsf{pk}, A_3, D_m, D_n)$	$\boxed{\mathcal{V}erifier}$
$(V, m_1, \ldots, m_L, n_i, a_3,$ $r_m, r_n, \Sigma_1, \Sigma_2, \Sigma_3, s, t)$		
$a_1, a_2, b_1, b_2, b_3, b_4 \in_{\mathcal{R}} \mathbb{Z}_p$		
$\hat{W}_i = u_0^{\prod_{j=1, j\neq i}^{K} (\alpha + n_j)}$		
$A_1 = g^{a_1} g_1^{a_2}$		
$A_2 = \hat{W}_i g_1^{a_1}$		
$B_1 = \Sigma_1 g_1^{b_1}$		
$B_2 = \Sigma_2 g_1^{b_2}$		
$B_3 = \Sigma_3 h_1^{b_3}$		
$B_4 = g_0^{b_3} g_1^{b_4}$		
	$\xrightarrow{A_1, A_2, B_1, \ldots, B_4}$	
$SoK\Big[(r_m, r_n, a_1, a_2, a_3, \delta_1, \delta_2, b_1, \ldots, b_4, \beta_{2,3}, \beta_{2,4}, \beta_{3t}, \beta_{4t}, \gamma_{3,3}, \gamma_{3,4}, s, t, m_1, \ldots, m_L,$ $n_i) : A_1 = g^{a_1} g_1^{a_2} \wedge 1 = A_1^{n_i} g^{-\delta_1} g_1^{-\delta_2} \wedge B_4 = g_0^{b_3} g_1^{b_4} \wedge 1 = B_4^{b_2} g_0^{-\beta_{2,3}} g_1^{-\beta_{2,4}} \wedge$ $1 = B_4^{a_3} g_0^{-\gamma_{3,3}} g_1^{-\gamma_{3,4}} \wedge 1 = B_4^t g_0^{-\beta_{3t}} g_1^{-\beta_{4t}} \wedge D_m = g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L} \wedge D_n = g_0^{n_i} g_1^{r_n} \wedge$ $\hat{e}(A_3, v_0)\, \hat{e}(A_2, v_1)^{-1} = \hat{e}\left(g_a^{a_3} g_1^{-\delta_1} A_2^{n_i}, v_0\right) \hat{e}\left(g_1^{-a_1}, v_1\right) \wedge$ $Z\hat{e}(A_3, B_3)\, \hat{e}(B_1, h)^{-1} = \hat{e}\left(g_1^{-b_1}, h\right) \hat{e}\left(g_a^{\beta_{3t} - \gamma_{3,3}} A_3^{b_3}, h_1\right) \hat{e}(g_a^{a_3 - t}, B_3) \wedge$ $\hat{e}(B_2, XB_3)\, \hat{e}(g, h)^{-1} = \hat{e}(g_b^t g_0^s g_1^{m_1} \cdots g_L^{m_L}, h)\, \hat{e}\left(g_1^{-\beta_{2,3}} B_2^{b_3}, h_1\right) \hat{e}\left(g_1^{b_2}, XB_3\right)\Big]$ Verifiziere SoK		
		1 oder 0

Abbildung 2.8: Signaturbeweisprotokoll Prove der ESS+-Signatur für einen Akkumulator

Analog kann mit dem in Unterunterabschnitt 2.11.1.2 vorgestellten Proof-of-Knowledge bewiesen werden, dass mehrere Werte $n_1, \ldots, n_k$ in V akkumuliert sind, indem anstelle der Gleichung $\hat{e}(A_3, v_0)\, \hat{e}(A_2, v_1)^{-1} = \hat{e}\left(g_a^{a_3} g_1^{-\delta_1} A_2^{n_i}, v_0\right) \hat{e}\left(g_1^{-a_1}, v_1\right)$ die Gleichung $\hat{e}(A_3, v_0)\, \hat{e}(A_2, v_I)^{-1} = \hat{e}(g_a^{a_3}, v_0)\, \hat{e}\left(g_1^{-a_1}, v_I\right)$ bewiesen wird, wobei $A_2 = \hat{W}_I g_1^{a_1}$ mit $\hat{W}_I^{\prod_{j=1}^{k} (\alpha + n_j)} = V$ und $v_I = v_0^{\prod_{j=1}^{k} (\alpha + n_j)}$ ist. Entsprechend werden die Zahlen $r_n, a_2, \delta_1, \delta_2$, $n_i \in \mathbb{Z}_p$ und die Elemente $D_n, A_1 \in \mathbb{G}_1$ nicht benötigt.

KAPITEL 3

EIN NEUES SIGNATURVERFAHREN MIT EFFIZIENTEN PROTOKOLLEN (BBS+A)

Wie bereits im vorherigen Kapitel 2 erwähnt, verwenden aktuelle kompakte und teilbare elektronische Geldsysteme (z.B. [AWSM07, ASM08, CG10]) das Nguyen-Akkumulator-Schema, um mehrere Werte zu akkumulieren, mit denen anschließend die Münzen generiert werden. Für diese elektronischen Geldsysteme wird somit ein digitales Signaturverfahren mit effizienten Protokollen benötigt, so dass die Menge aller Nachrichten $\mathbb{M}_{\mathsf{Sign}}$ jeden Nguyen-Akkumulator $V \in \mathbb{G}_1$ und jeden Nachrichtenblock $(m_1, \ldots, m_L) \in \mathbb{Z}_p^L$ für ein $L \in \mathbb{N}$ beinhaltet. Au *et al.* haben in [AWSM07, ASM08] ein solches Signaturverfahren mit $\mathbb{M}_{\mathsf{Sign}} = \mathbb{G}_1 \times \mathbb{Z}_p^L$ entwickelt, dessen Sicherheit auf der SXDH-Annahme basiert und folglich nur in einem Typ-3-Pairing sicher ist (vgl. Unterabschnitt 2.12.2). Dieses ESS+-Signaturverfahren ist unter der AWSM-Annahme ([ASM08]) existentiell unfälschbar unter einem adaptiven Angriff mit gewählten Nachrichten, wobei AWSM keine Standard-Annahme darstellt. Somit verbleiben Au *et al.* in [AWSM07] mit:

> „[...] it remains an open problem to propose such signature scheme with more standard assumptions."

Weiter ist zu beachten, dass die AWSM-Annahme im Wesentlichen der Annahme entspricht, dass das ESS-Signaturverfahren ([AWSM07]) *stark* existentiell unfälschbar unter einem adaptiven Angriff mit gewählten Nachrichten ist, was jedoch nicht bewiesen wurde. In [AWSM07] wurde lediglich die existentielle Unfälschbarkeit unter einem adaptiven Angriff mit gewählten Nachrichten bewiesen, wobei für den Beweis explizit benötigt wird, dass für ein gefälschtes Nachrichten-Signatur-Paar $(m^\star, \sigma^\star)$ die Bedingung $m^\star \notin \{m_1, \ldots, m_q\}$ gelten muss, wobei $m_1, \ldots, m_q$ die vom Angreifer adaptiv gewählten Nachrichten sind (vgl. auch Abschnitt 2.8).

In diesem Kapitel wird nun ein neues Signaturverfahren mit effizienten Protokollen entwickelt, so dass wie gewünscht die Menge aller Nachrichten $\mathbb{M}_{\mathsf{Sign}}$ jeden Nguyen-Akkumulator $V \in \mathbb{G}_1$ und jeden Nachrichtenblock $(m_1, \ldots, m_L) \in \mathbb{Z}_p^L$ für ein $L \in \mathbb{N}$ beinhaltet, wobei die Sicherheit auf der q-SDH-Annahme basiert. Wie die ESS+-Signatur, stellt auch das neue Signaturverfahren eine Erweiterung der BBS+-Signatur dar und vereint somit Eigenschaften des BBS+-Signaturverfahrens (vgl. Unterabschnitt 2.12.1) mit dem Akkumulator-Schema von Nguyen (vgl. Unterabschnitt 2.11.1), weshalb es im weiteren Verlauf mit *BBS+A* bezeichnet wird. Wie die BBS+-Signatur ist auch die BBS+A-Signatur unter der q-SDH-Annahme in allen Pairing Typen stark existentiell unfälschbar unter einem adaptiven Angriff mit gewählten Nachrichten (siehe Unterabschnitt 3.2.3). Somit stellt die BBS+A-Signatur eine Alternative zur ESS+-Signatur dar, die einerseits in allen Pairing Typen unter einer Standard-Annahme sicher und andererseits effizienter ist. Dadurch bietet das BBS+A-Signaturverfahren im Vergleich zu ESS+ mehr Anwendungsmöglichkeiten in Bezug auf kompakte und teilbare Geldsysteme.

Zur Konstruktion der BBS+A-Signatur werden, im Gegensatz zur ESS+-Signatur, die spezifischen Eigenschaften des Nguyen-Akkumulator-Schemas ausgenutzt. Denn es ist zu beachten, dass die Berechnung des Nguyen-Akkumulators bei einem gegebenen Schlüsselpaar $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenAcc}\left(1^\lambda, K\right)$ bzgl. einer Wertemenge $X = \{x_1, \ldots, x_K\} \subset \mathbb{Z}_p \setminus \{-\alpha\}$ genau der Berechnung des Polynom-Commitments (vgl. Unterabschnitt 2.9.2) für das Polynom $\phi(\mathsf{x}) = \prod_{j=1}^{K}(\mathsf{x} + x_j) \in \mathbb{Z}_p[\mathsf{x}]$ entspricht, wobei GenAcc und GenCom die gleichen Algorithmen sind:

$$\mathsf{Acc}(\mathsf{pk}, X) = u_0^{\prod_{j=1}^{K}(\alpha + x_j)} = u_0^{\sum_{j=0}^{K}\phi_j \alpha^j} = u_0^{\phi(\alpha)} = \mathsf{PolyCom}(\mathsf{pk}, \phi(\mathsf{x})). \tag{3.1}$$

Insbesondere gilt für jede Primzahl p, jede Zahl $\alpha \in \mathbb{Z}_p^*$ und jede obere Schranke $K \in \mathbb{N}$:

$$\left\{ \prod_{j=1}^{k}(\mathsf{x} + x_j) \in \mathbb{Z}_p[\mathsf{x}] \ : \ \{x_1, \ldots, x_k\} \subset \mathbb{Z}_p \setminus \{-\alpha\}, k \leq K \right\} \subset \{\phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}] \ : \ \deg(\phi) \leq K\}.$$

Daher wird das BBS+A-Signaturverfahren so entwickelt, dass ein Nachrichtenblock $(m_1, \ldots, m_L) \in \mathbb{Z}_p^L$ zusammen mit einem Polynom $\phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ mit $\deg(\phi) \leq K$ signiert werden kann, da jedes für das Nguyen-Akkumulator-Schema mögliche Produkt $\prod_{j=1}^{k}(\mathsf{x} + x_j)$ mit $x_j \in \mathbb{X}$ und $k \leq K$ in der Menge aller Nachrichten $\mathbb{M}_{\mathsf{Com}}$ des Polynom-Commitment-Schemas enthalten ist.

Da die Sicherheit der drei verwendeten Bausteine – Polynom-Commitment-Schema, Akkumulator-Schema von Nguyen und BBS+-Signaturverfahren – auf der q-SDH-Annahme basieren, können diese sehr effizient kombiniert werden, so dass das resultierende BBS+A-Signaturverfahren ebenfalls unter der q-SDH-Annahme sicher ist.

Dazu wird im folgenden Abschnitt 3.1 zunächst ein perfekt verbergendes Commitment-Schema für das BBS+A-Signaturverfahren entwickelt, bevor dieses in Abschnitt 3.2 beschrieben wird.

3.1 Ein Commitment-Schema für das BBS+A-Signaturverfahren

Ein perfekt verbergendes Commitment C für eine Nachricht $m = (m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}]$ mit $\deg(\phi) \leq K$ entsteht durch das Produkt des Pedersen-Commitments (siehe Unterabschnitt 2.9.1) und des Polynom-Commitments (siehe Unterabschnitt 2.9.2):

$$C = g_0^r g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)} \leftarrow \mathsf{PedCom}(m_1, \ldots, m_L) \cdot \mathsf{PolyCom}(\phi(\mathsf{x})).$$

Im Folgenden wird die Konstruktion des Commitment-Schemas beschrieben:

GenCom führt bei Eingabe des Sicherheitsparameters 1^λ und zweier Zahlen $L, K \in \mathbb{N}$ den Algorithmus $\mathsf{Setup}_{\mathsf{Bil}}$ aus und erhält die Systemparameter $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, h) \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right)$, wobei $\mathbb{G}_1 = \langle u_0 \rangle, \mathbb{G}_2 = \langle h \rangle$ und $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ eine bilineare Abbildung ist. Dann wählt der Algorithmus zufällig ein Generatortupel $(g_0, \ldots, g_L) \in_{\mathcal{R}} \mathbb{G}_1^{L+1}$ sowie eine Zahl $\alpha \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Elemente $h_1 = h^\alpha$ sowie $u_i = u_0^{\alpha^i}$ für $1 \leq i \leq K$. Die Ausgabe ist der öffentliche Schlüssel

$$\mathsf{pk} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g_0, \ldots, g_L, u_0, \ldots, u_K, h, h_1) \leftarrow \mathsf{GenCom}\left(1^\lambda, L, K\right),$$

wobei für die Menge aller Nachrichten $\mathbb{M}_{\mathsf{Com}} = \left\{m : m = (m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}], \deg(\phi) \leq K\right\}$ und für die Menge aller Zufallswerte und Indizes $\mathbb{R} = \mathbb{I} = \mathbb{Z}_p$ gilt.

Com bindet sich bei Eingabe des öffentlichen Schlüssels pk an eine Nachricht $m = (m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}]$ mit $\deg(\phi) \leq K$, indem der Algorithmus zufällig eine Zahl $r \in_{\mathcal{R}} \mathbb{Z}_p$ wählt und das Commitment

$$C = g_0^r g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)} \leftarrow \mathsf{Com}(\mathsf{pk}, m)$$

berechnet und ausgibt.

VerifyCom verifiziert bei Eingabe des öffentlichen Schlüssels pk, einer Nachricht $m = (m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}]$ mit $\deg(\phi) \leq K$ und einer Zufallszahl $r \in \mathbb{Z}_p$ die korrekte Berechnung des Commitments $C \in \mathbb{G}_1$, indem der Algorithmus die Gleichung

$$C \stackrel{?}{=} g_0^r g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}$$

überprüft. Entsprechend ist die Ausgabe $\mathsf{VerifyCom}(\mathsf{pk}, C, m, r) = \{1, 0\}$.

EvalWit berechnet bei Eingabe des öffentlichen Schlüssels pk, eines Polynoms $\phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ mit $\deg(\phi) \leq K$ und einer Zahl $i \in \mathbb{Z}_p$ den Wert $\phi(i) \in \mathbb{Z}_p$ und den Zeugen $W_i \in \mathbb{G}_1$ für die Berechnung $\phi(i)$, indem der Algorithmus das Polynom

$$\phi_i(\mathsf{x}) = \frac{\phi(\mathsf{x}) - \phi(i)}{\mathsf{x} - i} \in \mathbb{Z}_p[\mathsf{x}] \text{ und den Zeugen } W_i = u_0^{\phi_i(\alpha)}$$

berechnet. Die Ausgabe ist $\mathsf{EvalWit}(\mathsf{pk}, \phi(\mathsf{x}), i) = (i, \phi(i), W_i)$.

VerifyEval verifiziert bei Eingabe des öffentlichen Schlüssels pk, eines Commitments $C \in \mathbb{G}_1$, einiger Zahlen $m_1, \ldots, m_L, r, i \in \mathbb{Z}_p$ und eines Zeugen $W_i \in \mathbb{G}_1$ die korrekte Berechnung $\phi(i) \in \mathbb{Z}_p$, indem der Algorithmus die Gleichung

$$\hat{e}(C, h) \stackrel{?}{=} \hat{e}(g_0^r g_1^{m_1} \cdots g_L^{m_L}, h)\, \hat{e}\left(W_i, h_1 h^{-i}\right) \hat{e}\left(u_0^{\phi(i)}, h\right)$$

überprüft. Entsprechend ist die Ausgabe $\mathsf{VerifyEval}(\mathsf{pk}, C, (m_1, \ldots, m_L), r, (i, \phi(i), W_i)) = \{1, 0\}$.

Die Durchführbarkeit folgt dabei direkt aus der Durchführbarkeit des Polynom-Commitment-Schemas aus Unterabschnitt 2.9.2:

$$\hat{e}(g_0^r g_1^{m_1} \cdots g_L^{m_L}, h)\, \hat{e}\left(W_i, h_1 h^{-i}\right) \hat{e}\left(u_0^{\phi(i)}, h\right) = \hat{e}(g_0^r g_1^{m_1} \cdots g_L^{m_L}, h)\, \hat{e}\left(u_0^{\phi(\alpha)}, h\right) = \hat{e}(C, h).$$

3.1.1 Sicherheitsanalyse des Commitment-Schemas

In diesem Unterabschnitt werden in Lemma 3.1.1 bis 3.1.3 die perfekte Verborgenheit sowie die Gebundenheit gezeigt. Anschließend wird eine weitere, benötigte Sicherheitseigenschaft definiert und diese sowie die Beschränktheit in Lemma 3.1.4 und 3.1.5 bewiesen.

Lemma 3.1.1. *Das Commitment-Schema* ($\mathsf{GenCom}, \mathsf{Com}, \mathsf{VerifyCom}, \mathsf{EvalWit}, \mathsf{VerifyEval}$) *ist perfekt verbergend.*

Beweis. Analog zum Pedersen-Commitment ist dieses ebenfalls durch die Zufallszahl $r \in_{\mathcal{R}} \mathbb{Z}_p$ perfekt verbergend. Denn für jedes Commitment $C \in \mathbb{G}_1$ und jede Nachricht $m = (m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}]$ (auch mit $\deg(\phi) > K$) gibt es genau eine Zahl $r \in \mathbb{Z}_p$ mit $C = g_0^r g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}$. Seien $\delta_0, \ldots, \delta_L \in \mathbb{Z}_p$ mit $u_0 = g_0^{\delta_0}, g_1 = g_0^{\delta_1}, \ldots, g_L = g_0^{\delta_L}$ die diskreten Logarithmen zur Basis g_0 und sei $C = g_0^c$ für eine Zahl $c \in \mathbb{Z}_p$. Dann ist

$$r = c - (\delta_1 m_1 + \cdots + \delta_L m_L + \delta_0 \phi(\alpha)) \mod p$$

eindeutig bestimmt. □

Nun wird gezeigt, dass das Commitment-Schema rechnerisch bindend ist.

Lemma 3.1.2. *Das Commitment-Schema* ($\mathsf{GenCom}, \mathsf{Com}, \mathsf{VerifyCom}, \mathsf{EvalWit}, \mathsf{VerifyEval}$) *ist rechnerisch bindend, falls die q-SDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gilt.*

Beweis. Angenommen, es existiert ein polynomieller Angreifer $\mathcal{A}$, der mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ ein Commitment $C \in \mathbb{G}_1$ und zwei verschiedene Tupel $(r, m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^{L+1} \times \mathbb{Z}_p[\mathsf{x}]$ sowie $(r', m_1', \ldots, m_L', \phi'(\mathsf{x})) \in \mathbb{Z}_p^{L+1} \times \mathbb{Z}_p[\mathsf{x}]$ mit $\deg(\phi), \deg(\phi') \leq K'$ und

$$C = g_0^r g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)} = g_0^{r'} g_1^{m_1'} \cdots g_L^{m_L'} u_0^{\phi'(\alpha)}$$

ausgibt, wobei wir sogar $K' > K$ für eine polynomiell beschränkte Zahl $K' \in \mathbb{N}$ erlauben. Wir müssen die beiden Fälle (1) $(r, m_1, \ldots, m_L) \neq (r', m'_1, \ldots, m'_L)$ und (2) $(r, m_1, \ldots, m_L) = (r', m'_1, \ldots, m'_L)$ unterscheiden. Da das Commitment C das Produkt aus Pedersen- und Polynom-Commitment ist, lässt sich der erste Fall auf die Gebundenheit des Pedersen-Commitments und der zweite Fall auf auf die Gebundenheit des Polynom-Commitments reduzieren.

Fall 1: Wir konstruieren einen polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, die Gebundenheit des Pedersen-Commitments zu brechen, was ein Widerspruch zur DLog-Annahme (vgl. Unterabschnitt 2.9.1) und folglich ein Widerspruch zur q-SDH-Annahme ist.

Schlüsselgenerierung: Der Angreifer $\mathcal{B}$ erhält den öffentlichen Schlüssel $\mathsf{pk}' = (p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \hat{e}, g, g_0, \ldots, g_L, h)$ des Pedersen-Commitment-Schemas für $\mathsf{Setup}_{\mathsf{Bil}}$. Danach definiert $\mathcal{B}$ den Generator $u_0 := g$, wählt zufällig eine Zahl $\alpha \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Elemente $h_1 = h^{\alpha}$ sowie $u_i = u_0^{\alpha^i}$ für $1 \leq i \leq K$. Anschließend sendet $\mathcal{B}$ den perfekt simulierten öffentlichen Schlüssel $\mathsf{pk} = (p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \hat{e}, g_0, \ldots, g_L, u_0, \ldots, u_K, h, h_1)$ an den Angreifer $\mathcal{A}$.

Ausgabe: Der Angreifer $\mathcal{A}$ gibt ein Commitment $C \in \mathbb{G}_1$ und zwei verschiedene Tupel $(r, m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^{L+1} \times \mathbb{Z}_p[\mathsf{x}]$ sowie $(r', m'_1, \ldots, m'_L, \phi'(\mathsf{x})) \in \mathbb{Z}_p^{L+1} \times \mathbb{Z}_p[\mathsf{x}]$ mit

$$(r, m_1, \ldots, m_L) \neq (r', m'_1, \ldots, m'_L) \text{ und} \tag{3.2}$$

$$C = g_0^r g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)} = g_0^{r'} g_1^{m'_1} \cdots g_L^{m'_L} u_0^{\phi'(\alpha)} \tag{3.3}$$

aus. Der Angreifer $\mathcal{B}$ berechnet die Werte $\phi(\alpha) \in \mathbb{Z}_p$ sowie $\phi'(\alpha) \in \mathbb{Z}_p$ und gibt das Commitment $C \in \mathbb{G}_1$ sowie zwei Tupel $(\phi(\alpha), r, m_1, \ldots, m_L) \in \mathbb{Z}_p^{L+2}$ und $(\phi'(\alpha), r', m'_1, \ldots, m'_L) \in \mathbb{Z}_p^{L+2}$ aus, wobei die beiden Tupel nach (3.2) verschieden sind und nach (3.3) gilt: $C = g^{\phi(\alpha)} g_0^r g_1^{m_1} \cdots g_L^{m_L} = g^{\phi'(\alpha)} g_0^{r'} g_1^{m'_1} \cdots g_L^{m'_L}$.

Somit bricht der Angreifer $\mathcal{B}$ die Gebundenheit des Pedersen-Commitment-Schemas mit derselben nicht vernachlässigbaren Wahrscheinlichkeit $\epsilon(\lambda)$.

Fall 2: Wir konstruieren einen polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, die Gebundenheit des Polynom-Commitments zu brechen, was ein Widerspruch zur q-SDH-Annahme ist (vgl. Unterabschnitt 2.9.2). Beachte, dass das Polynom-Commitment-Schema die Gebundenheit ebenfalls dann garantiert, falls $\deg(\phi), \deg(\phi') \leq K'$ für eine polynomiell beschränkte Zahl $K' > K$ ist (vgl. [KZG10b]).

Schlüsselgenerierung: Der Angreifer $\mathcal{B}$ erhält den öffentlichen Schlüssel $\mathsf{pk}' = (p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \hat{e}, u_0, \ldots, u_K, h, h_1)$ des Polynom-Commitment-Schemas. Dann wählt $\mathcal{B}$ zufällig ein Generatortupel $(g_0, \ldots, g_L) \in_{\mathcal{R}} \mathbb{G}_1^{L+1}$ und sendet den perfekt simulierten

öffentlichen Schlüssel $\mathsf{pk} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g_0, \ldots, g_L, u_0, \ldots, u_K, h, h_1)$ an den Angreifer $\mathcal{A}$.

Ausgabe: Der Angreifer $\mathcal{A}$ gibt ein Commitment $C \in \mathbb{G}_1$ und zwei verschiedene Tupel $(r, m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^{L+1} \times \mathbb{Z}_p[\mathsf{x}]$ sowie $(r', m_1', \ldots, m_L', \phi'(\mathsf{x})) \in \mathbb{Z}_p^{L+1} \times \mathbb{Z}_p[\mathsf{x}]$ mit

$$(r, m_1, \ldots, m_L) = (r', m_1', \ldots, m_L') \text{ und} \tag{3.4}$$

$$C = g_0^r g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)} = g_0^{r'} g_1^{m_1'} \cdots g_L^{m_L'} u_0^{\phi'(\alpha)} \tag{3.5}$$

aus. Der Angreifer $\mathcal{B}$ berechnet das Commitment $C^\star = C g_0^{-r} g_1^{-m_1} \cdots g_L^{-m_L} \in \mathbb{G}_1$ und gibt $C^\star$ sowie zwei Tupel $\phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ und $\phi'(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ aus, wobei die beiden Polynome nach (3.4) verschieden sind und mit (3.5) gilt: $C^\star = u_0^{\phi(\alpha)} = u_0^{\phi'(\alpha)}$.

Somit bricht der Angreifer $\mathcal{B}$ die Gebundenheit des Polynom-Commitment-Schemas mit derselben nicht vernachlässigbaren Wahrscheinlichkeit $\epsilon(\lambda)$. □

Nun wird gezeigt, dass das Commitment-Schema berechnungsbindend ist. Übertragen auf dieses Commitment-Schema bedeutet dies, dass jeder polynomielle Angreifer $\mathcal{A}$ nur mit vernachlässigbarer Wahrscheinlichkeit ein Commitment $C \in \mathbb{G}_1$ sowie zwei Tupel $(r, m_1, \ldots, m_L, i, \phi(i), W_i) \in \mathbb{Z}_p^{L+3} \times \mathbb{G}_1$ und $\left(r', m_1', \ldots, m_L', i, \phi(i)', W_i'\right) \in \mathbb{Z}_p^{L+3} \times \mathbb{G}_1$ mit $(r, m_1, \ldots, m_L, \phi(i)) \neq \left(r', m_1', \ldots, m_L', \phi(i)'\right)$ ausgeben kann, so dass $\mathsf{VerifyEval}$ beide Eingaben akzeptiert.

Lemma 3.1.3. *Das Commitment-Schema* ($\mathsf{GenCom}, \mathsf{Com}, \mathsf{VerifyCom}, \mathsf{EvalWit}, \mathsf{VerifyEval}$) *ist berechnungsbindend, falls die q-SDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gilt.*

Beweis. Angenommen, es existiert ein polynomieller Angreifer $\mathcal{A}$, der mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ ein Commitment $C \in \mathbb{G}_1$ und zwei Tupel $(r, m_1, \ldots, m_L, i, \phi(i), W_i) \in \mathbb{Z}_p^{L+3} \times \mathbb{G}_1$ sowie $\left(r', m_1', \ldots, m_L', i, \phi(i)', W_i'\right) \in \mathbb{Z}_p^{L+3} \times \mathbb{G}_1$ mit $(r, m_1, \ldots, m_L, \phi(i)) \neq \left(r', m_1', \ldots, m_L', \phi(i)'\right)$ und

$$\begin{aligned}\hat{e}(C, h) &= \hat{e}(g_0^r g_1^{m_1} \cdots g_L^{m_L}, h)\, \hat{e}\left(W_i, h_1 h^{-i}\right) \hat{e}\left(u_0^{\phi(i)}, h\right) \\ &= \hat{e}\left(g_0^{r'} g_1^{m_1'} \cdots g_L^{m_L'}, h\right) \hat{e}\left(W_i', h_1 h^{-i}\right) \hat{e}\left(u_0^{\phi(i)'}, h\right)\end{aligned}$$

ausgibt. Wir müssen die beiden Fälle (1) $W_i = W_i'$ und (2) $W_i \neq W_i'$ unterscheiden. Erneut lässt sich der erste Fall auf die Gebundenheit des Pedersen-Commitments und der zweite Fall auf auf die Berechnungsgebundenheit des Polynom-Commitments reduzieren.

Fall 1: Wir konstruieren einen polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, die Gebundenheit des Pedersen-Commitments zu brechen, was ein Widerspruch zur DLog-Annahme (vgl. Unterabschnitt 2.9.1) und somit auch zur q-SDH-Annahme ist.

Schlüsselgenerierung: Der Angreifer $\mathcal{B}$ erhält den öffentlichen Schlüssel $\mathsf{pk}' = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_0, \ldots, g_L, h)$ des Pedersen-Commitment-Schemas für $\mathsf{Setup}_{\mathsf{Bil}}$. Danach definiert $\mathcal{B}$ den Generator $u_0 := g$, wählt zufällig eine Zahl $\alpha \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Elemente $h_1 = h^{\alpha}$ sowie $u_j = u_0^{\alpha^j}$ für $1 \leq j \leq K$. Anschließend sendet $\mathcal{B}$ den perfekt simulierten öffentlichen Schlüssel $\mathsf{pk} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g_0, \ldots, g_L, u_0, \ldots, u_K, h, h_1)$ an den Angreifer $\mathcal{A}$.

Ausgabe: Der Angreifer $\mathcal{A}$ gibt ein Commitment $C \in \mathbb{G}_1$ und zwei verschiedene Tupel $(r, m_1, \ldots, m_L, i, \phi(i), W_i) \in \mathbb{Z}_p^{L+3} \times \mathbb{G}_1$ sowie $\left(r', m_1', \ldots, m_L', i, \phi(i)', W_i'\right) \in \mathbb{Z}_p^{L+3} \times \mathbb{G}_1$ mit $(r, m_1, \ldots, m_L, \phi(i)) \neq \left(r', m_1', \ldots, m_L', \phi(i)'\right)$ und

$$W_i = W_i' \text{ sowie} \tag{3.6}$$

$$\begin{aligned} \hat{e}(C, h) &= \hat{e}\left(g_0^r g_1^{m_1} \cdots g_L^{m_L}, h\right) \hat{e}\left(W_i, h_1 h^{-i}\right) \hat{e}\left(u_0^{\phi(i)}, h\right) \\ &= \hat{e}\left(g_0^{r'} g_1^{m_1'} \cdots g_L^{m_L'}, h\right) \hat{e}\left(W_i', h_1 h^{-i}\right) \hat{e}\left(u_0^{\phi(i)'}, h\right) \\ \Leftrightarrow \hat{e}(C, h) &= \hat{e}\left(g_0^r g_1^{m_1} \cdots g_L^{m_L} W_i^{\alpha - i} u_0^{\phi(i)}, h\right) \\ &= \hat{e}\left(g_0^{r'} g_1^{m_1'} \cdots g_L^{m_L'} W_i'^{\alpha - i} u_0^{\phi(i)'}, h\right) \end{aligned} \tag{3.7}$$

aus. Der Angreifer $\mathcal{B}$ berechnet das Commitment $C^{\star} = C \cdot W_i^{i-\alpha}$ und gibt das Commitment $C^{\star} \in \mathbb{G}_1$ sowie zwei Tupel $(r, m_1, \ldots m_L, \phi(i)) \in \mathbb{Z}_p^{L+2}$ und $\left(r', m_1', \ldots m_L', \phi(i)'\right) \in \mathbb{Z}_p^{L+2}$ aus, wobei die beiden Tupel nach Voraussetzung verschieden sind und nach (3.7) gilt: $C^{\star} = g^{\phi(i)} g_0^r g_1^{m_1} \cdots g_L^{m_L} = g^{\phi(i)'} g_0^{r'} g_1^{m_1'} \cdots g_L^{m_L'}$.

Somit bricht der Angreifer $\mathcal{B}$ die Gebundenheit des Pedersen-Commitment-Schemas mit derselben nicht vernachlässigbaren Wahrscheinlichkeit $\epsilon(\lambda)$.

Fall 2: Wir konstruieren einen polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, die Berechnungsgebundenheit des Polynom-Commitments zu brechen, was ein Widerspruch zur q-SDH-Annahme ist (vgl. Unterabschnitt 2.9.2).

Schlüsselgenerierung: Der Angreifer $\mathcal{B}$ erhält den öffentlichen Schlüssel $\mathsf{pk}' = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, \ldots, u_K, h, h_1)$ des Polynom-Commitment-Schemas. Dann wählt $\mathcal{B}$ zufällig $L+1$ paarweise verschiedene Zahlen $\delta_0, \ldots, \delta_L \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Generatoren $g_0 = u_0^{\delta_0}, \ldots, g_L = u_0^{\delta_L}$. Anschließend sendet $\mathcal{B}$ den perfekt simulierten öffentlichen Schlüssel $\mathsf{pk} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g_0, \ldots, g_L, u_0, \ldots, u_K, h, h_1)$ an den Angreifer $\mathcal{A}$.

Ausgabe: Der Angreifer $\mathcal{A}$ gibt ein Commitment $C \in \mathbb{G}_1$ und zwei verschiedene Tupel $(r, m_1, \ldots, m_L, i, \phi(i), W_i) \in \mathbb{Z}_p^{L+3} \times \mathbb{G}_1$ sowie $\left(r', m_1', \ldots, m_L', i, \phi(i)', W_i'\right) \in \mathbb{Z}_p^{L+3} \times \mathbb{G}_1$ mit $(r, m_1, \ldots, m_L, \phi(i)) \neq \left(r', m_1', \ldots, m_L', \phi(i)'\right)$ und

$$W_i \neq W_i' \text{ sowie} \tag{3.8}$$

$$\begin{aligned} \hat{e}(C, h) &= \hat{e}\left(g_0^r g_1^{m_1} \cdots g_L^{m_L}, h\right) \hat{e}\left(W_i, h_1 h^{-i}\right) \hat{e}\left(u_0^{\phi(i)}, h\right) \\ &= \hat{e}\left(g_0^{r'} g_1^{m_1'} \cdots g_L^{m_L'}, h\right) \hat{e}\left(W_i', h_1 h^{-i}\right) \hat{e}\left(u_0^{\phi(i)'}, h\right) \end{aligned} \tag{3.9}$$

aus. Der Angreifer $\mathcal{B}$ berechnet die Werte $\phi(i)^\star = \phi(i) + \delta_0 r + \delta_1 m_1 + \cdots + \delta_L m_L \mod p$ sowie $\phi(i)'^\star = \phi(i)' + \delta_0 r' + \delta_1 m_1' + \cdots + \delta_L m_L' \mod p$. Dann gilt nach (3.9): $\hat{e}(C, h) = \hat{e}\left(W_i, h_1 h^{-i}\right) \hat{e}\left(u_0^{\phi(i)^\star}, h\right) = \hat{e}\left(W_i', h_1 h^{-i}\right) \hat{e}\left(u_0^{\phi(i)'^\star}, h\right)$. Daraus folgt $C = W_i^{\alpha - i} u_0^{\phi(i)^\star} = W_i'^{\alpha - i} u_0^{\phi(i)'^\star}$. Damit resultiert aus (3.8): $\phi(i)^\star \neq \phi(i)'^\star$. Der Angreifer $\mathcal{B}$ gibt das Commitment $C \in \mathbb{G}_1$ sowie zwei verschiedene Tripel $(i, \phi(i)^\star, W_i) \in \mathbb{Z}_p^2 \times \mathbb{G}_1$ und $\left(i, \phi(i)'^\star, W_i'\right) \in \mathbb{Z}_p^2 \times \mathbb{G}_1$ mit $\phi(i)^\star \neq \phi(i)'^\star$ aus.

Somit bricht der Angreifer $\mathcal{B}$ die Berechnungsgebundenheit des Polynom-Commitment-Schemas mit derselben nicht vernachlässigbaren Wahrscheinlichkeit $\epsilon(\lambda)$. □

Bevor wir die Sicherheit des im nächsten Abschnitt 3.2 beschriebenen BBS+A-Signaturverfahrens beweisen können, benötigen wir eine weitere Sicherheitseigenschaft, die wir *stark berechnungsbindend* nennen. Denn kein polynomieller Angreifer soll in der Lage sein, ein Commitment C und für zwei verschiedene Zahlen $i, i' \in \mathbb{Z}_p$ zwei Tupel $(r, m_1, \ldots, m_L, i, \phi(i), W_i) \in \mathbb{Z}_p^{L+3} \times \mathbb{G}_1$ und $(r', m_1', \ldots, m_L', i', \phi'(i'), W_{i'}') \in \mathbb{Z}_p^{L+3} \times \mathbb{G}_1$ mit $(r, m_1, \ldots, m_L) \neq (r', m_1', \ldots, m_L')$ auszugeben, so dass $\mathsf{VerifyEval}$ beide Eingaben akzeptiert. Denn daraus würde folgen, dass $\phi(\mathsf{x}) \neq \phi'(\mathsf{x})$ wäre und somit $\phi(i)$ und $\phi'(i')$ zwei Auswertungen von zwei verschiedenen Polynomen wären. Dies hätte zur Folge, dass die Sicherheit des BBS+A-Signaturverfahrens nicht beweisbar wäre.

Definition 3.1.1 (**Starke Berechnungsgebundenheit**). *Das Commitment-Schema* $(\mathsf{GenCom}, \mathsf{Com}, \mathsf{VerifyCom}, \mathsf{EvalWit}, \mathsf{VerifyEval})$ *aus Abschnitt 3.1 mit Sicherheitsparameter* $\lambda \in \mathbb{N}$ *heißt* stark berechnungsbindend, *falls es für jeden polynomiellen Sender* $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ *eine vernachlässigbare Funktion* ν *gibt, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\begin{gathered} \Pr\Big[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenCom}\left(1^\lambda, L, K\right);\ (C, \mathsf{state}) \leftarrow \mathcal{S}_0(\mathsf{pk});\ i, i' \in_{\mathcal{R}} \mathbb{Z}_p; \\ \left((r, m_1, \ldots, m_L, i, \phi(i), W_i), (r', m_1', \ldots, m_L', i', \phi'(i'), W_{i'}')\right) \leftarrow \mathcal{S}_1(\mathsf{pk}, C, \mathsf{state}, i, i') : \\ (r, m_1, \ldots, m_L) \neq (r', m_1', \ldots, m_L') \ \wedge \\ \mathsf{VerifyEval}(\mathsf{pk}, C, (m_1, \ldots, m_L), r, (i, \phi(i), W_i)) = 1 \ \wedge \\ \mathsf{VerifyEval}(\mathsf{pk}, C, (m_1', \ldots, m_L'), r', (i', \phi'(i'), W_{i'}')) = 1\Big] \leq \nu(\lambda), \end{gathered}$$

wobei state *Informationen zum aktuellen Zustand von* $\mathcal{S}_0$ *enthält.*

Beachte, dass die Sicherheitseigenschaft stark berechnungsbindend nicht die Eigenschaft berechnungsbindend aus Definition 2.9.3 impliziert. Denn bei der Berechnungsgebundenheit darf der Sender den Index i wählen, während dies bei der starken Berechnungsgebundenheit nicht der Fall ist.

Bevor wir beweisen können, dass das obige Commitment-Schema stark berechnungsbindend ist, halten wir das folgende Korollar fest, welches aus [KZG10a, KZG10b] folgt.

Korollar 3.1.1. *Gegeben sei das Polynom Commitment-Schema* $(\mathsf{GenCom}, \mathsf{PolyCom}, \mathsf{VerifyCom}, \mathsf{EvalWit}, \mathsf{VerifyEval})$ *von [KZG10a, KZG10b] (siehe Unterabschnitt 2.9.2).*

Angenommen, es gelten die q-SDH-Annahme und die q-polyDH-Annahme für $\mathsf{Setup}_{\mathsf{Bil}}$. *Dann gibt es für jeden polynomiellen Sender* $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ *eine vernachlässigbare Funktion* ν*, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *und für* $K' > K$ *gilt:*

$$\begin{aligned}
&\Pr\Big[(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenCom}\left(1^\lambda, K\right);\ (C, \mathsf{state}) \leftarrow \mathcal{S}_0\left(\mathsf{pk}\right);\ i^{(1)}, \ldots, i^{(K')} \in \mathbb{Z}_p; \\
&\left(\left(i^{(1)}, \phi\left(i^{(1)}\right), W_{i^{(1)}}\right), \ldots, \left(i^{(K')}, \phi\left(i^{(K')}\right), W_{i^{(K')}}\right)\right) \leftarrow \mathcal{S}_1\left(\mathsf{pk}, C, \mathsf{state}, i^{(1)}, \ldots, i^{(K')}\right)\ : \\
&\mathsf{VerifyEval}\left(\mathsf{pk}, C, i^{(1)}, \phi\left(i^{(1)}\right), W_{i^{(1)}}\right) = \cdots = \mathsf{VerifyEval}\left(\mathsf{pk}, C, i^{(K')}, \phi\left(i^{(K')}\right), W_{i^{(K')}}\right) = 1 \\
&\wedge\ C \neq \mathsf{PolyCom}\left(\mathsf{pk}, \phi(\mathsf{x})\right)\Big] \leq \nu(\lambda),
\end{aligned}$$

wobei state *Informationen zum aktuellen Zustand von* $\mathcal{S}_0$ *enthält und* $\phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ *das Polynom mit* $\deg(\phi) < K'$ *ist, das durch Interpolation der Paare* $\left(i^{(1)}, \phi\left(i^{(1)}\right)\right), \ldots,$ $\left(i^{(K')}, \phi\left(i^{(K')}\right)\right)$ *entsteht.*

Dies wird zwar nicht direkt bewiesen, folgt allerdings aus der Berechnungsgebundenheit sowie der Beschränktheit des Polynom-Commitment-Schemas und wird implizit in [KZG10b] verwendet.

Beweis. Angenommen, es existiert ein polynomieller Angreifer $\mathcal{A}$, der mit Wahrscheinlichkeit $\nu(\lambda)$ ein Commitment C und für $K' > K$ beliebige Zahlen $i^{(1)}, \ldots, i^{(K')} \in \mathbb{Z}_p$ die Tripel $\left(i^{(1)}, \phi\left(i^{(1)}\right), W_{i^{(1)}}\right), \ldots, \left(i^{(K')}, \phi\left(i^{(K')}\right), W_{i^{(K')}}\right)$ mit

$$\hat{e}(C, h) = \hat{e}\left(W_{i^{(j)}}, h_1 h^{-i^{(j)}}\right) \hat{e}\left(u_0^{\phi\left(i^{(j)}\right)}, h\right)$$

für alle $1 \leq j \leq K'$ ausgibt, obwohl $C \neq \mathsf{PolyCom}\left(\mathsf{pk}, \phi(\mathsf{x})\right)$ gilt, wobei $\phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ das interpolierte Polynom wie in Korollar 3.1.1 ist. Dann gibt es die beiden Möglichkeiten $C \neq u_0^{\phi(\alpha)}$ oder $\deg(\phi) > K$.

Wir zeigen, dass ν eine vernachlässigbare Funktion ist.

Aus der Berechnungsgebundenheit des Polynom-Commitment-Schemas folgt, dass, außer mit vernachlässigbarer Wahrscheinlichkeit, der Wert $\phi\left(i^{(j)}\right)$ für alle $1 \leq j \leq K'$ die korrekte Berechnung des Polynoms $\phi(\mathsf{x})$ mit $C = u_0^{\phi(\alpha)}$ an der Stelle $i^{(j)}$ ist. Somit gilt für das interpolierte Polynom $\phi(\mathsf{x})$, außer mit vernachlässigbarer Wahrscheinlichkeit:

$C = u_0^{\phi(\alpha)}$. Dies wurde bereits im Beweis von Theorem 3.5 in [KZG10b] verwendet, um die Beschränktheit des Polynom-Commitment-Schemas zu beweisen.

Weiter folgt aus der Beschränktheit des Polynom-Commitment-Schemas, dass, außer mit vernachlässigbarer Wahrscheinlichkeit, $\deg(\phi) \leq K$ gilt.

Aus beiden Eigenschaften folgt schließlich, dass nur mit vernachlässigbarer Wahrscheinlichkeit $C \neq \mathsf{PolyCom}(\mathsf{pk}, \phi(\mathsf{x}))$ gilt. □

Das Korollar 3.1.1 wurde bereits implizit im Beweis von Theorem 4.1 in [KZG10b] verwendet, um die Sicherheit des in [KZG10a, KZG10b] vorgestellten verifizierbaren Secret-Sharing-Verfahrens zu beweisen.

Aus obigem Korollar 3.1.1 folgt:

Korollar 3.1.2. *Gegeben sei das Polynom-Commitment-Schema* $(\mathsf{GenCom}, \mathsf{PolyCom}, \mathsf{VerifyCom}, \mathsf{EvalWit}, \mathsf{VerifyEval})$ *von [KZG10a, KZG10b] (siehe Unterabschnitt 2.9.2) mit Sicherheitsparameter* $\lambda \in \mathbb{N}$. *Sei* $\mathcal{A}$ *ein polynomieller Angreifer, der als Eingabe den öffentlichen Schlüssel* $\mathsf{pk} \leftarrow \mathsf{GenCom}\left(1^\lambda, K\right)$ *des Polynom-Commitment-Schemas erhält und sei* $C \in \mathbb{G}_1$.

Angenommen, es gelten die q-SDH-Annahme und die q-polyDH-Annahme für $\mathsf{Setup}_{\mathsf{Bil}}$. *Dann kennt* $\mathcal{A}$ *genau dann ein Polynom* $\phi(\mathsf{x}) \in \mathbb{M}_{\mathsf{Com}}$ *mit* $C = \mathsf{PolyCom}\left(\mathsf{pk}, \phi(\mathsf{x})\right)$, *wenn* $\mathcal{A}$ *zu einer von einem Challenger* $\mathcal{C}$ *zufällig gewählten Zahl* $i \in_{\mathcal{R}} \mathbb{Z}_p$ *mit nicht vernachlässigbarer Wahrscheinlichkeit* $\epsilon(\lambda)$ *ein Tripel* $\left(i, \phi\left(i\right), W_i\right)$ *mit* $\mathsf{VerifyEval}(\mathsf{pk}, C, i, \phi(i), W_i) = 1$ *ausgeben kann.*

Beweis. $\Rightarrow$: Folgt direkt aus der Durchführbarkeit des Polynom-Commitment-Schemas. $\Leftarrow$: Angenommen, der Angreifer $\mathcal{A}$ kann mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon\left(\lambda\right)$ zur Zufallszahl $i^{(1)} \in_{\mathcal{R}} \mathbb{Z}_p$ einen Wert $\phi\left(i^{(1)}\right) \in \mathbb{Z}_p$ sowie einen Zeugen $W_{i^{(1)}} \in \mathbb{G}_1$ mit $\hat{e}\left(C, h\right) = \hat{e}\left(W_{i^{(1)}}, h_1 h^{-i^{(1)}}\right) \hat{e}\left(u_0^{\phi\left(i^{(1)}\right)}, h\right)$ ausgeben. Dann gibt es einen probabilistischen voraussichtlich-polynomiellen Extractor $\mathcal{E}$, der die Rolle des Challengers $\mathcal{C}$ übernimmt und mit nicht vernachlässigbarer Wahrscheinlichkeit ein Polynom $\phi\left(\mathsf{x}\right) \in \mathbb{Z}_p\left[\mathsf{x}\right]$ mit $C = \mathsf{PolyCom}\left(\mathsf{pk}, \phi(\mathsf{x})\right)$ extrahiert.

Sei $n = 2$. Der Extractor $\mathcal{E}$ spult den Angreifer $\mathcal{A}$ zurück, wählt eine neue Zufallszahl $i^{(n)} \in_{\mathcal{R}} \mathbb{Z}_p \setminus \{i^{(1)}, \ldots, i^{(n-1)}\}$ und erhält ein Tripel $\left(i^{(n)}, \phi\left(i^{(n)}\right), W_{i^{(n)}}\right)$ mit $\hat{e}\left(C, h\right) = \hat{e}\left(W_{i^{(n)}}, h_1 h^{-i^{(n)}}\right) \hat{e}\left(u_0^{\phi\left(i^{(n)}\right)}, h\right)$. Anschließend erhöht der Extractor $\mathcal{E}$ den Wert n zu $n+1$ und wiederholt diesen Schritt K-mal. Somit kennt $\mathcal{E}$ insgesamt $K+1$ Tripel $\left(i^{(1)}, \phi\left(i^{(1)}\right), W_{i^{(1)}}\right), \ldots, \left(i^{(K')}, \phi\left(i^{(K')}\right), W_{i^{(K')}}\right)$ mit $\hat{e}\left(C, h\right) = \hat{e}\left(W_{i^{(j)}}, h_1 h^{-i^{(j)}}\right) \hat{e}\left(u_0^{\phi\left(i^{(j)}\right)}, h\right)$ für alle $1 \leq j \leq K+1$. Nun kann $\mathcal{E}$ durch Polynominterpolation der $K+1$ Punkte $\left(i^{(1)}, \phi\left(i^{(1)}\right)\right), \ldots, \left(i^{(K+1)}, \phi\left(i^{(K+1)}\right)\right)$ das Polynom $\phi\left(\mathsf{x}\right) \in \mathbb{Z}_p\left[\mathsf{x}\right]$ berechnen (vgl. z.B. Secret-Sharing-Verfahren von A. Shamir [Sha79] oder [KZG10a, KZG10b]). Dann folgt aus Korollar 3.1.1, dass nur mit vernachlässigbarer Wahrscheinlichkeit $C \neq \mathsf{PolyCom}\left(\mathsf{pk}, \phi(\mathsf{x})\right)$ gilt.

Folglich hat der Extractor $\mathcal{E}$ das Polynom $\phi(\mathsf{x})$ mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$ extrahiert, wobei ν eine vernachlässigbare Funktion ist (vgl. auch *Forking Lemma* in [PS96, PS00, KLL06]). □

Mit Korollar 3.1.2 und Lemma 3.1.2 kann nun die starke Berechnungsgebundenheit des Commitment-Schemas bewiesen werden.

Lemma 3.1.4. *Das Commitment-Schema* (GenCom, Com, VerifyCom, EvalWit, VerifyEval) *ist stark berechnungsbindend, falls die q-SDH-Annahme und die q-polyDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gelten.*

Beweis. Angenommen, es existiert ein polynomieller Angreifer $\mathcal{A}$, der mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ die starke Berechnungsgebundenheit gemäß Definition 3.1.1 bricht.

Dann hat der Angreifer $\mathcal{A}$ bzgl. des Commitments $C^\star := Cg_0^{-r}g_1^{-m_1}\cdots g_L^{-m_L} \in \mathbb{G}_1$ zur Zufallszahl $i \in_{\mathcal{R}} \mathbb{Z}_p$ ein Tripel $(i, \phi(i), W_i)$ mit $\hat{e}(C^\star, h) = \hat{e}(W_i, h_1h^{-i})\,\hat{e}\left(u_0^{\phi(i)}, h\right)$ ausgegeben. Mit Korollar 3.1.2 folgt, dass $\mathcal{A}$ ein Polynom $\phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ mit $\deg(\phi) \leq K$ und $C^\star = u_0^{\phi(\alpha)}$ kennt. Analog kennt $\mathcal{A}$ ein Polynom $\phi'(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ mit $\deg(\phi') \leq K$ und $\hat{e}(C'^\star, h) = \hat{e}\left(W'_{i'}, h_1h^{-i'}\right)\hat{e}\left(u_0^{\phi'(i')}, h\right)$ für $C'^\star := Cg_0^{-r'}g_1^{-m'_1}\cdots g_L^{-m'_L} = u_0^{\phi'(\alpha)}$. Insgesamt kennt $\mathcal{A}$ also mit nicht vernachlässigbarer Wahrscheinlichkeit die beiden verschiedenen Tupel $(r, m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^{L+1} \times \mathbb{Z}_p[\mathsf{x}]$ und $(r', m'_1, \ldots, m'_L, \phi'(\mathsf{x})) \in \mathbb{Z}_p^{L+1} \times \mathbb{Z}_p[\mathsf{x}]$ mit $\deg(\phi), \deg(\phi') \leq K$ und $C = g_0^r g_1^{m_1}\cdots g_L^{m_L}u_0^{\phi(\alpha)} = g_0^{r'}g_1^{m'_1}\cdots g_L^{m'_L}u_0^{\phi'(\alpha)}$. Dies ist allerdings ein Widerspruch zur Gebundenheit des Commitment-Schemas (vgl. Lemma 3.1.2). □

Nun muss gezeigt werden, dass das Commitment-Schema beschränkt ist.

Lemma 3.1.5. *Das Commitment-Schema* (GenCom, Com, VerifyCom, EvalWit, VerifyEval) *ist beschränkt, falls die q-SDH-Annahme und die q-polyDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gelten.*

Beweis. Angenommen, es existiert ein polynomieller Angreifer $\mathcal{A}$, der mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ ein Commitment $C \in \mathbb{G}_1$ mit $C = g_0^r g_1^{m_1}\cdots g_L^{m_L}u_0^{\phi(\alpha)}$ und eine Zahl $K' \in \mathbb{N}$ mit $\deg(\phi) = K' > K$ ausgibt. Wir konstruieren einen polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, die Beschränktheit des Polynom-Commitments zu brechen, indem $\mathcal{B}$ mit einem Challenger $\mathcal{C}$ das Spiel 2.9.1 durchführt und gewinnt, was ein Widerspruch zur q-polyDH-Annahme mit $q = K$ ist (vgl. Unterabschnitt 2.9.2).

Schlüsselgenerierung: Der Angreifer $\mathcal{B}$ erhält den öffentlichen Schlüssel $\mathsf{pk}' = (p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \hat{e}, u_0, \ldots, u_K, h, h_1)$ des Polynom-Commitment-Schemas. Dann wählt $\mathcal{B}$ zufällig $L+1$ paarweise verschiedene Zahlen $\delta_0, \ldots, \delta_L \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Generatoren $g_0 = u_0^{\delta_0}, \ldots, g_L = u_0^{\delta_L}$. Anschließend sendet $\mathcal{B}$ den perfekt simulierten öffentlichen Schlüssel $\mathsf{pk} = (p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \hat{e}, g_0, \ldots, g_L, u_0, \ldots, u_K, h, h_1)$ an den Angreifer $\mathcal{A}$.

Ausgabe Commitment: Der Angreifer $\mathcal{A}$ gibt ein Commitment $C \in \mathbb{G}_1$ und eine Zahl $K' \in \mathbb{N}$ mit $K' > K$ aus. Der Angreifer $\mathcal{B}$ leitet das Commitment C und die Zahl K' weiter an den Challenger $\mathcal{C}$.

Challenge: Der Challenger $\mathcal{C}$ sendet $K'+1$ zufällige Zahlen $i^{(1)}, \ldots, i^{(K'+1)} \in_{\mathcal{R}} \mathbb{Z}_p$ an den Angreifer $\mathcal{B}$, welcher die Zahlen an den Angreifer $\mathcal{A}$ weiterleitet.

Ausgabe Responses: Der Angreifer $\mathcal{A}$ gibt insgesamt $K'+1$ Tupel $\left(r^{(j)}, m_1^{(j)}, \ldots, m_L^{(j)}, i^{(j)}, \phi\left(i^{(j)}\right), W_{i^{(j)}}\right) \in \mathbb{Z}_p^{L+3} \times \mathbb{G}_1$ für $1 \leq j \leq K'+1$ mit

$$\hat{e}\left(C, h\right) = \hat{e}\left(g_0^{r^{(j)}} g_1^{m_1^{(j)}} \cdots g_L^{m_L^{(j)}}, h\right) \hat{e}\left(W_{i^{(j)}}, h_1 h^{-i^{(j)}}\right) \hat{e}\left(u_0^{\phi\left(i^{(j)}\right)}, h\right) \tag{3.10}$$

aus. Dann gilt nach obigem Lemma 3.1.4 für alle $1 \leq j \leq K'+1$, dass, außer mit vernachlässigbarer Wahrscheinlichkeit, $\left(r^{(1)}, m_1^{(1)}, \ldots, m_L^{(1)}\right) = \cdots = \left(r^{(K'+1)}, m_1^{(K'+1)}, \ldots, m_L^{(K'+1)}\right)$ ist. Sei $\phi\left(\mathsf{x}\right) \in \mathbb{Z}_p\left[\mathsf{x}\right]$ das Polynom, das durch Interpolation der $K'+1$ Punkte $\left(i^{(1)}, \phi\left(i^{(1)}\right)\right), \ldots, \left(i^{(K'+1)}, \phi\left(i^{(K'+1)}\right)\right)$ entsteht und sei $\phi^\star\left(\mathsf{x}\right) := \phi\left(\mathsf{x}\right) + \delta_0 r^{(1)} + \delta_1 m_1^{(1)} + \cdots + \delta_L m_L^{(1)} \in \mathbb{Z}_p\left[\mathsf{x}\right]$ mit $\deg(\phi) = \deg\left(\phi^\star\right)$. Der Angreifer $\mathcal{B}$ berechnet die Werte $\phi^\star\left(i^{(j)}\right) = \phi\left(i^{(j)}\right) + \delta_0 r^{(1)} + \delta_1 m_1^{(1)} + \cdots + \delta_L m_L^{(1)} \mod p$ für alle $1 \leq j \leq K'+1$ und leitet die $K'+1$ Tripel $\left(i^{(j)}, \phi^\star\left(i^{(j)}\right), W_{i^{(j)}}\right) \in \mathbb{Z}_p^2 \times \mathbb{G}_1$ weiter an den Challenger $\mathcal{C}$, wobei mit Gleichung 3.10 für alle $1 \leq j \leq K'+1$ gilt: $\hat{e}\left(C, h\right) = \hat{e}\left(W_{i^{(j)}}, h_1 h^{-i^{(j)}}\right) \hat{e}\left(u_0^{\phi\left(i^{(j)}\right)^\star}, h\right)$.

Somit bricht der Angreifer $\mathcal{B}$ die Beschränktheit des Polynom-Commitment-Schemas mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon\left(\lambda\right) - \nu\left(\lambda\right)$, wobei ν eine vernachlässigbare Funktion ist. □

Aus den obigen Lemmata folgt der Satz:

Satz 3.1.1. *Das Commitment-Schema* (GenCom, Com, VerifyCom, EvalWit, VerifyEval) *ist perfekt verbergend, rechnerisch bindend, berechnungsbindend, stark berechnungsbindend und beschränkt, falls die q-SDH-Annahme und die q-polyDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gelten.*

Insbesondere gilt das folgende Korollar, welches fundamental für die Konstruktion sowie die Sicherheit der Protokolle Issue und Prove des BBS+A-Signaturverfahrens ist.

Korollar 3.1.3. *Gegeben sei das Commitment-Schema* (GenCom, Com, VerifyCom, EvalWit, VerifyEval) *mit Sicherheitsparameter* $\lambda \in \mathbb{N}$. *Sei* $\mathcal{A}$ *ein polynomieller Angreifer, der als Eingabe den öffentlichen Schlüssel* $\mathsf{pk} \leftarrow \mathsf{GenCom}\left(1^\lambda, L, K\right)$ *des Commitment-Schemas erhält und sei* $C \in \mathbb{G}_1$.

Angenommen, es gelten die q-SDH-Annahme und die q-polyDH-Annahme für $\mathsf{Setup}_{\mathsf{Bil}}$. *Dann kennt* $\mathcal{A}$ *genau dann eine Nachricht* $m \in \mathbb{M}_{\mathsf{Com}}$ *und einen Zufallswert* $r \in \mathbb{R}$ *mit* $C = \mathsf{Com}\,(\mathsf{pk}, m; r)$, *wenn* $\mathcal{A}$ *zu einer von einem Challenger* $\mathcal{C}$ *zufällig gewählten Zahl* $i \in_{\mathcal{R}} \mathbb{Z}_p$ *mit nicht vernachlässigbarer Wahrscheinlichkeit* $\epsilon(\lambda)$ *ein Tupel* $(r, m_1, \ldots, m_L, i, \phi\,(i)\,, W_i)$ *mit* $\mathsf{VerifyEval}\,(\mathsf{pk}, C, (m_1, \ldots, m_L), r, (i, \phi(i), W_i)) = 1$ *ausgeben kann.*

Beweis. Der Beweis erfolgt analog zu Korollar 3.1.2.
⇒: Folgt direkt aus der Durchführbarkeit des Commitment-Schemas.
⇐: Angenommen, der Angreifer $\mathcal{A}$ kann mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon\,(\lambda)$ zur Zufallszahl $i^{(1)} \in_{\mathcal{R}} \mathbb{Z}_p$ ein Tupel $\left(r^{(1)}, m_1^{(1)}, \ldots, m_L^{(1)}, i^{(1)}, \phi\left(i^{(1)}\right), W_{i^{(1)}}\right) \in \mathbb{Z}_p^{L+3} \times \mathbb{G}_1$ mit $\hat{e}\,(C, h) = \hat{e}\left(g_0^{r^{(1)}} g_1^{m_1^{(1)}} \cdots g_L^{m_L^{(1)}}, h\right) \hat{e}\left(W_{i^{(1)}}, h_1 h^{-i^{(1)}}\right) \hat{e}\left(u_0^{\phi\left(i^{(1)}\right)}, h\right)$ ausgeben. Dann gibt es einen probabilistischen voraussichtlich-polynomiellen Extractor $\mathcal{E}$, der die Rolle des Challengers $\mathcal{C}$ übernimmt und mit nicht vernachlässigbarer Wahrscheinlichkeit eine Nachricht $m = (m_1, \ldots, m_L, \phi\,(\mathsf{x})) \in \mathbb{Z}_p^L \times \mathbb{Z}_p\,[\mathsf{x}]$ mit $\deg(\phi) \leq K$ sowie einen Zufallswert $r \in \mathbb{Z}_p$ mit $C = \mathsf{Com}\,(\mathsf{pk}, m; r)$ extrahiert.

Sei $n = 2$. Der Extractor $\mathcal{E}$ spult den Angreifer $\mathcal{A}$ zurück, wählt eine neue Zufallszahl $i^{(n)} \in_{\mathcal{R}} \mathbb{Z}_p \setminus \{i^{(1)}, \ldots, i^{(n-1)}\}$ und erhält ein Tupel $\left(r^{(n)}, m_1^{(n)}, \ldots, m_L^{(n)}, i^{(n)}, \phi\left(i^{(n)}\right), W_{i^{(n)}}\right)$ mit $\hat{e}\,(C, h) = \hat{e}\left(g_0^{r^{(n)}} g_1^{m_1^{(n)}} \cdots g_L^{m_L^{(n)}}, h\right) \hat{e}\left(W_{i^{(n)}}, h_1 h^{-i^{(n)}}\right) \hat{e}\left(u_0^{\phi'\left(i^{(n)}\right)}, h\right)$. Dann folgt aus Lemma 3.1.4, dass, außer mit vernachlässigbarer Wahrscheinlichkeit, $\left(r^{(n)}, m_1^{(n)}, \ldots, m_L^{(n)}\right) = \left(r^{(1)}, m_1^{(1)}, \ldots, m_L^{(1)}\right)$ ist. Zur Vereinfachung sei $(r, m_1, \ldots, m_L) := \left(r^{(1)}, m_1^{(1)}, \ldots, m_L^{(1)}\right)$. Der Extractor $\mathcal{E}$ erhöht n zu $n+1$ und wiederholt diesen Schritt K-mal. Somit kennt $\mathcal{E}$ insgesamt $K+1$ verschiedene Punkte $\left(i^{(1)}, \phi\left(i^{(1)}\right)\right), \ldots, \left(i^{(K+1)}, \phi\left(i^{(K+1)}\right)\right)$ des Polynoms $\phi\,(\mathsf{x})$, denn nur mit vernachlässigbarer Wahrscheinlichkeit sind nicht alle Werte $\phi\left(i^{(n)}\right)$ für $1 \leq n \leq K+1$ die korrekte Berechnung des Polynoms $\phi\,(\mathsf{x})$ an der Stelle $i^{(n)}$ (Berechnungsgebundenheit, vgl. Lemma 3.1.3) und nur mit vernachlässigbarer Wahrscheinlichkeit ist $\deg\,(\phi) > K$ (Beschränktheit, vgl. Lemma 3.1.5). Somit kann $\mathcal{E}$ durch Polynominterpolation das Polynom $\phi\,(\mathsf{x}) \in \mathbb{Z}_p\,[\mathsf{x}]$ mit $\deg\,(\phi) \leq K$ und $C = g_0^r g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}$ berechnen. Folglich hat der Extractor $\mathcal{E}$ eine Nachricht $m = (m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{M}_{\mathsf{Com}}$ und einen Zufallswert $r \in \mathbb{R}$ mit $C = \mathsf{Com}(\mathsf{pk}, m; r)$ mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu\,(\lambda)$ extrahiert, wobei ν eine vernachlässigbare Funktion ist. □

3.1.2 Bemerkungen zum Commitment-Schema

Es ist zu beachten, dass für ein Commitment $C = g_0^r g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}$ sowohl die Beziehung

$$C = \mathsf{PedCom}(\mathsf{pk}, m; r) \cdot \mathsf{PolyCom}(\mathsf{pk}, \phi(\mathsf{x}))$$

auch als

$$C = \mathsf{PedCom}(\mathsf{pk}, m; r) \cdot \mathsf{Acc}(\mathsf{pk}, X)$$

für die Nachricht $m = (m_1, \ldots, m_L) \in \mathbb{Z}_p^L$ und eine Wertemenge $X = \{x_1, \ldots, x_K\} \subset \mathbb{Z}_p \setminus \{-\alpha\}$ mit $\phi(\mathsf{x}) = \prod_{j=1}^{K} (\alpha + x_j) \in \mathbb{Z}_p[\mathsf{x}]$ betrachtet werden können (vgl. Gleichung 3.1).

Durch das Korollar 3.1.3 wird deutlich, weshalb die Verwendung des Polynom-SignaturCommitment-SignaturSchemas elementar für das BBS+A-Signaturverfahren ist. Denn ohne den Einsatz des Polynom-SignaturCommitment-SignaturSchemas ist eine zu Korollar 3.1.3 entsprechende Aussage, wie etwa

„der Angreifer $\mathcal{A}$ kennt genau dann eine Nachricht $m \in \mathbb{M}_{\mathsf{Com}}$, einen Zufallswert $r \in \mathbb{R}$ und eine Wertemenge $X \subset \mathbb{X}$ mit $C = \mathsf{PedCom}(\mathsf{pk}, m; r) \cdot \mathsf{Acc}(\mathsf{pk}, X)$, wenn $\mathcal{A}$ zu einer von einem Challenger $\mathcal{C}$ zufällig gewählten Zahl $i \in_{\mathcal{R}} \{1, \ldots, K\}$ mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ ein Tupel $\left(r, m_1, \ldots, m_L, x_i, \hat{W}_i\right)$ mit $\hat{e}(C, h) = \hat{e}(g_0^r g_1^{m_1} \cdots g_L^{m_L}, h)\, \hat{e}\left(\hat{W}_i, h_1 h^{x_i}\right)$ ausgeben kann“,

aus zweierlei Gründen nicht möglich:

1. Der Extractor $\mathcal{E}$ kann den Angreifer $\mathcal{A}$ zwar analog zu Korollar 3.1.3 ebenfalls K-mal zurückspulen und immer eine andere Zahl $i \in \{1, \ldots, K\}$ wählen, doch da das Akkumulator-Schema quasi kommutativ ist, kann nicht überprüft werden, ob $\mathcal{A}$ tatsächlich das korrekte Werte-Zeugen-Paar $(x_i, \hat{W}_i)$ bzgl. i ausgibt. Stattdessen kann $\mathcal{A}$ beispielsweise stets dasselbe Paar $(x_1, \hat{W}_1)$ ausgeben. Somit kann der Extractor $\mathcal{E}$ nicht die gesamte Wertemenge $X = \{x_1, \ldots, x_K\}$ extrahieren.

2. Selbst falls der Extractor $\mathcal{E}$ eine vollständige Wertemenge $X = \{x_1, \ldots, x_K\}$ extrahieren könnte, ist dadurch nicht garantiert, dass $V = \mathsf{Acc}(\mathsf{pk}, X)$ gilt. Denn der Angreifer $\mathcal{A}$ kann eine Wertemenge $X = \{x_1, \ldots, x_K\}$ wählen, aber den Akkumulator nicht korrekt mit $V = \mathsf{Acc}(\mathsf{pk}, X) = u_0^{\prod_{j=1}^{K}(\alpha + x_j)}$, sondern für eine Zufallszahl $r \in_{\mathcal{R}} \mathbb{Z}_p$ durch $V^\star = u_0^{r \cdot \prod_{j=1}^{K}(\alpha + x_j)}$ berechnen. Einen korrekten Zeugen kann $\mathcal{A}$ dann entsprechend durch $\hat{W}_i^\star = u_0^{r \cdot \prod_{j=1, j \neq i}^{K}(\alpha + x_j)}$ berechnen, so dass wie gewünscht die Gleichung $\left(\hat{W}_i^\star\right)^{\alpha + x_i} = V^\star$ erfüllt ist.

3.2 Das BBS+A-Signaturverfahren

In diesem Abschnitt wird die Konstruktion des BBS+A-Signaturverfahrens beschrieben. Dazu wird im folgenden Unterabschnitt 3.2.1 zunächst ein Überblick über das Signaturverfahren angegeben. Nach der detaillierten Konstruktion in Unterabschnitt 3.2.2 folgen die Sicherheitsanalyse in Unterabschnitt 3.2.3 sowie ein Effizienzvergleich mit dem ESS+-Signaturverfahren in Unterabschnitt 3.2.4.

3.2.1 Überblick über das BBS+A-Signaturverfahren

GenSign generiert bei Eingabe des Sicherheitsparameters 1^λ und zweier Zahlen $L, K \in \mathbb{N}$ den öffentlichen Schlüssel $\mathsf{pk}_{\mathsf{Com}} \leftarrow \mathsf{GenCom}\left(1^\lambda, L, K\right)$ des im vorherigen Abschnitt 3.1 vorgestellten Commitment-Schemas, inklusive des öffentlichen Schlüssels des Akkumulator-Schemas $\mathsf{pk}_{\mathsf{Acc}}$, sowie das Schlüsselpaar $\left(\mathsf{pk}_{\mathsf{BBS+}}, \mathsf{sk}_{\mathsf{BBS+}}\right) = (X, x) \in \mathbb{G}_2 \times \mathbb{Z}_p^*$ der BBS+-Signatur mit $X = h^x$. Insgesamt ist der öffentliche Schlüssel

$$\mathsf{pk} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_0, \ldots, g_L, u_0, \ldots, u_K, h, h_1, \ldots, h_K, X)\,.$$

Sign signiert die Nachricht $m = (m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}]$ mit $\deg(\phi) \leq K$ durch

$$\sigma = \left(\Sigma = \left(g g_0^s g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}\right)^{\frac{1}{x+e}}, e, s\right) \in \left(\mathbb{G}_1 \times \mathbb{Z}_p^2\right).$$

Dadurch entspricht die BBS+A-Signatur nahezu der BBS+-Signatur, weshalb die Sicherheit der BBS+A-Signatur analog zur Sicherheit der BBS+-Signatur bewiesen werden kann.

VerifySign verifiziert die Signatur $\sigma = (\Sigma, e, s)$ auf eine Nachricht $m = (m_1, \ldots, m_L, \phi(\mathsf{x}))$ durch Überprüfung der Gleichung

$$\hat{e}\left(\Sigma, X h^e\right) \stackrel{?}{=} \hat{e}\left(g g_0^s g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}, h\right).$$

Issue: Im Issue-Protokoll wird eine Signatur σ auf eine Nachricht $m = (m_1, \ldots, m_L, \phi(\alpha)) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}]$ erstellt, wobei $\phi(\mathsf{x})$ ein Polynom mit $\deg(\phi) \leq K$ und $V = u_0^{\phi(\alpha)}$ ein Akkumulator ist. Dazu berechnet der User $\mathcal{U}$ ein Commitment $C = g_0^r g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}$ gemäß Abschnitt 3.1 und muss dem Signer in zero-knowledge beweisen, dass er die Nachricht $m = (m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}]$ mit $\phi(\mathsf{x}) \leq K$ sowie die Zufallszahl $r \in \mathbb{Z}_p$ kennt.

Dazu muss $\mathcal{U}$ gemäß Korollar 3.1.3 zu einer zufälligen Zahl $l \in_{\mathcal{R}} \mathbb{Z}_p$ beweisen, dass er das Tupel $(r, m_1, \ldots, m_L, l, \phi(l), W_l)$ kennt, so dass VerifyEval akzeptiert.

Dies ist für die Sicherheit des Issue-Protokolls elementar, denn um die Signer-Privacy (vgl. Abschnitt 2.12) des Protokolls zu beweisen, muss ein Simulator $\mathcal{SIM}$ ohne Kenntnis von $\mathsf{sk}_{\mathsf{BBS+A}}$ die Rolle des Signers $\mathcal{S}$ simulieren, wobei $\mathcal{SIM}$ den (möglicherweise betrügerischen) User $\mathcal{U}^*$ als zurückspulbare Blackbox verwendet und Zugriff auf ein Signatur-Orakel hat. Somit muss der Simulator $\mathcal{SIM}$ genügend Informationen von $\mathcal{U}^*$ erhalten, um das Polynom $\phi(\mathsf{x})$ zu berechnen. Durch die Rewinding-Technik (vgl. [KLL06]) kann der Simulator $\mathcal{SIM}$ die Werte aus dem Proof-of-Knowledge extrahieren und den User $\mathcal{U}^*$ zurückspulen, um analog zu Korollar 3.1.3 das Polynom $\phi(\mathsf{x})$ zu interpolieren.

Es wird jedoch eine nichtinteraktive Signature-of-Knowledge anstelle eines interaktiven Proof-of-Knowledge verwendet.

Prove: Das Prove-Protokoll ist ein Zero-Knowledge-Proof-of-Knowledge-Protokoll um zu zeigen, dass sich ein geheimer Wert (o.B.d.A. sei dies) $n_1 \in \mathbb{Z}_p \setminus \{-\alpha\}$ in einem geheimen Akkumulator $V = u_0^{\phi(\alpha)}$ befindet und der User $\mathcal{U}$ im Besitz einer gültigen BBS+A-Signatur (Σ, e, s) auf eine Nachricht $m = (m_1, \ldots, m_L, \phi(\mathsf{x}))$ ist. Somit ist dieses Protokoll eine Kombinationen der Beweise, dass sich ein geheimer Wert in einem geheimen Akkumulator befindet (vgl. Unterunterabschnitt 2.11.1.2) und über die Kenntnis einer BBS+-Signatur (vgl. Unterabschnitt 2.12.1). Daher müssen die folgenden Relationen (in zero-knowledge) bewiesen werden:

$$\hat{e}(V, h) = \hat{e}\left(\hat{W}_1, h_1 h^{n_1}\right), \tag{3.11}$$

$$\hat{e}(\Sigma, Xh^e) = \hat{e}(g g_0^s g_1^{m_1} \ldots g_L^{m_L} V, h). \tag{3.12}$$

Zusätzlich muss der User $\mathcal{U}$ analog zum Issue-Protokoll beweisen, dass er das Polynom $\phi(\mathsf{x})$ mit $V = u_0^{\phi(\alpha)}$ kennt. Dazu genügt es zu beweisen, dass $\mathcal{U}$ das Polynom $\phi_1(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ mit $\hat{W}_1 = u_0^{\phi_1(\alpha)}$ kennt:

$$\hat{e}\left(\hat{W}_1, h\right) = \hat{e}\left(W_{1,l}, h_1 h^{-l}\right) \hat{e}\left(u_0^{\phi_1(l)}, h\right). \tag{3.13}$$

Dies ist für die Beweiskraft wichtig, da auch hier der User K-mal zurückgespult wird, damit der Extractor das Polynom $\phi_1(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ durch Interpolation bestimmen und anschließend das Polynom $\phi(\mathsf{x}) = \phi_1(\mathsf{x}) \cdot (\mathsf{x} + n_1)$ mit $V = u_0^{\phi(\alpha)}$ berechnen kann.

Die drei obigen Gleichungen 3.11 bis 3.13 können jedoch zu einer einzigen Gleichung zusammengefasst werden. Denn die beiden Gleichungen 3.11 und 3.12 können zu folgender Gleichung kombiniert werden:

$$\hat{e}(\Sigma, Xh^e) = \hat{e}(g g_0^s g_1^{m_1} \ldots g_L^{m_L}, h)\, \hat{e}\left(\hat{W}_1, h_1 h^{n_1}\right). \tag{3.14}$$

Weiter ist die Gleichung 3.13 äquivalent zu

$$\hat{e}\left(\hat{W}_1, h_1 h^{n_1}\right) = \hat{e}\left(W_{1,l}, h_2 h_1^{n_1 - l} h^{-n_1 l}\right) \hat{e}\left(u_0^{\phi_1(l)}, h_1 h^{n_1}\right). \tag{3.15}$$

Somit können schließlich die beiden Gleichungen 3.14 und 3.15 zu folgender kombiniert werden:

$$\hat{e}(\Sigma, Xh^e) = \hat{e}(g g_0^s g_1^{m_1} \ldots g_L^{m_L}, h)\, \hat{e}\left(W_{1,l}, h_2 h_1^{n_1 - l} h^{-n_1 l}\right) \hat{e}\left(u_0^{\phi_1(l)}, h_1 h^{n_1}\right). \tag{3.16}$$

Prove-k: Das Prove-k-Protokoll ist ebenfalls ein Zero-Knowledge-Proof-of-Knowledge-Protokoll. Allerdings wird hier im Gegensatz zu Prove gezeigt, dass sich mehrere bekannte Werte (o.B.d.A. seien dies) $n_1, \ldots, n_k$ für $1 \leq k \leq K - 1$ in einem geheimen Akkumulator V befinden. Analog zu den Gleichungen 3.11 bis 3.13 müssen die folgenden Gleichungen bewiesen werden:

$$\hat{e}(V, h) = \hat{e}\left(\hat{W}_I, h^{\prod_{j=1}^{k}(\alpha + n_j)}\right),$$

$$\hat{e}\left(\Sigma, Xh^{e}\right) = \hat{e}\left(gg_0^s g_1^{m_1} \ldots g_L^{m_L} V, h\right),$$
$$\hat{e}\left(\hat{W}_I, h\right) = \hat{e}\left(W_{I,l}, h_1 h^{-l}\right) \hat{e}\left(u_0^{\phi_I(l)}, h\right).$$

Wie beim Prove-Protokoll können diese drei Gleichungen zu einer einzigen Gleichung kombiniert werden:

$$\hat{e}\left(\Sigma, Xh^{e}\right) = \hat{e}\left(gg_0^s g_1^{m_1} \ldots g_L^{m_L}, h\right) \hat{e}\left(W_{I,l}, h^{(\alpha-l)\prod_{j=1}^{k}(\alpha+n_j)}\right) \hat{e}\left(u_0^{\phi_I(l)}, h^{\prod_{j=1}^{k}(\alpha+n_j)}\right). \tag{3.17}$$

Da $k \leq K-1$ ist, kann das Element $h^{(\alpha-l)\prod_{j=1}^{k}(\alpha+n_j)}$ mit den öffentlichen Generatoren $h, h_1, \ldots, h_K$ effizient berechnet werden.

Beachte, dass die Gleichung 3.16 ein Sonderfall der Gleichung 3.17 für $k = 1$ ist. Es werden allerdings beide Protokolle benötigt, da der Wert n_1 im Prove-Protokoll geheim bleibt, während er für das Prove-k-Protokoll bekannt sein muss.

Prove-K: Das Prove-K-Protokoll ist das Zero-Knowledge-Proof-of-Knowledge-Protokoll, mit dem bewiesen wird, dass sich alle Werte $n_1, \ldots, n_K$ in einem Akkumulator V befinden:

$$\hat{e}\left(\Sigma, Xh^{e}\right) = \hat{e}\left(gg_0^s g_1^{m_1} \ldots g_L^{m_L} u_0^{\prod_{j=1}^{K}(\alpha+n_j)}, h\right).$$

Da für den Zeugen $\hat{W}_I = u_0$ gilt und das Polynom $\phi(\mathsf{x}) = \prod_{j=1}^{K}(\mathsf{x}+n_j)$ mit $V = u_0^{\phi(\alpha)}$ vollständig bekannt ist, ist diese Gleichung ausreichend.

Somit ist das Prove-Protokoll für kompakte elektronische Geldsysteme (wie in Kapitel 5) und das Prove-k bzw. Prove-K-Protokoll für teilbare elektronische Geldsysteme (wie in Kapitel 6) geeignet.

Nun muss gezeigt werden, dass durch die obige Gleichung 3.17 weiterhin die entsprechenden Sicherheitseigenschaften Berechnungsgebundenheit und Beschränktheit des Commitment-Schemas erfüllt sind. Dazu unterscheiden wir die beiden Fälle, in denen die Zahlen $n_1, \ldots, n_K \in \mathbb{Z}_p \setminus \{-\alpha\}$ (1) beliebig vom User gewählt werden oder (2) die Ausgabe einer Hashfunktion sind. Der zweite Fall führt zu einem effizienteren Protokoll und trifft auf die teilbaren elektronischen Geldsysteme in Kapitel 6 bis 9 sowie in [ASM08] zu. Die Sicherheit ist dann jedoch nur im Random-Oracle-Modell beweisbar.

Fall 1 (Allgemeines Verfahren): Im ersten Fall ist es notwendig, dass der User im Prove- bzw. Prove-k-Protokoll an die Zahlen $e, s, m_1, \ldots, m_L \in \mathbb{Z}_p$ gebunden ist (beispielsweise durch die Berechnung eines Pedersen-Commitments). Dann folgt direkt, dass die entsprechende Version der Berechnungsgebundenheit weiterhin erfüllt ist.

Lemma 3.2.1. *Angenommen, es gilt die q-SDH-Annahme für* $\mathsf{Setup_{Bil}}$*. Dann gibt es für jeden polynomiellen Angreifer* $\mathcal{A}$ *eine vernachlässigbare Funktion* ν*, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *und für* $k < K = q$ *gilt:*

$$\begin{aligned}
&\Pr\Big[(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{GenSign}\left(1^{\lambda}, L, K\right); \\
&\left(\Sigma, e, s, m_1, \ldots, m_L, n_1, \ldots, n_k, l, \phi_I(l), W_{I,l}, \phi_I(l)', W'_{I,l}\right) \leftarrow \mathcal{A}(\mathsf{pk}) \ : \\
&\hat{e}(\Sigma, Xh^e) = \hat{e}\left(gg_0^s g_1^{m_1} \ldots g_L^{m_L}, h\right) \hat{e}\left(W_{I,l}, h^{(\alpha-l)\prod_{j=1}^{k}(\alpha+n_j)}\right) \hat{e}\left(u_0^{\phi_I(l)}, h^{\prod_{j=1}^{k}(\alpha+n_j)}\right) \\
&= \hat{e}\left(gg_0^s g_1^{m_1} \ldots g_L^{m_L}, h\right) \hat{e}\left(W'_{I,l}, h^{(\alpha-l)\prod_{j=1}^{k}(\alpha+n_j)}\right) \hat{e}\left(u_0^{\phi_I(l)'}, h^{\prod_{j=1}^{k}(\alpha+n_j)}\right) \\
&\wedge \phi_I(l) \neq \phi_I(l)'\Big] \leq \nu(\lambda).
\end{aligned}$$

Beweis. Angenommen, es existiert ein polynomieller Angreifer $\mathcal{A}$, der mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ ein Tupel mit den Eigenschaften wie im obigen Lemma 3.2.1 ausgibt. Wir konstruieren einen polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das q-SDH-Problem mit $q = K$ zu lösen.

Systemparameter: Der Angreifer $\mathcal{B}$ erhält eine zufällige q-SDH Probleminstanz $(p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \hat{e}, u_0, \ldots, u_K, h, h_1, \ldots, h_K)$. Dann wählt $\mathcal{B}$ zufällig ein Generatortupel $(g, g_0, \ldots, g_L) \in_{\mathcal{R}} \mathbb{G}_1^{L+2}$ sowie eine Zahl $x \in_{\mathcal{R}} \mathbb{Z}_p^*$, berechnet den öffentlichen Signaturschlüssel $X = h^x$ und sendet den perfekt simulierten öffentlichen Schlüssel $\mathsf{pk} = (p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \hat{e}, g, g_0, \ldots, g_L, u_0, \ldots, u_K, h, h_1, \ldots, h_K, X)$ an den Angreifer $\mathcal{A}$.

Ausgabe: Der Angreifer $\mathcal{A}$ gibt nun ein Tupel $\left(\Sigma, e, s, m_1, \ldots, m_L, n_1, \ldots, n_k, l, \phi_I(l), W_{I,l}, \phi_I(l)', W'_{I,l}\right)$ für $1 \leq k \leq K-1$ mit $\phi_I(l) \neq \phi_I(l)'$ und den folgenden Eigenschaften aus:

$$\begin{aligned}
\hat{e}(\Sigma, Xh^e) &= \hat{e}\left(gg_0^s g_1^{m_1} \ldots g_L^{m_L}, h\right) \hat{e}\left(W_{I,l}, h^{(\alpha-l)\prod_{j=1}^{k}(\alpha+n_j)}\right) \hat{e}\left(u_0^{\phi_I(l)}, h^{\prod_{j=1}^{k}(\alpha+n_j)}\right) \\
&= \hat{e}\left(gg_0^s g_1^{m_1} \ldots g_L^{m_L}, h\right) \hat{e}\left(W'_{I,l}, h^{(\alpha-l)\prod_{j=1}^{k}(\alpha+n_j)}\right) \hat{e}\left(u_0^{\phi_I(l)'}, h^{\prod_{j=1}^{k}(\alpha+n_j)}\right) \\
&\Leftrightarrow \hat{e}\left(gg_0^s g_1^{m_1} \ldots g_L^{m_L}, h\right) \left(\hat{e}\left(W_{I,l}, h^{\alpha-l}\right) \hat{e}\left(u_0^{\phi_I(l)}, h\right)\right)^{\prod_{j=1}^{k}(\alpha+n_j)} \\
&= \hat{e}\left(gg_0^s g_1^{m_1} \ldots g_L^{m_L}, h\right) \left(\hat{e}\left(W'_{I,l}, h^{\alpha-l}\right) \hat{e}\left(u_0^{\phi_I(l)'}, h\right)\right)^{\prod_{j=1}^{k}(\alpha+n_j)} \\
&\Leftrightarrow \hat{e}\left(W_{I,l}, h^{\alpha-l}\right) \hat{e}\left(u_0^{\phi_I(l)}, h\right) = \hat{e}\left(W'_{I,l}, h^{\alpha-l}\right) \hat{e}\left(u_0^{\phi_I(l)'}, h\right),
\end{aligned}$$

da wir $n_j \neq -\alpha$ für alle $1 \leq j \leq k$ sowie $l \neq \alpha$ voraussetzen können, da $\mathcal{B}$ andernfalls direkt das q-SDH-Problem löst. Dann gilt wie in [KZG10a, KZG10b]:

$$\left(\frac{W_{I,l}}{W'_{I,l}}\right)^{\frac{1}{\phi_I(l)'-\phi_I(l)}} = u_0^{\frac{1}{\alpha-l}}.$$

Somit löst $\mathcal{B}$ das q-SDH-Problem mit derselben nicht vernachlässigbaren Wahrscheinlichkeit $\epsilon(\lambda)$. □

Fall 2 (Spezielles Verfahren im Random-Oracle-Modell): Als erstes muss gezeigt werden, dass die Eigenschaft Berechnungsgebundenheit weiterhin erfüllt ist. Jedoch wird dies lediglich in einer schwächeren Variante bewiesen, die für die Sicherheit des Signaturverfahrens allerdings immer noch ausreichend ist. In der schwächeren Variante darf der Angreifer $\mathcal{A}$ die Zahl $l \in \mathbb{Z}_p$ nicht selbst wählen, sondern diese Zahl ist ebenfalls die Ausgabe einer Hashfunktion (was auch für das Signaturverfahren und die folgenden teilbaren elektronischen Geldsysteme in Kapitel 6 bis Kapitel 9 zutrifft).

Lemma 3.2.2. *Angenommen, es gilt die q-SDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *und die zu akkumulierenden Werte* $n_1, \ldots, n_K$ *sowie die Zahl* l *sind die Ausgaben einer Hashfunktion, wobei sich eine Anfrage, um einen Hashwert* n_i *für* $1 \leq i \leq K$ *zu erhalten, von einer Anfrage, um den Hashwert* l *zu erhalten, unterscheidet (z.B. durch unterschiedliche Eingabelängen). Dann gibt es im Random-Oracle-Modell für jeden polynomiellen Angreifer* $\mathcal{A}$*, der* q_H *Orakel-Anfragen stellt, eine vernachlässigbare Funktion* ν*, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *und für* $k < K$ *sowie* $q = \max\{q_H, K\}$ *gilt:*

$$\begin{aligned}
&\Pr\Big[(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{GenSign}\left(1^\lambda, L, K\right); \\
&\left(\Sigma, n_1, \ldots, n_k, l, (e, s, m_1, \ldots, m_L, \phi_I(l), W_{I,l}), \left(e', s', m'_1, \ldots, m'_L, \phi_I(l)', W'_{I,l}\right)\right) \\
&\leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{H}}}(\mathsf{pk}) \ : \\
&\hat{e}(\Sigma, Xh^e) = \hat{e}\left(gg_0^s g_1^{m_1} \ldots g_L^{m_L}, h\right) \hat{e}\left(W_{I,l}, h^{(\alpha-l)\prod_{j=1}^{k}(\alpha+n_j)}\right) \hat{e}\left(u_0^{\phi_I(l)}, h^{\prod_{j=1}^{k}(\alpha+n_j)}\right) \ \wedge \\
&\hat{e}\left(\Sigma, Xh^{e'}\right) = \hat{e}\left(gg_0^{s'} g_1^{m'_1} \ldots g_L^{m'_L}, h\right) \hat{e}\left(W'_{I,l}, h^{(\alpha-l)\prod_{j=1}^{k}(\alpha+n_j)}\right) \hat{e}\left(u_0^{\phi_I(l)'}, h^{\prod_{j=1}^{k}(\alpha+n_j)}\right) \\
&\wedge \ (e, s, m_1, \ldots, m_L, \phi_I(l)) \neq \left(e', s', m'_1, \ldots, m'_L, \phi_I(l)'\right)\Big] \leq \nu(\lambda).
\end{aligned}$$

Beweis. Angenommen, es existiert ein polynomieller Angreifer $\mathcal{A}$, der mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ ein Tupel mit den Eigenschaften wie im obigen Lemma 3.2.2 ausgibt. Seien q_{H_n} bzw. q_{H_l} die Anzahl der Anfragen von $\mathcal{A}$ an das Random-Orakel $\mathcal{O}^{\mathsf{H}}$, um einen Hashwert n_j bzw. l_i zu erhalten. Der Angreifer $\mathcal{B}$ wählt zufällig q_{H_n} Zahlen $n_1, \ldots, n_{q_{H_n}} \in_{\mathcal{R}} \mathbb{Z}_p$ sowie q_{H_l} Zahlen $l_1, \ldots, l_{q_{H_l}} \in_{\mathcal{R}} \mathbb{Z}_p$ als Ausgabe des Random-Oracles. Falls $n_j = -\alpha$ oder $l_i = \alpha$ für ein $1 \leq j \leq q_{H_n}$ bzw. $1 \leq i \leq q_{H_l}$ ist, kann $\mathcal{B}$ sofort das q-SDH-Problem lösen. Daher sei nun $n_j \neq -\alpha$ und $l_i \neq \alpha$ für alle $1 \leq j \leq q_{H_n}$ und $1 \leq i \leq q_{H_l}$. Seien o.B.d.A. die ersten k Zahlen $n_1, \ldots, n_k$ sowie $l_1 =: l$ die Ausgabe von $\mathcal{A}$. (Wir können davon ausgehen, dass $\mathcal{A}$ nur Werte ausgibt, die er als Antwort auf eine Orakel-Anfrage erhalten hat. Ansonsten müsste $\mathcal{A}$ die vom Random-Oracle zufällig gewählte Ausgabe raten, was nur mit vernachlässigbarer Wahrscheinlichkeit möglich ist.)

Nach der Ausgabe gilt für $h_{l,I} := h^{(\alpha-l)\prod_{j=1}^{k}(\alpha+n_j)}$ und $h_I := h^{\prod_{j=1}^{k}(\alpha+n_j)}$:

$$\begin{aligned}\hat{e}(\Sigma, h) &= \hat{e}\left(g^{\frac{1}{x+e}} g_0^{\frac{s}{x+e}} g_1^{\frac{m_1}{x+e}} \dots g_L^{\frac{m_L}{x+e}}, h\right) \hat{e}\left(W_{I,l}^{\frac{1}{x+e}}, h_{l,I}\right) \hat{e}\left(u_0^{\frac{\phi_I(l)}{x+e}}, h_I\right) \\ &= \hat{e}\left(g^{\frac{1}{x+e'}} g_0^{\frac{s'}{x+e'}} g_1^{\frac{m'_1}{x+e'}} \dots g_L^{\frac{m'_L}{x+e'}}, h\right) \hat{e}\left(W_{I,l}'^{\frac{1}{x+e'}}, h_{l,I}\right) \hat{e}\left(u_0^{\frac{\phi_I(l)'}{x+e'}}, h_I\right).\end{aligned}$$

Dann müssen wir die beiden Fälle (1) $\frac{\phi_I(l)}{x+e} \neq \frac{\phi_I(l)'}{x+e'}$ und (2) $\frac{\phi_I(l)}{x+e} = \frac{\phi_I(l)'}{x+e'}$ unterscheiden.

Fall 1: Wir konstruieren einen polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das q-SDH-Problem mit $q_{H_n} + q_{H_l} = q_H$ und $q = \max\{q_H, K\}$ zu lösen (üblicherweise ist $q_H \gg K$).

Systemparameter: Der Angreifer $\mathcal{B}$ erhält eine zufällige q-SDH Probleminstanz $(p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, \dots, u_q, h, h_1, \dots, h_q)$. Dann wählt $\mathcal{B}$ zufällig $L+2$ paarweise verschiedene Zahlen $\delta, \delta_0, \dots, \delta_L \in_{\mathcal{R}} \mathbb{Z}_p^*$ sowie eine Zahl $x \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet den Generator $g = u_0^{\delta \prod_{i=1}^{q_{H_l}}(\alpha-l_i) \prod_{j=1}^{q_{H_n}}(\alpha+n_j)}$, die Generatoren $g_0 = g^{\delta_0}, \dots, g_L = g^{\delta_L}$ und das Element $X = h^x$. Anschließend sendet $\mathcal{B}$ den perfekt simulierten öffentlichen Schlüssel $\mathsf{pk} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_0, \dots, g_L, u_0, \dots, u_K, h, h_1, \dots, h_K, X)$ an den Angreifer $\mathcal{A}$. Beachte, dass $K \leq q$ ist.

Ausgabe: Der Angreifer $\mathcal{A}$ gibt nun obiges Tupel aus und es gilt:

$$\begin{aligned}&\hat{e}\left(\left(g^{\frac{1}{x+e}} g_0^{\frac{s}{x+e}} g_1^{\frac{m_1}{x+e}} \dots g_L^{\frac{m_L}{x+e}}\right)^{\frac{1}{\prod_{j=1}^{k}(\alpha+n_j)}}, h\right) \hat{e}\left(W_{I,l}^{\frac{1}{x+e}}, h_1 h^{-l}\right) \hat{e}\left(u_0^{\frac{\phi_I(l)}{x+e}}, h\right) \\ &= \hat{e}\left(\left(g^{\frac{1}{x+e'}} g_0^{\frac{s'}{x+e'}} g_1^{\frac{m'_1}{x+e'}} \dots g_L^{\frac{m'_L}{x+e'}}\right)^{\frac{1}{\prod_{j=1}^{k}(\alpha+n_j)}}, h\right) \hat{e}\left(W_{I,l}'^{\frac{1}{x+e'}}, h_1 h^{-l}\right) \hat{e}\left(u_0^{\frac{\phi_I(l)'}{x+e'}}, h\right) \\ &\Leftrightarrow \hat{e}\left(u_0^{\frac{\delta\left(1+\delta_0 s+\delta_1 m_1+\dots+\delta_L m_L\right)\prod_{i=1}^{q_{H_l}}(\alpha-l_i)\prod_{j=k+1}^{q_{H_n}}\left(\alpha+n_j\right)}{x+e}}, h\right) \hat{e}\left(W_{I,l}^{\frac{1}{x+e}}, h_1 h^{-l}\right) \\ &\quad \hat{e}\left(u_0^{\frac{\phi_I(l)}{x+e}}, h\right) \\ &= \hat{e}\left(u_0^{\frac{\delta\left(1+\delta_0 s'+\delta_1 m'_1+\dots+\delta_L m'_L\right)\prod_{i=1}^{q_{H_l}}(\alpha-l_i)\prod_{j=k+1}^{q_{H_n}}\left(\alpha+n_j\right)}{x+e'}}, h\right) \hat{e}\left(W_{I,l}'^{\frac{1}{x+e'}}, h_1 h^{-l}\right) \\ &\quad \hat{e}\left(u_0^{\frac{\phi_I(l)'}{x+e'}}, h\right).\end{aligned}$$

Mit (o.B.d.A.) $l := l_1$ folgt weiter:

$$\left(u_0^{\frac{\delta\left(1+\delta_0 s+\delta_1 m_1+\cdots+\delta_L m_L\right)\prod_{i=2}^{q_{H_l}}(\alpha-l_i)\prod_{j=k+1}^{q_{H_n}}\left(\alpha+n_j\right)}{x+e}} W_{I,l}^{\frac{1}{x+e}}\right)^{\alpha-l} u_0^{\frac{\phi_I(l)}{x+e}}$$
$$= \left(u_0^{\frac{\delta\left(1+\delta_0 s'+\delta_1 m_1'+\cdots+\delta_L m_L'\right)\prod_{i=2}^{q_{H_l}}(\alpha-l_i)\prod_{j=k+1}^{q_{H_n}}\left(\alpha+n_j\right)}{x+e'}} W_{I,l}'^{\frac{1}{x+e'}}\right)^{\alpha-l} u_0^{\frac{\phi_I(l)'}{x+e'}}.$$

Somit ist

$$\left(\frac{u_0^{\frac{\delta\left(1+\delta_0 s+\delta_1 m_1+\cdots+\delta_L m_L\right)\prod_{i=2}^{q_{H_l}}(\alpha-l_i)\prod_{j=k+1}^{q_{H_n}}\left(\alpha+n_j\right)}{x+e}} W_{I,l}^{\frac{1}{x+e}}}{u_0^{\frac{\delta\left(1+\delta_0 s'+\delta_1 m_1'+\cdots+\delta_L m_L'\right)\prod_{i=2}^{q_{H_l}}(\alpha-l_i)\prod_{j=k+1}^{q_{H_n}}\left(\alpha+n_j\right)}{x+e'}} W_{I,l}'^{\frac{1}{x+e'}}}\right)^{\frac{1}{\frac{\phi_I(l)'}{x+e'}-\frac{\phi_I(l)}{x+e}}} = u_0^{\frac{1}{\alpha-l}}.$$

Folglich löst der Angreifer $\mathcal{B}$ das q-SDH-Problem mit derselben nicht vernachlässigbaren Wahrscheinlichkeit $\epsilon(\lambda)$.

Fall 2: Wir konstruieren einen polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ erneut dazu verwendet, das q-SDH-Problem mit $q_{H_n} + q_{H_l} = q_H$ und $q = \max\{q_H, K\}$ zu lösen. Allerdings wählt $\mathcal{B}$ zufällig q_{H_n} Zahlen $n_1, \ldots, n_{q_{H_n}} \in_{\mathcal{R}} \mathbb{Z}_p$ sowie q_{H_l} Zahlen $l_1, \ldots, l_{q_{H_l}} \in_{\mathcal{R}} \mathbb{Z}_p \setminus \{-n_1, \ldots, -n_{q_{H_n}}\}$ als Ausgabe des Random-Oracles (an dieser Stelle wird benötigt, dass sich die Anfragen unterscheiden). Wie im ersten Fall, kann wieder $n_j \neq -\alpha$ für alle $1 \leq j \leq q_{H_n}$ sowie $l_i \neq \alpha$ für alle $1 \leq i \leq q_{H_l}$ vorausgesetzt werden. Seien erneut o.B.d.A. die ersten k Zahlen $n_1, \ldots, n_k$ sowie $l_1 =: l$ die Ausgabe von $\mathcal{A}$. Die Zahlen $l_1, \ldots, l_{q_{H_l}}$ sind im Random-Oracle-Modell rechnerisch ununterscheidbar simuliert.

Systemparameter: Der Angreifer $\mathcal{B}$ erhält eine zufällige q-SDH Probleminstanz $(p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, \ldots, u_q, h, h_1, \ldots, h_q)$. Dann wählt $\mathcal{B}$ zufällig $L+2$ paarweise verschiedene Zahlen $\delta, \delta_0, \ldots, \delta_L \in_{\mathcal{R}} \mathbb{Z}_p^*$ sowie eine Zahl $x \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet den Generator $g = u_0^{\delta \prod_{j=1}^{q_{H_n}}(\alpha+n_j)}$, die Generatoren $g_0 = g^{\delta_0}, \ldots, g_L = g^{\delta_L}$ und das Element $X = h^x$. Anschließend sendet $\mathcal{B}$ den perfekt simulierten öffentlichen Schlüssel $\mathsf{pk} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_0, \ldots, g_L, u_0, \ldots, u_K, h, h_1, \ldots, h_K, X)$ an den Angreifer $\mathcal{A}$. Beachte, dass $K \leq q$ ist.

Ausgabe: Der Angreifer $\mathcal{A}$ gibt nun obiges Tupel aus und da nach Voraussetzung $\frac{\phi_I(l)}{x+e} = \frac{\phi_I(l)'}{x+e'}$ ist, folgt:

$$\hat{e}\left(\left(g^{\frac{1}{x+e}}g_0^{\frac{s}{x+e}}g_1^{\frac{m_1}{x+e}}\ldots g_L^{\frac{m_L}{x+e}}\right)^{\frac{1}{\prod_{j=1}^{k}\left(\alpha+n_j\right)}},h\right)\hat{e}\left(W_{I,l}^{\frac{1}{x+e}},h_1h^{-l}\right)\hat{e}\left(u_0^{\frac{\phi_I(l)}{x+e}},h\right)$$
$$=\hat{e}\left(\left(g^{\frac{1}{x+e'}}g_0^{\frac{s'}{x+e'}}g_1^{\frac{m_1'}{x+e'}}\ldots g_L^{\frac{m_L'}{x+e'}}\right)^{\frac{1}{\prod_{j=1}^{k}\left(\alpha+n_j\right)}},h\right)\hat{e}\left(W_{I,l}'^{\frac{1}{x+e'}},h_1h^{-l}\right)\hat{e}\left(u_0^{\frac{\phi_I(l)'}{x+e'}},h\right)$$
$$\Leftrightarrow\hat{e}\left(u_0^{\frac{\delta\left(1+\delta_0s+\delta_1m_1+\cdots+\delta_Lm_L\right)\prod_{j=k+1}^{q_{H_n}}\left(\alpha+n_j\right)}{x+e}},h\right)\hat{e}\left(W_{I,l}^{\frac{1}{x+e}},h_1h^{-l}\right)$$
$$=\hat{e}\left(u_0^{\frac{\delta\left(1+\delta_0s'+\delta_1m_1'+\cdots+\delta_Lm_L'\right)\prod_{j=k+1}^{q_{H_n}}\left(\alpha+n_j\right)}{x+e'}},h\right)\hat{e}\left(W_{I,l}'^{\frac{1}{x+e'}},h_1h^{-l}\right).$$

Seien $\varphi(\mathsf{x})\in\mathbb{Z}_p[\mathsf{x}]$ das Polynom und $R\in\mathbb{Z}_p^*$ die Zahl mit $\prod_{j=k+1}^{q_{H_n}}(\mathsf{x}+n_j)=\varphi(\mathsf{x})(\mathsf{x}-l)+R$, denn da $l_i\neq -n_j$ für alle $1\leq i\leq q_{H_l}$ und $1\leq j\leq q_{H_n}$ gilt, ist das Polynom $\prod_{j=k+1}^{q_{H_n}}(\mathsf{x}+n_j)\in\mathbb{Z}_p[\mathsf{x}]$ nicht ohne Rest durch das Polynom $\mathsf{x}-l$ teilbar. Seien weiter $z,z'\in\mathbb{Z}_p$ zwei Zahlen mit $z:=\frac{\delta(1+\delta_0s+\delta_1m_1+\cdots+\delta_Lm_L)}{x+e}$ und $z':=\frac{\delta\left(1+\delta_0s'+\delta_1m_1'+\cdots+\delta_Lm_L'\right)}{x+e'}$. Dann ist nur mit vernachlässigbarer Wahrscheinlichkeit $z=z'$, da $\mathcal{A}$ ansonsten die Gebundenheit des Pedersen-Commitment-Schemas gebrochen hätte:

Angenommen, es gilt: $z=z'$. Dann gilt für das Commitment $C:=g^{\frac{1}{x+e}}g_0^{\frac{s}{x+e}}g_1^{\frac{m_1}{x+e}}\ldots g_L^{\frac{m_L}{x+e}}=g^{\frac{1}{x+e'}}g_0^{\frac{s'}{x+e'}}g_1^{\frac{m_1'}{x+e'}}\ldots g_L^{\frac{m_L'}{x+e'}}$. Wegen $(e,s,m_1,\ldots,m_L,\phi_I(l))\neq\left(e',s',m_1',\ldots,m_L',\phi_I(l)'\right)$ und $\frac{\phi_I(l)}{x+e}=\frac{\phi_I(l)'}{x+e'}$ ist auch $(e,s,m_1,\ldots,m_L)\neq(e',s',m_1',\ldots,m_L')$. Falls $e\neq e'$ ist, handelt es sich um zwei verschiedene Darstellungen von C bzgl. $(g,g_0,\ldots,g_L)$. Falls $e=e'$ ist, folgt $(s,m_1,\ldots,m_L)\neq(s',m_1',\ldots,m_L')$. Somit handelt es sich erneut um zwei verschiedene Darstellungen von C bzgl. $(g,g_0,\ldots,g_L)$.

Mit den obigen Definitionen von $\varphi(\mathsf{x}),R,z$ und z' gilt:

$$u_0^{(\varphi(\alpha)(\alpha-l)+R)z}W_{I,l}^{\frac{\alpha-l}{x+e}}=u_0^{(\varphi(\alpha)(\alpha-l)+R)z'}W_{I,l}'^{\frac{\alpha-l}{x+e'}}.$$

Daraus folgt:

$$\left(\frac{W_{I,l}^{\frac{1}{x+e}}u_0^{\varphi(\alpha)z}}{W_{I,l}'^{\frac{1}{x+e'}}u_0^{\varphi(\alpha)z'}}\right)^{\frac{1}{R(z'-z)}}=u_0^{\frac{1}{\alpha-l}}.$$

Somit löst der Angreifer $\mathcal{B}$ das q-SDH-Problem mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)-\nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist. □

Nun kann die analoge Version der starken Berechnungsgebundenheit bewiesen werden. (Diese muss für den Fall 1 nicht gezeigt werden, da dort ja verlangt wird, dass sich $\mathcal{A}$ zusätzlich an die Zahlen $e,s,m_1,\ldots,m_L\in\mathbb{Z}_p$ binden muss.)

Lemma 3.2.3. *Angenommen, es gelten die q-SDH-Annahme und die q-polyDH-Annahme für* $\mathsf{Setup_{Bil}}$. *Dann gibt es für jeden polynomiellen Angreifer* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ *eine vernachlässigbare Funktion* ν, *so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *und für* $k < K = q$ *gilt:*

$$\Pr\Big[(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{GenSign}\left(1^\lambda, L, K\right);\ (\Sigma, \mathsf{state}) \leftarrow \mathcal{A}_0\left(\mathsf{pk}\right);\ i, i' \in_{\mathcal{R}} \mathbb{Z}_p;$$
$$\Big((e, s, m_1, \ldots, m_L, i, \phi_I\left(i\right), W_{I,i}), \Big(e', s', m_1', \ldots, m_L', i', \phi_I'\left(i'\right), W_{I,i'}'\Big)\Big) \leftarrow \mathcal{A}_1\left(\mathsf{pk}, \Sigma, \mathsf{state}, i, i'\right):$$
$$\hat{e}\left(\Sigma, Xh^e\right) = \hat{e}\left(gg_0^s g_1^{m_1} \cdots g_L^{m_L}, h\right) \hat{e}\left(W_{I,i}, h^{(\alpha-i)\prod_{l=1}^{k}(\alpha+n_l)}\right) \hat{e}\left(u_0^{\phi_I(i)}, h^{\prod_{l=1}^{k}(\alpha+n_l)}\right)$$
$$\hat{e}\left(\Sigma, Xh^{e'}\right) = \hat{e}\left(gg_0^{s'} g_1^{m_1'} \cdots g_L^{m_L'}, h\right) \hat{e}\left(W_{I,i'}', h^{(\alpha-i')\prod_{l=1}^{k}(\alpha+n_l)}\right) \hat{e}\left(u_0^{\phi_I'(i')}, h^{\prod_{l=1}^{k}(\alpha+n_l)}\right)$$
$$\wedge\ (e, s, m_1, \ldots, m_L) \neq (e', s', m_1', \ldots, m_L')\Big] \leq \nu(\lambda).$$

Beweis. Angenommen, es existiert ein polynomieller Angreifer $\mathcal{A}$, der mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon\left(\lambda\right)$ ein Element $\Sigma \in \mathbb{G}_1$ und zu zwei Zufallszahlen $i, i' \in_{\mathcal{R}} \mathbb{Z}_p$ ein Tupel $\Big((e, s, m_1, \ldots, m_L, i, \phi_I\left(i\right), W_{I,i}), \Big(e', s', m_1', \ldots, m_L', i', \phi_I'\left(i'\right), W_{I,i'}'\Big)\Big)$ mit den Eigenschaften wie in Lemma 3.2.3 ausgibt.

Selbstverständlich kennt der Angreifer $\mathcal{A}$ die geheime Zahl $x \in \mathbb{Z}_p^*$ nur mit vernachlässigbarer Wahrscheinlichkeit. Wir gehen nun aber davon aus, dass der Angreifer $\mathcal{A}$ die Zahl x kennt.

Dann hat $\mathcal{A}$ bzgl. des Commitments $C^\star := \left(\Sigma^{x+e} g^{-1} g_0^{-s} g_1^{-m_1} \cdots g_L^{-m_L}\right)^{\frac{1}{\prod_{l=1}^{k}(\alpha+n_l)}} \in \mathbb{G}_1$ zur Zahl $i \in_{\mathcal{R}} \mathbb{Z}_p$ ein Tripel $\left(i, \phi_I\left(i\right), W_{I,i}\right)$ mit $\hat{e}\left(C^\star, h\right) = \hat{e}\left(W_{I,i}, h_1 h^{-i}\right) \hat{e}\left(u_0^{\phi_I(i)}, h\right)$ ausgegeben. Mit Korollar 3.1.2 folgt, dass $\mathcal{A}$ ein Polynom $\phi_I(\mathsf{x}) \in \mathbb{Z}_p\left[\mathsf{x}\right]$ mit $\deg\left(\phi_I\right) \leq K$ und $C^\star = u_0^{\phi_I(\alpha)}$ kennt.

Analog kennt $\mathcal{A}$ bzgl. $C'^\star := \left(\Sigma^{x+e'} g^{-1} g_0^{-s'} g_1^{-m_1'} \cdots g_L^{-m_L'}\right)^{\frac{1}{\prod_{l=1}^{k}(\alpha+n_l)}} \in \mathbb{G}_1$ ein Polynom $\phi_I'(\mathsf{x}) \in \mathbb{Z}_p\left[\mathsf{x}\right]$ mit $\deg\left(\phi_I'\right) \leq K$ und $\hat{e}\left(C'^\star, h\right) = \hat{e}\left(W_{I,i'}', h_1 h^{-i'}\right) \hat{e}\left(u_0^{\phi_I'(i')}, h\right)$ sowie $C'^\star = u_0^{\phi_I'(\alpha)}$.

Seien $\phi(\mathsf{x}), \phi'(\mathsf{x}) \in \mathbb{Z}_p\left[\mathsf{x}\right]$ die beiden Polynome mit $\phi(\mathsf{x}) := \phi_I(\mathsf{x}) \prod_{l=1}^{k}\left(\mathsf{x} + n_l\right)$ und $\phi'(\mathsf{x}) := \phi_I'(\mathsf{x}) \prod_{l=1}^{k}\left(\mathsf{x} + n_l\right)$.

Insgesamt kennt $\mathcal{A}$ also mit nicht vernachlässigbarer Wahrscheinlichkeit die beiden verschiedenen Tupel $\left(\frac{1}{x+e}, \frac{s}{x+e}, \frac{m_1}{x+e}, \ldots, \frac{m_L}{x+e}, \frac{\phi(\mathsf{x})}{x+e}\right) \in \mathbb{Z}_p^{L+2} \times \mathbb{Z}_p\left[\mathsf{x}\right]$ und $\left(\frac{1}{x+e'}, \frac{s'}{x+e'}, \frac{m_1'}{x+e'}, \ldots, \frac{m_L'}{x+e'}, \frac{\phi'(\mathsf{x})}{x+e'}\right) \in \mathbb{Z}_p^{L+2} \times \mathbb{Z}_p\left[\mathsf{x}\right]$ mit $\deg(\phi), \deg(\phi') \leq K + k =: K' < 2K$ für eine polynomiell beschränkte Zahl $K' \in \mathbb{N}$ und

$$\Sigma = g^{\frac{1}{x+e}} g_0^{\frac{s}{x+e}} g_1^{\frac{m_1}{x+e}} \cdots g_L^{\frac{m_L}{x+e}} u_0^{\frac{\phi(\alpha)}{x+e}} = g^{\frac{1}{x+e'}} g_0^{\frac{s'}{x+e'}} g_1^{\frac{m_1'}{x+e'}} \cdots g_L^{\frac{m_L'}{x+e'}} u_0^{\frac{\phi'(\alpha)}{x+e'}}.$$

Dies ist allerdings ein Widerspruch zur Gebundenheit des Commitment-Schemas gemäß Lemma 3.1.2.

Somit gilt obiges Lemma 3.2.3 sogar dann, falls der Angreifer $\mathcal{A}$ die geheime Zahl $x \in \mathbb{Z}_p^*$ mit $X = h^x$ kennt. □

Nun müssen wir die analoge Version der Beschränktheit beweisen, also zeigen, dass sich kein polynomieller Angreifer $\mathcal{A}$ an ein Polynom $\phi_I(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ mit $\deg(\phi_I) > K - k$ binden kann. Dies kann analog zu Lemma 3.1.5 gezeigt werden. Da sich der User während des Prove-k-Protokolls entweder im Fall 1 zusätzlich an die Zahlen $e, s, m_1, \ldots, m_L \in \mathbb{Z}_p$ binden muss, oder im Fall 2 nach obigem Lemma 3.2.3 nur mit vernachlässigbarer Wahrscheinlichkeit verschiedene Tupel $(e, s, m_1, \ldots, m_L) \neq (e', s', m_1', \ldots, m_L')$ ausgeben kann, definieren wir das Spiel folgendermaßen:

Spiel 3.2.1. *Das Spiel wird zwischen einem polynomiellen Angreifer $\mathcal{A}$ und einem Challenger $\mathcal{C}$ durchgeführt. Sei $K' \in \mathbb{N}$ polynomiell beschränkt mit $K' > K - k$.*

Setup: *Der Challenger $\mathcal{C}$ führt bei Eingabe des Sicherheitsparameters 1^λ den Algorithmus* GenSign *aus und erhält das Schlüsselpaar* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenSign}\left(1^\lambda, L, K\right)$ *des Signaturverfahrens. Anschließend sendet $\mathcal{C}$ den öffentlichen Schlüssel* pk *an den Angreifer $\mathcal{A}$.*

Commit: *Der Angreifer $\mathcal{A}$ sendet ein Element $\Sigma \in \mathbb{G}_1$, k Zahlen $n_1, \ldots, n_k \in \mathbb{Z}_p$ und eine Zahl $K' \in \mathbb{N}$ an $\mathcal{C}$.*

Challenge: *Der Challenger wählt zufällig $K'+1$ paarweise verschiedene Zahlen $i^{(1)}, \ldots, i^{(K'+1)} \in_{\mathcal{R}} \mathbb{Z}_p$ und sendet diese an $\mathcal{A}$.*

Ausgabe: *Der Angreifer $\mathcal{A}$ gibt $K'+1$ Tupel* $\left(e, s, m_1, \ldots, m_L, i^{(j)}, \phi_I\left(i^{(j)}\right), W_{I,i^{(j)}}\right) \in \mathbb{Z}_p^{L+4} \times \mathbb{G}_1$ *aus und gewinnt das Spiel, falls*

1. *für alle $1 \leq j \leq K'+1$ die Gleichung*

$$\hat{e}(\Sigma, Xh^e) = \hat{e}(gg_0^s g_1^{m_1} \cdots g_L^{m_L}, h)\, \hat{e}\left(W_{I,i^{(j)}}, h^{\left(\alpha - i^{(j)}\right)\prod_{l=1}^{k}(\alpha+n_l)}\right) \hat{e}\left(u_0^{\phi_I\left(i^{(j)}\right)}, h^{\prod_{l=1}^{k}(\alpha+n_l)}\right)$$

   erfüllt ist,

2. *die Interpolation der $K'+1$ Werte $\phi\left(i^{(1)}\right), \ldots, \phi\left(i^{(K'+1)}\right)$ ein Polynom vom Grad K' erzeugt und*
3. *die Interpolation (in den Exponenten) der $K'+1$ Zeugen $W_{i^{(1)}}, \ldots, W_{i^{(K'+1)}}$ ein Polynom vom Grad $K'-1$ erzeugt.*

Lemma 3.2.4. *Angenommen, es gelten die q-SDH-Annahme und die q-polyDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$. *Dann gibt es für jeden polynomiellen Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion ν, so dass für alle hinreichend großen $\lambda \in \mathbb{N}$ und für $k < K = q$ gilt:*

$$\Pr[\mathcal{A} \text{ gewinnt Spiel 3.2.1}] \leq \nu(\lambda).$$

Beweis. Angenommen, es existiert ein polynomieller Angreifer $\mathcal{A}$, der mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ das obige Spiel 3.2.1 gewinnt. Wir konstruieren einen polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das q-polyDH-Problem zu lösen, wobei $q = K$ ist.

Systemparameter: Der Angreifer $\mathcal{B}$ erhält eine zufällige q-polyDH Probleminstanz $(p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, \ldots, u_K, h, h_1, \ldots, h_K)$. Dann wählt dieser zufällig ein Generatortupel $(g, g_0, \ldots, g_L) \in_{\mathcal{R}} \mathbb{G}_1^{L+1}$ sowie eine Zahl $x \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet das Element $X = h^x$. Anschließend sendet $\mathcal{B}$ den perfekt simulierten öffentlichen Schlüssel $\mathsf{pk} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_0, \ldots, g_L, u_0, \ldots, u_K, h, h_1, \ldots, h_K, X)$ an den Angreifer $\mathcal{A}$.

Ausgabe Commitment: Der Angreifer $\mathcal{A}$ gibt ein Element $\Sigma \in \mathbb{G}_1$, die Zahlen $n_1, \ldots, n_k \in \mathbb{Z}_p$ und eine Zahl $K' \in \mathbb{N}$ mit $K' > K - k$ aus.

Challenge: Der Angreifer $\mathcal{B}$ sendet $K' + 1$ zufällige, paarweise verschiedene Zahlen $i^{(1)}, \ldots, i^{(K'+1)} \in_{\mathcal{R}} \mathbb{Z}_p$ an den Angreifer $\mathcal{A}$.

Ausgabe Responses: Der Angreifer $\mathcal{A}$ gibt insgesamt $K' + 1$ Tupel $\left(e, s, m_1, \ldots, m_L, i^{(j)}, \phi_I\left(i^{(j)}\right), W_{I,i^{(j)}}\right)$ mit

$$\hat{e}(\Sigma, Xh^e) = \hat{e}(gg_0^s g_1^{m_1} \cdots g_L^{m_L}, h)\, \hat{e}\left(W_{I,i^{(j)}}, h^{\left(\alpha - i^{(j)}\right)\prod_{l=1}^{k}(\alpha+n_l)}\right)$$
$$\hat{e}\left(u_0^{\phi_I\left(i^{(j)}\right)}, h^{\prod_{l=1}^{k}(\alpha+n_l)}\right)$$

für alle $1 \leq j \leq K' + 1$ aus. Für alle $1 \leq j \leq K' + 1$ ist nur mit vernachlässigbarer Wahrscheinlichkeit $\phi_I\left(i^{(j)}\right)$ nicht die korrekte Berechnung des Polynoms $\phi_I(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ an der Stelle $i^{(j)}$, da $\mathcal{A}$ ansonsten die Berechnungsgebundenheit gemäß Lemma 3.2.1 gebrochen hätte, wobei $\phi_I(\mathsf{x})$ das Polynom mit $\Sigma^{x+e} = gg_0^s g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi_I(\alpha)\cdot\prod_{l=1}^{k}(\alpha+n_l)}$ ist. Der Angreifer $\mathcal{B}$ berechnet durch Polynominterpolation das Polynom $\phi_I(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ und anschließend das Polynom $\phi(\mathsf{x}) = \phi_I(\mathsf{x}) \cdot \prod_{l=1}^{k}(\mathsf{x} + n_l) \in \mathbb{Z}_p[\mathsf{x}]$. Folglich ist $\deg(\phi) = \deg(\phi_I) + k$. Weiter berechnet $\mathcal{B}$ das Commitment $C = \Sigma^{x+e}(gg_0^s g_1^{m_1} \cdots g_L^{m_L})^{-1}$ mit $C = u_0^{\phi(\alpha)}$. Da nach Voraussetzung $\deg(\phi_I) > K - k$ ist, folgt $\deg(\phi) > K$.

Somit löst der Angreifer $\mathcal{B}$ das q-polyDH-Problem mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist. □

3.2.2 Konstruktion des BBS+A-Signaturverfahrens

Im Folgenden wird das BBS+A-Signaturverfahren detailliert beschrieben. Wie in Unterabschnitt 2.6.4 erwähnt, gilt für das Typ-1-Pairing die Beziehung $u_0 = h$ und analog für das Typ-2-Pairing $u_0 = \psi(h)$, wobei $\psi\colon \mathbb{G}_2 \to \mathbb{G}_1$ der bekannte Isomorphismus ist.

GenSign führt bei Eingabe des Sicherheitsparameters 1^λ und zweier Zahlen $L, K \in \mathbb{N}$ den Algorithmus $\mathsf{Setup}_{\mathsf{Bil}}$ aus und erhält die Systemparameter $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, h) \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right)$, wobei $\mathbb{G}_1 = \langle u_0 \rangle, \mathbb{G}_2 = \langle h \rangle$ und $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ eine bilineare Abbildung ist. Dann wählt der Algorithmus zufällig ein Generatortupel $(g, g_0, \ldots, g_L) \in_{\mathcal{R}} \mathbb{G}_1^{L+2}$ sowie zwei Zahlen $x, \alpha \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Elemente $X = h^x$ sowie $u_i = u_0^{\alpha^i}$ und $h_i = h^{\alpha^i}$ für $1 \leq i \leq K$. Schließlich wählt der Algorithmus eine Hashfunktion $\mathsf{H}\colon \{0,1\}^* \to \mathbb{Z}_p$, die weder die Einweg-Eigenschaft noch die schwache Kollisionsresistenz erfüllen muss. Die Ausgabe ist das Schlüsselpaar

$$(\mathsf{pk}, \mathsf{sk}) = \left((p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_0, \ldots, g_L, u_0, \ldots, u_K, h, h_1, \ldots, h_K, X, \mathsf{H}), x\right) \leftarrow \mathsf{GenSign}\left(1^\lambda, L, K\right),$$

wobei für die Menge aller Nachrichten $\mathbb{M}_{\mathsf{Sign}} = \left\{m : m = (m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}], \deg(\phi) \leq K\right\}$ gilt.

Sign signiert bei Eingabe des Schlüsselpaares $(\mathsf{pk}, \mathsf{sk})$ eine Nachricht $m = (m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^L \times \in \mathbb{Z}_p[\mathsf{x}]$ mit $\deg(\phi) \leq K$, indem der Algorithmus zufällig zwei Zahlen $e, s \in_{\mathcal{R}} \mathbb{Z}_p$ wählt und das Element

$$\Sigma = \left(g g_0^s g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}\right)^{\frac{1}{x+e}}$$

berechnet. Die Ausgabe ist die Signatur

$$\sigma = (\Sigma, e, s) \leftarrow \mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, m).$$

VerifySign verifiziert bei Eingabe des öffentlichen Schlüssels pk eine Signatur $\sigma = (\Sigma, e, s)$ auf eine Nachricht $m = (m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}]$, indem der Algorithmus die Gleichung

$$\hat{e}(\Sigma, X h^e) \stackrel{?}{=} \hat{e}\left(g g_0^s g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}, h\right)$$

überprüft. Entsprechend ist die Ausgabe $\mathsf{VerifySign}(\mathsf{pk}, m, \sigma) = \{1, 0\}$.

Issue: Damit ein User $\mathcal{U}$ eine Signatur σ auf eine Nachricht $m = (m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}]$ mit $\deg(\phi) \leq K$ bzgl. des öffentlichen Schlüssels pk erhält und der Signer $\mathcal{S}$ keinerlei Informationen über die Nachricht erlangt, berechnet $\mathcal{U}$ das perfekt verbergende Commitment

$$C = g_0^{s'} g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)} \leftarrow \mathsf{Com}(\mathsf{pk}, m)$$

gemäß Abschnitt 3.1 für eine Zufallszahl $s' \in_{\mathcal{R}} \mathbb{Z}_p$. Anschließend führen $\mathcal{U}$ und $\mathcal{S}$ das interaktive Signaturerstellungsprotokoll Issue durch, welches in folgender Abbildung 3.1 dargestellt ist.

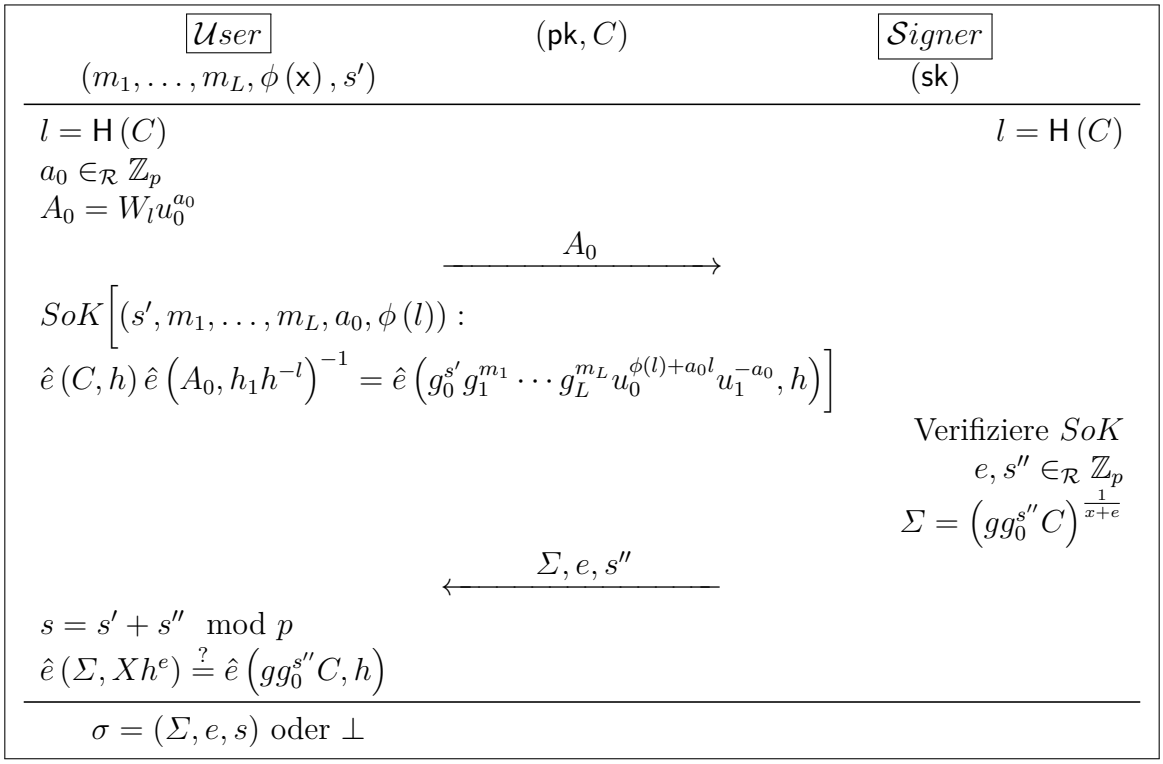

Abbildung 3.1: Signaturerstellungsprotokoll Issue der BBS+A-Signatur

Schritt 1: Der User $\mathcal{U}$ berechnet den Hashwert $l = \mathsf{H}(C)$, den Wert $\phi(l) \in \mathbb{Z}_p$ und das Polynom $\phi_l(\mathsf{x}) = (\phi(\mathsf{x}) - \phi(l))(\mathsf{x} - l)^{-1}$. Sei $W_l = u_0^{\phi_l(\alpha)}$ der Zeuge für die Berechnung $\phi(l)$, der analog zum Commitment mit den Generatoren $u_0, \ldots, u_{K-1}$ berechnet werden kann. Dann wählt der User $\mathcal{U}$ zufällig eine Zahl $a_0 \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Element $A_0 = W_l u_0^{a_0}$, wodurch der Zeuge W_l perfekt verborgen ist. Anschließend erstellt $\mathcal{U}$ die folgende Signature-of-Knowledge (siehe auch Abschnitt A.1):

$$SoK\Big[(s', m_1, \ldots, m_L, a_0, \phi(l)) : \hat{e}(C, h)\,\hat{e}\left(A_0, h_1 h^{-l}\right)^{-1} = \hat{e}\left(g_0^{s'} g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(l)+a_0 l} u_1^{-a_0}, h\right)\Big].$$

Durch SoK beweist $\mathcal{U}$ in zero-knowledge, dass $\mathcal{U}$ die Nachricht $m \in \mathbb{M}_{\mathsf{Com}}$ und den Zufallswert $s' \in \mathbb{R}$ mit $C = \mathsf{Com}(\mathsf{pk}, m; s')$ kennt (vgl. Korollar 3.1.3). Der Beweis wird nichtinteraktiv ausgeführt, damit SoK von einer dritten Partei verifiziert werden kann, da dies für die folgenden elektronischen Geldsysteme benötigt wird.

Schritt 2: Der Signer $\mathcal{S}$ verifiziert SoK, wählt zufällig zwei Zahlen $e, s'' \in_{\mathcal{R}} \mathbb{Z}_p$

und berechnet das Element

$$\Sigma = \left(g g_0^{s''} C\right)^{\frac{1}{x+e}} = \left(g g_0^{s} g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}\right)^{\frac{1}{x+e}}$$

für $s = s' + s'' \mod p$. Dann sendet $\mathcal{S}$ das Tripel (Σ, e, s'') an $\mathcal{U}$.

Schritt 3: Der User $\mathcal{U}$ berechnet $s = s' + s'' \mod p$, verifiziert die Signatur durch Überprüfung der Gleichung $\hat{e}(\Sigma, X h^e) \stackrel{?}{=} \hat{e}\left(g g_0^{s''} C, h\right)$ und speichert die Signatur $\sigma = (\Sigma, e, s)$ ab.

Prove: Damit ein User $\mathcal{U}$ einem Verifier $\mathcal{V}$ beweisen kann, im Besitz einer gültigen Signatur $\sigma = (\Sigma, e, s)$ bzgl. pk auf eine Nachricht $m = (m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}]$ zu sein, wobei ein geheimer Wert (dies sei o.B.d.A.) $n_1 \in \mathbb{Z}_p$ in $V = u_0^{\phi(\alpha)}$ akkumuliert ist und der Verifier $\mathcal{V}$ keinerlei Informationen über die Nachricht und die Signatur erhält, führen $\mathcal{U}$ und $\mathcal{V}$ das Signaturbeweisprotokoll Prove durch, welches ein Zero-Knowledge-Proof-of-Knowledge-Protokoll und in Abbildung 3.2 dargestellt ist.

Zur Vereinfachung nehmen wir an, dass der User $\mathcal{U}$ beweisen möchte, dass er für die perfekt verbergenden Commitments $D_m = g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L}$ sowie $D_n = g_0^{r_n} g_1^{n_1}$ mit $r_m, r_n \in_{\mathcal{R}} \mathbb{Z}_p$ eine gültige Signatur $\sigma = (\Sigma, e, s)$ kennt.

Schritt 1: Der User $\mathcal{U}$ wählt zufällig eine Zahl $r_\sigma \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das perfekt verbergende Pedersen-Commitment $D_\sigma = g_0^{r_\sigma} g_1^{e} g_2^{s}$, um sich an die Zahlen $e, s \in \mathbb{Z}_p$ zu binden, da dies für den Sicherheitsbeweis benötigt wird. Danach wählt $\mathcal{U}$, wie bei der BBS+-Signatur, zufällig die beiden Zahlen $b_1, b_2 \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Elemente $B_1 = g_0^{b_1} g_1^{b_2}$ sowie $B_2 = \Sigma g_1^{b_1}$, damit das Element $\Sigma \in \mathbb{G}_1$ perfekt verborgen ist. Anschließend berechnet $\mathcal{U}$ den Hashwert $l = \mathsf{H}(D_m || D_n || D_\sigma || B_1 || B_2)$. Es sei $\phi_1(\mathsf{x}) = \prod_{j=2}^{K} (\mathsf{x} + n_j) \in \mathbb{Z}_p[\mathsf{x}]$ das Polynom mit $\phi(\mathsf{x}) = \phi_1(\mathsf{x})(\mathsf{x} + n_1)$, dann berechnet $\mathcal{U}$, wie beim Polynom-Commitment, den Wert $\phi_1(l) = \prod_{j=2}^{K} (l + n_j) \in \mathbb{Z}_p$ und das Polynom $\phi_{1,l}(\mathsf{x}) = (\phi_1(\mathsf{x}) - \phi_1(l))(\mathsf{x} - l)^{-1}$. Sei $W_{1,l} = u_0^{\phi_{1,l}(\alpha)}$ der Zeuge für die Berechnung $\phi_1(l)$. Weiter wählt $\mathcal{U}$ zufällig zwei Zahlen $a_1, r_a \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Elemente $A_2 = g_0^{r_a} g_1^{a_1} g_2^{\phi_1(l)}$ und $A_1 = W_{1,l} u_0^{a_1}$, um den Zeugen $W_{1,l} \in \mathbb{G}_1$ perfekt zu verbergen. Anschließend sendet $\mathcal{U}$ die Elemente $A_1, A_2, B_1, B_2, D_\sigma$ an $\mathcal{V}$ und erstellt die Signature-of-Knowledge SoK, wobei $\delta_1 = a_1 n_1, \delta_2 = \phi_1(l) n_1, \delta_r = r_a n_1, \beta_1 = b_1 e$ und $\beta_2 = b_2 e$ ist (siehe auch Abschnitt A.2):

$$\begin{aligned} SoK\Big[(r_m, r_n, r_\sigma, r_a, a_1, \delta_1, \delta_2, \delta_r, b_1, b_2, \beta_1, \beta_2, e, s, m_1, \ldots, m_L, n_1, \phi_1(l)) : \\ A_2 = g_0^{r_a} g_1^{a_1} g_2^{\phi_1(l)} \wedge 1 = A_2^{n_1} g_0^{-\delta_r} g_1^{-\delta_1} g_2^{-\delta_2} \wedge B_1 = g_0^{b_1} g_1^{b_2} \wedge 1 = B_1^{e} g_0^{-\beta_1} g_1^{-\beta_2} \\ \wedge D_m = g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L} \wedge D_n = g_0^{r_n} g_1^{n_1} \wedge D_\sigma = g_0^{r_\sigma} g_1^{e} g_2^{s} \wedge \\ \hat{e}(B_2, X) \left(\hat{e}(g, h)\, \hat{e}\left(A_1, h_2 h_1^{-l}\right)\right)^{-1} = \hat{e}\left(g_1^{b_1}, X\right) \hat{e}(A_1^{n_1}, h_1) \end{aligned}$$

$$\hat{e}\left(g_0^s g_1^{m_1+\beta_1} g_2^{m_2} \cdots g_L^{m_L} u_0^{l\delta_1+\delta_2} u_1^{la_1-\delta_1+\phi_1(l)} u_2^{-a_1} A_1^{-ln_1} B_2^{-e}, h\right)\Big].$$

Durch SoK beweist $\mathcal{U}$ in zero-knowledge, dass er im Besitz einer gültigen Signatur $\sigma = (\Sigma, e, s)$ auf eine Nachricht $m = (m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}]$ mit $V = u_0^{\phi(\alpha)} = u_0^{\phi_1(\alpha)(\alpha+n_1)}$ ist. Dabei beweisen die ersten sieben Gleichungen die korrekte Berechnung der Elemente A_2, B_1, D_m, D_n und D_σ sowie $\delta_1 = a_1 n_1, \delta_2 = \phi_1(l) n_1$ und $\beta_1 = b_1 e$, was für die folgende Gleichung notwendig ist. Durch die letzte Gleichung wird bewiesen, dass $\mathcal{U}$ eine gültige Signatur σ kennt und dass der Wert $\phi_1(l) \in \mathbb{Z}_p$ die korrekte Berechnung des Polynoms $\phi_1(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ an der Stelle $l \in \mathbb{Z}_p$ ist (vgl. Unterabschnitt 3.2.1). Der Beweis wird nichtinteraktiv ausgeführt, damit SoK von einer dritten Partei verifiziert werden kann, da dies für die folgenden elektronischen Geldsysteme benötigt wird.

Schritt 2: Der Verifier $\mathcal{V}$ berechnet den Hashwert $l = \mathsf{H}(D_m||D_n||D_\sigma||B_1||B_2)$ und verifiziert SoK.

$\boxed{\mathcal{U}ser}$	(pk, D_m, D_n)	$\boxed{\mathcal{V}erifier}$
$(m_1, \ldots, m_L, n_1, \ldots, n_K, \Sigma, e, s, r_m, r_n)$		
$a_1, b_1, b_2, r_\sigma, r_a \in_{\mathcal{R}} \mathbb{Z}_p$ $D_\sigma = g_0^{r_\sigma} g_1^e g_2^s$ $B_1 = g_0^{b_1} g_1^{b_2}$ $B_2 = \Sigma g_1^{b_1}$ $l = \mathsf{H}(D_m\|\|D_n\|\|D_\sigma\|\|B_1\|\|B_2)$ $A_2 = g_0^{r_a} g_1^{a_1} g_2^{\phi_1(l)}$ $A_1 = W_{1,l} u_0^{a_1}$		
	$\xrightarrow{A_1, A_2, B_1, B_2, D_\sigma}$	
$SoK\Big[(r_m, r_n, r_\sigma, r_a, a_1, \delta_1, \delta_2, \delta_r, b_1, b_2, \beta_1, \beta_2, e, s, m_1, \ldots, m_L, n_1, \phi_1(l)):$ $A_2 = g_0^{r_a} g_1^{a_1} g_2^{\phi_1(l)} \wedge 1 = A_2^{n_1} g_0^{-\delta_r} g_1^{-\delta_1} g_2^{-\delta_2} \wedge B_1 = g_0^{b_1} g_1^{b_2} \wedge 1 = B_1^e g_0^{-\beta_1} g_1^{-\beta_2} \wedge$ $D_m = g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L} \wedge D_n = g_0^{r_n} g_1^{n_1} \wedge D_\sigma = g_0^{r_\sigma} g_1^e g_2^s \wedge$ $\hat{e}(B_2, X)\left(\hat{e}(g, h)\,\hat{e}\left(A_1, h_2 h_1^{-l}\right)\right)^{-1} = \hat{e}\left(g_1^{b_1}, X\right)\hat{e}(A_1^{n_1}, h_1)$ $\hat{e}\left(g_0^s g_1^{m_1+\beta_1} g_2^{m_2} \cdots g_L^{m_L} u_0^{l\delta_1+\delta_2} u_1^{la_1-\delta_1+\phi_1(l)} u_2^{-a_1} A_1^{-ln_1} B_2^{-e}, h\right)\Big]$		
		$l = \mathsf{H}(D_m\|\|D_n\|\|D_\sigma\|\|B_1\|\|B_2)$ Verifiziere SoK
		1 oder 0

Abbildung 3.2: Signaturbeweisprotokoll Prove der BBS+A-Signatur

Prove-k: Damit ein User $\mathcal{U}$ einem Verifier $\mathcal{V}$ beweisen kann, im Besitz einer gültigen Signatur $\sigma = (\Sigma, e, s)$ bzgl. pk auf eine Nachricht $m = (m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}]$ zu sein, wobei o.B.d.A. die Werte $n_1, \ldots, n_k \in \mathbb{Z}_p$ mit $k \leq K$ in $V = u_0^{\phi(\alpha)}$ akkumuliert sind und der Verifier $\mathcal{V}$ keinerlei Informationen über die Nachricht und die Signatur erhält, führen $\mathcal{U}$ und $\mathcal{V}$ das Signaturbeweisprotokoll Prove-k durch, welches ein Zero-Knowledge-Proof-of-Knowledge-Protokoll und in Abbildung 3.3 dargestellt ist.

Zur Vereinfachung nehmen wir an, dass der User $\mathcal{U}$ beweisen möchte, dass er für das perfekt verbergende Pedersen-Commitment $D_m = g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L}$ mit $r_m \in_{\mathcal{R}} \mathbb{Z}_p$ eine gültige Signatur $\sigma = (\Sigma, e, s)$ kennt.

Schritt 1: Der User $\mathcal{U}$ wählt zufällig eine Zahl $r_\sigma \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das perfekt verbergende Pedersen-Commitment $D_\sigma = g_0^{r_\sigma} g_1^e g_2^s$, um sich an die Zahlen $e, s \in \mathbb{Z}_p$ zu binden, da dies für den Sicherheitsbeweis benötigt wird. Danach wählt $\mathcal{U}$, wie bei der BBS+-Signatur, zufällig die beiden Zahlen $b_1, b_2 \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Elemente $B_1 = g_0^{b_1} g_1^{b_2}$ sowie $B_2 = \Sigma g_1^{b_1}$, damit das Element $\Sigma \in \mathbb{G}_1$ perfekt verborgen ist. Anschließend berechnet $\mathcal{U}$ den Hashwert $l = \mathsf{H}(D_m || D_\sigma || B_1 || B_2)$. Es sei $\phi_I(\mathsf{x}) = \prod_{j=k+1}^{K} (\mathsf{x} + n_j) \in \mathbb{Z}_p[\mathsf{x}]$ das Polynom mit $\phi(\mathsf{x}) = \phi_I(\mathsf{x}) \prod_{j=1}^{k} (\mathsf{x} + n_j)$, dann berechnet $\mathcal{U}$, wie beim Polynom-Commitment, den Wert $\phi_I(l) = \prod_{j=k+1}^{K} (l + n_j) \in \mathbb{Z}_p$ und das Polynom $\phi_{I,l}(\mathsf{x}) = (\phi_I(\mathsf{x}) - \phi_I(l))(\mathsf{x} - l)^{-1}$. Sei $W_{I,l} = u_0^{\phi_{I,l}(\alpha)}$ der Zeuge für die Berechnung $\phi_I(l)$. Weiter wählt $\mathcal{U}$ zufällig eine Zahl $a_1 \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Element $A_1 = W_{I,l} u_0^{a_1}$, um den Zeugen $W_{I,l} \in \mathbb{G}_1$ perfekt zu verbergen. Anschließend sendet $\mathcal{U}$ die Elemente A_1, B_1, B_2, D_σ an $\mathcal{V}$ und erstellt die Signature-of-Knowledge SoK, wobei $\beta_1 = b_1 e, \beta_2 = b_2 e, u_I = u_0^{\prod_{j=1}^{k} (\alpha + n_j)}, u_{l,I} = u_0^{(\alpha - l) \prod_{j=1}^{k} (\alpha + n_j)}$ und $h_{l,I} = h^{(\alpha - l) \prod_{j=1}^{k} (\alpha + n_j)}$ ist (siehe auch Abschnitt A.3):

$$
\begin{aligned}
SoK\Bigg[& (r_m, r_\sigma, a_1, b_1, b_2, \beta_1, \beta_2, e, s, m_1, \ldots, m_L, \phi_I(l)) : B_1 = g_0^{b_1} g_1^{b_2} \wedge \\
& 1 = B_1^e g_0^{-\beta_1} g_1^{-\beta_2} \ \wedge \ D_m = g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L} \ \wedge \ D_\sigma = g_0^{r_\sigma} g_1^e g_2^s \wedge \\
& \frac{\hat{e}(B_2, X)}{\hat{e}(g, h)\, \hat{e}(A_1, h_{l,I})} = \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(g_0^s g_1^{m_1 + \beta_1} g_2^{m_2} \ldots g_L^{m_L} u_{l,I}^{-a_1} u_I^{\phi_I(l)} B_2^{-e}, h\right) \Bigg].
\end{aligned}
$$

Durch SoK beweist $\mathcal{U}$ in zero-knowledge, dass er im Besitz einer gültigen Signatur $\sigma = (\Sigma, e, s)$ auf eine Nachricht $m = (m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}]$ mit $V = u_0^{\phi(\alpha)} = u_0^{\phi_I(\alpha) \prod_{j=1}^{k} (\alpha + n_j)}$ ist. Dabei beweisen die ersten vier Gleichungen die korrekte Berechnung der Elemente B_1, D_m und D_σ sowie $\beta_1 = b_1 e$, was für die folgende Gleichung benötigt wird. Durch die letzte Gleichung wird bewiesen, dass $\mathcal{U}$ eine gültige Signatur σ kennt und dass der Wert $\phi_I(l) \in \mathbb{Z}_p$ die korrekte Berechnung des Polynoms $\phi_I(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ an der

Stelle $l \in \mathbb{Z}_p$ ist (vgl. Unterabschnitt 3.2.1). Der Beweis wird nichtinteraktiv ausgeführt, damit SoK von einer dritten Partei verifiziert werden kann, da dies für die folgenden elektronischen Geldsysteme notwendig ist.

Schritt 2: Der Verifier $\mathcal{V}$ berechnet den Hashwert $l = \mathsf{H}(D_m||D_\sigma||B_1||B_2)$ und verifiziert SoK.

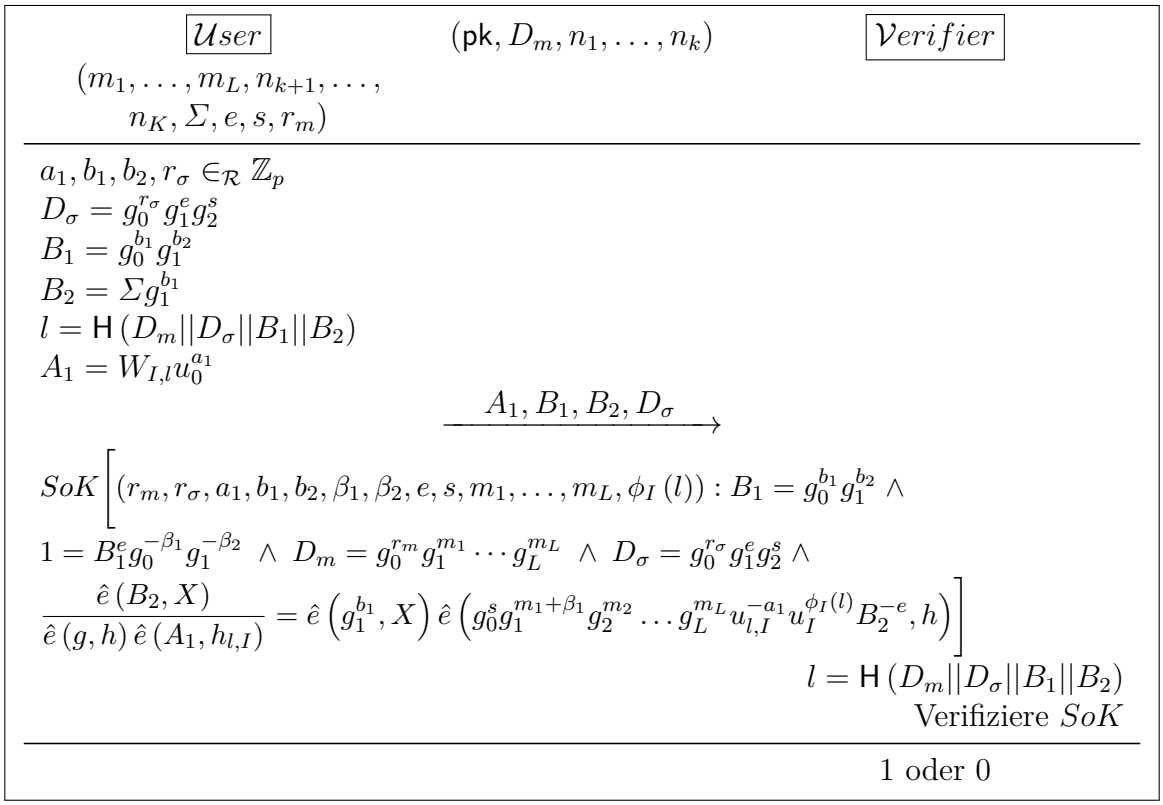

Abbildung 3.3: Signaturbeweisprotokoll Prove-k der BBS+A-Signatur

Prove*-k: Sind die Werte $n_1, \ldots, n_K \in \mathbb{Z}_p$ die Ausgabe einer Hashfunktion, ist es (im Random-Oracle-Modell) nicht notwendig, dass sich der User $\mathcal{U}$ an die Zahlen $e, s \in \mathbb{Z}_p$ bindet. Somit muss das Commitment D_σ nicht berechnet werden. Entsprechend werden die Berechnung von l und die Erstellung der Signature-of-Knowledge SoK angepasst.

Die folgende Abbildung 3.4 zeigt das Signaturbeweisprotokoll Prove*-k des BBS+A-Signaturverfahrens.

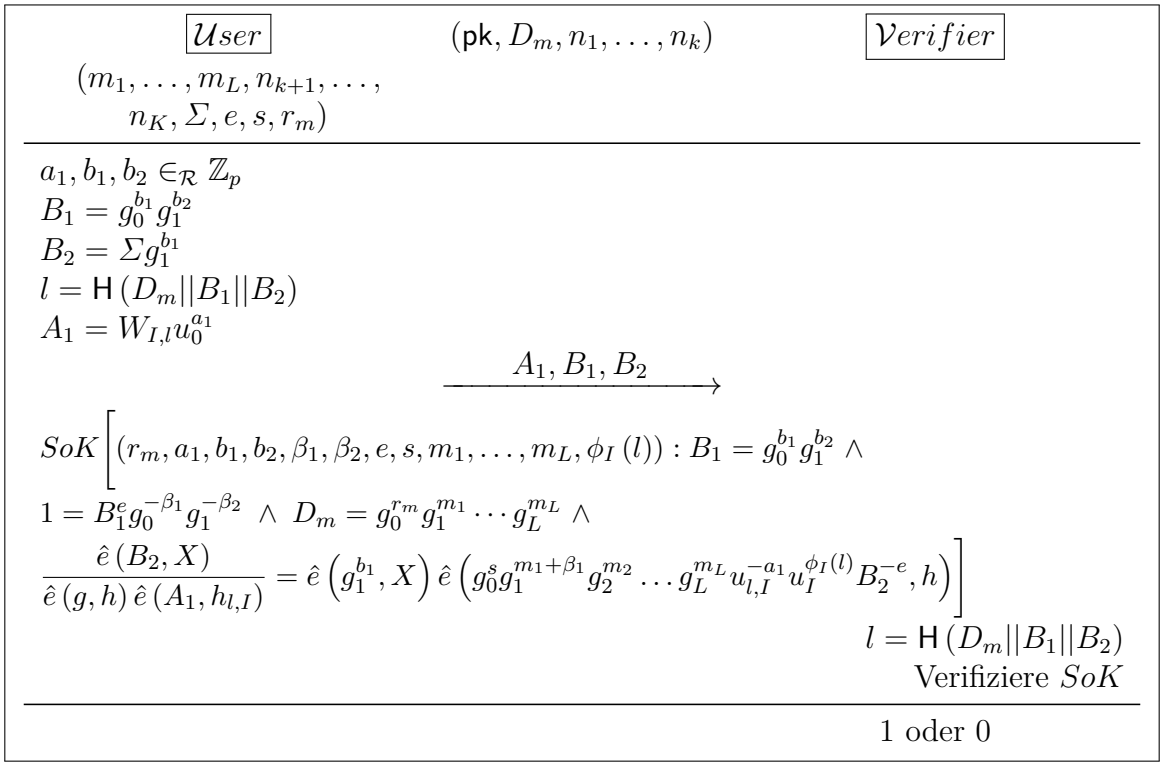

Abbildung 3.4: Signaturbeweisprotokoll Prove^*-k der BBS+A-Signatur

Prove-K: Damit ein User $\mathcal{U}$ einem Verifier $\mathcal{V}$ beweisen kann, im Besitz einer gültigen Signatur $\sigma = (\Sigma, e, s)$ bzgl. pk auf eine Nachricht $m = (m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}]$ zu sein, wobei die Werte $n_1, \ldots, n_K \in \mathbb{Z}_p$ in $V = u_0^{\phi(\alpha)}$ akkumuliert sind und der Verifier $\mathcal{V}$ keinerlei Informationen über die Nachricht und die Signatur erhält, führen $\mathcal{U}$ und $\mathcal{V}$ das Signaturbeweisprotokoll Prove-K durch, welches ein Zero-Knowledge-Proof-of-Knowledge-Protokoll und in Abbildung 3.5 dargestellt ist.

Zur Vereinfachung nehmen wir an, dass der User $\mathcal{U}$ beweisen möchte, dass er für das perfekt verbergende Pedersen-Commitment $D_m = g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L}$ mit $r_m \in_{\mathcal{R}} \mathbb{Z}_p$ sowie für die bekannten Werte $n_1, \ldots, n_K \in \mathbb{Z}_p$ eine gültige Signatur $\sigma = (\Sigma, e, s)$ kennt.

Schritt 1: Der User $\mathcal{U}$ wählt, wie bei der BBS+-Signatur, zufällig die beiden Zahlen $b_1, b_2 \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Elemente $B_1 = g_0^{b_1} g_1^{b_2}$ sowie $B_2 = \Sigma g_1^{b_1}$, damit das Element $\Sigma \in \mathbb{G}_1$ perfekt verborgen ist. Anschließend sendet $\mathcal{U}$ die Elemente B_1, B_2 an $\mathcal{V}$ und erstellt die Signature-of-Knowledge SoK, wobei $\beta_1 = b_1 e, \beta_2 = b_2 e$ und $V = u_0^{\prod_{j=1}^{K}(\alpha+n_j)}$ ist (siehe auch Abschnitt A.4):

$$SoK\Big[(r_m, b_1, b_2, \beta_1, \beta_2, e, s, m_1, \ldots, m_L) : B_1 = g_0^{b_1} g_1^{b_2} \wedge 1 = B_1^e g_0^{-\beta_1} g_1^{-\beta_2}$$

$$\wedge\ D_m = g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L} \wedge$$
$$\hat{e}\left(B_2, X\right) \hat{e}\left(gV, h\right)^{-1} = \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(g_0^{s} g_1^{m_1+\beta_1} g_2^{m_2} \cdots g_L^{m_L} B_2^{-e}, h\right)\Big].$$

Durch SoK beweist $\mathcal{U}$ in zero-knowledge, dass er im Besitz einer gültigen Signatur $\sigma = (\Sigma, e, s)$ auf eine Nachricht $m = (m_1, \ldots, m_L, \phi(x)) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}]$ mit $V = u_0^{\prod_{j=1}^{K}(\alpha+n_j)}$ ist. Dabei beweisen die ersten drei Gleichungen die korrekte Berechnung der Elemente B_1 und D_m sowie $\beta_1 = b_1 e$, was für die folgende Gleichung benötigt wird. Durch die letzte Gleichung wird gezeigt, dass $\mathcal{U}$ eine gültige Signatur σ kennt. Der Beweis wird nichtinteraktiv ausgeführt, damit SoK von einer dritten Partei verifiziert werden kann, da dies für die folgenden elektronischen Geldsysteme notwendig ist.

Schritt 2: Der Verifier $\mathcal{V}$ verifiziert SoK.

$\boxed{\mathcal{U}ser}$	$(\mathsf{pk}, D_m,$	$\boxed{\mathcal{V}erifier}$
$(m_1, \ldots, m_L, \Sigma, e, s, r_m)$	$n_1, \ldots, n_K)$	
$b_1, b_2 \in_{\mathcal{R}} \mathbb{Z}_p$		
$B_1 = g_0^{b_1} g_1^{b_2}$		
$B_2 = \Sigma g_1^{b_1}$		
	$\xrightarrow{\quad B_1, B_2 \quad}$	
$SoK\Big[(r_m, b_1, b_2, \beta_1, \beta_2, e, s, m_1, \ldots, m_L) : B_1 = g_0^{b_1} g_1^{b_2} \wedge$		
$1 = B_1^e g_0^{-\beta_1} g_1^{-\beta_2} \ \wedge\ D_m = g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L} \wedge$		
$\hat{e}\left(B_2, X\right) \hat{e}\left(gV, h\right)^{-1} = \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(g_0^{s} g_1^{m_1+\beta_1} g_2^{m_2} \cdots g_L^{m_L} B_2^{-e}, h\right)\Big]$		
		Verifiziere SoK
		1 oder 0

Abbildung 3.5: Signaturbeweisprotokoll Prove-K der BBS+A-Signatur

Bemerkung 3.2.1. *Werden in den Protokollen* Issue, Prove, Prove-k *und* Prove-K *jeweils die Zahl* $l \in_{\mathcal{R}} \mathbb{Z}_p$ *zufällig vom Signer* $\mathcal{S}$ *bzw. Verifier* $\mathcal{V}$ *gewählt und die jeweilige Signature-of-Knowledge als interaktiver Zero-Knowledge-Proof-of-Knowledge durchgeführt, erfüllen die Protokolle auch ohne das Random-Oracle-Modell die gewünschten Sicherheitseigenschaften. Da für die entwickelten Geldsysteme in Kapitel 5 bis 9 die Zahl* l *allerdings die Ausgabe einer Hashfunktion sein muss und der jeweilige Proof-of-Knowledge nichtinteraktiv als Signature-of-Knowledge durchgeführt werden muss (da die Händler ansonsten das Geld nicht bei der Bank einlösen könnten), wurden die Protokolle entsprechend beschrieben.*

3.2.3 Sicherheitsanalyse des BBS+A-Signaturverfahrens

Zunächst wird bewiesen, dass das BBS+A-Signaturverfahren unter der q-SDH-Annahme stark existentiell unfälschbar unter einem adaptiven Angriff mit gewählten Nachrichten ist. Danach wird die Sicherheit der Protokolle Issue und Prove analysiert.

Lemma 3.2.5. *Die BBS+A-Signatur* ($\mathsf{GenSign}, \mathsf{Sign}, \mathsf{VerifySign}$) *ist stark existentiell unfälschbar unter einem adaptiven Angriff mit gewählten Nachrichten, falls die q-SDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gilt.*

Beweis. Wir orientieren uns sehr stark an [Sch11, Lemma 6] und werden viele Grundzüge übernehmen.

Angenommen, es existiert ein polynomieller Angreifer $\mathcal{A}$, der die BBS+A-Signatur nach insgesamt q_S adaptiven Orakel-Anfragen mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ stark existentiell fälscht. Wir konstruieren einen polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das q-SDH-Problem mit $q = q_S$ (bzw. $q = q_S + 1$ im Typ-1- und Typ-2-Pairing) zu lösen. Dabei gilt im Folgenden für jedes vom Angreifer $\mathcal{A}$ gesendete Polynom $\phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ die Bedingung $\deg(\phi) \leq K'$ für eine polynomiell beschränkte Zahl $K' \in \mathbb{N}$, wobei sogar $K' > K$ erlaubt ist.

Der Angreifer $\mathcal{B}$ erhält eine zufällige q-SDH Probleminstanz $\left(p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, \mathrm{g}, \mathrm{g}^x, \mathrm{g}^{x^2}, \ldots, \mathrm{g}^{x^q}, \mathrm{h}, \mathrm{h}^x, \mathrm{h}^{x^2}, \ldots, \mathrm{h}^{x^q}\right)$, wobei für ein Typ-1-Pairing $\mathrm{g} = \mathrm{h}$ und für ein Typ-2-Pairing $\mathrm{g} = \psi(\mathrm{h})$ ist. Seien $\sigma_1 = (\Sigma_1, e_1, s_1), \ldots, \sigma_{q_S} = (\Sigma_{q_S}, e_{q_S}, s_{q_S})$ die simulierten Signaturen von $\mathcal{B}$ und $(m^\star, \sigma^\star) = ((m_1^\star, \ldots, m_L^\star, \phi^\star(\mathsf{x})), (\Sigma^\star, e^\star, s^\star))$ das gefälschte BBS+A-Nachrichten-Signatur-Paar, welches $\mathcal{A}$ am Ende ausgibt. Dann müssen die folgenden drei Fälle unterschieden werden:

Fall 1: $e^\star \notin \{e_1, \ldots, e_{q_S}\}$**:**

Setup: Der Angreifer $\mathcal{B}$ wählt zufällig die Zahlen $e_1, \ldots, e_{q_S}, s_1, \ldots, s_{q_S} \in_{\mathcal{R}} \mathbb{Z}_p$ sowie die Zahlen $r_0, r_1, \alpha \in_{\mathcal{R}} \mathbb{Z}_p^*$ und definiert die Polynome $\phi_{\bar{e}}(\mathsf{x}) := \prod_{l=1}^{q_S}(\mathsf{x} + e_l) \in \mathbb{Z}_p[\mathsf{x}]$, $\phi_{\bar{e}_i}(\mathsf{x}) := \prod_{l=1, l\neq i}^{q_S}(\mathsf{x} + e_l) \in \mathbb{Z}_p[\mathsf{x}]$ und $\phi_{\bar{e}_{i,j}}(\mathsf{x}) := \prod_{l=1, l\neq i, l\neq j}^{q_S}(\mathsf{x} + e_l) \in \mathbb{Z}_p[\mathsf{x}]$ sowie die Zahlen $\bar{e} := \phi_{\bar{e}}(x) \in \mathbb{Z}_p$, $\bar{e}_i := \phi_{\bar{e}_i}(x) \in \mathbb{Z}_p$ und $\bar{e}_{i,j} := \phi_{\bar{e}_{i,j}}(x) \in \mathbb{Z}_p$. Dann berechnet $\mathcal{B}$ die Generatoren $g = \mathrm{g}^{r_0\bar{e}}$ und $g_0 = \mathrm{g}^{r_1\bar{e}}$. (Falls $g = g_0 = 1$ wäre, wäre $\bar{e} = 0$ und somit $e_l = -x$ für ein $1 \leq l \leq q_S$. Dann würde $\mathcal{B}$ die Zahl x kennen und könnte das q-SDH-Problem direkt lösen.) Weiter wählt $\mathcal{B}$ zufällig $L+1$ paarweise verschiedene Zahlen $\mu_1, \ldots, \mu_{L+1} \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Generatoren $g_1 = g_0^{\mu_1}, \ldots, g_L = g_0^{\mu_L}$ sowie $u_0 = g_0^{\mu_{L+1}}$. Dann

- berechnet $\mathcal{B}$ im Typ-1- bzw. Typ-2-Pairing den Generator $h = \mathrm{h}^{r_1\bar{e}\mu_{L+1}}$ und das Element $X = (\mathrm{h}^x)^{r_1\bar{e}\mu_{L+1}}$, so dass die Beziehungen $u_0 = h$ bzw. $u_0 = \psi(h)$ und $X = h^x$ gelten, wobei das Element X nur für $q_S \leq q-1$ effizient berechnet werden kann.
- definiert $\mathcal{B}$ im Typ-3-Pairing den Generator $h := \mathrm{h}$ und das Element $X := \mathrm{h}^x$.

Anschließend berechnet $\mathcal{B}$ die Elemente $u_i = u_0^{\alpha^i}$ sowie $h_i = h^{\alpha^i}$ für $1 \leq i \leq K$ und sendet den perfekt simulierten öffentlichen Schlüssel $\mathsf{pk} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_0, \ldots, g_L, u_0, \ldots, u_K, h, h_1, \ldots, h_K, X)$ an den Angreifer $\mathcal{A}$.

Sign **Anfragen:** Sei $m_j = (m_{1,j}, \ldots, m_{L,j}, \phi_j(\mathsf{x})) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}]$ die j-te Anfrage von $\mathcal{A}$ für $1 \leq j \leq q_S$. Der Angreifer $\mathcal{B}$ setzt $z_j := s_j + \mu_1 m_{1,j} + \cdots + \mu_L m_{L,j} + \mu_{L+1}\phi_j(\alpha) \mod p$ und kann somit analog zu [Sch11] jede Signatur σ_j für $1 \leq j \leq q_S$ perfekt simulieren, indem $\mathcal{B}$ das Element

$$\Sigma_j = \mathrm{g}^{(r_0 + r_1 z_j)\bar{e}_j} = \left(g g_0^{z_j}\right)^{\frac{1}{x+e_j}} = \left(g g_0^{s_j} g_1^{m_{1,j}} \cdots g_L^{m_{L,j}} u_0^{\phi_j(\alpha)}\right)^{\frac{1}{x+e_j}}$$

berechnet und die Signatur $\sigma_j = (\Sigma_j, e_j, s_j)$ an den Angreifer $\mathcal{A}$ sendet.

Ausgabe: Sei $((m_1^\star, \ldots, m_L^\star, \phi^\star(\mathsf{x})), (\Sigma^\star, e^\star, s^\star,))$ das gefälschte BBS+A-Nachrichten-Signatur-Paar von $\mathcal{A}$. Der Angreifer $\mathcal{B}$ setzt $z^\star := s^\star + \mu_1 m_1^\star + \cdots + \mu_L m_L^\star + \mu_{L+1}\phi^\star(\alpha) \mod p$, so dass

$$\hat{e}\left(\Sigma^\star, X h^{e^\star}\right) = \hat{e}\left(g g_0^{s^\star} g_1^{m_1^\star} \cdots g_L^{m_L^\star} u_0^{\phi^\star(\alpha)}, h\right) = \hat{e}\left(g g_0^{z^\star}, h\right)$$

gilt. Dies ist äquivalent zu $\Sigma^\star = \left(g g_0^{z^\star}\right)^{\frac{1}{x+e^\star}} = \mathrm{g}^{\frac{\bar{e}(r_0 + r_1 z^\star)}{x+e^\star}}$. Da nach Voraussetzung $e^\star \notin \{e_1, \ldots, e_{q_S}\}$ gilt, ist das Polynom $\phi_{\bar{e}}(\mathsf{x})$ nicht ohne Rest durch das Polynom $\mathsf{x} + e^\star$ teilbar. Der Angreifer $\mathcal{B}$ berechnet nun die Zahl $R \in \mathbb{Z}_p^*$ und das Polynom $f(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ vom Grad $q_S - 1$ mit $\phi_{\bar{e}}(\mathsf{x}) = f(\mathsf{x})(\mathsf{x} + e^\star) + R$, woraus $\bar{e} = f(x)(x + e^\star) + R \mod p$ folgt. Weiter ist nur mit vernachlässigbarer Wahrscheinlichkeit das Produkt $g g_0^{s^\star} g_1^{m_1^\star} \cdots g_L^{m_L^\star} u_0^{\phi^\star(\alpha)} = 1$, da $\mathcal{A}$ ansonsten eine nichttriviale Darstellung der 1 bzgl. des Generatortupels $(g, g_0, \ldots, g_L, u_0)$ ausgegeben hätte. Folglich ist, außer mit vernachlässigbarer Wahrscheinlichkeit, $r_0 + r_1 z^\star \neq 0$ und kann somit in $\mathbb{Z}_p^*$ invertiert werden. Anschließend berechnet $\mathcal{B}$ das Element

$$\mathrm{g}^{\frac{1}{x+e^\star}} = \left((\Sigma^\star)^{\frac{1}{r_0 + r_1 z^\star}} \mathrm{g}^{-f(x)}\right)^{\frac{1}{R}}$$

und löst schließlich analog zu [Sch11] das q-SDH-Problem mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist.

Fall 2: $e^\star = e_k$ und $s^\star \neq s_k$ für ein $k \in \{1, \ldots, q_S\}$:

Setup: Der Angreifer $\mathcal{B}$ wählt zufällig q_S paarweise verschiedene Zahlen $e_1, \ldots, e_{q_S} \in_{\mathcal{R}} \mathbb{Z}_p$ und zufällig die Zahlen $s_1, \ldots, s_{q_S} \in_{\mathcal{R}} \mathbb{Z}_p$ sowie $r_0, r_1, r_2, \alpha \in_{\mathcal{R}} \mathbb{Z}_p^*$. Seien $\bar{e}, \bar{e}_i$ und $\bar{e}_{i,j}$ wie oben definiert. Dann berechnet $\mathcal{B}$ die Generatoren $g = \mathrm{g}^{r_0\bar{e} + \sum_{i=1}^{q_S} s_i \bar{e}_i}$, $g_0 = \mathrm{g}^{r_1\bar{e} - \sum_{i=1}^{q_S} \bar{e}_i}$ und $g_1 = \mathrm{g}^{r_2\bar{e}}$. (Falls $g = 1$, $g_0 = 1$ oder $g_1 = 1$ wäre, könnte $\mathcal{B}$ erneut

die Zahl x berechnen, da diese eine Nullstelle des entsprechenden Polynoms ist, vgl. auch [KZG10a, KZG10b]). Weiter wählt $\mathcal{B}$ zufällig L paarweise verschiedene Zahlen $\mu_2, \ldots, \mu_{L+1} \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Generatoren $g_2 = g_1^{\mu_2}, \ldots, g_L = g_1^{\mu_L}$ sowie $u_0 = g_1^{\mu_{L+1}}$. Dann

- berechnet $\mathcal{B}$ im Typ-1- bzw. Typ-2-Pairing den Generator $h = \mathrm{h}^{r_2 \bar{e} \mu_{L+1}}$ und das Element $X = (\mathrm{h}^x)^{r_2 \bar{e} \mu_{L+1}}$, so dass die Beziehungen $u_0 = h$ bzw. $u_0 = \psi(h)$ und $X = h^x$ gelten, wobei das Element X nur für $q_S \leq q-1$ effizient berechnet werden kann.
- definiert $\mathcal{B}$ im Typ-3-Pairing den Generator $h := \mathrm{h}$ und das Element $X := \mathrm{h}^x$.

Anschließend berechnet $\mathcal{B}$ die Elemente $u_i = u_0^{\alpha^i}$ sowie $h_i = h^{\alpha^i}$ für $1 \leq i \leq K$ und sendet den perfekt simulierten öffentlichen Schlüssel $\mathsf{pk} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_0, \ldots, g_L, u_0, \ldots, u_K, h, h_1, \ldots, h_K, X)$ an den Angreifer $\mathcal{A}$.

Sign **Anfragen:** Sei $m_j = (m_{1,j}, \ldots, m_{L,j}, \phi_j(\mathsf{x})) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}]$ die j-te Anfrage von $\mathcal{A}$ für $1 \leq j \leq q_S$. Der Angreifer $\mathcal{B}$ setzt $z_j := m_{1,j} + \mu_2 m_{2,j} + \cdots + \mu_L m_{L,j} + \mu_{L+1}\phi_j(\alpha) \mod p$ und kann somit analog zu [Sch11] jede Signatur σ_j für $1 \leq j \leq q_S$ rechnerisch ununterscheidbar simulieren, indem $\mathcal{B}$ das Element

$$\begin{aligned}\Sigma_j &= \mathrm{g}^{(r_0 + r_1 s_j + r_2 z_j)\bar{e}_j + \sum_{i=1, i\neq j}^{q_S}(s_i - s_j)\bar{e}_{i,j}} \\ &= \left(g g_0^{s_j} g_1^{z_j}\right)^{\frac{1}{x+e_j}} = \left(g g_0^{s_j} g_1^{m_{1,j}} \cdots g_L^{m_{L,j}} u_0^{\phi_j(\alpha)}\right)^{\frac{1}{x+e_j}}\end{aligned}$$

berechnet und die Signatur $\sigma_j = (\Sigma_j, e_j, s_j)$ an den Angreifer $\mathcal{A}$ sendet. Die Signaturen sind nicht perfekt simuliert, da die Zahlen $e_1, \ldots, e_{q_S}$ paarweise verschieden und somit nicht gemäß dem Algorithmus Sign gewählt sind.

Ausgabe: Sei $((m_1^\star, \ldots, m_L^\star, \phi^\star(\mathsf{x})), (\Sigma^\star, e^\star, s^\star,))$ das gefälschte BBS+A-Nachrichten-Signatur-Paar von $\mathcal{A}$. Der Angreifer $\mathcal{B}$ setzt $z^\star := m_1^\star + \mu_2 m_2^\star + \cdots + \mu_L m_L^\star + \mu_{L+1}\phi^\star(\alpha) \mod p$, so dass

$$\hat{e}\left(\Sigma^\star, X h^{e^\star}\right) = \hat{e}\left(g g_0^{s^\star} g_1^{m_1^\star} \cdots g_L^{m_L^\star} u_0^{\phi^\star(\alpha)}, h\right) = \hat{e}\left(g g_0^{s^\star} g_1^{z^\star}, h\right)$$

gilt. Dies ist äquivalent zu $\Sigma^\star = \left(g g_0^{s^\star} g_1^{z^\star}\right)^{\frac{1}{x+e^\star}} = \mathrm{g}^{\frac{(r_0 + r_1 s^\star + r_2 z^\star)\bar{e} + \sum_{i=1}^{q_S}(s_i - s^\star)\bar{e}_i}{x+e^\star}}$. Nach Voraussetzung gibt es ein $k \in \{1, \ldots, q_S\}$ mit $e^\star = e_k$ und $s^\star \neq s_k$. Daraus folgt weiter:

$$\Sigma^\star = \mathrm{g}^{\frac{(r_0 + r_1 s^\star + r_2 z^\star)\bar{e} + \sum_{i=1, i\neq k}^{q_S}(s_i - s^\star)\bar{e}_i + (s_k - s^\star)\bar{e}_k}{x+e_k}}.$$

Da alle $e_1, \ldots, e_{q_S}$ paarweise verschieden sind, ist das Polynom $\phi_{\bar{e}_k}(\mathsf{x})$ nicht ohne Rest durch das Polynom $\mathsf{x} + e_k$ teilbar. Der Angreifer $\mathcal{B}$ berechnet nun die Zahl $R \in \mathbb{Z}_p^*$ und das Polynom $f(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ vom Grad $q_S - 2$ mit $\phi_{\bar{e}_k}(\mathsf{x}) = f(\mathsf{x})(\mathsf{x} + e_k) + R$,

woraus $\bar{e}_k = f(x)(x + e_k) + R \mod p$ folgt. Da weiter $s^\star \neq s_k$ ist, kann $(s_k - s^\star)$ in $\mathbb{Z}_p^*$ invertiert werden. Somit berechnet $\mathcal{B}$ das Element

$$\mathrm{g}^{\frac{1}{x+e_k}} = \left(\Sigma^\star \mathrm{g}^{-\left((r_0+r_1 s^\star+r_2 z^\star)\bar{e}_k+\sum_{i=1,i\neq k}^{q_S}(s_i-s^\star)\bar{e}_{i,k}+(s_k-s^\star)f(x)\right)}\right)^{\frac{1}{R(s_k-s^\star)}}$$

und löst schließlich analog zu [Sch11] das q-SDH-Problem mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist.

Fall 3: $e^\star = e_k$ und $s^\star = s_k$ für ein $k \in \{1, \ldots, q_S\}$:

Setup: Der Angreifer $\mathcal{B}$ wählt zufällig q_S paarweise verschiedene Zahlen $e_1, \ldots, e_{q_S} \in_{\mathcal{R}} \mathbb{Z}_p$ und zufällig die Zahlen $z_1, \ldots, z_{q_S} \in_{\mathcal{R}} \mathbb{Z}_p$ sowie $r_0, r_1, \alpha \in_{\mathcal{R}} \mathbb{Z}_p^*$. Seien $\bar{e}, \bar{e}_i$ und $\bar{e}_{i,j}$ wie oben definiert. Dann berechnet $\mathcal{B}$ die Generatoren $g = \mathrm{g}^{r_0\bar{e}+\sum_{i=1}^{q_S} z_i\bar{e}_i}$ und $g_0 = \mathrm{g}^{r_1\bar{e}-\sum_{i=1}^{q_S}\bar{e}_i}$. (Erneut könnte $\mathcal{B}$ das q-SDH-Problem direkt lösen, falls $g = 1$ oder $g_0 = 1$ wäre.) Weiter wählt $\mathcal{B}$ zufällig $L + 1$ paarweise verschiedene Zahlen $\mu_1, \ldots, \mu_{L+1} \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Generatoren $g_1 = g_0^{\mu_1}, \ldots, g_L = g_0^{\mu_L}$ sowie $u_0 = g_0^{\mu_{L+1}}$. Dann

- berechnet $\mathcal{B}$ im Typ-1- bzw. Typ-2-Pairing den Generator $h = \mathrm{h}^{\mu_{L+1}\left(r_1\bar{e}-\sum_{i=1}^{q_S}\bar{e}_i\right)}$ und das Element $X = (\mathrm{h}^x)^{\mu_{L+1}\left(r_1\bar{e}-\sum_{i=1}^{q_S}\bar{e}_i\right)}$, so dass die Beziehungen $u_0 = h$ bzw. $u_0 = \psi(h)$ und $X = h^x$ gelten, wobei das Element X nur für $q_S \leq q - 1$ effizient berechnet werden kann.
- definiert $\mathcal{B}$ im Typ-3-Pairing den Generator $h := \mathrm{h}$ und das Element $X := \mathrm{h}^x$.

Anschließend berechnet $\mathcal{B}$ die Elemente $u_i = u_0^{\alpha^i}$ sowie $h_i = h^{\alpha^i}$ für $1 \leq i \leq K$ und sendet den perfekt simulierten öffentlichen Schlüssel $\mathsf{pk} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_0, \ldots, g_L, u_0, \ldots, u_K, h, h_1, \ldots, h_K, X)$ an den Angreifer $\mathcal{A}$.

Sign Anfragen: Sei $(m_{1,j}, \ldots, m_{L,j}, \phi_j(\mathsf{x})) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}]$ die j-te Anfrage von $\mathcal{A}$ für $1 \leq j \leq q_S$. Der Angreifer $\mathcal{B}$ berechnet die Zahl $s_j \in \mathbb{Z}_p$ mit $z_j = s_j + \mu_1 m_{1,j} + \cdots + \mu_L m_{L,j} + \mu_{L+1}\phi_j(\alpha) \mod p$ und kann somit analog zu [Sch11] jede Signatur σ_j für $1 \leq j \leq q_S$ rechnerisch ununterscheidbar simulieren, indem $\mathcal{B}$ das Element

$$\Sigma_j = \mathrm{g}^{(r_0+r_1 z_j)\bar{e}_j+\sum_{i=1,i\neq j}^{q_S}(z_i-z_j)\bar{e}_{i,j}} = \left(g g_0^{z_j}\right)^{\frac{1}{x+e_j}} = \left(g g_0^{s_j} g_1^{m_{1,j}} \cdots g_L^{m_{L,j}} u_0^{\phi_j(\alpha)}\right)^{\frac{1}{x+e_j}}$$

berechnet und die Signatur $\sigma_j = (\Sigma_j, e_j, s_j)$ an den Angreifer $\mathcal{A}$ sendet. Erneut sind die Signaturen nicht perfekt simuliert, da die Zahlen $e_1, \ldots, e_{q_S}$ paarweise verschieden und somit nicht gemäß dem Algorithmus Sign gewählt sind.

Ausgabe: Sei $m_j = ((m_1^\star, \ldots, m_L^\star, \phi^\star(\mathsf{x})), (\Sigma^\star, e^\star, s^\star,))$ das gefälschte BBS+A-Nachrichten-Signatur-Paar von $\mathcal{A}$. Der Angreifer $\mathcal{B}$ setzt $z^\star := s^\star + \mu_1 m_1^\star + \cdots + \mu_L m_L^\star + \mu_{L+1}\phi^\star(\alpha) \mod p$, so dass

$$\hat{e}\left(\Sigma^\star, Xh^{e^\star}\right) = \hat{e}\left(gg_0^{s^\star} g_1^{m_1^\star} \cdots g_L^{m_L^\star} u_0^{\phi^\star(\alpha)}, h\right) = \hat{e}\left(gg_1^{z^\star}, h\right)$$

gilt. Dies ist äquivalent zu $\Sigma^\star = \left(gg_0^{z^\star}\right)^{\frac{1}{x+e^\star}} = \mathrm{g}^{\frac{(r_0+r_1z^\star)\bar{e}+\sum_{i=1}^{q_S}(z_i-z^\star)\bar{e}_i}{x+e^\star}}$. Nach Voraussetzung gibt es ein $k \in \{1, \ldots, q_S\}$ mit $e^\star = e_k$ und $s^\star = s_k$. Daraus folgt weiter

$$\Sigma^\star = \mathrm{g}^{\frac{(r_0+r_1z^\star)\bar{e}+\sum_{i=1,i\neq k}^{q_S}(z_i-z^\star)\bar{e}_i+(z_k-z^\star)\bar{e}_k}{x+e_k}}.$$

Weiter muss $(m_{1,k}, \ldots, m_{L,k}, \phi_k(\mathsf{x})) \neq (m_1^\star, \ldots, m_L^\star, \phi^\star(\mathsf{x}))$ sein, denn sonst wäre $\Sigma^\star = \Sigma_k$ und der Angreifer $\mathcal{A}$ hätte somit keine Signatur gefälscht, sondern lediglich ein erhaltenes Nachrichten-Signatur-Paar ausgegeben. Weiter gilt nur mit vernachlässigbarer Wahrscheinlichkeit $z^\star = z_k$, da sonst $g_0^{m_1^\star} \cdots g_L^{m_L^\star} u_0^{\phi^\star(\alpha)} = g_0^{m_{1,k}} \cdots g_L^{m_{L,k}} u_0^{\phi_k(\alpha)}$ gilt, was ein Widerspruch zur Gebundenheit des Commitment-Schemas ist (siehe Lemma 3.1.2). Da alle $e_1, \ldots, e_{q_S}$ paarweise verschieden sind, ist das Polynom $\phi_{\bar{e}_k}(\mathsf{x})$ nicht ohne Rest durch das Polynom $\mathsf{x} + e_k$ teilbar. Der Angreifer $\mathcal{B}$ berechnet nun die Zahl $R \in \mathbb{Z}_p^*$ und das Polynom $f(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ vom Grad $q_S - 2$ mit $\phi_{\bar{e}_k}(\mathsf{x}) = f(\mathsf{x})(\mathsf{x} + e_k) + R$, woraus $\bar{e}_k = f(x)(x + e_k) + R \mod p$ folgt. Somit berechnet $\mathcal{B}$ das Element

$$\mathrm{g}^{\frac{1}{x+e_k}} = \left(\Sigma^\star \mathrm{g}^{-\left((r_0+r_1z^\star)\bar{e}_k+\sum_{i=1,i\neq k}^{q_S}(z_i-z^\star)\bar{e}_{i,k}+(z_k-z^\star)f(x)\right)}\right)^{\frac{1}{R(z_k-z^\star)}}$$

und löst schließlich analog zu [Sch11] das q-SDH-Problem mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist.

□

Für das elektronische Geldsystem in Abschnitt 9.2 benötigen wir folgendes Korollar:

Korollar 3.2.1. *Das obige Lemma 3.2.5 gilt auch dann, wenn der öffentliche Schlüssel* pk *zusätzlich einen weiteren, zufälligen Generator* $g_{\mathcal{O}} \in_{\mathcal{R}} \mathbb{G}_1$ *und das Element* $g_{\mathcal{O}}^x \in \mathbb{G}_1$ *enthält.*

Beweis. Der Angreifer $\mathcal{B}$ wählt zusätzlich in der Setup-Phase zufällig eine Zahl $\mu_{\mathcal{O}} \in_{\mathcal{R}} \mathbb{Z}_p^* \setminus \{\mu_1, \ldots, \mu_{L+1}\}$ und berechnet den Generator $g_{\mathcal{O}} = \mathrm{g}^{\mu_{\mathcal{O}}}$ sowie das Element $g_{\mathcal{O}}^x = (\mathrm{g}^x)^{\mu_{\mathcal{O}}}$. Ansonsten geht $\mathcal{B}$ wie im obigen Beweis zu Lemma 3.2.5 vor. □

Nun wird gezeigt, dass die Signaturbeweisprotokolle $\mathsf{Prove}, \mathsf{Prove}\text{-}k, \mathsf{Prove}^*\text{-}k$ sowie $\mathsf{Prove}\text{-}K$ durchführbar, valide und perfekt honest-verifier-zero-knowledge sind.

Lemma 3.2.6. *Das Signaturbeweisprotokoll* Prove *ist im Random-Oracle-Modell ein perfektes Zero-Knowledge-Proof-of-Knowledge-Protokoll über ein Nachrichten-Signatur-Paar mit vernachlässigbarer Knowledge-Error-Funktion κ, falls die q-SDH-Annahme und die q-polyDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gelten.*

Beweis. Wir beweisen zunächst die Durchführbarkeit, dann die Zero-Knowledge-Eigenschaft und anschließend die Beweiskraft.

Durchführbarkeit: Folgt aus der Durchführbarkeit von SoK (vgl. Abschnitt A.2).

Zero-Knowledge: Der Simulator $\mathcal{SIM}$ wählt zufällig die Elemente $D_m, D_n, D_\sigma, A_1, A_2, B_1, B_2 \in_{\mathcal{R}} \mathbb{G}_1$ sowie den Hashwert $l \in_{\mathcal{R}} \mathbb{Z}_p$ und simuliert die Signature-of-Knowledge SoK (vgl. Abschnitt A.2). Somit hat $\mathcal{SIM}$ das Protokoll perfekt simuliert.

Beweiskraft: Wir müssen zeigen, dass es einen probabilistischen Extractor $\mathcal{E}$ gibt, der den User $\mathcal{U}^*$ als zurückspulbare Blackbox verwendet und für $|x| = \lambda$ und eine Konstante $c > 0$ nach einer erwarteten Laufzeit von $\frac{\lambda^c}{\epsilon(\lambda) - \kappa(\lambda)}$ Schritten ein gültiges Nachrichten-Signatur-Paar extrahiert, falls der Verifier $\mathcal{V}$ mit Wahrscheinlichkeit $\epsilon(\lambda) > \kappa(\lambda)$ akzeptiert (vgl. Definition 2.10.3), wobei $\mathcal{E}$ die Rolle des Random-Oracles übernimmt.

Der Extractor $\mathcal{E}$ erhält als Eingabe das Tripel (pk, D_m, D_n) und muss den User $\mathcal{U}^*$ mehrmals zurückspulen, um das Nachrichten-Signatur-Paar zu extrahieren. Wir folgen der Notation von [KLL06], d.h. wir bezeichnen mit A eine binäre Matrix und mit ω die Zufallswerte des Protokolls. Dann setzen wir $A(\omega) = 1$, falls die Signature-of-Knowledge SoK akzeptiert wird und andernfalls $A(\omega) = 0$. Sei $\epsilon^*(\lambda) \geq \epsilon(\lambda)$ die Wahrscheinlichkeit, mit der die Signature-of-Knowledge SoK akzeptiert wird. Dann ist $\epsilon^*(\lambda)$ der Anteil der Matrix A mit $A(\omega) = 1$. Da der Extractor $\mathcal{E}$ den User mehrmals zurückspulen und verschiedene Werte erhalten muss, gibt es nicht nur eine, sondern K binäre Matrizen $A^{(1)}, \ldots, A^{(K)}$, wobei bei jeder Matrix $A^{(j)}$ ein anderer Zufallswert $l^{(j)} \in_{\mathcal{R}} \mathbb{Z}_p$ mit $l^{(j)} \neq l^{(i)}$ für $i \neq j$ und $1 \leq i, j \leq K$ als Ausgabe des Random-Oracles gewählt wird. Dadurch wird modelliert, dass der User $\mathcal{U}^*$ jeweils zu dem Zeitpunkt zurückgespult wird, bei dem $\mathcal{U}^*$ beim Random-Oracle den Hashwert für $\mathsf{H}(D_m||D_n||D_\sigma||B_1||B_2)$ anfragt und jeweils einen anderen Hashwert $l^{(j)}$ für $1 \leq j \leq K$ als Antwort erhält.

Für $\epsilon^*(\lambda) > \kappa_0(\lambda)$ besitzt der in [KLL06] entwickelte Rewind-Algorithmus eine erwartete Laufzeit von $2/(\epsilon^*(\lambda) - \kappa_0(\lambda))$ Schritten, um die Werte einer Signature-of-Knowledge zu extrahieren. Dabei bezeichnet κ_0 (auch im Folgenden) die Knowledge-Error-Funktion eines Proof-of-Knowledge-Systems und ist damit bei einer Signature-of-Knowledge eine vernachlässigbare Funktion (vgl. Unterabschnitt 2.10.1).

Somit kann $\mathcal{E}$ nach voraussichtlich $2K/(\epsilon^*(\lambda) - \kappa_0(\lambda))$ Schritten die Zahlen $r_m^{(j)}, r_n^{(j)}, r_\sigma^{(j)}, r_a^{(j)}, a_1^{(j)}, \delta_1^{(j)}, \delta_2^{(j)}, \delta_r^{(j)}, b_1^{(j)}, b_2^{(j)}, \beta_1^{(j)}, \beta_2^{(j)}, e^{(j)}, s^{(j)}, m_1^{(j)}, \ldots, m_L^{(j)}, n_1^{(j)}, \phi_1\left(l^{(j)}\right) \in \mathbb{Z}_p$ für $1 \leq j \leq K$ aus der Signature-of-Knowledge SoK extrahieren.

Falls $\left(e^{(j)}, s^{(j)}, m_1^{(j)}, \ldots, m_L^{(j)}, n_1^{(j)}, b_1^{(j)}\right) \neq \left(e^{(1)}, s^{(1)}, m_1^{(1)}, \ldots, m_L^{(1)}, n_1^{(1)}, b_1^{(1)}\right)$, $\delta_1^{(j)} \neq a_1^{(j)} n_1^{(j)}, \delta_2^{(j)} \neq \phi_1\left(l^{(j)}\right) n_1^{(j)}$ oder $\beta_1^{(j)} \neq b_1^{(j)} e^{(j)}$ für ein $j \in [1, K]$ ist, bricht $\mathcal{E}$ ab. Dieser Fall tritt allerdings nur mit vernachlässigbarer Wahrscheinlichkeit ein, da $\mathcal{U}^*$ ansonsten die Gebundenheit des Pedersen-Commitments bzgl. D_m, D_n, D_σ, A_2 oder B_1 gebrochen hätte.

Zur Vereinfachung sei $(e, s, m_1, \ldots, m_L, n_1, b_1) := \left(e^{(1)}, s^{(1)}, m_1^{(1)}, \ldots, m_L^{(1)}, n_1^{(1)}, b_1^{(1)}\right)$. Seien weiter $W_{1,l^{(j)}} := A_1^{(j)} u_0^{-a_1^{(j)}}$ sowie $\Sigma := B_2 g_1^{-b_1}$. Dann folgt mit der letzten Gleichung der Signature-of-Knowledge SoK für alle $1 \leq j \leq K$:

$$\begin{aligned}
&\frac{\hat{e}\left(B_2, X\right)}{\hat{e}\left(g, h\right) \hat{e}\left(A_1^{(j)}, h_2 h_1^{-l^{(j)}}\right)} = \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(\left(A_1^{(j)}\right)^{n_1}, h_1\right) \hat{e}\Big(g_0^s g_1^{m_1+\beta_1} g_2^{m_2} \cdots g_L^{m_L} \\
&\qquad u_0^{l^{(j)}\delta_1^{(j)}+\delta_2^{(j)}} u_1^{l^{(j)}a_1^{(j)}-\delta_1^{(j)}+\phi_1\left(l^{(j)}\right)} u_2^{-a_1^{(j)}} \left(A_1^{(j)}\right)^{-l^{(j)}n_1} B_2^{-e}, h\Big) \\
\Leftrightarrow \quad &\frac{\hat{e}\left(\Sigma g_1^{b_1}, X\right)}{\hat{e}\left(W_{1,l^{(j)}} u_0^{a_1^{(j)}}, h_2 h_1^{-l^{(j)}}\right)} = \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(\left(W_{1,l^{(j)}} u_0^{a_1^{(j)}}\right)^{n_1}, h_1\right) \hat{e}\Big(g g_0^s g_1^{m_1+b_1 e} g_2^{m_2} \cdots \\
&\qquad g_L^{m_L} u_0^{l^{(j)}a_1^{(j)}n_1+\phi_1(l^{(j)})n_1} u_1^{l^{(j)}a_1^{(j)}-a_1^{(j)}n_1+\phi_1\left(l^{(j)}\right)} \\
&\qquad u_2^{-a_1^{(j)}} \left(W_{1,l^{(j)}} u_0^{a_1^{(j)}}\right)^{-l^{(j)}n_1} \left(\Sigma g_1^{b_1}\right)^{-e}, h\Big) \\
\Leftrightarrow \quad &\hat{e}\left(\Sigma, X h^e\right) = \hat{e}\left(g g_0^s g_1^{m_1} \cdots g_L^{m_L}, h\right) \hat{e}\left(W_{1,l^{(j)}}^{\alpha-l^{(j)}} u_0^{\phi_1\left(l^{(j)}\right)}, h^{\alpha+n_1}\right). \qquad (3.18)
\end{aligned}$$

Sei nun $\phi_1(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ ein Polynom mit $\hat{e}\left(\Sigma, X h^e\right) = \hat{e}\left(g g_0^s g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi_1(\alpha)(\alpha+n_1)}\right)$. Dann folgt mit (3.18) für alle $1 \leq j \leq K$ die Gleichung $u_0^{\phi_1(\alpha)} = W_{1,l^{(j)}}^{\alpha-l^{(j)}} u_0^{\phi_1\left(l^{(j)}\right)}$ und mit Lemma 3.2.1 folgt, dass, außer mit vernachlässigbarer Wahrscheinlichkeit, $\phi_1\left(l^{(j)}\right)$ die korrekte Berechnung des Polynoms $\phi_1(\mathsf{x})$ an der Stelle $l^{(j)}$ ist. Weiter folgt mit Lemma 3.2.4, dass, außer mit vernachlässigbarer Wahrscheinlichkeit, $\deg(\phi_1) \leq K - 1$ ist.

Somit kennt der Extractor $\mathcal{E}$ insgesamt K Paare $\left(l^{(1)}, \phi_1\left(l^{(1)}\right)\right), \ldots, \left(l^{(K)}, \phi_1\left(l^{(K)}\right)\right)$ des Polynoms $\phi_1(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ mit $\deg(\phi_1) \leq K - 1$ und kann dieses interpolieren. Anschließend berechnet $\mathcal{E}$ das Polynom $\phi(\mathsf{x}) = \phi_1(\mathsf{x})(\mathsf{x} + n_1) \in \mathbb{Z}_p[\mathsf{x}]$. Dann folgt $\deg(\phi) \leq K$ und weiter

$$\hat{e}\left(\Sigma, X h^e\right) = \hat{e}\left(g g_0^s g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}, h\right).$$

Folglich ist $\left((m_1, \ldots, m_L, \phi(\mathsf{x})), (\Sigma, e, s)\right)$ ein gültiges Nachrichten-Signatur-Paar.

Somit ist $\kappa(\lambda) = \kappa_0(\lambda) + \nu(\lambda)$ der Anteil der Matrizen mit $A^{(j)}(\omega) = 1$ für $1 \leq j \leq K$, bei denen $\mathcal{E}$ aber dennoch kein gültiges Nachrichten-Signatur-Paar extrahieren kann, wobei ν eine vernachlässigbare Funktion ist. Insbesondere ist damit κ eine vernach-

lässigbare Funktion. Insgesamt benötigt $\mathcal{E}$ also voraussichtlich $2K/\left(\epsilon^*(\lambda)-\kappa(\lambda)\right) \leq 2K/\left(\epsilon(\lambda)-\kappa(\lambda)\right)$ Schritte, um das Nachrichten-Signatur-Paar zu extrahieren. □

Lemma 3.2.7. *Das Signaturbeweisprotokoll* Prove-k *ist im Random-Oracle-Modell ein perfektes Zero-Knowledge-Proof-of-Knowledge-Protokoll über ein Nachrichten-Signatur-Paar mit vernachlässigbarer Knowledge-Error-Funktion κ, falls die q-SDH-Annahme und die q-polyDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gelten.*

Beweis. Wir beweisen erneut Durchführbarkeit, Zero-Knowledge-Eigenschaft sowie Beweiskraft.

Durchführbarkeit: Folgt aus der Durchführbarkeit von SoK (vgl. Abschnitt A.3).

Zero-Knowledge: Der Simulator $\mathcal{SIM}$ wählt zufällig die Elemente $D_m, D_\sigma, A_1, B_1, B_2 \in_{\mathcal{R}} \mathbb{G}_1$ sowie den Hashwert $l \in_{\mathcal{R}} \mathbb{Z}_p$ und simuliert die Signature-of-Knowledge SoK (vgl. Abschnitt A.3). Somit hat $\mathcal{SIM}$ das Protokoll perfekt simuliert.

Beweiskraft: Der Extractor $\mathcal{E}$ benötigt analog zum Prove-Protokoll $K-k+1$ binäre Matrizen und kann analog nach voraussichtlich $2\left(K-k+1\right)/\left(\epsilon^*(\lambda)-\kappa_0(\lambda)\right)$ Schritten die Zahlen $r_m^{(j)}, r_\sigma^{(j)}, a_1^{(j)}, b_1^{(j)}, b_2^{(j)}, \beta_1^{(j)}, \beta_2^{(j)}, e^{(j)}, s^{(j)}, m_1^{(j)}, \ldots, m_L^{(j)}, \phi_I\left(l^{(j)}\right) \in \mathbb{Z}_p$ für $1 \leq j \leq K-k+1$ extrahieren.

Falls $\left(e^{(j)}, s^{(j)}, m_1^{(j)}, \ldots, m_L^{(j)}, b_1^{(j)}\right) \neq \left(e^{(1)}, s^{(1)}, m_1^{(1)}, \ldots, m_L^{(1)}, b_1^{(1)}\right)$ oder $\beta_1^{(j)} \neq b_1^{(j)} e^{(j)}$ für ein $j \in [1, K-k+1]$ ist, bricht $\mathcal{E}$ ab. Dieser Fall tritt allerdings nur mit vernachlässigbarer Wahrscheinlichkeit ein, da $\mathcal{U}^*$ ansonsten die Gebundenheit des Pedersen-Commitments bzgl. D_m, D_σ oder B_1 gebrochen hätte.

Zur Vereinfachung sei $(e, s, m_1, \ldots, m_L, b_1) := \left(e^{(1)}, s^{(1)}, m_1^{(1)}, \ldots, m_L^{(1)}, b_1^{(1)}\right)$. Seien weiter $W_{I,l^{(j)}} := A_1^{(j)} u_0^{-a_1^{(j)}}$ sowie $\Sigma := B_2 g_1^{-b_1}$. Dann folgt mit der letzten Gleichung der Signature-of-Knowledge SoK für alle $1 \leq j \leq K-k+1$:

$$\frac{\hat{e}\left(B_2, X\right)}{\hat{e}\left(g,h\right)\hat{e}\left(A_1^{(j)}, h_{l^{(j)},I}\right)} = \hat{e}\left(g_1^{b_1}, X\right)\hat{e}\left(g_0^s g_1^{m_1+\beta_1} g_2^{m_2} \ldots g_L^{m_L} u_{l^{(j)},I}^{-a_1^{(j)}} u_I^{\phi_I\left(l^{(j)}\right)} B_2^{-e}, h\right)$$

$$\Leftrightarrow \quad \frac{\hat{e}\left(\Sigma g_1^{b_1}, X\right)}{\hat{e}\left(W_{I,l^{(j)}} u_0^{a_1^{(j)}}, h_{l^{(j)},I}\right)} = \hat{e}\left(g_1^{b_1}, X\right)\hat{e}\left(g g_0^s g_1^{m_1+b_1 e} g_2^{m_2} \ldots g_L^{m_L} u_{l^{(j)},I}^{-a_1^{(j)}} u_I^{\phi_I\left(l^{(j)}\right)} \left(\Sigma g_1^{b_1}\right)^{-e}, h\right)$$

$$\Leftrightarrow \quad \hat{e}\left(\Sigma, X h^e\right) = \hat{e}\left(g g_0^s g_1^{m_1} \ldots g_L^{m_L}, h\right)\hat{e}\left(W_{I,l^{(j)}}^{\alpha-l^{(j)}} u_0^{\phi_I\left(l^{(j)}\right)}, h_I\right). \tag{3.19}$$

Sei nun $\phi_I\left(\mathsf{x}\right) \in \mathbb{Z}_p\left[\mathsf{x}\right]$ ein Polynom mit $\hat{e}\left(\Sigma, X h^e\right) = \hat{e}\left(g g_0^s g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi_I(\alpha)\prod_{j=1}^{k}(\alpha+n_j)}\right)$.

Dann folgt mit (3.19) für alle $1 \leq j \leq K-k+1$ die Gleichung $u_0^{\phi_I(\alpha)} = W_{I,l^{(j)}}^{\alpha-l^{(j)}} u_0^{\phi_I\left(l^{(j)}\right)}$ und mit Lemma 3.2.1 folgt, dass, außer mit vernachlässigbarer Wahrscheinlichkeit, $\phi_I\left(l^{(j)}\right)$

die korrekte Berechnung des Polynoms $\phi_I(\mathsf{x})$ an der Stelle $l^{(j)}$ ist. Weiter folgt mit Lemma 3.2.4, dass, außer mit vernachlässigbarer Wahrscheinlichkeit, $\deg(\phi_I) \leq K-k$ ist.

Somit kennt der Extractor $\mathcal{E}$ insgesamt $K-k+1$ Paare $\left(l^{(1)}, \phi_I\left(l^{(1)}\right)\right), \ldots, \left(l^{(K-k+1)}, \phi_I\left(l^{(K-k+1)}\right)\right)$ des Polynoms $\phi_I(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ mit $\deg(\phi_I) \leq K-k$ und kann dieses interpolieren. Anschließend berechnet $\mathcal{E}$ das Polynom $\phi(\mathsf{x}) = \phi_I(\mathsf{x}) \prod_{j=1}^{k}(\mathsf{x}+n_j) \in \mathbb{Z}_p[\mathsf{x}]$. Dann folgt $\deg(\phi) \leq K$ und weiter

$$\hat{e}(\Sigma, Xh^e) = \hat{e}\left(gg_0^s g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}, h\right).$$

Folglich ist $((m_1, \ldots, m_L, \phi(\mathsf{x})), (\Sigma, e, s))$ ein gültiges Nachrichten-Signatur-Paar.

Somit ist $\kappa(\lambda) = \kappa_0(\lambda) + \nu(\lambda)$ der Anteil der Matrizen mit $A^{(j)}(\omega) = 1$ für $1 \leq j \leq K-k+1$, bei denen $\mathcal{E}$ aber dennoch kein gültiges Nachrichten-Signatur-Paar extrahieren kann, wobei ν eine vernachlässigbare Funktion ist. Insbesondere ist damit κ eine vernachlässigbare Funktion. Insgesamt benötigt $\mathcal{E}$ also voraussichtlich $2(K-k+1)/(\epsilon^*(\lambda) - \kappa(\lambda)) \leq 2(K-k+1)/(\epsilon(\lambda) - \kappa(\lambda))$ Schritte, um das Nachrichten-Signatur-Paar zu extrahieren. □

Lemma 3.2.8. *Das Signaturbeweisprotokoll* Prove^**-k ist im Random-Oracle-Modell ein perfektes Zero-Knowledge-Proof-of-Knowledge-Protokoll über ein Nachrichten-Signatur-Paar mit vernachlässigbarer Knowledge-Error-Funktion* κ*, falls die q-SDH-Annahme und die q-polyDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gelten.*

Beweis. Wieder beweisen wir Durchführbarkeit, Zero-Knowledge-Eigenschaft sowie Beweiskraft.

Durchführbarkeit: Folgt aus der Durchführbarkeit von *SoK* (vgl. Abschnitt A.3).

Zero-Knowledge: Der Simulator $\mathcal{SIM}$ wählt zufällig die Elemente $D_m, A_1, B_1, B_2 \in_{\mathcal{R}} \mathbb{G}_1$ sowie den Hashwert $l \in_{\mathcal{R}} \mathbb{Z}_p$ und simuliert die Signature-of-Knowledge *SoK* (vgl. Abschnitt A.3). Somit hat $\mathcal{SIM}$ das Protokoll perfekt simuliert.

Beweiskraft: Der Extractor $\mathcal{E}$ kann analog zu obigem Lemma 3.2.7 nach voraussichtlich $2(K-k+1)/(\epsilon^*(\lambda) - \kappa_0(\lambda))$ Schritten die Zahlen $r_m^{(j)}, a_1^{(j)}, b_1^{(j)}, b_2^{(j)}, \beta_1^{(j)}, \beta_2^{(j)}, e^{(j)}, s^{(j)}, m_1^{(j)}, \ldots, m_L^{(j)}, \phi_I\left(l^{(j)}\right) \in \mathbb{Z}_p$ für $1 \leq j \leq K-k+1$ extrahieren.

Falls $b^{(j)} \neq b^{(1)}$ oder $\beta_1^{(j)} \neq b_1^{(j)} e^{(j)}$ für ein $j \in [1, K-k+1]$ ist, bricht $\mathcal{E}$ ab. Dieser Fall tritt allerdings nur mit vernachlässigbarer Wahrscheinlichkeit ein, da $\mathcal{U}^*$ ansonsten die Gebundenheit des Pedersen-Commitments bzgl. B_1 gebrochen hätte.

Seien wie oben $W_{I,l^{(j)}} := A_1^{(j)} u_0^{-a_1^{(j)}}$ sowie $\Sigma := B_2 g_1^{-b_1}$, dann folgt für alle $1 \leq j \leq K-k+1$:

$$\hat{e}\left(\Sigma, Xh^{e^{(j)}}\right) = \hat{e}\left(gg_0^{s^{(j)}} g_1^{m_1^{(j)}} \ldots g_L^{m_L^{(j)}}, h\right) \hat{e}\left(W_{I,l^{(j)}}^{\alpha - l^{(j)}} u_0^{\phi_I\left(l^{(j)}\right)}, h_I\right). \tag{3.20}$$

Seien nun analog zu obigem Beweis $\phi_I^{(j)}(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ Polynome mit $\hat{e}\left(\Sigma, Xh^{e^{(j)}}\right) = \hat{e}\left(gg_0^{s^{(j)}} g_1^{m_1^{(j)}} \cdots g_L^{m_L^{(j)}} u_0^{\phi_I^{(j)}(\alpha)} \prod_{j=1}^{k}(\alpha+n_j)\right)$ für alle $1 \leq j \leq K-k+1$. Dann folgt mit (3.20) für alle $1 \leq j \leq K-k+1$ die Gleichung $u_0^{\phi_I^{(j)}(\alpha)} = W_{I,l^{(j)}}^{\alpha-l^{(j)}} u_0^{\phi_I^{(j)}\left(l^{(j)}\right)}$. Weiter folgt mit Lemma 3.2.3, dass, außer mit vernachlässigbarer Wahrscheinlichkeit, $\left(e^{(1)}, s^{(1)}, m_1^{(1)}, \ldots, m_L^{(1)}\right) = \cdots = \left(e^{(K-k+1)}, s^{(K-k+1)}, m_1^{(K-k+1)}, \ldots, m_L^{(K-k+1)}\right)$ und folglich $\phi_I^{(1)}(\mathsf{x}) = \cdots = \phi_I^{(K-k+1)}(\mathsf{x})$ gelten. Des Weiteren folgt, dass, außer mit vernachlässigbarer Wahrscheinlichkeit,

- $\phi_I^{(1)}\left(l^{(j)}\right)$ die korrekte Berechnung des Polynoms $\phi_I^{(1)}(\mathsf{x})$ an der Stelle $l^{(j)}$ (gemäß Lemma 3.2.2) und
- $\deg\left(\phi_I^{(1)}\right) \leq K-k$ (gemäß Lemma 3.2.4) ist.

Sei nun zur Vereinfachung $(e, s, m_1, \ldots, m_L, \phi_I(\mathsf{x})) := \left(e^{(1)}, s^{(1)}, m_1^{(1)}, \ldots, m_L^{(1)}, \phi_I^{(1)}(\mathsf{x})\right)$, dann kennt der Extractor $\mathcal{E}$ insgesamt $K-k+1$ Paare $\left(l^{(1)}, \phi_I\left(l^{(1)}\right)\right), \ldots, \left(l^{(K-k+1)}, \phi_I\left(l^{(K-k+1)}\right)\right)$ des Polynoms $\phi_I(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ mit $\deg(\phi_I) \leq K-k$ und kann dieses interpolieren. Anschließend berechnet $\mathcal{E}$ das Polynom $\phi(\mathsf{x}) = \phi_I(\mathsf{x}) \prod_{j=1}^{k}(\mathsf{x}+n_j) \in \mathbb{Z}_p[\mathsf{x}]$. Dann folgt $\deg(\phi) \leq K$ und weiter mit (3.20):

$$\hat{e}(\Sigma, Xh^e) = \hat{e}\left(gg_0^s g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}, h\right).$$

Folglich ist $((m_1, \ldots, m_L, \phi(\mathsf{x})), (\Sigma, e, s))$ ein gültiges Nachrichten-Signatur-Paar.

Somit ist $\kappa(\lambda) = \kappa_0(\lambda) + \nu(\lambda)$ der Anteil der Matrizen mit $A^{(j)}(\omega) = 1$ für $1 \leq j \leq K-k+1$, bei denen $\mathcal{E}$ aber dennoch kein gültiges Nachrichten-Signatur-Paar extrahieren kann, wobei ν eine vernachlässigbare Funktion ist. Insbesondere ist damit κ eine vernachlässigbare Funktion. Insgesamt benötigt $\mathcal{E}$ also voraussichtlich $2(K-k+1)/(\epsilon^*(\lambda) - \kappa(\lambda)) \leq 2(K-k+1)/(\epsilon(\lambda) - \kappa(\lambda))$ Schritte, um das Nachrichten-Signatur-Paar zu extrahieren. □

Lemma 3.2.9. *Das Signaturbeweisprotokoll* Prove-K *ist im Random-Oracle-Modell ein perfektes Zero-Knowledge-Proof-of-Knowledge-Protokoll über ein Nachrichten-Signatur-Paar mit vernachlässigbarer Knowledge-Error-Funktion κ, falls die q-SDH-Annahme und die q-polyDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gelten.*

Beweis. Wir beweisen erneut Durchführbarkeit, Zero-Knowledge-Eigenschaft sowie Beweiskraft.

Durchführbarkeit: Folgt aus der Durchführbarkeit von *SoK* (vgl. Abschnitt A.4).

Zero-Knowledge: Der Simulator $\mathcal{SIM}$ wählt zufällig die Elemente $D_m, B_1, B_2 \in_{\mathcal{R}} \mathbb{G}_1$ und simuliert die Signature-of-Knowledge SoK (vgl. Abschnitt A.4). Somit hat $\mathcal{SIM}$ das Protokoll perfekt simuliert.

Beweiskraft: Der Extractor $\mathcal{E}$ muss den User $\mathcal{U}^*$ nur während der Signature-of-Knowledge SoK zurückspulen und extrahiert analog zu obigen Lemmata nach voraussichtlich $2/\left(\epsilon^*(\lambda) - \kappa_0(\lambda)\right)$ Schritten die Zahlen $r_m, b_1, b_2, \beta_1, \beta_2, e, s, m_1, \ldots, m_L \in \mathbb{Z}_p$.

Falls $\beta_1 \neq b_1 e$ ist, bricht $\mathcal{E}$ ab. Dieser Fall tritt allerdings nur mit vernachlässigbarer Wahrscheinlichkeit ein, da $\mathcal{U}^*$ ansonsten die Gebundenheit des Pedersen-Commitments bzgl. B_1 gebrochen hätte.

Der Extractor $\mathcal{E}$ berechnet das Element $\Sigma = B_2 g_1^{-b_1}$ und mit der letzten Gleichung der Signature-of-Knowledge SoK folgt:

$$\begin{aligned}
& \frac{\hat{e}\left(B_2, X\right)}{\hat{e}\left(gV, h\right)} = \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(g_0^s g_1^{m_1+\beta_1} g_2^{m_2} \cdots g_L^{m_L} B_2^{-e}, h\right) \\
\Leftrightarrow \quad & \frac{\hat{e}\left(\Sigma g_1^{b_1}, X\right)}{\hat{e}\left(gV, h\right)} = \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(g_0^s g_1^{m_1+\beta_1} g_2^{m_2} \cdots g_L^{m_L} \left(\Sigma g_1^{b_1}\right)^{-e}, h\right) \\
\Leftrightarrow \quad & \hat{e}\left(\Sigma, X h^e\right) = \hat{e}\left(g g_0^s g_1^{m_1} \cdots g_L^{m_L} V, h\right),
\end{aligned}$$

wobei $V = u_0^{\phi(\alpha)}$ mit $\phi\left(\mathsf{x}\right) = \prod_{i=j}^{K} \left(\mathsf{x} + n_j\right)$ ist.

Folglich ist $\left(\left(m_1, \ldots, m_L, \phi\left(\mathsf{x}\right)\right), \left(\Sigma, e, s\right)\right)$ ein gültiges Nachrichten-Signatur-Paar.

Weiter ist $\kappa = \kappa_0$ und damit eine vernachlässigbare Funktion. Der Extractor $\mathcal{E}$ benötigt also voraussichtlich $2/\left(\epsilon^*(\lambda) - \kappa(\lambda)\right) \leq 2/\left(\epsilon(\lambda) - \kappa(\lambda)\right)$ Schritte, um das Nachrichten-Signatur-Paar zu extrahieren. □

Nun bleibt zu zeigen, dass das Signaturerstellungsprotokoll Issue die Signer-Privacy sowie die User-Privacy erfüllt.

Lemma 3.2.10. *Das Signaturerstellungsprotokoll* Issue *erfüllt die Signer-Privacy, falls die q-SDH-Annahme und die q-polyDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gelten.*

Beweis. Wir müssen zeigen, dass es einen probabilistischen voraussichtlich-polynomiellen Simulator $\mathcal{SIM}$ gibt, der mit Zugriff auf das Signatur-Orakel $\mathcal{O}_{\mathsf{Sign}}$ die Protokollansicht $\mathsf{view}_{\mathcal{U}^*}^{\mathcal{S}}\left(\mathsf{Issue}\right)$ für jeden polynomiellen User $\mathcal{U}^*$ rechnerisch ununterscheidbar simulieren kann.

Seien wie oben $\epsilon^*(\lambda)$ die Wahrscheinlichkeit, mit der die Signature-of-Knowledge akzeptiert wird und κ_0 die dazugehörige vernachlässigbare Knowledge-Error-Funktion. Analog zur Beweiskraft des Signaturbeweisprotokolls Prove kann $\mathcal{SIM}$ nach voraussichtlich $2\left(K+1\right) / \left(\epsilon^*(\lambda) - \kappa_0(\lambda)\right)$ Schritten die Zahlen $s'^{(j)}, a_0^{(j)}, m_1^{(j)}, \ldots, m_L^{(j)}, \phi^{(j)}\left(l^{(j)}\right) \in \mathbb{Z}_p$ für $1 \leq j \leq K+1$ extrahieren. Falls $\epsilon^*(\lambda) \leq \kappa_0(\lambda)$ ist, bricht $\mathcal{SIM}$ ab und sendet das Symbol $\perp$ an den User $\mathcal{U}^*$.

Falls $\left(s'^{(j)}, m_1^{(j)}, \ldots, m_L^{(j)}\right) \neq \left(s'^{(1)}, m_1^{(1)}, \ldots, m_L^{(1)}\right)$ für ein $j \in [2, K+1]$ ist, bricht $\mathcal{SIM}$ ab. Dieser Fall tritt allerdings nur mit vernachlässigbarer Wahrscheinlichkeit ein (vgl. Lemma 3.1.4). Zur Vereinfachung sei $(s', m_1, \ldots, m_L) := \left(s'^{(1)}, m_1^{(1)}, \ldots, m_L^{(1)}\right)$.

Somit kennt der Simulator $\mathcal{SIM}$ insgesamt $K+1$ Paare $\left(l^{(1)}, \phi\left(l^{(1)}\right)\right), \ldots, \left(l^{(K+1)}, \phi\left(l^{(K+1)}\right)\right)$ des Polynoms $\phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ und erhält dieses durch Polynominterpolation. Falls $C \neq g_0^{s'} g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}$ ist, bricht $\mathcal{SIM}$ ab. Dies tritt aber analog zu Korollar 3.1.1 nur mit vernachlässigbarer Wahrscheinlichkeit ein, da, außer mit vernachlässigbarer Wahrscheinlichkeit, jeder Wert $\phi\left(l^{(j)}\right)$ die korrekte Berechnung des Polynoms $\phi(\mathsf{x})$ an der Stelle $l^{(j)}$ für $1 \leq j \leq K+1$ (vgl. Lemma 3.1.3) und $\deg(\phi) \leq K$ (vgl. Lemma 3.1.5) ist.

Anschließend sendet $\mathcal{SIM}$ die Nachricht $m = (m_1, \ldots, m_L, \phi(\mathsf{x})) \in \mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}]$ mit $\deg(\phi) \leq K$ an das Signatur-Orakel $\mathcal{O}_{\mathsf{Sign}}$ und erhält eine BBS+A-Signatur $\sigma = (\Sigma, e, s)$ mit

$$\hat{e}(\Sigma, Xh^e) = \hat{e}\left(g g_0^s g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}, h\right).$$

Dann berechnet $\mathcal{SIM}$ die Zahl $s'' = s - s' \mod p$ und sendet das perfekt simulierte Tripel (Σ, e, s'') an den User $\mathcal{U}^*$.

Somit hat der Simulator $\mathcal{SIM}$ das Issue-Protokoll perfekt simuliert, falls $\mathcal{SIM}$ nicht abbricht. Dies passiert allerdings nur mit vernachlässigbarer Wahrscheinlichkeit $\kappa_0(\lambda) + \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist. Folglich kann jeder polynomielle Algorithmus nur mit vernachlässigbarer Wahrscheinlichkeit unterscheiden, ob $\mathcal{U}^*$ das Issue-Protokoll mit dem richtigen Signer $\mathcal{S}$ oder einem Simulator $\mathcal{SIM}$ durchführt. □

Lemma 3.2.11. *Das Signaturerstellungsprotokoll* Issue *erfüllt die User-Privacy.*

Beweis. Dies folgt direkt aus der perfekten Verborgenheit des verwendeten Commitment-Schemas sowie des Zeugen W_l und der Honest-Verifier-Zero-Knowledge-Eigenschaft von SoK. Der Simulator $\mathcal{SIM}(\mathsf{pk}, C)$ wählt zufällig das Element $A_0 \in_{\mathcal{R}} \mathbb{G}_1$ und simuliert SoK. Somit hat $\mathcal{SIM}$ das Protokoll perfekt simuliert. □

Aus den obigen Lemmata folgt der folgende Satz:

Satz 3.2.1. *Das BBS+A-Signaturverfahren* (GenSign, Sign, VerifySign, Issue, Prove) *ist sicher, falls die q-SDH-Annahme und die q-polyDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gelten.*

3.2.4 Effizienz des BBS+A-Signaturverfahrens

In diesem Unterabschnitt wird die Effizienz des BBS+A-Signaturverfahrens mit dem bisher einzig vergleichbaren ESS+-Signaturverfahren (vgl. Unterabschnitt 2.12.2) verglichen, da bei beiden Signaturverfahren mit effizienten Protokollen ein Nachrichtenblock $(m_1, \ldots, m_L) \in \mathbb{Z}_p^L$ zusammen mit dem Nguyen-Akkumulator $V \in \mathbb{G}_1$ signiert werden kann.

Der bedeutendste Vorteil des BBS+A-Signaturverfahrens ist, dass dieses unter einer Standard-Annahme in allen Pairing Typen sicher ist, da es speziell für das Nguyen-Akkumulator-Schema entwickelt wurde. Gleichzeitig ist dies allerdings auch ein Nachteil, da *nur* der Nguyen-Akkumulator $V \in \mathbb{G}_1$ bzw. das entsprechende Polynom $\phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ mit $V = u_0^{\phi(\alpha)}$ signiert werden kann, während das ESS+-Signaturverfahren die Signatur auf ein beliebiges Gruppenelement $M \in \mathbb{G}_1$ ermöglicht. Wenn mit dem ESS+-Signaturverfahren ebenfalls der Nguyen-Akkumulator signiert werden soll, wie beispielsweise bei den teilbaren elektronischen Geldsystemen in [ASM08] und [CG10], ist im Signaturerstellungsprotokoll Issue somit kein Beweis für die korrekte Berechnung des Akkumulators notwendig, wie es beim BBS+A-Signaturverfahren der Fall ist. Aus diesem Grund muss der Signer $\mathcal{S}$ beim ESS+-Signaturverfahren weniger Berechnungen ausführen. In allen anderen Aspekten ist das BBS+A-Signaturverfahren allerdings effizienter.

In der folgenden Tabelle 3.1 werden die drei Signaturverfahren BBS+, ESS+ und BBS+A hinsichtlich ihrer Effizienz verglichen, wobei bei den Signaturbeweisprotokollen von ESS+, wie bei BBS+A, ebenfalls bewiesen wird, dass sich entsprechende Werte in einem Akkumulator befinden (vgl. Unterabschnitt 2.12.2). Um die drei Signaturverfahren sinnvoll miteinander vergleichen zu können, werden bei sämtlichen Protokollen die Berechnungen und das Senden der jeweiligen Commitments mit berücksichtigt, vereinfachend jeder interaktive Proof-of-Knowledge als nichtinteraktive Signature-of-Knowledge betrachtet und keine Pairings vorab berechnet oder veröffentlicht. Des Weiteren wird das Typ-3-Pairing als Grundlage der Signaturverfahren gewählt, da das ESS+-Signaturverfahren nur in diesem sicher ist.

Wie in Unterabschnitt 2.5.1 beschrieben, wählen wir p als 256-Bit Primzahl, was aktuell und auch in naher Zukunft ein hohes Sicherheitsniveau garantiert. Somit besitzt bei einem Typ-3-Pairing jedes Element der Gruppe $\mathbb{G}_1$ eine Länge von 257 Bits, der Gruppe $\mathbb{G}_2$ von 513 Bits und der Gruppe $\mathbb{G}_T$ von 1.024 Bits (siehe Tabelle 2.3).

Bei dieser und allen folgenden Tabellen werden die Anzahl der Elemente der Systemparameter, der öffentlichen Schlüssel und der gespeicherten Inhalte angegeben. Zudem wird bei jedem Protokoll die Anzahl der zwischen den beiden Parteien gesendeten Daten aufgelistet. Weiter wird der Berechnungsaufwand der einzelnen Parteien bei der Durchführung des entsprechenden Protokolls angegeben, wobei im Folgenden die Abkürzungen *ME* für *multibased exponentiationen* (wozu auch das Potenzieren mit einer festen Basis gezählt wird) und P für *Pairings* stehen.

Alternativ können für das BBS+A-Signaturverfahren auch die Generatoren $\hat{e}(g_1, X)$, $\hat{e}(g, h), \hat{e}(g_0, h), \ldots, \hat{e}(g_L, h), \hat{e}(u_0, h), \hat{e}(u_1, h) \in \mathbb{G}_T$ als Teil des öffentlichen Schlüssels pk veröffentlicht werden. Dann muss der User $\mathcal{U}$ im Signaturerstellungsprotokoll Issue nur ein Pairing, im Signaturbeweisprotokoll Prove nur zwei Pairings und in den Signaturbeweisprotokollen Prove-k, Prove*-k sowie Prove-K nur ein Pairing berechnen. Dazu müssen die entsprechenden Exponentiationen allerdings in der Gruppe $\mathbb{G}_T$ statt in $\mathbb{G}_1$ durchgeführt werden.

Des Weiteren kann die im Vergleich zum ESS+-Signaturverfahren einzige Schwach-

	BBS+	**ESS+**	**BBS+A**
Annahme	q-SDH	AWSM	q-SDH, q-polyDH
$\mathbb{M}_{\mathsf{Sign}}$	$\mathbb{Z}_p^L$	$\mathbb{G}_1 \times \mathbb{Z}_p^L$	$\mathbb{Z}_p^L \times \mathbb{Z}_p[\mathsf{x}]$ mit $\deg \leq K$
pk	$\mathbb{G}_1^{L+2} \times \mathbb{G}_2^2$	$\mathbb{G}_1^{L+K+5} \times \mathbb{G}_2^{K+4} \times \mathbb{G}_T$	$\mathbb{G}_1^{L+K+3} \times \mathbb{G}_2^{K+2}$
Bits	$1.540 + 257L$	$4.361 + 257L + 770K$	$1.797 + 257L + 770K$
sk	$\mathbb{Z}_p$	$\mathbb{G}_1 \times \mathbb{Z}_p$	$\mathbb{Z}_p$
Bits	256	513	256
σ	$\mathbb{G}_1 \times \mathbb{Z}_p^2$	$\mathbb{G}_1^2 \times \mathbb{G}_2 \times \mathbb{Z}_p^2$	$\mathbb{G}_1 \times \mathbb{Z}_p^2$
Bits	769	1.539	769
Issue	$\mathbb{G}_1^2 \times \mathbb{Z}_p^{L+4}$	$\mathbb{G}_1^4 \times \mathbb{G}_2 \times \mathbb{Z}_p^{L+4}$	$\mathbb{G}_1^3 \times \mathbb{Z}_p^{L+6}$
Bits	$1.538 + 256L$	$2.565 + 256L$	$2.307 + 256L$
$\mathcal{U}ser$	4 ME + 2 P	4 ME + 4 P	5 ME + 3 P
$\mathcal{S}igner$	2 ME	4 ME	3 ME + 2 P
Prove	$\mathbb{G}_1^3 \times \mathbb{Z}_p^{L+8}$	$\mathbb{G}_1^8 \times \mathbb{G}_2 \times \mathbb{Z}_p^{L+21}$	$\mathbb{G}_1^7 \times \mathbb{Z}_p^{L+17}$
Bits	$2.819 + 256L$	$7.945 + 256L$	$6.151 + 256L$
$\mathcal{U}ser$	8 ME + 2 P	25 ME + 8 P	17 ME + 3 P
$\mathcal{V}erifier$	5 ME + 2 P	16 ME + 8 P	11 ME + 4 P
Prove-k		$\mathbb{G}_1^6 \times \mathbb{G}_2 \times \mathbb{Z}_p^{L+16}$	$\mathbb{G}_1^5 \times \mathbb{Z}_p^{L+11}$
Bits		$6.151 + 256L$	$4.101 + 256L$
$\mathcal{U}ser$		20 ME + 8 P	11 ME + 2 P
$\mathcal{V}erifier$		14 ME + 8 P	7 ME + 3 P
Prove*-k			$\mathbb{G}_1^4 \times \mathbb{Z}_p^{L+10}$
Bits			$3.588 + 256L$
$\mathcal{U}ser$			9 ME + 2 P
$\mathcal{V}erifier$			6 ME + 3 P
Prove-K		$\mathbb{G}_1^4 \times \mathbb{G}_2 \times \mathbb{Z}_p^{L+12}$	$\mathbb{G}_1^3 \times \mathbb{Z}_p^{L+8}$
Bits		$4.613 + 256L$	$2.819 + 256L$
$\mathcal{U}ser$		15 ME + 6 P	8 ME + 2 P
$\mathcal{V}erifier$		10 ME + 6 P	5 ME + 2 P

Tabelle 3.1: Effizienzvergleich der BBS+, ESS+ und BBS+A Signaturverfahren

stelle etwas verbessert werden, wenn der Signer $\mathcal{S}$ die geheime Zahl $\alpha \in \mathbb{Z}_p^*$ speichert. Denn dann muss $\mathcal{S}$ im Signaturerstellungsprotokoll Issue entsprechend nur zwei Exponentiationen und ein Pairing berechnen.

KAPITEL 4

ÜBERBLICK ELEKTRONISCHER GELDSYSTEME

In diesem Kapitel werden elektronische Geldsysteme und deren Sicherheitsziele definiert. Dazu werden in Abschnitt 4.1 die Algorithmen und Protokolle elektronischer Geldsysteme, in Abschnitt 4.2 die zusätzlichen Algorithmen und Protokolle fairer elektronischer Geldsysteme und in Abschnitt 4.3 ein zusätzliches Protokoll fairer elektronischer Geldsysteme mit Observern formalisiert.

Anschließend werden die unterschiedlichen Sicherheitsziele der elektronischen Geldsysteme in Abschnitt 4.4 definiert.

Schließlich wird in Abschnitt 4.5 der aktuelle Forschungsstand elektronischer Geldsysteme beschrieben und in Unterabschnitt 4.5.1 die unterschiedlichen Sicherheitsdefinitionen in der Literatur analysiert.

4.1 Elektronische Geldsysteme

Elektronische Geldsysteme bestehen, wie bereits kurz in Abschnitt 1.1 erwähnt, aus den drei Parteien *Bank*, *Kunden* und *Händler*. Zur Initialisierung eines elektronischen Geldsystems werden zunächst die benötigten Systemparameter sp von einem Parametergenerierungsalgorithmus Setup und dann das Schlüsselpaar $(\mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{B}})$ der Bank $\mathcal{B}$ von einem Schlüsselgenerierungsalgorithmus BGen erzeugt. Der Parametergenerierungsalgorithmus Setup kann dabei von einer Trusted-Third-Party ausgeführt werden und wird zusätzlich zum Schlüsselgenerierungsalgorithmus BGen benötigt, damit die Systemparameter sp nicht von einer betrügerischen Bank generiert werden können. Die Methode, zwei verschiedene Algorithmen zur Initialisierung zu verwenden, wird ebenfalls häufig

von S. Canard und A. Gouget benutzt [CG07, CG08, CDG+09, CG10]. Diese Methode ist damit vergleichbar, dass die benötigten Systemparameter sp von BGen generiert werden, aber zur Erstellung von sp eine öffentliche Hashfunktion mit öffentlichen Eingabewerten eingesetzt wird. Letztere Möglichkeit wird häufig von Au *et al.* verwendet [ASM07, AWSM07, ASM08, Au09].

Nach der Initialisierung können die Kunden und Händler dem elektronischen Geldsystem beitreten, indem sie das Kontoeröffnungsprotokoll $\mathsf{Account}$ mit der Bank $\mathcal{B}$ durchführen. In der Literatur wird anstelle des Kontoeröffnungsprotokolls nahezu immer lediglich ein Schlüsselgenerierungsalgorithmus verwendet. Da die Bank allerdings Informationen über den Kunden speichert und in den elektronischen Geldsystemen mit Observern, die in Kapitel 9 beschriebenen sind, ebenfalls Daten an den Kunden sendet, verwenden wir das $\mathsf{Account}$-Protokoll anstelle eines Algorithmus.

Nun kann jeder Kunde mit dem Abhebeprotokoll $\mathsf{Withdraw}$ eine elektronische Münze bei der Bank abheben und mit dem Bezahlprotokoll Spend bei einem Händler ausgeben. Im Gegensatz zu herkömmlichen Bargeld kann der Händler eine elektronische Münze aber nicht als Wechselgeld oder Zahlungsmittel verwenden, sondern muss die erhaltene Münze mit dem Einlöseprotokoll $\mathsf{Deposit}$ bei der Bank einlösen, damit ihm der entsprechende Geldbetrag gutgeschrieben wird.

In einem *kompakten elektronischen Geldsystem* (wie bspw. [CHL05, Wei05, ASM07, AWSM07, CDG+09, BCKL09]) kann durch das Abhebeprotokoll $\mathsf{Withdraw}$ eine elektronische Geldbörse $\mathbb{W}$ abgehoben werden, mit der mehrere elektronische Münzen erzeugt werden können. Mit dem Bezahlprotokoll Spend können die einzelnen Münzen dann bei Händlern ausgegeben werden.

Auch in einem *teilbaren elektronischen Geldsystem* (z.B. [CG07, ASM08, CG10, IL13]) kann durch das Abhebeprotokoll $\mathsf{Withdraw}$ eine elektronische Geldbörse $\mathbb{W}$ abgehoben werden. Jedoch kann die Geldbörse $\mathbb{W}$ im Bezahlprotokoll Spend in Münzen mit unterschiedlichen Geldbeträgen (Denominierungen) *geteilt* werden. Durch diese Eigenschaft können sehr effiziente elektronische Geldsysteme konstruiert werden.

Der Begriff *Geldbörse* bzw. *Wallet* wird in der Literatur für kompakte elektronische Geldsysteme, wie bspw. [CHL05, ASM07, AWSM07, BCKL09], verwendet. Im Gegensatz dazu wird bei teilbaren Geldsystemen häufig die Bezeichnung *Münze* oder *teilbare Münze* gebraucht (bspw. [Oka95, CFT98, CG07, CG10]). Da sich die Abhebeprotokolle der teilbaren Geldsysteme nach der Veröffentlichung von [CHL05] allerdings stark am Abhebeprotokoll von [CHL05] orientieren (indem ebenfalls Signaturverfahren mit effizienten Protokollen eingesetzt werden), verwenden wir, wie auch in [ASM08, Au09], den Begriff *Geldbörse* ebenfalls für teilbare Geldsysteme.

Jede Geldbörse der in Kapitel 6 bis 9 beschriebenen teilbaren elektronischen Geldsysteme enthält eine Signatur σ der Bank sowie K eindeutige Werte, die wir als *Serienschlüssel* bezeichnen, wobei K der (Geld-)Wert der Geldbörse ist. Jeder dieser K Serienschlüssel besitzt den Geldwert 1, so dass der Geldwert der Geldbörse insgesamt K beträgt. Eine Münze coin ist ein Tupel der Form $\mathsf{coin} = (k, S, T, R, \cdots, \Phi = (\cdots, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$. Somit enthält jede Münze

- den (Geld-)Wert k der Münze,
- eine *Seriennummer* S, aus der die k zugehörigen Serienschlüssel berechnet werden, welche zur Double-Spending-Detection benötigt werden,
- ein *Sicherheitstag* T, welches zur Identifizierung eines Double-Spenders benötigt wird,
- einen einzigartigen Wert R, durch den die Münze eindeutig festgelegt wird,
- einen Beweis Φ, der unter anderem
  - eine Signature-of-Knowledge SoK, zur Verifikation der Münze,
  - den Bezahlzeitpunkt ts und
  - den bezahlten Händler $\mathsf{pk}_{\mathcal{H}}$ enthält.

Bemerkt die Bank, dass einige Serienschlüssel mehrfach eingelöst wurden, wird der Double-Spender durch den Identifizierungsalgorithmus Identify identifiziert. Durch den Verifizierungsalgorithmus VerifyGuilt kann verifiziert werden, dass ein bestimmter Kunde in der Tat ein Double-Spender ist.

In [Tro06] hat M. Trolin eine stärkere Definition für elektronische Geldsysteme angegeben, als bisher in der Literatur zu finden war. Während vorherige Sicherheitsdefinitionen eher die Interessen der Bank vertraten, verlangt M. Trolin, dass ehrliche Kunden ebenfalls vor einer betrügerischen Bank geschützt werden müssen. Deshalb wird in [Tro06] erstmals ein zusätzlicher Algorithmus zur Verifizierung der Abhebeprotokolle verwendet, der ehrliche Kunden davor schützen soll, dass eine betrügerische Bank einem Kunden Geld abbucht, obwohl das entsprechende Abhebeprotokoll nicht bzw. nicht korrekt durchgeführt wurde. Aus diesem Grund wird zusätzlich der Verifizierungsalgorithmus VerifyWithdraw definiert.

Definition 4.1.1 (**Elektronisches Geldsystem**). *Sei $\lambda \in \mathbb{N}$ der Sicherheitsparameter. Ein* elektronisches (Offline-) Geldsystem *besteht aus einer probabilistischen polynomiellen Turing-Maschine $\mathcal{B}$ (der Bank), zwei Mengen probabilistischer polynomieller Turing-Maschinen $\hat{\mathcal{K}}$ (den Kunden) und $\hat{\mathcal{H}}$ (Händlern) sowie den folgenden polynomiellen Algorithmen bzw. Protokollen:*

1. *Der* Parametergenerierungsalgorithmus Setup *ist ein probabilistischer Algorithmus, der bei Eingabe des Sicherheitsparameters 1^{λ} die öffentlichen Systemparameter* sp *erzeugt und ausgibt.*
2. *Der* Schlüsselgenerierungsalgorithmus BGen *ist ein probabilistischer Algorithmus, der bei Eingabe der Systemparameter* sp *einen privaten Schlüssel* $\mathsf{sk}_{\mathcal{B}}$ *und einen öffentlichen Schlüssel* $\mathsf{pk}_{\mathcal{B}}$ *der Bank $\mathcal{B}$ erzeugt. Die Ausgabe von* BGen *ist das Schlüsselpaar* $(\mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{B}}) \leftarrow \mathsf{BGen}(\mathsf{sp})$.

3. *Das* Kontoeröffnungsprotokoll Account *ist ein Protokoll*

$$(\{\mathsf{pk}_{\mathcal{K}}, \mathsf{sk}_{\mathcal{K}}), \perp\} \leftarrow \mathcal{K}(\mathsf{sp}) \leftrightarrow \mathcal{B}(\mathsf{sp}) \rightarrow \{\mathsf{pk}_{\mathcal{K}}, \perp\}$$

zwischen einem Kunden $\mathcal{K} \in \hat{\mathcal{K}}$ *und der Bank* $\mathcal{B}$*. Die gemeinsame Eingabe enthält die Systemparameter* sp*. Nach erfolgreicher Durchführung gibt* $\mathcal{K}$ *das Schlüsselpaar* $(\mathsf{pk}_{\mathcal{K}}, \mathsf{sk}_{\mathcal{K}})$ *und* $\mathcal{B}$ *den öffentlichen Schlüssel* $\mathsf{pk}_{\mathcal{K}}$ *aus.*

Auf dieselbe Weise wird das Protokoll zwischen einem Händler $\mathcal{H} \in \hat{\mathcal{H}}$ *und der Bank* $\mathcal{B}$ *durchgeführt, wobei* $\mathcal{H}$ *nach erfolgreicher Durchführung das Schlüsselpaar* $(\mathsf{pk}_{\mathcal{H}}, \mathsf{sk}_{\mathcal{H}})$ *und* $\mathcal{B}$ *den öffentlichen Schlüssel* $\mathsf{pk}_{\mathcal{H}}$ *ausgibt.*

4. *Das* Abhebeprotokoll Withdraw *ist ein interaktives Protokoll*

$$\{\mathbb{W}, \perp\} \leftarrow \mathcal{K}(\mathsf{sp}, K, \mathsf{ts}, \mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{K}}) \leftrightarrow \mathcal{B}(\mathsf{sp}, K, \mathsf{ts}, \mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{K}}) \rightarrow \{\mathsf{view}, \perp\}$$

zwischen einem Kunden $\mathcal{K} \in \hat{\mathcal{K}}$ *und der Bank* $\mathcal{B}$*. Die gemeinsame Eingabe enthält die Systemparameter* sp*, den Wert* K *der Geldbörse und den aktuellen Zeitpunkt* ts*. Die private Eingabe von* $\mathcal{K}$ *enthält den öffentlichen Schlüssel* $\mathsf{pk}_{\mathcal{B}}$ *der Bank* $\mathcal{B}$ *und seinen privaten Schlüssel* $\mathsf{sk}_{\mathcal{K}}$*. Die private Eingabe von* $\mathcal{B}$ *enthält ihren privaten Schlüssel* $\mathsf{sk}_{\mathcal{B}}$ *und den öffentlichen Schlüssel* $\mathsf{pk}_{\mathcal{K}}$ *des Kunden* $\mathcal{K}$*. Nach erfolgreicher Durchführung gibt* $\mathcal{K}$ *eine Geldbörse* $\mathbb{W}$ *und* $\mathcal{B}$ *eine Protokollansicht* view *aus.*

5. *Das* Bezahlprotokoll Spend *ist ein nichtinteraktives Protokoll*

$$\{\mathbb{W}', \perp\} \leftarrow \mathcal{K}(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, k, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}, \mathsf{sk}_{\mathcal{K}}, \mathbb{W}) \leftrightarrow \mathcal{H}(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, k, \mathsf{ts}, \mathsf{sk}_{\mathcal{H}}) \rightarrow \{\mathsf{coin}, \perp\}$$

zwischen einem Kunden $\mathcal{K} \in \hat{\mathcal{K}}$ *und einem Händler* $\mathcal{H} \in \hat{\mathcal{H}}$*. Die gemeinsame Eingabe enthält die Systemparameter* sp*, den öffentlichen Schlüssel* $\mathsf{pk}_{\mathcal{B}}$ *der Bank* $\mathcal{B}$*, den Wert* k *der Münze und den aktuellen Zeitpunkt* ts*. Die private Eingabe von* $\mathcal{K}$ *enthält den öffentliche Schlüssel* $\mathsf{pk}_{\mathcal{H}}$ *des Händlers* $\mathcal{H}$*, seinen privaten Schlüssel* $\mathsf{sk}_{\mathcal{K}}$ *und eine Geldbörse* $\mathbb{W}$*. Die private Eingabe von* $\mathcal{H}$ *ist sein privater Schlüssel* $\mathsf{sk}_{\mathcal{H}}$*. Nach erfolgreicher Durchführung gibt* $\mathcal{K}$ *eine aktualisierte Geldbörse* $\mathbb{W}'$ *aus. Der Händler* $\mathcal{H}$ *gibt die Münze* coin *aus, wenn sie akzeptiert wird und andernfalls das Symbol* $\perp$*.*

6. *Das* Einlöseprotokoll Deposit *ist ein nichtinteraktives Protokoll*

$$\mathcal{H}(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{H}}, \mathsf{coin}) \leftrightarrow \mathcal{B}(\mathsf{sp}, \mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{H}}) \rightarrow \{1, 0\}$$

zwischen einem Händler $\mathcal{H} \in \hat{\mathcal{H}}$ *und der Bank* $\mathcal{B}$*. Die gemeinsame Eingabe enthält die Systemparameter* sp*. Die private Eingabe von* $\mathcal{H}$ *enthält den öffentlichen Schlüssel* $\mathsf{pk}_{\mathcal{B}}$ *der Bank* $\mathcal{B}$*, seinen privaten Schlüssel* $\mathsf{sk}_{\mathcal{H}}$ *und eine Münze* coin*. Die private Eingabe von* $\mathcal{B}$ *enthält ihren privater Schlüssel* $\mathsf{sk}_{\mathcal{B}}$ *und den öffentlichen Schlüssel* $\mathsf{pk}_{\mathcal{H}}$ *des Händlers* $\mathcal{H}$*. Die Bank* $\mathcal{B}$ *gibt die Zahl* 1 *aus, wenn* coin *akzeptiert wird und andernfalls* 0*.*

7. *Der* Identifizierungsalgorithmus $\mathsf{Identify}$ *ist ein deterministischer Algorithmus, der bei Eingabe der Systemparameter* sp*, des öffentlichen Schlüssels* $\mathsf{pk}_{\mathcal{B}}$ *der Bank* $\mathcal{B}$ *und zweier Münzen* coin *sowie* coin' *entweder einen öffentlichen Schlüssel* $\mathsf{pk}_{\mathcal{K}}$ *eines Kunden* $\mathcal{K} \in \hat{\mathcal{K}}$ *mit einem Beweis* Π_G *oder das Symbol* $\perp$ *ausgibt.*

8. *Der* Verifizierungsalgorithmus $\mathsf{VerifyGuilt}$ *ist ein deterministischer Algorithmus, der bei Eingabe der Systemparameter* sp*, des öffentlichen Schlüssels* $\mathsf{pk}_{\mathcal{B}}$ *der Bank* $\mathcal{B}$*, eines öffentlichen Schlüssels* $\mathsf{pk}_{\mathcal{K}}$ *und eines Beweises* Π_G *verifiziert, ob der Kunde mit dem öffentlichen Schlüssel* $\mathsf{pk}_{\mathcal{K}}$ *ein Double-Spending begangen hat. Nach erfolgreicher Verifikation wird* 1 *und andernfalls* 0 *ausgegeben.*

9. *Der* Verifizierungsalgorithmus $\mathsf{VerifyWithdraw}$ *ist ein deterministischer Algorithmus, der bei Eingabe der Systemparameter* sp*, des öffentlichen Schlüssels* $\mathsf{pk}_{\mathcal{B}}$ *der Bank* $\mathcal{B}$*, eines öffentlichen Schlüssels* $\mathsf{pk}_{\mathcal{K}}$ *und einer Protokollansicht* view *verifiziert, ob der Kunde mit dem öffentlichen Schlüssel* $\mathsf{pk}_{\mathcal{K}}$ *das Abhebeprotokoll bzgl.* view *mit der Bank* $\mathcal{B}$ *durchgeführt hat. Nach erfolgreicher Verifikation wird* 1 *und andernfalls* 0 *ausgegeben.*

4.2 Faire elektronische Geldsysteme

Wie in Abschnitt 1.1 beschrieben, ist die *Anonymität* der ehrlichen Kunden eines der wichtigsten Ziele elektronischer Geldsysteme. Diese ist im Vergleich zu gewöhnlichem Bargeld allerdings stärker, da beispielsweise herkömmliche Geldscheine anhand ihrer Seriennummer identifiziert und Transaktionen von Konto zu Konto verfolgt werden können. Daher kann die Anonymität elektronischer Geldsysteme von Kunden und Händlern missbraucht werden, um kriminelle Aktivitäten wie Geldwäsche oder Erpressungen durchzuführen (vgl. [SN92, SPC95]).

Um diese kriminellen Handlungen zu verhindern, kann ein elektronisches Geldsystem zu einem *fairen* elektronischen Geldsystem erweitert werden (bspw. [BGK95, CMS96, CPS96, FTY96, FTY98, GT03, XY03, Sch09]). So ist es beispielsweise im Fall einer Geldwäsche wichtig, den Eigentümer der ausgegebenen Münzen zu ermitteln. Weiter ist es bei einer Erpressung nützlich, die entsprechenden Münzen während des Bezahlens zu erkennen und gegebenenfalls zu sperren. Daher werden zur Bekämpfung der kriminellen Aktivitäten zwei verschiedene Mechanismen zur *Kundenverfolgung* und *Münzverfolgung* benötigt.

Um die Anonymität der ehrlichen Kunden jedoch nicht zu verletzen, dürfen die genannten Verfolgungstechniken nicht alleine durch die Bank durchgeführt werden. Aus diesem Grund wird bei einem fairen elektronischen Geldsystem zusätzlich eine vertrauenswürdige dritte Partei, die sogenannte *Trusted-Third-Party (TTP)*, die wir auch mit *Deanonymisierer* $\mathcal{D}$ bezeichnen, benötigt, die nur in Verdachtsfällen die Anonymität aufhebt. Durch das Kundenverfolgungsprotokoll UTrace kann der Eigentümer einer bestimmten, eingelösten Münze identifiziert werden, um Geldwäsche zu bekämpfen. Wei-

ter kann durch dieses Protokoll ein Kunde ermittelt werden, der einen normalerweise anonymen Service nutzt, welcher in Verdachtsfällen allerdings eine Identifizierung erfordert (vgl. [FTY96]). Durch das Münzverfolgungsprotokoll CTrace können die zu einer bestimmten Geldbörse (und somit zu einem bestimmten Kunden) gehörenden, ausgegebenen Münzen lokalisiert werden. Somit kann mit einer Münzverfolgung die in [SN92] beschriebene Erpressung verhindert werden. Des Weiteren kann eine Münzverfolgung dazu genutzt werden, die Aktivitäten eines verdächtigen Kunden zu überwachen (vgl. [FTY96]).

Während bei den fairen elektronischen Geldsystemen [BGK95, CPS96] der Deanonymisierer bei der Kontoeröffnung beteiligt sein muss, wurden unabhängig voneinander in [CMS96] und [FTY96] erstmals faire elektronische Geldsysteme entwickelt, bei denen alle bisherigen Protokolle ohne den Deanonymisierer durchgeführt werden können. Somit werden für ein faires elektronisches Geldsystem lediglich ein weiterer Schlüsselgenerierungsalgorithmus und die beiden Verfolgungsprotokolle benötigt.

Definition 4.2.1 (Faires elektronisches Geldsystem). *Ein* faires elektronisches (Offline-) Geldsystem *ist ein Geldsystem gemäß Definition 4.1.1 mit einer zusätzlichen probabilistischen polynomiellen Turing-Maschine $\mathcal{D}$ (dem Deanonymisierer) sowie den folgenden polynomiellen Algorithmen bzw. Protokollen:*

1. *Der* Schlüsselgenerierungsalgorithmus DGen *ist ein probabilistischer Algorithmus, der bei Eingabe der Systemparameter* sp *einen privaten Schlüssel* $\mathsf{sk}_{\mathcal{D}}$ *und einen öffentlichen Schlüssel* $\mathsf{pk}_{\mathcal{D}}$ *des Deanonymisierers* $\mathcal{D}$ *erzeugt. Die Ausgabe von* DGen *ist das Schlüsselpaar* $(\mathsf{pk}_{\mathcal{D}}, \mathsf{sk}_{\mathcal{D}}) \leftarrow \mathsf{DGen}(\mathsf{sp})$.

2. *Das* Kundenverfolgungsprotokoll UTrace *ist ein interaktives Protokoll*

$$\{\mathsf{pk}_{\mathcal{K}}, \perp\} \leftarrow \mathcal{B}(\mathsf{sp}, \mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{D}}, \mathsf{coin}) \leftrightarrow \mathcal{D}(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{D}}) \rightarrow \{\mathsf{pk}_{\mathcal{K}}, \perp\}$$

   zwischen der Bank $\mathcal{B}$ und dem Deanonymisierer $\mathcal{D}$. Die gemeinsame Eingabe enthält die Systemparameter sp. *Die private Eingabe von $\mathcal{B}$ enthält seinen privaten Schlüssel* $\mathsf{sk}_{\mathcal{B}}$, *den öffentlichen Schlüssel* $\mathsf{pk}_{\mathcal{D}}$ *des Deanonymisierers $\mathcal{D}$ und eine Münze* coin. *Die private Eingabe von $\mathcal{D}$ enthält den öffentliche Schlüssel* $\mathsf{pk}_{\mathcal{B}}$ *der Bank $\mathcal{B}$ und seinen privaten Schlüssel* $\mathsf{sk}_{\mathcal{D}}$. *Die Ausgabe beider Parteien ist entweder ein öffentlicher Schlüssel* $\mathsf{pk}_{\mathcal{K}}$ *eines Kunden $\mathcal{K} \in \hat{\mathcal{K}}$ oder das Symbol $\perp$.*

3. *Das* Münzverfolgungsprotokoll CTrace *ist ein interaktives Protokoll*

$$\{\mathsf{COINS}, \perp\} \leftarrow \mathcal{B}(\mathsf{sp}, \mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{D}}, \mathsf{view}) \leftrightarrow \mathcal{D}(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{D}}) \rightarrow \{\mathsf{COINS}, \perp\}$$

   zwischen der Bank $\mathcal{B}$ und dem Deanonymisierer $\mathcal{D}$. Die gemeinsame Eingabe enthält die Systemparameter sp. *Die private Eingabe von $\mathcal{B}$ enthält seinen privaten Schlüssel* $\mathsf{sk}_{\mathcal{B}}$, *den öffentlichen Schlüssel* $\mathsf{pk}_{\mathcal{D}}$ *des Deanonymisierers $\mathcal{D}$ und eine*

Protokollansicht view*. Die private Eingabe von* $\mathcal{D}$ *enthält den öffentlichen Schlüssel* $\mathsf{pk}_{\mathcal{B}}$ *der Bank* $\mathcal{B}$ *und seinen privaten Schlüssel* $\mathsf{sk}_{\mathcal{D}}$*. Die Ausgabe beider Parteien ist entweder eine Menge von Münzen* COINS*, die auch zusätzliche Informationen zum Sperren von Münzen enthalten kann, oder das Symbol* $\perp$*.*

Zusätzlich ist der öffentliche Schlüssel $\mathsf{pk}_{\mathcal{D}}$ *eine weitere gemeinsame Eingabe der in Definition 4.1.1 definierten Protokolle* Withdraw, Spend *und* Deposit *sowie eine weitere Eingabe der Algorithmen* Identify, VerifyGuilt *und* VerifyWithdraw*.*

4.3 Elektronische Geldsysteme mit Observern

Da jede elektronische Münze bloß ein digitaler Datensatz ist, kann sie von unehrlichen Kunden leicht dupliziert und mehrmals zum Bezahlen verwendet werden, was als *Double-Spending* bezeichnet wird. Durch die Konstruktion elektronischer (Offline-) Geldsysteme muss also gewährleistet sein, dass ein sogenannter *Double-Spender* nach der Tat identifiziert werden kann, was nicht im Widerspruch zur Anonymität steht.

Damit ein Double-Spender bei einem Offline-Geldsystem allerdings nicht nur nach seiner Tat identifiziert wird, sondern eine Mehrfachausgabe unmittelbar zum Bezahlzeitpunkt verhindert wird, setzen elektronische Geldsysteme eine zusätzliche manipulationssichere Hardware, den sogenannten *Observer*, ein (vgl. [Bra93, CP94, NS03, Sch09]). Der Observer ist dabei insbesondere beim Bezahlvorgang beteiligt und kann ein Double-Spending somit unmittelbar verhindern. Da der Observer nur mit dem entsprechenden Kunden kommuniziert, werden keine weiteren Algorithmen oder Protokolle benötigt.

Bei den elektronischen Geldsystemen in den Abschnitten 9.3 und 9.4 werden die Observer allerdings von einer Trusted-Third-Party $\mathcal{TTP}$ ausgehändigt. Somit besitzen diese Geldsysteme ein zusätzliches Registrierungsprotokoll Registry.

- Das Registrierungsprotokoll Registry ist ein interaktives Protokoll

$$\left(\{\mathsf{pk}_{\mathcal{K}}, \mathsf{sk}_{\mathcal{K}}, \mathsf{pk}_{\mathcal{O}}, SoK\}, \perp\right) \leftarrow \mathcal{K}(\mathsf{sp}) \leftrightarrow \mathcal{TTP}(\mathsf{sp})$$

  zwischen einem Kunden $\mathcal{K} \in \hat{\mathcal{K}}$ und der Trusted-Third-Party $\mathcal{TTP}$. Die gemeinsame Eingabe enthält die Systemparameter sp. Nach erfolgreicher Durchführung gibt $\mathcal{K}$ das Schlüsselpaar $(\mathsf{pk}_{\mathcal{K}}, \mathsf{sk}_{\mathcal{K}})$, den öffentlichen Schlüssel des Observers $\mathsf{pk}_{\mathcal{O}}$ sowie eine Signature-of-Knowledge SoK aus.

Zusätzlich ist der öffentliche Schlüssel $\mathsf{pk}_{\mathcal{O}}$ eine weitere gemeinsame Eingabe des Protokolls Withdraw und eine weitere private Eingabe des Kunden bei dem Protokoll Spend.

4.4 Sicherheitsziele (fairer) elektronischer Geldsysteme (mit Observern)

In diesem Abschnitt werden die Sicherheitsziele elektronischer Geldsysteme, fairer elektronischer Geldsysteme und elektronischer Geldsysteme mit Observern definiert.

Wie bereits in Abschnitt 1.1 erwähnt, ist die *Anonymität* das entscheidende Sicherheitsmerkmal elektronischer Geldsysteme. Ein weiteres zentrales Ziel ist die *Unfälschbarkeit* elektronischer Münzen. Dadurch wird garantiert, dass betrügerische Kunden und Händler keine elektronischen Münzen fälschen können und ein Double-Spender identifiziert wird. Dabei darf es einer unehrlichen Bank allerdings nicht möglich sein, einen Kunden fälschlicherweise eines Double-Spendings zu beschuldigen, welches er nicht begangen hat. Diese Eigenschaft nennen wir *Double-Spending-Beschuldigung* und entspricht dem Grundgedanken der Sicherheitseigenschaft *strong Exculpability* aus [CHL05] bzw. *Non-Frameability* aus [Tro06] (vgl. Unterabschnitt 4.5.1).

Wie bereits in Abschnitt 4.1 beschrieben, hat M. Trolin in [Tro06] eine stärkere Definition für elektronische Geldsysteme gefordert, um insbesondere ehrliche Kunden vor einer betrügerischen Bank zu schützen. So fordert er unter anderem, dass eine Bank nur dann einem Kunden Geld abbuchen kann, wenn das entsprechende Abhebeprotokoll von beiden Parteien korrekt durchgeführt wurde. Dieses Sicherheitsziel bezeichnen wir mit *Abhebe-Beschuldigung*.

Neben den genannten vier Sicherheitszielen für elektronische Geldsysteme benötigen faire elektronische Geldsysteme weitere Sicherheitseigenschaften. Einerseits müssen die Kunden- und Münzverfolgung aus Sicht der Bank und des Deanonymisierers sicher sein, so dass beide Parteien in Verdachtsfällen die entsprechende Verfolgung durchführen können. Diese Sicherheitsziele nennen wir *korrekte Kundenverfolgung* bzw. *korrekte Münzverfolgung*. Andererseits müssen die Verfolgungstechniken aber auch aus Sicht der Kunden sicher sein, damit unschuldige Kunden nicht irrtümlich beschuldigt werden können (vgl. [Sch09]). Bei den meisten fairen elektronischen Geldsystemen in der Literatur (z.B. [CMS96, FTY96, FTY98, GT03, XY03]) bleibt dies allerdings völlig unberücksichtigt. Aus diesem Grund führen wir die beiden Sicherheitsziele *Kundenverfolgung-Beschuldigung* und *Münzverfolgung-Beschuldigung* ein.

Des Weiteren werden die beiden Sicherheitseigenschaften *Over-Spending-Prevention* und *Double-Spending-Prevention* für elektronische Geldsysteme mit Observern definiert.

Bei den folgenden formalen Sicherheitsdefinitionen für elektronische Geldsysteme orientieren wir uns an [Au09].

Die Sicherheitsziele werden als Spiele beschrieben, die ein polynomieller Angreifer $\mathcal{A}$ gewinnen muss. Jedes Spiel beginnt mit einer *Initialisierung*, in der der Angreifer die benötigten Systemparameter und öffentlichen Schlüssel von einem Challenger $\mathcal{C}$ erhält. Anschließend kann $\mathcal{A}$ in der *Probe-Phase* die einzelnen Protokolle durchführen. Dazu befragt der Angreifer unterschiedliche Orakel, die sich wie die ehrlichen Parteien gemäß dem entsprechenden Protokoll verhalten. Am *Ende* versucht nun der Angreifer das

entsprechende Spiel zu gewinnen.

Um eine möglichst starke Sicherheit zu gewährleisten, erhält der Angreifer $\mathcal{A}$ die Möglichkeit jedes Protokoll mit einem Orakel sequentiell durchzuführen, wobei der Angreifer die Rolle einer der beiden zugehörigen Parteien übernimmt. Das Orakel verhält sich dabei wie die jeweils andere Partei gemäß dem entsprechenden Protokoll. Zusätzlich darf der Angreifer $\mathcal{A}$ Protokolle zwischen zwei ehrlichen Parteien ansehen, wodurch das Abhören eines Protokolls modelliert wird. Bei allen Protokollen darf der Angreifer die äußeren Parameter, wie die beteiligten Kunden und Händler, die Geldbeträge sowie die Zeitpunkte, bestimmen.

Im Folgenden werden Kunden, Händler oder die Bank als *korrupt* bezeichnet, wenn sie vom Angreifer $\mathcal{A}$ kontrolliert werden. Andernfalls handelt es sich um *ehrliche* Kunden und Händler bzw. die ehrliche Bank.

Die Orakel verwalten mit dem Challenger $\mathcal{C}$ die Listen $\mathsf{Hash}, \mathcal{U}_{\mathcal{A}}, \mathcal{U}_H, \mathbb{W}_H, \mathsf{view}_{\mathcal{A}}, \mathsf{view}_H$, $\mathbb{T}_H, \mathbb{S}, \mathbb{C}_H, \mathbb{C}, \mathbb{C}_{H,\mathcal{A}}, \mathbb{C}_{\mathcal{A}}, \mathbb{C}, \mathbb{D}_H$ und $\mathbb{D}_{\mathcal{A}}$ mit den folgenden Eigenschaften:

Hash : Hier werden die Ausgaben des Random-Oracles gespeichert.

$\mathcal{U}_{\mathcal{A}}$: Hier werden die öffentlichen Schlüssel der korrupten Kunden und Händler gespeichert.

$\mathcal{U}_H$: Hier werden die Schlüsselpaare der ehrlichen Kunden und Händler gespeichert.

$\mathbb{W}_H$: Hier werden die Geldbörsen der ehrlichen Kunden gespeichert.

$\mathsf{view}_{\mathcal{A}}$: Hier werden die Protokollansichten $\mathsf{view} = (\mathbb{T}, \sigma')$ der Abhebeanfragen bzgl. korrupter Kunden gespeichert. Jede Protokollansicht view lässt sich in die beiden Teile $\mathbb{T}$ und σ' unterteilen, wobei die Abhebeinformationen $\mathbb{T}$ alle vom Kunden erzeugten und an die Bank gesendeten Daten inkl. aller öffentlicher Informationen (wie bspw. der (Geld-)Wert der Geldbörse) sowie berechneter Hashwerte enthalten und σ' die von der Bank erzeugten Werte enthält.

view_H : Hier werden die Protokollansichten $\mathsf{view} = (\mathbb{T}, \sigma')$ der Abhebeanfragen bzgl. ehrlicher Kunden gespeichert, wobei $\mathbb{T}$ und σ' wie oben definiert sind.

$\mathbb{T}_H$: Hier werden die Abhebeinformationen $\mathbb{T}$ der Abhebeanfragen bzgl. ehrlicher Kunden mit der korrupten Bank gespeichert.

$\mathbb{S}$: Hier werden Informationen über Bezahlprotokolle mit ehrlichen Händlern gespeichert.

$\mathbb{C}_H$ **und** $\mathbb{C}$: Hier werden die Münzen gespeichert, die von ehrlichen Kunden bei ehrlichen Händlern ausgegeben werden.

$\mathbb{C}_{H,\mathcal{A}}$: Hier werden die Münzen gespeichert, die von ehrlichen Kunden bei korrupten Händlern ausgegeben werden.

$\mathbb{C}_{\mathcal{A}}$: Hier werden die Münzen gespeichert, die von korrupten Kunden bei ehrlichen Händlern ausgegeben werden.

$\mathbb{D}_{H}$: Hier werden die Münzen gespeichert, die von ehrlichen Händlern bei der Bank eingelöst werden.

$\mathbb{D}_{\mathcal{A}}$: Hier werden die Münzen gespeichert, die von korrupten Händlern bei der Bank eingelöst werden.

Zusätzlich gibt es die beiden Werte \$ und \$*, die zu Beginn jeweils mit dem Wert 0 initialisiert werden. Dabei ist \$ der Geldbetrag, der von korrupten Kunden abgehoben und von korrupten Händlern eingenommen wird. Dagegen ist \$* der Geldbetrag, der von korrupten Kunden ausgegeben und von korrupten Händlern eingelöst wird.

Weiter gibt es die drei Zähler $i_{\mathcal{A}}, i$ und j, die zu Beginn jeweils mit dem Wert 1 initialisiert werden. Der Zähler $i_{\mathcal{A}}$ wird für die Abhebeprotokolle korrupter Kunden, der Zähler i für die Abhebeprotokolle ehrlicher Kunden und der Zähler j für die Bezahlprotokolle mit ehrlichen Händlern verwendet.

Bei den folgenden Orakel Bezeichnungen $\mathcal{O}^{\mathsf{Z}}_{\mathcal{X},\mathcal{Y}}$ steht Z für das jeweilige Protokoll und $\mathcal{X}, \mathcal{Y}$ stehen für die ehrlichen Parteien, deren Rollen vom Orakel übernommen werden.

- $\mathcal{O}^{\mathsf{H}}$: Dieses Orakel ist das Random-Oracle. Mit diesem Orakel kann der Angreifer $\mathcal{A}$ eine Hashanfrage durchführen. Dazu sendet der Angreifer $\mathcal{A}$ einen String $m \in \{0,1\}^*$ an das Orakel. Falls bereits ein Eintrag $(m, \mathsf{H}(m)) \in \mathsf{Hash}$ existiert, sendet $\mathcal{O}^{\mathsf{H}}$ den String $\mathsf{H}(m)$ zurück. Andernfalls wählt $\mathcal{O}^{\mathsf{H}}$ einen String $\mathsf{H}(m) \in_{\mathcal{R}} A$, wobei A das Bild der Hashfunktion ist, sendet $\mathsf{H}(m)$ an $\mathcal{A}$ und speichert den Eintrag $(m, \mathsf{H}(m))$ in der Liste Hash. Dieses Orakel darf auch simultan zu anderen Orakeln angefragt werden.

- $\mathcal{O}^{\mathsf{A}}_{\mathcal{B}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ das Kontoeröffnungsprotokoll $\mathsf{Account}$ durchführen und korrupte Kunden (bzw. Händler) in das System hinzufügen. Der Angreifer $\mathcal{A}$ übernimmt die Rolle des Kunden, während sich das Orakel $\mathcal{O}^{\mathsf{A}}_{\mathcal{B}}$ wie die Bank gemäß Protokoll verhält. Nach erfolgreicher Durchführung wird der entsprechende öffentliche Schlüssel $\mathsf{pk}_{\mathcal{K}}$ in der Liste der korrupten Kunden $\mathcal{U}_{\mathcal{A}}$ gespeichert.

- $\mathcal{O}^{\mathsf{A}}_{\mathcal{O},\mathcal{B}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ das Kontoeröffnungsprotokoll $\mathsf{Account}$ durchführen und korrupte Kunden (bzw. Händler) in das System hinzufügen. Der Angreifer $\mathcal{A}$ übernimmt die Rolle des Kunden, während sich das Orakel $\mathcal{O}^{\mathsf{A}}_{\mathcal{O},\mathcal{B}}$ wie der Observer und wie die Bank, die ein Schlüsselpaar $(\mathsf{pk}_{\mathcal{O}}, \mathsf{sk}_{\mathcal{O}})$ generiert, jeweils gemäß Protokoll verhält. Nach erfolgreicher Durchführung wird das entsprechende Tripel $(\mathsf{pk}_{\mathcal{K}}, \mathsf{pk}_{\mathcal{O}}, \mathsf{sk}_{\mathcal{O}})$ in der Liste der korrupten Kunden $\mathcal{U}_{\mathcal{A}}$ gespeichert.

- $\mathcal{O}^{\mathsf{A}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ korrupte Kunden (bzw. Händler) in das System hinzufügen. Dazu sendet der Angreifer $\mathcal{A}$ einen öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{K}}$ an das Orakel, welches diesen in der Liste der korrupten Kunden $\mathcal{U}_{\mathcal{A}}$ speichert.

- $\mathcal{O}^{\mathsf{A}}_{\mathcal{K}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ das Kontoeröffnungsprotokoll Account durchführen und ehrliche Kunden (bzw. Händler) in das System hinzufügen. Der Angreifer $\mathcal{A}$ übernimmt die Rolle der Bank, während sich das Orakel $\mathcal{O}^{\mathsf{A}}_{\mathcal{K}}$ wie ein Kunde gemäß Protokoll verhält und ein Schlüsselpaar $(\mathsf{pk}_{\mathcal{K}}, \mathsf{sk}_{\mathcal{K}})$ generiert. Nach erfolgreicher Durchführung wird der öffentliche Schlüssel $\mathsf{pk}_{\mathcal{K}}$ an $\mathcal{A}$ gesendet und das Schlüsselpaar $(\mathsf{pk}_{\mathcal{K}}, \mathsf{sk}_{\mathcal{K}})$ in der Liste der ehrlichen Kunden $\mathcal{U}_H$ gespeichert.

- $\mathcal{O}^{\mathsf{A}}_{\mathcal{K},\mathcal{B}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ eine Durchführung des Kontoeröffnungsprotokolls Account ansehen, um ehrliche Kunden (bzw. Händler) in das System hinzuzufügen. Das Orakel $\mathcal{O}^{\mathsf{A}}_{\mathcal{K},\mathcal{B}}$ verhält sich wie ein Kunde, der ein Schlüsselpaar $(\mathsf{pk}_{\mathcal{K}}, \mathsf{sk}_{\mathcal{K}})$ generiert, und wie die Bank jeweils gemäß Protokoll. Der Angreifer erhält alle Daten, die zwischen dem Kunden und der Bank gesendet werden. Anschließend wird der öffentliche Schlüssel $\mathsf{pk}_{\mathcal{K}}$ an $\mathcal{A}$ gesendet und das Schlüsselpaar $(\mathsf{pk}_{\mathcal{K}}, \mathsf{sk}_{\mathcal{K}})$ in der Liste der ehrlichen Kunden $\mathcal{U}_H$ gespeichert.

- $\mathcal{O}^{\mathsf{W}}_{\mathcal{B}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ das Abhebeprotokoll Withdraw durchführen, so dass ein korrupter Kunde eine Geldbörse erhält. Der Angreifer sendet einen öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_{\mathcal{A}}$, einen Zeitpunkt ts und einen Geldbetrag K an das Orakel und übernimmt die Rolle des Kunden mit dem öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{K}}$, während sich $\mathcal{O}^{\mathsf{W}}_{\mathcal{B}}$ wie die Bank gemäß Protokoll verhält. Nach erfolgreicher Durchführung wird ein Eintrag $(i_{\mathcal{A}}, \mathsf{view} = (\mathbb{T}, \sigma'))$ in der Liste $\mathsf{view}_{\mathcal{A}}$ gespeichert, der Wert \$ um K erhöht und der Zähler $i_{\mathcal{A}}$ um 1 erhöht.

- $\mathcal{O}^{\mathsf{W}}_{\mathcal{O},\mathcal{B}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ das Abhebeprotokoll Withdraw durchführen, so dass ein korrupter Kunde eine Geldbörse erhält. Der Angreifer sendet einen öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{K}}$ mit $(\mathsf{pk}_{\mathcal{K}}, \mathsf{pk}_{\mathcal{O}}, \cdot) \in \mathcal{U}_{\mathcal{A}}$, einen Zeitpunkt ts und einen Geldbetrag K an das Orakel und übernimmt die Rolle des korrupten Kunden mit dem öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{K}}$, während sich $\mathcal{O}^{\mathsf{W}}_{\mathcal{O},\mathcal{B}}$ wie der Observer mit dem öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{O}}$ und wie die Bank jeweils gemäß Protokoll verhält. Nach erfolgreicher Durchführung wird ein Eintrag $(i_{\mathcal{A}}, \mathsf{view} = (\mathbb{T}, \sigma'))$ in der Liste $\mathsf{view}_{\mathcal{A}}$ gespeichert, der Wert \$ um K erhöht und der Zähler $i_{\mathcal{A}}$ um 1 erhöht.

- $\mathcal{O}^{\mathsf{W}}_{\mathcal{K}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ das Abhebeprotokoll Withdraw durchführen, so dass ein ehrlicher Kunde eine Geldbörse erhält. Der Angreifer sendet einen öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H$, einen Zeitpunkt ts und einen Geldbetrag K an das Orakel und übernimmt die Rolle der Bank, während sich $\mathcal{O}^{\mathsf{W}}_{\mathcal{K}}$ wie der Kunde mit dem öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{K}}$ gemäß Protokoll verhält. Nach erfolgreicher Durchführung wird ein Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, K)$ in der Liste $\mathbb{W}_H$ sowie ein Eintrag $(i, \mathbb{T})$ in der Liste $\mathbb{T}_H$ gespeichert und der Zähler i um 1 erhöht.

Falls die Anfrage nicht erfolgreich durchgeführt wurde, wird nur ein Eintrag $(i, \mathbb{T})$ in der Liste $\mathbb{T}_H$ gespeichert und der Zähler i um 1 erhöht.

- $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ eine Durchführung des Abhebeprotokolls Withdraw ansehen, so dass ein ehrlicher Kunde eine Geldbörse erhält. Der Angreifer sendet einen öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H$, einen Zeitpunkt ts und einen Geldbetrag K an das Orakel, welches sich wie der Kunde mit dem öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{K}}$ und wie die Bank jeweils gemäß Protokoll verhält. Der Angreifer erhält alle Daten, die zwischen dem Kunden und der Bank gesendet werden. Anschließend wird ein Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, K)$ in der Liste $\mathbb{W}_H$ sowie ein Eintrag (i, view) in der Liste view_H gespeichert und der Zähler i um 1 erhöht.

- $\mathcal{O}^{\mathsf{S}}_{\mathcal{H}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ das Bezahlprotokoll Spend durchführen, so dass ein korrupter Kunde mit einer Münze coin bezahlt. Der Angreifer sendet einen öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{H}} \in \mathcal{U}_H$, einen *gültigen* Zeitpunkt ts und einen Wert k an das Orakel und übernimmt die Rolle eines korrupten Kunden, während sich $\mathcal{O}^{\mathsf{S}}_{\mathcal{H}}$ wie der Händler mit dem öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{H}}$ gemäß Protokoll verhält. Nach erfolgreicher Durchführung wird ein Eintrag (j, coin) in der Liste $\mathbb{C}_{\mathcal{A}}$ sowie ein Eintrag $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts})$ in der Liste $\mathbb{S}$ gespeichert, der Wert \$* um k erhöht und der Zähler j um 1 erhöht.

  Ein Zeitpunkt ts ist gültig, falls $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts}) \notin \mathbb{S}$ ist.

- $\mathcal{O}^{\mathsf{S}}_{\mathcal{O},\mathcal{H}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ das Bezahlprotokoll Spend durchführen, so dass ein korrupter Kunde mit einer Münze coin bezahlt, wobei sich das Orakel $\mathcal{O}^{\mathsf{S}}_{\mathcal{O},\mathcal{H}}$ in die beiden Orakel $\mathcal{O}^{\mathsf{S}}_{\mathcal{O}}$ und $\mathcal{O}^{\mathsf{S}}_{\mathcal{H}}$ unterteilt. Der Angreifer sendet einen öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{H}} \in \mathcal{U}_H$, einen *gültigen* Zeitpunkt ts und einen Wert k für das Orakel $\mathcal{O}^{\mathsf{S}}_{\mathcal{H}}$ sowie einen öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{O}} \in \mathcal{U}_{\mathcal{A}}$ und einen Wert $\widetilde{k}$ für das Orakel $\mathcal{O}^{\mathsf{S}}_{\mathcal{O}}$. Der Angreifer $\mathcal{A}$ übernimmt die Rolle eines korrupten Kunden, während sich $\mathcal{O}^{\mathsf{S}}_{\mathcal{O}}$ wie der Observer mit dem öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{O}}$ bzgl. des Münzwertes $\widetilde{k}$ gemäß Protokoll und sich $\mathcal{O}^{\mathsf{S}}_{\mathcal{H}}$ wie der Händler mit dem öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{H}}$ bzgl. des Münzwertes k gemäß Protokoll verhält. Nach erfolgreicher Durchführung wird ein Eintrag (j, coin) in der Liste $\mathbb{C}_{\mathcal{A}}$ sowie ein Eintrag $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts})$ in der Liste $\mathbb{S}$ gespeichert, der Wert \$* um k erhöht und der Zähler j um 1 erhöht.

  Ein Zeitpunkt ts ist gültig, falls $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts}) \notin \mathbb{S}$ ist.

- $\mathcal{O}^{\mathsf{S}^\star}_{\mathcal{H}}$: Wie $\mathcal{O}^{\mathsf{S}}_{\mathcal{H}}$, allerdings muss ts nicht gültig sein.

- $\mathcal{O}^{\mathsf{S}^\star}_{\mathcal{O},\mathcal{H}}$: Wie $\mathcal{O}^{\mathsf{S}}_{\mathcal{O},\mathcal{H}}$, allerdings muss ts nicht gültig sein.

- $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ das Bezahlprotokoll Spend durchführen, so dass ein ehrlicher Kunde mit einer Münze coin bei einem korrupten Händler bezahlt. Der Angreifer sendet einen Index i mit $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i) \in \mathbb{W}_H$, einen öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{H}} \in \mathcal{U}_{\mathcal{A}}$, einen Zeitpunkt ts und einen *gültigen* Wert k an

das Orakel und übernimmt die Rolle des Händlers mit dem öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{H}}$, während sich $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$ wie der Kunde bzgl. des Eintrags $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i)$ gemäß Protokoll verhält. Nach erfolgreicher Durchführung wird dieser Eintrag zu $(i, \mathbb{W}'_i, \mathsf{pk}_{\mathcal{K}}, \$_i - k)$ aktualisiert, wobei $\mathbb{W}'_i$ die aktualisierte Geldbörse ist. Weiter wird ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in der Liste $\mathbb{C}_{H,\mathcal{A}}$ gespeichert und der Wert \$ um k erhöht.

Ein Wert k ist gültig, falls $k \leq \$_i$ ist.

- $\mathcal{O}^{\mathsf{S}^\star}_{\mathcal{K}}$: Wie $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$, allerdings muss k nicht gültig sein. Somit kann ein ehrlicher Kunde durch $\mathcal{O}^{\mathsf{S}^\star}_{\mathcal{K}}$ zu einem Double-Spending gezwungen werden. Wird ein Double-Spending bzgl. eines Eintrags $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i)$ erzwungen indem $k > \$_i$ ist, wird die Geldbörse $\mathbb{W}_i$ zuerst wieder zu dem ursprünglichen Zustand zurückgesetzt, an dem sie noch nie zum Bezahlen verwendet wurde.

- $\mathcal{O}^{\mathsf{S}}_{\mathcal{K},\mathcal{H}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ eine Durchführung des Bezahlprotokolls Spend ansehen, so dass ein ehrlicher Kunde mit einer Münze coin bei einem ehrlichen Händler bezahlt. Der Angreifer sendet einen Index i mit $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i) \in \mathbb{W}_H$, einen öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{H}} \in \mathcal{U}_H$, einen gültigen Zeitpunkt ts und einen gültigen Wert k an das Orakel, welches sich wie der Kunde bzgl. des Eintrags $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i)$ und wie der Händler mit dem öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{H}}$ jeweils gemäß Protokoll verhält. Der Angreifer erhält alle Daten, die zwischen dem Kunden und dem Händler gesendet werden. Anschließend wird der Eintrag zu $(i, \mathbb{W}'_i, \mathsf{pk}_{\mathcal{K}}, \$_i - k)$ aktualisiert, wobei $\mathbb{W}'_i$ die aktualisierte Geldbörse ist. Weiter wird ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in der Liste $\mathbb{C}_H$, ein Eintrag (j, coin) in der Liste $\mathbb{C}$ sowie ein Eintrag $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts})$ in der Liste $\mathbb{S}$ gespeichert und der Zähler j um 1 erhöht.

- $\mathcal{O}^{\mathsf{S}^\star}_{\mathcal{K},\mathcal{H}}$: Wie $\mathcal{O}^{\mathsf{S}}_{\mathcal{K},\mathcal{H}}$, allerdings müssen ts und k nicht gültig sein. Somit kann ein ehrlicher Kunde durch $\mathcal{O}^{\mathsf{S}^\star}_{\mathcal{K},\mathcal{H}}$ zu einem Double-Spending gezwungen werden. Wird ein Double-Spending bzgl. eines Eintrags $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i)$ erzwungen indem $k > \$_i$ ist, wird die Geldbörse $\mathbb{W}_i$ zuerst wieder zu dem ursprünglichen Zustand zurückgesetzt, an dem sie noch nie zum Bezahlen verwendet wurde.

- $\mathcal{O}^{\mathsf{D}}_{\mathcal{H}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ das Einlöseprotokoll $\mathsf{Deposit}$ durchführen, so dass ein ehrlicher Händler eine Münze coin einlöst. Der Angreifer sendet einen Index j mit $(j, \mathsf{coin}) \in \mathbb{C} \cup \mathbb{C}_{\mathcal{A}}$ an das Orakel und übernimmt die Rolle der korrupten Bank, während sich $\mathcal{O}^{\mathsf{D}}_{\mathcal{H}}$ wie der Händler bzgl. des Eintrags (j, coin) gemäß Protokoll verhält. Nach erfolgreicher Durchführung wird ein Eintrag (j, coin) in der Liste $\mathbb{D}_H$ gespeichert.

- $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ das Einlöseprotokoll $\mathsf{Deposit}$ durchführen, so dass ein korrupter Händler eine Münze coin im Wert von k einlöst. Der Angreifer sendet einen öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{H}} \in \mathcal{U}_{\mathcal{A}}$ und übernimmt die Rolle des korrupten Händlers, während sich $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$ wie die Bank gemäß Protokoll verhält.

Nach erfolgreicher Durchführung wird die Münze coin in einer Liste $\mathbb{D}_{\mathcal{A}}$ gespeichert und der Wert \$* um k erhöht.

- $\mathcal{O}^{\mathsf{D}}_{\mathcal{H},\mathcal{B}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ eine Durchführung des Einlöseprotokolls $\mathsf{Deposit}$ ansehen, so dass ein ehrlicher Händler eine Münze coin bei der ehrlichen Bank einlöst. Der Angreifer sendet einen Index j mit $(j, \mathsf{coin}) \in \mathbb{C} \cup \mathbb{C}_{\mathcal{A}}$ an das Orakel $\mathcal{O}^{\mathsf{D}}_{\mathcal{H},\mathcal{B}}$, welches sich wie der Händler bzgl. des Eintrags (j, coin) und wie die Bank jeweils gemäß Protokoll verhält. Der Angreifer erhält alle Daten, die zwischen dem Händler und der Bank gesendet werden. Anschließend wird ein Eintrag (j, coin) in der Liste $\mathbb{D}_H$ gespeichert.

- $\mathcal{O}^{\mathsf{VW}}_{\mathcal{B}}$: Durch dieses Orakel erhält der Angreifer $\mathcal{A}$ eine Protokollansicht view der Bank. Dazu sendet der Angreifer einen Index $i_{\mathcal{A}}$ mit $(i_{\mathcal{A}}, (\mathbb{T}, \cdot)) \in \mathsf{view}_{\mathcal{A}}$ an das Orakel, welches eine Protokollansicht $\mathsf{view} = (\mathbb{T}, \sigma')$ an den Angreifer $\mathcal{A}$ sendet, wobei σ' gemäß dem Abhebeprotokoll $\mathsf{Withdraw}$ erzeugt wird.

- $\mathcal{O}^{\mathsf{UT}}_{\mathcal{D}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ das Kundenverfolgungsprotokoll UTrace durchführen. Der Angreifer sendet eine Münze coin an das Orakel und übernimmt die Rolle der Bank, während sich $\mathcal{O}^{\mathsf{UT}}_{\mathcal{D}}$ wie der Deanonymisierer gemäß Protokoll verhält.

- $\mathcal{O}^{\mathsf{UT}}_{\mathcal{B},\mathcal{D}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ eine Durchführung des Kundenverfolgungsprotokolls UTrace ansehen. Der Angreifer sendet eine Münze coin an das Orakel, welches sich wie die Bank bzgl. coin und wie der Deanonymisierer jeweils gemäß Protokoll verhält. Der Angreifer erhält alle Daten, die zwischen der Bank und dem Deanonymisierer gesendet werden.

- $\mathcal{O}^{\mathsf{CT}}_{\mathcal{D}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ das Münzverfolgungsprotokoll CTrace durchführen. Der Angreifer sendet eine Protokollansicht $\mathsf{view} = (\mathbb{T}, \sigma')$ an das Orakel und übernimmt die Rolle der Bank, während sich $\mathcal{O}^{\mathsf{CT}}_{\mathcal{D}}$ wie der Deanonymisierer gemäß Protokoll verhält.

- $\mathcal{O}^{\mathsf{CT}}_{\mathcal{B},\mathcal{D}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ eine Durchführung des Münzverfolgungsprotokolls CTrace ansehen. Der Angreifer sendet eine Protokollansicht $\mathsf{view} = (\mathbb{T}, \sigma')$ an das Orakel, welches sich wie die Bank bzgl. view und wie der Deanonymisierer jeweils gemäß Protokoll verhält. Der Angreifer erhält alle Daten, die zwischen der Bank und dem Deanonymisierer gesendet werden.

Wenn es bei einem elektronischen Geldsystem mit Observern ein zusätzliches Registrierungsprotokoll $\mathsf{Registry}$ gibt, das vor dem Kontoeröffnungsprotokoll $\mathsf{Account}$ durchgeführt wird, werden die beiden folgenden Orakel benötigt. Weiter werden bereits bei diesen Orakel-Anfragen die öffentlichen Schlüssel der Kunden und Observer erzeugt und gespeichert.

- $\mathcal{O}^{\mathsf{R}}_{\mathcal{TTP}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ das Registrierungsprotokoll $\mathsf{Registry}$ durchführen und korrupte Kunden (bzw. Händler) in das System hinzufügen. Der Angreifer $\mathcal{A}$ übernimmt die Rolle des Kunden, während sich das Orakel $\mathcal{O}^{\mathsf{R}}_{\mathcal{TTP}}$ wie die Trusted-Third-Party gemäß Protokoll verhält und ein Schlüsselpaar $(\mathsf{pk}_{\mathcal{O}}, \mathsf{sk}_{\mathcal{O}})$ generiert. Nach erfolgreicher Durchführung wird ein Tripel $(\mathsf{pk}_{\mathcal{K}}, \mathsf{pk}_{\mathcal{O}}, \mathsf{sk}_{\mathcal{O}})$ in der Liste der korrupten Kunden $\mathcal{U}_{\mathcal{A}}$ gespeichert.

- $\mathcal{O}^{\mathsf{R}}_{\mathcal{K},\mathcal{TTP}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ eine Durchführung des Registrierungsprotokolls $\mathsf{Registry}$ ansehen, um ehrliche Kunden (bzw. Händler) in das System hinzuzufügen. Das Orakel $\mathcal{O}^{\mathsf{R}}_{\mathcal{K},\mathcal{TTP}}$ verhält sich wie ein Kunde, der ein Schlüsselpaar $(\mathsf{pk}_{\mathcal{K}}, \mathsf{sk}_{\mathcal{K}})$ generiert, und wie die Trusted-Third-Party, die ein Schlüsselpaar $(\mathsf{pk}_{\mathcal{O}}, \mathsf{sk}_{\mathcal{O}})$ generiert, jeweils gemäß Protokoll. Der Angreifer erhält alle Daten, die zwischen dem Kunden und der Trusted-Third-Party gesendet werden. Anschließend wird der öffentliche Schlüssel $\mathsf{pk}_{\mathcal{K}}$ an $\mathcal{A}$ gesendet und der Eintrag $(\mathsf{pk}_{\mathcal{K}}, \mathsf{sk}_{\mathcal{K}}, \mathsf{pk}_{\mathcal{O}}, \mathsf{sk}_{\mathcal{O}})$ in der Liste der ehrlichen Kunden $\mathcal{U}_H$ gespeichert.

Dabei ist die Ausgabe der Orakel $\mathcal{O}^{\mathsf{UT}}_{\mathcal{D}}, \mathcal{O}^{\mathsf{UT}}_{\mathcal{B},\mathcal{D}}, \mathcal{O}^{\mathsf{CT}}_{\mathcal{D}}, \mathcal{O}^{\mathsf{CT}}_{\mathcal{B},\mathcal{D}}$ und des Algorithmus DGen stets ein leerer String, wenn es sich nicht um ein faires elektronisches Geldsystem handelt, da dann die entsprechenden Algorithmen und Protokolle nicht definiert sind.

Entsprechend ist die Ausgabe der Orakel $\mathcal{O}^{\mathsf{R}}_{\mathcal{TTP}}$ und $\mathcal{O}^{\mathsf{R}}_{\mathcal{K},\mathcal{TTP}}$ ebenfalls stets ein leerer String, wenn das Protokoll $\mathsf{Registry}$ nicht definiert ist.

Wenn es sich um ein elektronisches Geldsystem mit Observern handelt, werden die Orakel $\mathcal{O}^{\mathsf{A}}_{\mathcal{B}}, \mathcal{O}^{\mathsf{W}}_{\mathcal{B}}, \mathcal{O}^{\mathsf{S}}_{\mathcal{H}}, \mathcal{O}^{\mathsf{S}^*}_{\mathcal{H}}$ bei den folgenden Spielen entsprechend durch die Orakel $\mathcal{O}^{\mathsf{A}}_{\mathcal{O},\mathcal{B}}$, $\mathcal{O}^{\mathsf{W}}_{\mathcal{O},\mathcal{B}}, \mathcal{O}^{\mathsf{S}}_{\mathcal{O},\mathcal{H}}, \mathcal{O}^{\mathsf{S}^*}_{\mathcal{O},\mathcal{H}}$ ersetzt.

Unfälschbarkeit: Wenn eine Kollusion korrupter Kunden und Händler in Abhebeprotokollen Geldbörsen im Wert von $\$_1$ bei der Bank abhebt und in Bezahlprotokollen Münzen im Wert von $\$_2$ von ehrlichen Kunden erhält, soll sie nicht in der Lage sein, Münzen im Wert von mehr als $\$_1 + \$_2$ bei ehrlichen Händlern durch Bezahlprotokolle auszugeben bzw. durch Einlöseprotokolle bei der Bank einzulösen, ohne dass ein betrügerischer Kunde identifiziert wird. Somit ist dies das wichtigste Sicherheitsziel für die Bank.

Die Unfälschbarkeit wird in der Literatur auch häufig als *Balance* bezeichnet (z.B. [CHL05, ASM07, AWSM07]) und bei einigen Autoren (bspw. [CHL05, CG07, CG10, IL13]) wird die Identifikation eines Double-Spenders als getrenntes Sicherheitsziel *Identification of Double-Spenders* definiert. Wie in [Tro06, ASM07, AWSM07, ASM08, Au09] kann dies allerdings auch gemeinsam definiert werden.

Spiel 4.4.1 (Unfälschbarkeit). *Das Spiel wird zwischen einem polynomiellen Angreifer $\mathcal{A}$ und einem Challenger $\mathcal{C}$ durchgeführt.*

Initialisierung: *Der Challenger $\mathcal{C}$ wählt einen hinreichend großen Sicherheitsparameter $\lambda \in \mathbb{N}$ und führt die Algorithmen $\mathsf{Setup}, \mathsf{BGen}$ sowie DGen aus. Dann sendet $\mathcal{C}$ die*

Systemparameter sp *sowie die öffentlichen Schlüssel* $\mathsf{pk}_{\mathcal{B}}$ *und* $\mathsf{pk}_{\mathcal{D}}$ *an den Angreifer* $\mathcal{A}$*.*

Probe-Phase: *Der Angreifer* $\mathcal{A}$ *kann adaptiv eine polynomiell beschränkte Anzahl an Anfragen an die Orakel* $\mathcal{O}^{\mathsf{H}}, \mathcal{O}^{\mathsf{R}}_{\mathcal{TTP}}, \mathcal{O}^{\mathsf{R}}_{\mathcal{K},\mathcal{TTP}}, \mathcal{O}^{\mathsf{A}}_{\mathcal{B}}, \mathcal{O}^{\mathsf{A}}_{\mathcal{K},\mathcal{B}}, \mathcal{O}^{\mathsf{W}}_{\mathcal{B}}, \mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}, \mathcal{O}^{\mathsf{S}}_{\mathcal{H}}, \mathcal{O}^{\mathsf{S}}_{\mathcal{K}}, \mathcal{O}^{\mathsf{S}}_{\mathcal{K},\mathcal{H}}, \mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$, $\mathcal{O}^{\mathsf{D}}_{\mathcal{H},\mathcal{B}}, \mathcal{O}^{\mathsf{VW}}_{\mathcal{B}}, \mathcal{O}^{\mathsf{UT}}_{\mathcal{B},\mathcal{D}}$ *und* $\mathcal{O}^{\mathsf{CT}}_{\mathcal{B},\mathcal{D}}$ *stellen, um Kunden zu generieren, Abhebe-, Bezahl- sowie Einlöseprotokolle durchzuführen und Verfolgungsprotokolle anzusehen.*

Ende: *Wenn der Angreifer* $\mathcal{A}$ *entscheidet das Spiel zu beenden, löst der Challenger* $\mathcal{C}$ *zunächst jede Münze* coin *mit* $(\cdot,\cdot,\mathsf{coin}) \in \mathbb{C}_{\mathcal{A}} \setminus \mathbb{D}_H$ *mit dem Einlöseprotokoll* $\mathsf{Deposit}$ *ein. Der Angreifer* $\mathcal{A}$ *gewinnt nun das Spiel, falls* $\$^* > \$$ *ist und der* $\mathsf{Identify}$*-Algorithmus keinen öffentlichen Schlüssel* $\mathsf{pk} \in \mathcal{U}_{\mathcal{A}}$ *ausgibt.*

Definition 4.4.1 (**Unfälschbarkeit**)**.** *Ein elektronisches Geldsystem erfüllt die Sicherheitseigenschaft* Unfälschbarkeit*, wenn es für jeden polynomiellen Angreifer* $\mathcal{A}$ *eine vernachlässigbare Funktion* ν *gibt, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\Pr[\mathcal{A} \textit{ gewinnt Spiel 4.4.1}] \leq \nu(\lambda).$$

Anonymität: Keine Kollusion korrupter Kunden, Händler und der Bank soll entscheiden können, welche Münze von welchem ehrlichen Kunden ausgegeben wurde, solange diese Münze nicht mehrfach ausgegeben und kein entsprechendes Verfolgungsprotokoll durchgeführt wird. Selbst wenn ein ehrlicher Kunde einige Münzen mehrfach ausgibt, soll die Anonymität der übrigen Münzen weiterhin gewährleistet sein.

Durch die Anonymität soll also impliziert sein, dass eine Münze weder zum entsprechenden Abhebeprotokoll (und damit zum Eigentümer dieser Münze) noch zu anderen Münzen desselben Abhebeprotokolls verknüpft werden kann, was bei manchen Autoren als unterschiedliche Eigenschaften angesehen werden (z.B. [CPS94, Oka95, Tsi97, CFT98, NS00, CG07]). Daher existieren in der Literatur verschiedene Begriffe wie *Anonymity*, *Untraceability* und *Unlinkability*, die unterschiedliche Bedeutungen haben können.

Spiel 4.4.2 (**Anonymität**)**.** *Das Spiel wird zwischen einem polynomiellen Angreifer* $\mathcal{A}$ *und einem Challenger* $\mathcal{C}$ *durchgeführt.*

Initialisierung: *Der Challenger* $\mathcal{C}$ *wählt einen hinreichend großen Sicherheitsparameter* $\lambda \in \mathbb{N}$ *und führt die Algorithmen* Setup *und* DGen *aus. Dann sendet* $\mathcal{C}$ *die Systemparameter* sp *sowie den öffentlichen Schlüssel* $\mathsf{pk}_{\mathcal{D}}$ *an* $\mathcal{A}$*. Der Angreifer* $\mathcal{A}$ *generiert das Schlüsselpaar der Bank und sendet den öffentlichen Schlüssel* $\mathsf{pk}_{\mathcal{B}}$ *an* $\mathcal{C}$*.*

Probe-Phase: *Der Angreifer* $\mathcal{A}$ *kann adaptiv eine polynomiell beschränkte Anzahl an Anfragen an die Orakel* $\mathcal{O}^{\mathsf{H}}, \mathcal{O}^{\mathsf{R}}_{\mathcal{TTP}}, \mathcal{O}^{\mathsf{R}}_{\mathcal{K},\mathcal{TTP}}, \mathcal{O}^{\mathsf{A}}, \mathcal{O}^{\mathsf{A}}_{\mathcal{K}}, \mathcal{O}^{\mathsf{W}}_{\mathcal{K}}, \mathcal{O}^{\mathsf{S}^\star}_{\mathcal{H}}, \mathcal{O}^{\mathsf{S}^\star}_{\mathcal{K}}, \mathcal{O}^{\mathsf{S}^\star}_{\mathcal{K},\mathcal{H}}, \mathcal{O}^{\mathsf{D}}_{\mathcal{H}}, \mathcal{O}^{\mathsf{UT}}_{\mathcal{D}}$ *und* $\mathcal{O}^{\mathsf{CT}}_{\mathcal{D}}$ *stellen, um Kunden zu generieren sowie Abhebe-, Bezahl- und Einlöse- und die Verfolgungsprotokolle durchzuführen.*

Challenge-Phase: *Der Angreifer $\mathcal{A}$ sendet zwei verschiedene Indizes i_0, i_1 mit $(i_0, \mathbb{W}_0, \mathsf{pk}_0, \$_0), (i_1, \mathbb{W}_1, \mathsf{pk}_1, \$_1) \in \mathbb{W}_H$, zwei öffentliche Schlüssel $\mathsf{pk}_{\mathcal{H}_b}, \mathsf{pk}_{\mathcal{H}_{\bar{b}}} \in \mathcal{U}_{\mathcal{A}} \cup \mathcal{U}_H$, zwei Zeitpunkte $\mathsf{ts}_b, \mathsf{ts}_{\bar{b}}$ und zwei Münzwerte $k_b, k_{\bar{b}}$, wobei auch $\mathsf{pk}_0 = \mathsf{pk}_1$ und $\left(\mathsf{pk}_{\mathcal{H}_b}, \mathsf{ts}_b, k_b\right) = \left(\mathsf{pk}_{\mathcal{H}_{\bar{b}}}, \mathsf{ts}_{\bar{b}}, k_{\bar{b}}\right)$ erlaubt sind, so dass die folgenden Bedingungen gelten:*

- *Es gilt $0 \leq k_b, k_{\bar{b}} \leq \min\{\$_0, \$_1\}$, da ein Kunde andernfalls zu einem Double-Spending gezwungen wird.*
- *Der Angreifer $\mathcal{A}$ hat das Orakel $\mathcal{O}_{\mathcal{D}}^{\mathsf{CT}}$ nicht bzgl. Protokollansichten $\mathsf{view}_0 = (\mathbb{T}_0, \cdot)$ und $\mathsf{view}_1 = (\mathbb{T}_1, \cdot)$ mit $(i_0, \mathbb{T}_0), (i_1, \mathbb{T}_1) \in \mathbb{T}_H$ angefragt.*

Der Challenger $\mathcal{C}$ wählt zufällig ein Bit $b \in_{\mathcal{R}} \{0,1\}$ und setzt $\bar{b} = 1 - b$. Dann führt $\mathcal{C}$ zwei Bezahlprotokolle

$$\mathbb{W}'_b \leftarrow \mathcal{C}\left(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{D}}, k_b, \mathsf{ts}_b, \mathsf{pk}_{\mathcal{H}_b}, \mathsf{sk}_b, \mathbb{W}_b\right) \leftrightarrow \mathcal{A}(\cdot) \rightarrow \mathsf{coin}_b$$

und

$$\mathbb{W}'_{\bar{b}} \leftarrow \mathcal{C}\left(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{D}}, k_{\bar{b}}, \mathsf{ts}_{\bar{b}}, \mathsf{pk}_{\mathcal{H}_{\bar{b}}}, \mathsf{sk}_{\bar{b}}, \mathbb{W}_{\bar{b}}\right) \leftrightarrow \mathcal{A}(\cdot) \rightarrow \mathsf{coin}_{\bar{b}}$$

mit dem Angreifer $\mathcal{A}$ aus, so dass $\mathcal{A}$ die beiden Münzen coin_b und $\mathsf{coin}_{\bar{b}}$ erhält. Anschließend werden die beiden Einträge in der Liste $\mathbb{W}_H$ zu $(i_0, \mathbb{W}'_0, \mathsf{pk}_0, \$_0 - k_0)$ und $(i_1, \mathbb{W}'_1, \mathsf{pk}_1, \$_1 - k_1)$ aktualisiert. Wenn $k_{\hat{b}} = 0$ für $\hat{b} \in \{0,1\}$ ist, wird kein Bezahlprotokoll bzgl. $i_{\hat{b}}$ durchgeführt.

Post-Challenge-Phase: *Der Angreifer $\mathcal{A}$ kann erneut adaptiv eine polynomiell beschränkte Anzahl an Anfragen an die Orakel $\mathcal{O}^{\mathsf{H}}, \mathcal{O}^{\mathsf{R}}_{\mathcal{TTP}}, \mathcal{O}^{\mathsf{R}}_{\mathcal{K},\mathcal{TTP}}, \mathcal{O}^{\mathsf{A}}, \mathcal{O}^{\mathsf{A}}_{\mathcal{K}}, \mathcal{O}^{\mathsf{W}}_{\mathcal{K}}, \mathcal{O}^{\mathsf{S}^\star}_{\mathcal{H}}, \mathcal{O}^{\mathsf{S}^\star}_{\mathcal{K}}, \mathcal{O}^{\mathsf{S}^\star}_{\mathcal{K},\mathcal{H}}, \mathcal{O}^{\mathsf{D}}_{\mathcal{H}}, \mathcal{O}^{\mathsf{UT}}_{\mathcal{D}}$ und $\mathcal{O}^{\mathsf{CT}}_{\mathcal{D}}$ mit folgenden Einschränkungen stellen. Der Angreifer $\mathcal{A}$ darf*

- *die Orakel $\mathcal{O}^{\mathsf{S}^\star}_{\mathcal{K}}$ und $\mathcal{O}^{\mathsf{S}^\star}_{\mathcal{K},\mathcal{H}}$ bzgl. des in der Challenge gewählten Index i_0 maximal bzgl. Münzen im Gesamtwert von $\$_0 - \max\{k_b, k_{\bar{b}}\}$ und bzgl. des Index i_1 maximal bzgl. Münzen im Gesamtwert von $\$_1 - \max\{k_b, k_{\bar{b}}\}$ anfragen (andernfalls könnte $\mathcal{A}$ ein Double-Spending bzgl. der Geldbörsen $\mathbb{W}_0$ oder $\mathbb{W}_1$ erzwingen und die Anonymität brechen) sowie*
- *das Orakel $\mathcal{O}^{\mathsf{UT}}_{\mathcal{D}}$ nicht bzgl. der Münzen coin_b und $\mathsf{coin}_{\bar{b}}$ sowie das Orakel $\mathcal{O}^{\mathsf{CT}}_{\mathcal{D}}$ nicht bzgl. Protokollansichten $\mathsf{view}_0 = (\mathbb{T}_0, \cdot)$ und $\mathsf{view}_1 = (\mathbb{T}_1, \cdot)$ mit $(i_0, \mathbb{T}_0)$, $(i_1, \mathbb{T}_1) \in \mathbb{T}_H$ anfragen.*

Ende: *Der Angreifer $\mathcal{A}$ gibt ein Bit b' aus und gewinnt das Spiel, falls $b' = b$ ist.*

Definition 4.4.2 (Anonymität). *Ein elektronisches Geldsystem erfüllt die Sicherheitseigenschaft* Anonymität*, wenn es für jeden polynomiellen Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion ν gibt, so dass für alle hinreichend großen $\lambda \in \mathbb{N}$ gilt:*

$$\Pr\left[\mathcal{A} \text{ gewinnt Spiel 4.4.2}\right] \leq \frac{1}{2} + \nu(\lambda).$$

Double-Spending-Beschuldigung: Keine Kollusion korrupter Kunden, Händler und der Bank soll in der Lage sein, einen ehrlichen Kunden des Double-Spendings bzgl. zweier Münzen zu beschuldigen, die nicht beide von diesem Kunden ausgegeben wurden. Wenn ein Kunde mehrere Münzen mehrfach ausgibt, soll er also nur für diese Münzen des Double-Spendings beschuldigt werden können und nicht fälschlicherweise für andere.

Somit entspricht die Double-Spending-Beschuldigung dem Grundgedanken der Eigenschaft *strong Exculpability* aus [CHL05] bzw. *Non-Frameability* aus [Tro06], ist aber stärker als diese (siehe Unterabschnitt 4.5.1).

Spiel 4.4.3 (**Double-Spending-Beschuldigung**). *Das Spiel wird zwischen einem polynomiellen Angreifer $\mathcal{A}$ und einem Challenger $\mathcal{C}$ durchgeführt.*

Initialisierung: *Wie bei Spiel 4.4.2 (Anonymität).*

Probe-Phase: *Wie bei Spiel 4.4.2 (Anonymität).*

Ende: *Der Angreifer $\mathcal{A}$ gibt einen öffentlichen Schlüssel $\mathsf{pk} \in \mathcal{U}_H$ sowie einen Beweis Π_G, der zwei Münzen coin sowie coin' enthält, aus und gewinnt das Spiel, falls $\mathsf{VerifyGuilt}(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{D}}, \mathsf{pk}, \Pi_G) = 1$ gilt, obwohl nicht $(\cdot, \mathsf{pk}, \mathsf{coin}), (\cdot, \mathsf{pk}, \mathsf{coin}') \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ ist.*

Definition 4.4.3 (**Double-Spending-Beschuldigung**). *Ein elektronisches Geldsystem erfüllt die Sicherheitseigenschaft* Double-Spending-Beschuldigung*, wenn es für jeden polynomiellen Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion ν gibt, so dass für alle hinreichend großen $\lambda \in \mathbb{N}$ gilt:*

$$\Pr[\mathcal{A} \text{ gewinnt Spiel 4.4.3}] \leq \nu(\lambda).$$

Abhebe-Beschuldigung: Keine Kollusion korrupter Kunden, Händler und der Bank soll in der Lage sein, eine Protokollansicht bzgl. eines ehrlichen Kunden zu erzeugen, der das entsprechende Abhebeprotokoll nicht durchgeführt hat. Dies ist wichtig, damit die Bank einem Kunden nur Geld abbuchen kann, wenn der Kunde dieses Geld wirklich abgehoben hat (vgl. [Tro06]). Weiter könnte die Bank somit aber auch einer dritten Partei nachweisen, welcher Kunde zu welchem Zeitpunkt Geld abgehoben hat und die Bank somit überwacht werden (vgl. [STS99a, STS99c]).

In [CG07] wird behauptet, dass die Eigenschaft Abhebe-Beschuldigung durch die in [CHL05] definierte *weak Exculpability* (was eine schwächere Form der Double-Spending-Beschuldigung ist) impliziert wird. Dies ist allerdings falsch. Ein Gegenbeispiel ist das Geldsystem in [ASM08] (siehe Unterabschnitt 4.5.1).

Spiel 4.4.4 (**Abhebe-Beschuldigung**). *Das Spiel wird zwischen einem polynomiellen Angreifer $\mathcal{A}$ und einem Challenger $\mathcal{C}$ durchgeführt.*

Initialisierung: *Wie bei Spiel 4.4.2 (Anonymität).*

Probe-Phase: *Wie bei Spiel 4.4.2 (Anonymität).*

Ende: *Der Angreifer $\mathcal{A}$ gibt einen öffentlichen Schlüssel $\mathsf{pk} \in \mathcal{U}_H$ sowie eine Protokollansicht $\mathsf{view} = (\mathbb{T}, \sigma')$ aus und gewinnt das Spiel, falls* $\mathsf{VerifyWithdraw}(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{D}}, \mathsf{pk}, \mathsf{view}) = 1$ *gilt, obwohl $(\cdot, \mathbb{T}) \notin \mathbb{T}_H$ ist.*

Definition 4.4.4 (**Abhebe-Beschuldigung**). *Ein elektronisches Geldsystem erfüllt die Sicherheitseigenschaft* Abhebe-Beschuldigung*, wenn es für jeden polynomiellen Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion ν gibt, so dass für alle hinreichend großen $\lambda \in \mathbb{N}$ gilt:*

$$\Pr[\mathcal{A} \text{ gewinnt Spiel 4.4.4}] \leq \nu(\lambda).$$

Korrekte Kundenverfolgung: Keine Kollusion korrupter Kunden und Händler soll in der Lage sein, eine Münze auszugeben oder einzulösen, so dass nicht der korrupte Eigentümer der Münze durch das Kundenverfolgungsprotokoll identifiziert wird.

Spiel 4.4.5 (**Korrekte Kundenverfolgung**). *Das Spiel wird zwischen einem polynomiellen Angreifer $\mathcal{A}$ und einem Challenger $\mathcal{C}$ durchgeführt.*

Initialisierung: *Wie bei Spiel 4.4.1 (Unfälschbarkeit).*

Probe-Phase: *Wie bei Spiel 4.4.1 (Unfälschbarkeit).*

Ende: *Der Angreifer $\mathcal{A}$ gibt eine Münze* coin *mit $(\cdot, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ aus und gewinnt das Spiel, falls das Kundenverfolgungsprotokoll* UTrace *bzgl.* coin *nicht genau einen öffentlichen Schlüssel $\mathsf{pk} \in \mathcal{U}_{\mathcal{A}}$ ausgibt.*

Definition 4.4.5 (**Korrekte Kundenverfolgung**). *Ein faires elektronisches Geldsystem erfüllt die Sicherheitseigenschaft* korrekte Kundenverfolgung*, wenn es für jeden polynomiellen Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion ν gibt, so dass für alle hinreichend großen $\lambda \in \mathbb{N}$ gilt:*

$$\Pr[\mathcal{A} \text{ gewinnt Spiel 4.4.5}] \leq \nu(\lambda).$$

Kundenverfolgung-Beschuldigung: Keine Kollusion korrupter Kunden, Händler und der Bank soll in der Lage sein, eine Münze zu erzeugen, so dass fälschlicherweise ein Kunde durch das Kundenverfolgungsprotokoll beschuldigt wird, obwohl dieser die entsprechende Münze nicht ausgegeben hat. Dabei gibt es die beiden Möglichkeiten, dass (1) ein ehrlicher Kunde beschuldigt wird, obwohl dieser diese Münze nicht ausgegeben hat, oder (2) der ehrliche Kunde nicht „beschuldigt“ wird, obwohl dieser die Münze ausgegeben hat.

(Beachte, dass Fall (2) nicht notwendigerweise erforderlich ist, da dies für bisherige und die in dieser Arbeit entwickelten elektronischen Geldsysteme keinen Vorteil für die ehrlichen Kunden darstellt. Diese Eigenschaft ist allerdings dann notwendig, wenn ein Kunde nachweisen will, dass eine bestimmte Münze tatsächlich von ihm ausgegeben wurde und er der ursprüngliche Eigentümer ist.)

Spiel 4.4.6 (**Kundenverfolgung-Beschuldigung**). *Das Spiel wird zwischen einem polynomiellen Angreifer $\mathcal{A}$ und einem Challenger $\mathcal{C}$ durchgeführt.*

Initialisierung: *Wie bei Spiel 4.4.2 (Anonymität).*

Probe-Phase: *Wie bei Spiel 4.4.2 (Anonymität).*

Ende: *Der Angreifer $\mathcal{A}$ führt mit dem Challenger $\mathcal{C}$ das Kundenverfolgungsprotokoll* UTrace *bzgl. einer Münze* coin *aus und gewinnt das Spiel, falls die Ausgabe ein öffentlicher Schlüssel* pk *ist, wobei* $\mathsf{pk} \in \mathcal{U}_H$ *oder* $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ *gilt, obwohl* $(\cdot, \mathsf{pk}, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ *ist.*

Definition 4.4.6 (**Kundenverfolgung-Beschuldigung**). *Ein faires elektronisches Geldsystem erfüllt die Sicherheitseigenschaft* Kundenverfolgung-Beschuldigung*, wenn es für jeden polynomiellen Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion ν gibt, so dass für alle hinreichend großen $\lambda \in \mathbb{N}$ gilt:*

$$\Pr[\mathcal{A} \text{ gewinnt Spiel 4.4.6}] \leq \nu(\lambda).$$

Korrekte Münzverfolgung: Keine Kollusion korrupter Kunden und Händler soll in der Lage sein, eine Münze auszugeben oder einzulösen, so dass diese Münze nicht eindeutig durch das Münzverfolgungsprotokoll verfolgt wird.

Spiel 4.4.7 (**Korrekte Münzverfolgung**). *Das Spiel wird zwischen einem polynomiellen Angreifer $\mathcal{A}$ und einem Challenger $\mathcal{C}$ durchgeführt.*

Initialisierung: *Wie bei Spiel 4.4.1 (Unfälschbarkeit).*

Probe-Phase: *Wie bei Spiel 4.4.1 (Unfälschbarkeit).*

Ende: *Der Angreifer $\mathcal{A}$ gibt eine Münze* coin *mit* $(\cdot, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ *aus. Der Challenger $\mathcal{C}$ führt jetzt zu jeder Protokollansicht* view *mit* $(\cdot, \mathsf{view}) \in \mathsf{view}_{\mathcal{A}} \cup \mathsf{view}_H$ *das Münzverfolgungsprotokoll* CTrace *durch und der Angreifer $\mathcal{A}$ gewinnt das Spiel, falls die Münze* coin *nicht bei genau einer* CTrace *Durchführung ausgegeben wird.*

Definition 4.4.7 (**Korrekte Münzverfolgung**). *Ein faires elektronisches Geldsystem erfüllt die Sicherheitseigenschaft* korrekte Münzverfolgung*, wenn es für jeden polynomiellen Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion ν gibt, so dass für alle hinreichend großen $\lambda \in \mathbb{N}$ gilt:*

$$\Pr[\mathcal{A} \text{ gewinnt Spiel 4.4.7}] \leq \nu(\lambda).$$

Münzverfolgung-Beschuldigung: Keine Kollusion korrupter Kunden, Händler und der Bank soll in der Lage sein, eine Protokollansicht sowie eine Münze zu erzeugen, so dass die Münze fälschlicherweise durch das Münzverfolgungsprotokoll verfolgt wird, obwohl sie nicht vom entsprechendem Abhebeprotokoll stammt. Dabei gibt es die beiden Möglichkeiten, dass (1) eine Münze eines ehrlichen Kunden verfolgt wird, obwohl dieser das entsprechende Abhebeprotokoll nicht durchgeführt hat, oder (2) eine Münze verfolgt wird, obwohl diese nicht vom entsprechenden Abhebeprotokoll eines ehrlichen Kunden stammt.

Die Eigenschaft (1) garantiert, dass Münzen eines ehrlichen Kunden nicht fälschlicherweise verfolgt und gesperrt werden können. Der Fall (2) wird benötigt, wenn durch das Münzverfolgungsprotokoll die Aktivitäten verdächtiger Kunden überwacht werden (vgl. [FTY96] und Abschnitt 4.2). Denn durch diese Eigenschaft können ehrliche Kunden nicht irrtümlich beschuldigt werden, gewisse Münzen ausgegeben zu haben. Diese Eigenschaft kann dabei direkt durch die Konstruktion der Münzverfolgung realisiert werden, oder dadurch, dass im Anschluss bei jeder verfolgten Münze zusätzlich das Kundenverfolgungsprotokoll durchgeführt wird (siehe Unterabschnitt 7.6.2).

Spiel 4.4.8 (Münzverfolgung-Beschuldigung). *Das Spiel wird zwischen einem polynomiellen Angreifer $\mathcal{A}$ und einem Challenger $\mathcal{C}$ durchgeführt.*

Initialisierung: *Wie bei Spiel 4.4.2 (Anonymität).*

Probe-Phase: *Wie bei Spiel 4.4.2 (Anonymität).*

Ende: *Der Angreifer $\mathcal{A}$ führt mit dem Challenger $\mathcal{C}$ das Münzverfolgungsprotokoll* C-Trace *bzgl. einer Protokollansicht* $\mathsf{view} = (\mathbb{T}, \sigma')$ *durch und gewinnt das Spiel, falls die Ausgabe eine Münze* coin *enthält, obwohl*

- $(i, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ *und* $(i, \mathbb{T}) \notin \mathbb{T}_H$ *oder*
- $(i, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ *und* $(i, \mathbb{T}) \in \mathbb{T}_H$ *ist.*

Somit gibt es die drei Fälle,

- $(i, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ *und* $(i', \mathbb{T}) \in \mathbb{T}_H$ *für* $i' \neq i$*, oder*
- $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ *und* $(\cdot, \mathbb{T}) \notin \mathbb{T}_H$*, oder*
- $(\cdot, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ *und* $(\cdot, \mathbb{T}) \in \mathbb{T}_H$*.*

Definition 4.4.8 (Münzverfolgung-Beschuldigung). *Ein faires elektronisches Geldsystem erfüllt die Sicherheitseigenschaft* Münzverfolgung-Beschuldigung*, wenn es für jeden polynomiellen Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion ν gibt, so dass für alle hinreichend großen $\lambda \in \mathbb{N}$ gilt:*

$$\Pr[\mathcal{A} \textit{ gewinnt Spiel 4.4.8}] \leq \nu(\lambda).$$

Over-Spending-Prevention: Wenn eine Kollusion korrupter Kunden und Händler Geldbörsen im Wert von $\$_1$ von der Bank und Münzen im Wert von $\$_2$ von ehrlichen Kunden erhält, soll sie nicht in der Lage sein, Münzen im Wert von mehr als $\$_1 + \$_2$ bei ehrlichen Händlern auszugeben bzw. bei der Bank einzulösen.

Spiel 4.4.9 (**Over-Spending-Prevention**). *Das Spiel wird zwischen einem polynomiellen Angreifer $\mathcal{A}$ und einem Challenger $\mathcal{C}$ durchgeführt.*

Initialisierung: *Wie bei Spiel 4.4.1 (Unfälschbarkeit).*

Probe-Phase: *Wie bei Spiel 4.4.1 (Unfälschbarkeit).*

Ende: *Der Angreifer $\mathcal{A}$ gewinnt das Spiel, falls $\$^* > \$$ ist.*

Definition 4.4.9 (**Over-Spending-Prevention**). *Ein elektronisches Geldsystem mit Observer erfüllt die Sicherheitseigenschaft* Over-Spending-Prevention*, wenn es für jeden polynomiellen Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion ν gibt, so dass für alle hinreichend großen $\lambda \in \mathbb{N}$ gilt:*

$$\Pr[\mathcal{A} \textit{ gewinnt Spiel 4.4.9}] \leq \nu(\lambda).$$

Double-Spending-Prevention: Keine Kollusion korrupter Kunden und Händler soll in der Lage sein, einen Serienschlüssel mehrmals auszugeben.

Spiel 4.4.10 (**Double-Spending-Prevention**). *Das Spiel wird zwischen einem polynomiellen Angreifer $\mathcal{A}$ und einem Challenger $\mathcal{C}$ durchgeführt.*

Initialisierung: *Wie bei Spiel 4.4.1 (Unfälschbarkeit).*

Probe-Phase: *Wie bei Spiel 4.4.1 (Unfälschbarkeit).*

Ende: *Der Angreifer $\mathcal{A}$ gibt zwei Münzen* coin *und* coin' *aus und gewinnt das Spiel, falls die Ausgabe von* $\mathsf{Identify}(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{D}}, \mathsf{coin}, \mathsf{coin}')$ *einen öffentlichen Schlüssel* $\mathsf{pk} \in \mathcal{U}_{\mathcal{A}}$ *enthält.*

Definition 4.4.10 (**Double-Spending-Prevention**). *Ein elektronisches Geldsystem mit Observer erfüllt die Sicherheitseigenschaft* Double-Spending-Prevention*, wenn es für jeden polynomiellen Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion ν gibt, so dass für alle hinreichend großen $\lambda \in \mathbb{N}$ gilt:*

$$\Pr[\mathcal{A} \textit{ gewinnt Spiel 4.4.10}] \leq \nu(\lambda).$$

Eingehende und ausgehende Informationen: Keine Kollusion korrupter Kunden, Händler und der Bank kann einem Observer, der einem ehrlichen Kunden gehört, zusätzliche Informationen zukommen lassen, oder von diesem zusätzliche Informationen erhalten. Somit wird zwischen *eingehenden* und *ausgehenden* Informationen unterschieden (vgl. [CP93, Sch09]).

Eingehende Informationen sind zusätzliche Informationen, die die Bank bzw. ein Händler einem Observer, der einem ehrlichen Kunden gehört, im Abhebe- bzw. Bezahlprotokoll zukommen lässt, indem Bank, Händler und Observer vom Abhebe- bzw. Bezahlprotokoll abweichen.

Ausgehende Informationen sind zusätzliche Informationen, die ein Observer, der einem ehrlichen Kunden gehört, der Bank bzw. einem Händler im Abhebe- bzw. Bezahlprotokoll zukommen lässt, indem Bank, Händler und Observer vom Abhebe- bzw. Bezahlprotokoll abweichen.

Definition 4.4.11 (**Sicherheit eines elektronischen Geldsystems**)**.** *Ein elektronisches Geldsystem heißt* sicher*, wenn es die Sicherheitseigenschaften Unfälschbarkeit, Anonymität, Double-Spending-Beschuldigung und Abhebe-Beschuldigung erfüllt.*

Definition 4.4.12 (**Sicherheit eines fairen elektronischen Geldsystems**)**.** *Ein faires elektronisches Geldsystem heißt* sicher*, wenn es die Sicherheitseigenschaften Unfälschbarkeit, Anonymität, Double-Spending-Beschuldigung, Abhebe-Beschuldigung, korrekte Kundenverfolgung, Kundenverfolgung-Beschuldigung, korrekte Münzverfolgung und Münzverfolgung-Beschuldigung erfüllt.*

Definition 4.4.13 (**Sicherheit eines elektronischen Geldsystems mit Observern**)**.** *Ein elektronisches Geldsystem mit Observern heißt* sicher*, wenn es die Sicherheitseigenschaften Unfälschbarkeit, Anonymität, Double-Spending-Beschuldigung, Abhebe-Beschuldigung und Double-Spending-Prevention erfüllt und sowohl eingehende als auch ausgehende Informationen verhindert werden.*

Definition 4.4.14 (**Sicherheit eines fairen elektronischen Geldsystems mit Observern**)**.** *Ein faires elektronisches Geldsystem mit Observern heißt* sicher*, wenn es die Sicherheitseigenschaften Unfälschbarkeit, Anonymität, Double-Spending-Beschuldigung, Abhebe-Beschuldigung, korrekte Kundenverfolgung, Kundenverfolgung-Beschuldigung, korrekte Münzverfolgung, Münzverfolgung-Beschuldigung und Double-Spending-Prevention erfüllt und sowohl eingehende als auch ausgehende Informationen verhindert werden.*

4.5 Aktueller Forschungsstand

Aktuell sind im Wesentlichen zwei kompakte elektronische Geldsysteme [CHL05] (bzw. die ausführliche Version [CHL06b]) und [AWSM07] sowie zwei teilbare elektronische

Geldsysteme [CG07] und [IL13] vorhanden, die die zentralen Sicherheitsanforderungen Unfälschbarkeit, Anonymität und Double-Spending-Beschuldigung (jedoch mit schwächeren Definitionen, vgl. Unterabschnitt 4.5.1) erfüllen. Das teilbare elektronische Geldsystem [CG10] erfüllt zwar nach Aussage der Autoren die zentralen Sicherheitsanforderungen, allerdings wird in Unterabschnitt 6.3.1 ein Angriff entwickelt, mit dem die Unfälschbarkeit (in [CG10] entspricht dies der Sicherheitseigenschaft Unforgeability) des Geldsystems gebrochen werden kann. Die weiteren kompakten elektronischen Geldsysteme [Wei05, CHL06a, ASM07, BCKL09] bauen auf dem in [CHL05] vorgestellten System auf. Die elektronischen Geldsysteme [BCKL09] und [IL13] sind sogar im Standard-Modell sicher, während alle anderen genannten elektronischen Geldsysteme auf das Random-Oracle-Modell angewiesen sind. Die teilbaren Geldsysteme [CG07] und [IL13] sind allerdings durch die aufwendigen Zero-Knowledge-Beweise wenig effizient und es ist sogar fraglich, ob das System in [CG07] überhaupt implementiert werden kann (vgl. [ASM08, CG10, IL13]). Von den oben genannten Geldsystemen ist allerdings keines fair oder verwendet Observer.

Andere Systeme erfüllen die Sicherheitsziele nicht oder nur teilweise bzw. „abgeschwächt“. So gewährleisten nahezu alle teilbaren Geldsysteme, die vor 2005 entwickelt wurden (z.B. [EO94, Oka95, CFT98]), keine Anonymität, sondern lediglich eine Art *Pseudonymität*, so dass einige Münzen eines Kunden verknüpft, aber nicht dem jeweiligen Kunden zugeordnet werden können. Das teilbare Geldsystem [ASM08] gewährleistet nur eine *statistische Unfälschbarkeit*, so dass für die Bank zwar auf lange Sicht ein Verlust vermieden wird, aber ein Angreifer mit einer Wahrscheinlichkeit von 50 Prozent elektronische Münzen fälschen kann. Eine weitere Schwachstelle einiger elektronischer Geldsysteme ist, dass während des Bezahlprotokolls einige Informationen bezüglich der verwendeten Münzen preisgegeben werden, wodurch die Anonymität nur noch unter bestimmten Voraussetzungen gewährleistet ist (bspw. [NS00, NSS02, CDG$^+$09]). Zudem ist in den Systemen [NS00, NSS02, CDG$^+$09] eine Double-Spending-Detection nur mit Hilfe einer Trusted-Third-Party möglich.

Nach bestem Wissen wurde in der Literatur noch kein kompaktes oder teilbares elektronisches Geldsystem vorgestellt, das die Möglichkeit zur Aufhebung der Anonymität in Verdachtsfällen durch eine Trusted-Third-Party oder eine Double-Spending-Prevention durch den Einsatz von Observern garantiert, so dass die in Abschnitt 4.4 definierten Sicherheitsziele erfüllt sind.

Das Geldsystem [CDG$^+$09] wird von den Autoren zwar als fair betitelt, jedoch wird die Trusted-Third-Party ausschließlich für eine Double-Spending-Detection verwendet. Daher ist anzunehmen, dass die Autoren die Bezeichnung *fair* lediglich verwenden, um anzudeuten, dass eine Trusted-Third-Party involviert ist.

In [Au09, Abschnitt 5.6] wird eine Methode vorgeschlagen, wie jedes kompakte oder teilbare Geldsystem zu einem fairen Geldsystem erweitert werden kann. Die Münzverfolgung ist aber eher ineffizient und gewährleistet nicht das Sicherheitsziel Münzverfolgung-Beschuldigung gemäß Definition 4.4.8 (siehe Abschnitt 7.4 und Unterabschnitt 7.6.1).

Einige kompakte elektronische Geldsysteme (z.B. [CHL05]) ermöglichen zwar eine

Münzverfolgung, diese ist jedoch *nur* nach einem Double-Spending möglich, weshalb es sich nicht um faire elektronische Geldsysteme handelt. Das kompakte elektronische Geldsystem [CHL06a] ist eine Erweiterung von [CHL05] und bekämpft Geldwäsche, indem jeder Kunde nur eine bestimmte Anzahl von Münzen an einen bestimmten Händler zahlen darf. Die Identität eines Kunden kann, wie nach einem Double-Spending, aufgedeckt werden, wenn er zu viele Münzen bei einen Händler ausgibt. Im Anschluss kann die Bank alle Münzen des Kunden verfolgen, wobei es sich auch bei [CHL06a] nicht um ein faires Geldsystem handelt.

Neben den beiden elektronischen Geldsystemen [CHL05] und [CHL06a], gibt es noch weitere, die versuchen, kriminelle Aktivitäten ohne den Einsatz einer Trusted-Third-Party zu bekämpfen (z.B. [STS99a, STS99b, STS99c, KV02]).

4.5.1 Analyse der Sicherheitsdefinitionen

Die im obigen Abschnitt 4.4 definierten Sicherheitsziele Unfälschbarkeit, Anonymität und Double-Spending-Beschuldigung sind stärker als bei allen bisherigen elektronischen Geldsystemen in der Literatur. Dies liegt bereits daran, dass sich der Angreifer $\mathcal{A}$ auch beliebige Protokolle zweier ehrlicher Parteien ansehen kann. Es sei jedoch bemerkt, dass der Angreifer $\mathcal{A}$ dadurch keinen entscheidenden Vorteil besitzt, ein Spiel zu gewinnen.

Im Folgenden werden die Definitionen Unfälschbarkeit, Anonymität, Double-Spending-Beschuldigung und Abhebe-Beschuldigung mit denen der ab [CHL05] (die ausführlichen Sicherheitsdefinitionen und Beweise befinden sich in der vollständigen Version [CHL06b]) veröffentlichten elektronischen Geldsysteme [CHL06b, Tro06, ASM07, AWSM07, CG07, ASM08, Au09, CDG$^+$09, CG10] und [IL13] verglichen. (Die Geldsysteme in [CHL06a, Hoh06] und [BCKL09] verwenden dieselben Definitionen wie [CHL06b].) Dabei werden die wichtigsten Unterschiede im Vergleich zu den Definitionen in Abschnitt 4.4 aufgelistet, wobei zur Vereinfachung dieselben Orakelbezeichnungen verwendet werden.

Es ist allerdings zu beachten, dass lediglich die Unterschiede der Definitionen im Vergleich zu Abschnitt 4.4 genannt werden. Es ist durchaus möglich, dass die folgenden elektronischen Geldsysteme dennoch ebenfalls die in Abschnitt 4.4 definierten Sicherheitsziele erfüllen.

Unfälschbarkeit: Die Sicherheitseigenschaft Unfälschbarkeit stimmt im Wesentlichen bei den meisten elektronischen Geldsystemen in der Literatur überein. Bei den Definitionen in [CHL06b, Tro06, ASM07, AWSM07, CG07, ASM08, CDG$^+$09, CG10] und [IL13] kann der Angreifer $\mathcal{A}$ entweder nur die Orakel $\mathcal{O}^{\mathsf{W}}_{\mathcal{B}}$ und $\mathcal{O}^{\mathsf{S}}_{\mathcal{H}}$ oder $\mathcal{O}^{\mathsf{W}}_{\mathcal{B}}$ und $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$ befragen, um entweder Abhebe- und Bezahlprotokolle oder Abhebe- und Einlöseprotokolle durchzuführen.

Das Spiel 4.4.1 orientiert sich an [Au09]. Denn hier kann der Angreifer $\mathcal{A}$ die Orakel $\mathcal{O}^{\mathsf{A}}_{\mathcal{B}}, \mathcal{O}^{\mathsf{A}}_{\mathcal{K},\mathcal{B}}, \mathcal{O}^{\mathsf{W}}_{\mathcal{B}}, \mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}, \mathcal{O}^{\mathsf{S}}_{\mathcal{H}}, \mathcal{O}^{\mathsf{S}}_{\mathcal{K}}, \mathcal{O}^{\mathsf{S}}_{\mathcal{K},\mathcal{H}}$ und $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$ befragen, um Kunden hinzuzufügen sowie Abhebe- Bezahl- und Einlöseprotokolle durchzuführen.

Anonymität: Im Gegensatz zur Unfälschbarkeit, besitzt nahezu jedes elektronisches Geldsystem in der Literatur eine eigene Definition der Anonymität.

In [CHL06b] und [IL13] wird die Anonymität *simulationsbasierend* definiert. Dazu darf der Angreifer $\mathcal{A}$ die Orakel $\mathcal{O}^{\mathsf{A}}_{\mathcal{K}}, \mathcal{O}^{\mathsf{W}}_{\mathcal{K}}$ und $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$ befragen und muss entscheiden, ob die $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$-Anfragen vom Orakel (und somit vom ehrlichen Kunden) gemäß Protokoll oder von einem Simulator durchgeführt werden. Dadurch ist die Anonymität nur dann garantiert, falls niemals ein Kunde eine Münze mehrfach ausgibt. Wenn der Angreifer $\mathcal{A}$, wie in Spiel 4.4.2 das Orakel $\mathcal{O}^{\mathsf{S}^\star}_{\mathcal{K}}$ befragen dürfte und somit Kunden zu einem Double-Spending zwingen könnte, wäre die Anonymität nicht mehr mit den entsprechenden Beweisen aus [CHL06b, IL13] beweisbar. Denn der Simulator müsste dazu, ohne Kenntnis der öffentlichen Schlüssel der Kunden, die Sicherheitstags so berechnen, dass durch den $\mathsf{Identify}$-Algorithmus der öffentliche Schlüssel des gewählten Kunden ausgegeben wird.

Bei den meisten elektronischen Geldsystemen (z.B. [Tro06, ASM07, AWSM07, CG07, ASM08, CDG$^+$09, CG10]) wird die Anonymität *spielbasierend* definiert und unterteilt sich ebenfalls wie das Spiel 4.4.2 in eine Probe-Phase und eine Challenge-Phase.

Gemeinsam haben die oben genannten Geldsysteme, dass der Challenger für ein zufälliges Bit $b \in_{\mathcal{R}} \{0,1\}$ in der Challenge-Phase lediglich ein Bezahlprotokoll bzgl. des Index i_b bzw. des Kunden pk_b mit dem Angreifer $\mathcal{A}$ durchführt. Ebenso dürfen bei diesen Definitionen die Kunden nicht zu einem Double-Spending gezwungen werden.

In [Tro06] kann der Angreifer $\mathcal{A}$ in der Probe-Phase die Orakel $\mathcal{O}^{\mathsf{A}}_{\mathcal{K}}, \mathcal{O}^{\mathsf{W}}_{\mathcal{K}}$ und $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$ befragen. Das besondere an der Definition in [Tro06] ist, dass der Angreifer $\mathcal{A}$ in der Challenge-Phase zusätzlich alle privaten Schlüssel der Kunden erhält.

Bei den Definitionen in [ASM07, AWSM07] und [ASM08] kann der Angreifer $\mathcal{A}$ nur mit zwei Kunden pk_0 und pk_1 Abhebe- und Bezahlprotokolle durchführen. Somit sind diese beiden Kunden auch für die Challenge festgelegt. Während der Angreifer $\mathcal{A}$ in [ASM07] für die Challenge die beiden Indizes i_0 und i_1 bzgl. pk_0 und pk_1 wählen darf, werden diese in [AWSM07] und [ASM08] ebenfalls vom Challenger bestimmt.

In [CG07] und [CG10] kann der Angreifer $\mathcal{A}$ in der Probe-Phase die Orakel $\mathcal{O}^{\mathsf{A}}, \mathcal{O}^{\mathsf{A}}_{\mathcal{K}}, \mathcal{O}^{\mathsf{W}}_{\mathcal{K}}$ und $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$ befragen. In der Challenge-Phase wählt der Angreifer zwei öffentliche Schlüssel $\mathsf{pk}_0, \mathsf{pk}_1$ und führt mit dem Kunden bzgl. pk_b ein Bezahlprotokoll durch.

In [CDG$^+$09] läuft das Spiel im Wesentlichen wie in [CG07, CG10] ab und der Angreifer $\mathcal{A}$ kann zusätzlich in einer Post-Challenge-Phase die Orakel $\mathcal{O}^{\mathsf{W}}_{\mathcal{K}}$ und $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$ befragen. Da allerdings für die Challenge verlangt wird, dass beide Kunden $\mathsf{pk}_0, \mathsf{pk}_1$ die *gleiche* Anzahl an Münzen ausgegeben haben und der Kunde $\mathsf{pk}_{\bar{b}}$ ebenfalls ein Bezahlprotokoll ausführen muss, das *nicht* vom Angreifer $\mathcal{A}$ gesehen werden darf, handelt es sich um einen sehr schwachen Anonymitätsbegriff. Insbesondere gewährleistet das Geldsystem [CDG$^+$09] nicht die oben definierte Anonymität gemäß Spiel 4.4.2, da jede Münze Informationen über die Anzahl der bisher ausgegebenen Münzen der Geldbörse enthält.

In [Au09] wird das Schlüsselpaar der Bank nicht vom Angreifer $\mathcal{A}$ sondern vom Challenger $\mathcal{C}$ generiert und an $\mathcal{A}$ gesendet. In der Probe-Phase darf der Angreifer $\mathcal{A}$ die Orakel $\mathcal{O}^{\mathsf{A}}, \mathcal{O}^{\mathsf{A}}_{\mathcal{K}}, \mathcal{O}^{\mathsf{W}}_{\mathcal{K}}, \mathcal{O}^{\mathsf{S}}_{\mathcal{H}}, \mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$ und $\mathcal{O}^{\mathsf{S}}_{\mathcal{K},\mathcal{H}}$ befragen. In der Challenge-Phase wählt der Angreifer zwei Indizes i_0 und i_1, wobei diese zum selben Kunden gehören dürfen.

Es ist leicht zu sehen, dass die durch das Spiel 4.4.2 definierte Anonymität stärker als alle obigen spielbasierenden Definitionen außer der von [Tro06] ist. Allerdings erfüllen die in Kapitel 6 bis 9 entwickelten Geldsysteme ebenfalls die in [Tro06] definierte Anonymität.

Double-Spending-Beschuldigung: In [CHL06b] werden die beiden Varianten *weak Exculpability* und *strong Exculpability* definiert. Die Eigenschaft weak Exculpability garantiert, dass ein ehrlicher Kunde nicht beschuldigt werden kann, wenn er niemals ein Double-Spending begeht. Diese Definition wird auch in [ASM07, AWSM07, CG07, ASM08, CDG+09, CG10] und [IL13] verwendet.

In [Tro06] wird ebenfalls die in [CHL06b] definierte strong Exculpability gefordert, die in [Tro06] allerdings mit *Non-Frameability* bezeichnet wird. Der Grundgedanke beider Definitionen ist, dass ein Kunde, der mehrere Double-Spendings begeht, dennoch *nur* für diese beschuldigt werden kann. Genau dieses Ziel wird durch die Definition 4.4.3 garantiert.

Allerdings wird dies mit der Definition in [CHL06b] nicht gewährleistet. So ist durch diese Definition nicht ausgeschlossen, dass ein ehrlicher Kunde eine Seriennummer zweimal ausgibt, aber der Angreifer $\mathcal{A}$ den Kunden beschuldigen kann, diese Seriennummer mehr als zweimal verwendet zu haben. Somit könnte die betrügerische Bank einen Double-Spender für beliebig viele Double-Spendings verantwortlich machen.

Auch in [Tro06] wird nicht das für teilbare elektronische Geldsysteme gewünschte Ziel definiert. Dort wird durch die Non-Frameability ausgeschlossen, dass ein Angreifer $\mathcal{A}$ einen ehrlichen Kunden für mehr Double-Spendings beschuldigen kann, als dieser tatsächlich begangen hat. Somit wird zwar das Problem der Definition in [CHL06b] abgedeckt, aber für teilbare Geldsysteme ist dies immer noch nicht ausreichend. Denn es wird nicht ausgeschlossen, dass ein ehrlicher Kunde ein Double-Spending bzgl. einer Münze im Wert von 1 Euro begeht, aber die betrügerische Bank den Kunden beschuldigen kann, eine Münze im Wert von 2 Euro mehrfach ausgegeben zu haben.

Abhebe-Beschuldigung: Diese Sicherheitseigenschaft wird nur in [Tro06] gefordert und dort mit *Exculpability* bezeichnet. Es ist allerdings davon auszugehen, dass dieses Ziel von jedem elektronischen Geldsystem erfüllt wird, wenn

1. der Kunde im Abhebeprotokoll durch einen Proof-of-Knowledge (bzw. eine Signature-of-Knowledge) die Kenntnis seines privaten Schlüssels $\mathsf{sk}_{\mathcal{K}}$ beweist, wobei $\mathsf{pk}_{\mathcal{K}} = g^{\mathsf{sk}_{\mathcal{K}}}$ sein öffentlicher Schlüssel für einen Generator g einer Gruppe $\mathbb{G}$ ist und

2. die Bank durch die Double-Spending-Detection nicht den privaten Schlüssel $\mathsf{sk}_{\mathcal{K}}$ des Double-Spenders erhält (wie bspw. bei [AWSM07]).

Denn um das Spiel 4.4.4 zu gewinnen, muss der Angreifer $\mathcal{A}$ den Proof-of-Knowledge (bzw. die Signature-of-Knowledge) fälschen, was ein Widerspruch zur Diskreter-Logarithmus-Annahme ist.

Wie bereits in Abschnitt 4.4 erwähnt, wird in [CG07] behauptet, dass das Sicherheitsziel Abhebe-Beschuldigung durch die in [CHL05] bzw. [CHL06b] definierte Sicherheitseigenschaft weak Exculpability impliziert wird. Dass diese Behauptung falsch ist, zeigt das elektronische Geldsystem in [ASM08]. Denn dieses erfüllt die Eigenschaft weak Exculpability, aber da der Kunde während des Abhebeprotokolls keinen Beweis bzgl. seines geheimen Schlüssels $\mathsf{sk}_{\mathcal{K}}$ erstellen muss, kann die betrügerische Bank einfach beliebig viele Protokollansichten mit dem öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{K}}$ erzeugen.

KAPITEL 5

KOMPAKTE ELEKTRONISCHE GELDSYSTEME

In diesem Kapitel werden zwei kompakte elektronische Geldsysteme entwickelt, welche sich sehr stark an dem kompakten Geldsystem in [ASM07, Abschnitt 4.2] (siehe auch [Au09, Abschnitt 5.3]) orientieren. In [ASM07] wird das ursprüngliche kompakte Geldsystem von [CHL05] effizienter gestaltet, dessen elementare Bausteine das CL-Signaturverfahren [CL02b] und die *Pseudozufallsfunktion* von Y. Dodis und A. Yampolskiy [DY05] sind. In [CHL05] kann jeder Kunde eine Geldbörse mit dem Wert K abheben und im Bezahlprotokoll mit einer Münze bezahlen, indem ein *Pseudozufallswert* $S = \mathfrak{g}_1^{\frac{1}{s+J+1}}$ berechnet wird, wobei $\mathfrak{g}_1$ ein Generator einer Gruppe $\mathbb{G}_p$ der Primzahlordnung p, die Zahl $s \in_{\mathcal{R}} \mathbb{Z}_p^*$ ein Geheimnis des Kunden und $J \in [0, K-1]$ (bzw. $J \in [1, K]$) ein Münzzähler ist. Dieses Element S kann unter der q-DDHI-Annahme (siehe [DY05, CHL06b, ASM07]) nicht effizient von einem zufälligen Element der Gruppe $\mathbb{G}_p$ unterschieden werden. Der Kunde muss nun im Bezahlprotokoll von [CHL05] in zero-knowledge beweisen, dass sich die Zahl J in dem Intervall $[0, K-1]$ (bzw. $[1, K]$) befindet (vgl. [Bou00, CHL05]). Dieser eher ineffiziente Zero-Knowledge-Beweis wird in [ASM07] nicht benötigt, da die Bank alle Zahlen $1, \ldots, K$ signiert, diese Signaturen $\sigma_1, \ldots, \sigma_K$ dem öffentlichen Schlüssel hinzufügt und der Kunde im Bezahlprotokoll nur beweisen muss, dass er σ_J kennt. Damit wird indirekt bewiesen, dass $1 \leq J \leq K$ ist. Die Komplexität des Bezahlprotokolls wird dadurch von $O(\lambda + \log(K))$ auf $O(\lambda)$ verbessert, während die Komplexität des öffentlichen Schlüssels allerdings von $O(\lambda)$ auf $O(\lambda \cdot K)$ erhöht wird, wobei λ der Sicherheitsparameter ist.

Des Weiteren werden in [ASM07] erstmals zwei weitere Bezahlprotokolle *Batch Spend* und *Compact Spend* vorgestellt, um mit mehreren bzw. mit allen Münzen zu bezahlen.

Die in diesem Kapitel beschriebenen kompakten elektronischen Geldsysteme verwenden anstelle der K Signaturen $\sigma_1, \ldots, \sigma_K$ den Nguyen-Akkumulator, um die Werte $1, \ldots, K$ zu akkumulieren. Aus Effizienzgründen werden alle K Zeugen $\hat{W}_1, \ldots, \hat{W}_K$ ebenfalls dem öffentlichen Schlüssel hinzugefügt. Im Bezahlprotokoll beweist der Kunde nun nicht, dass er eine Signatur auf die Zahl J kennt, sondern dass sich J im öffentlichen Akkumulator V befindet.

Im Geldsystem von [ASM07] wurde die BBS+-Signatur verwendet, um die Zahlen $1, \ldots, K$ zu signieren. Aus diesem Grund werden zusätzlich ein öffentlicher Schlüssel $X' \in \mathbb{G}_2$ sowie die Signaturen $\sigma_i = (\Sigma_i', e_i') \in \mathbb{G}_1 \times \mathbb{Z}_p$ mit $\hat{e}\left(\Sigma_i', X' h^{e_i'}\right) = \hat{e}\left(g g_0^i, h\right)$ für $1 \leq i \leq K$ dem öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}}$ hinzugefügt. Dadurch besteht der Schlüssel $\mathsf{pk}_{\mathcal{B}}$ aus $\mathbb{G}_1^K \times \mathbb{G}_2 \times \mathbb{Z}_p^K$ weiteren Elementen. In [Au09] wurde jedoch bemerkt, dass die *schwach-sichere* BB-Signatur von D. Boneh und X. Boyen [BB04, Abschnitt 3.1] für die gewünschte Sicherheit ausreichend ist. Dadurch besteht jede Signatur σ_i nur noch aus dem Element $\Sigma_i' \in \mathbb{G}_1$ mit $\hat{e}\left(\Sigma_i', X' h^i\right) = \hat{e}\left(g, h\right)$ für $1 \leq i \leq K$. Somit werden nur $\mathbb{G}_1^K \times \mathbb{G}_2$ weitere Elemente dem öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}}$ zugefügt.

Im folgenden kompakten Geldsystem KG-I (Abschnitt 5.1) wird anstelle der schwach-sicheren BB-Signatur der Nguyen-Akkumulator verwendet. Somit werden zusätzlich der Akkumulator $V \in \mathbb{G}_1$, ein Generator $h_1 \in \mathbb{G}_2$ und K Zeugen $\hat{W}_1, \ldots, \hat{W}_K \in \mathbb{G}_1$ mit $\hat{e}\left(\hat{W}_i, h_1 h^i\right) = \hat{e}\left(V, h\right)$ für $1 \leq i \leq K$ dem öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}}$ hinzugefügt. Dies ist im Vergleich zum oben erwähnten Geldsystem von [Au09] nur ein Gruppenelement in $\mathbb{G}_1$ mehr. Die Signatures-of-Knowledge bzgl. $\hat{e}\left(\Sigma_i', X' h^i\right) = \hat{e}\left(g, h\right)$ in [Au09] bzw. $\hat{e}\left(\hat{W}_i, h_1 h^i\right) = \hat{e}\left(V, h\right)$ im folgenden Geldsystem sind dabei im Prinzip identisch, da es sich jeweils um eine Signature-of-Knowledge über ein SDH-Paar handelt (vgl. Unterunterabschnitt 2.10.1.2).

Die Verwendung des Akkumulator-Schemas anstelle der schwach-sicheren BB-Signatur ist somit hinsichtlich der Komplexität identisch. Allerdings bietet dies in Verbindung mit dem in Kapitel 3 vorgestellten BBS+A-Signaturverfahren die Möglichkeit, dass jeder Kunde im Abhebeprotokoll eine beliebige Anzahl an Münzen abheben kann, während das Bezahlprotokoll im Gegensatz zu [ASM07] und [Au09] eine konstante Komplexität besitzt. Dazu wird das kompakte Geldsystem KG-I in Abschnitt 5.2 zu dem kompakten Geldsystem KG-II erweitert. Dabei wird besonders der Vorteil der BBS+A-Signatur im Vergleich zur ESS+-Signatur deutlich, da die Bank den Akkumulator im Abhebeprotokoll selbst berechnen kann und der Kunde somit nichts über die Berechnung des Akkumulators in zero-knowledge beweisen muss. Dadurch ist die Verwendung des BBS+A-Signaturverfahrens in allen Aspekten effizienter (vgl. Tabelle 3.1). Insgesamt kann das Abhebeprotokoll effizienter ausgeführt werden, als bei den teilbaren Geldsystemen in Kapitel 6.

Da die beiden kompakten elektronischen Geldsysteme KG-I und KG-II eine Modifizierung des kompakten Geldsystems aus [ASM07, Abschnitt 4.2] darstellen, werden lediglich die in [ASM07] definierten Sicherheitseigenschaften unter der Annahme bewiesen, dass das kompakte Geldsystem aus [ASM07, Abschnitt 4.2] diese Sicherheitseigenschaften

erfüllt. Aus diesem Grund orientieren sich die Sicherheitsbeweise ebenfalls an [ASM07].

5.1 Das kompakte elektronische Geldsystem KG-I

Im Folgenden werden die Algorithmen und Protokolle des kompakten elektronischen Geldsystems KG-I detailliert beschrieben, wobei nur das Bezahlprotokoll Spend vorgestellt wird. Die beiden in [ASM07] beschriebenen Bezahlprotokolle $\mathsf{Compact-Spend}$ und $\mathsf{Batch-Spend}$ können analog durchgeführt werden.

Wie in [ASM07, Au09] wird das System durch eine Trusted-Third-Party initialisiert, wobei diese im Gegensatz zu [ASM07, Au09] auch die Generatoren der Gruppen wählt. Dies ist allerdings damit vergleichbar, dass die Generatoren durch eine kryptografische Hashfunktion mit öffentlicher Eingabe berechnet werden. Anstelle eines Kontoeröffnungsprotokolls $\mathsf{Account}$ wird in [ASM07, Au09] nur ein Schlüsselgenerierungsalgorithmus KGen von jedem Kunden ausgeführt, um dem System beizutreten.

Setup führt bei Eingabe des Sicherheitsparameters 1^λ und der Zahl $K \in \mathbb{N}$ den Algorithmus $\mathsf{Setup_{Bil}}$ aus und erhält die Systemparameter $\mathsf{sp_{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, h) \leftarrow \mathsf{Setup_{Bil}}\left(1^\lambda\right)$, wobei $\mathbb{G}_1 = \langle u_0 \rangle, \mathbb{G}_2 = \langle h \rangle$ und $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ eine bilineare Abbildung ist. Dann wählt Setup eine kryptografische Hashfunktion H mit $\mathsf{H}\colon \{0,1\}^* \to \mathbb{Z}_p^*$, führt den Algorithmus $\mathsf{Setup_G}$ aus und erhält die Systemparameter $\mathsf{sp_G} = \left(p, \mathbb{G}_p', \mathfrak{g}_0\right) \leftarrow \mathsf{Setup_G}\left(1^\lambda, p\right)$, wobei $\mathbb{G}_p = \langle \mathfrak{g}_0 \rangle$ eine weitere zyklische Gruppe der Ordnung p ist. Anschließend wählt Setup zufällig fünf Generatoren $g, g_0, g_1, g_2, g_3 \in_\mathcal{R} \mathbb{G}_1$, einen Generator $\mathfrak{g}_1 \in_\mathcal{R} \mathbb{G}_p$ sowie eine Zahl $\alpha \in_\mathcal{R} \mathbb{Z}_p^*$ und berechnet das Element $h_1 = h^\alpha$, den Akkumulator $V = u_0^{\prod_{i=1}^{K}(\alpha+i)}$ sowie die Zeugen $\hat{W}_1, \ldots, \hat{W}_K$ mit $\hat{W}_i^{\alpha+i} = V$ für $1 \leq i \leq K$. Da die Zahl α nicht weiter benötigt wird, wird α wieder gelöscht, um zu verhindern, dass sie kompromittiert wird. Der Generator u_0 wurde nur zur Berechnung des Akkumulators sowie der Zeugen benötigt und muss nicht veröffentlicht werden. Die Ausgabe sind die Systemparameter

$$\mathsf{sp} = \left(p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \mathbb{G}_p', \hat{e}, g, g_0, g_1, g_2, g_3, V, \hat{W}_1, \ldots, \hat{W}_K, h, h_1, \mathfrak{g}_0, \mathfrak{g}_1, \mathsf{H}\right) \leftarrow \mathsf{Setup}\left(1^\lambda, K\right).$$

BGen wählt bei Eingabe der Systemparameter sp zufällig eine Zahl $x \in_\mathcal{R} \mathbb{Z}_p^*$ und berechnet das Element $X = h^x$. Die Ausgabe ist das Schlüsselpaar

$$(\mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{B}) = (X, x) \leftarrow \mathsf{BGen}\,(\mathsf{sp})\,.$$

Weiter legt $\mathcal{B}$ zwei Datenbanken $\mathbb{D}_W$ und $\mathbb{D}_C$ an, in denen die Abhebeinformationen bzw. die eingelösten Münzen gespeichert werden.

In der folgenden Tabelle 5.1 werden die benötigten Gruppen, deren öffentliche Generatoren, deren Verwendung und die zugrunde liegenden, kryptografischen Annahmen zusammengefasst.

Gruppen	Generatoren	Verwendung	Annahmen
$\mathbb{G}_1$	g, g_0, g_1, g_2, g_3 $V, \hat{W}_1, \ldots, \hat{W}_K$		
$\mathbb{G}_2$	h, h_1, X		
$(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$		Nguyen-Akkumulator und BBS+-Signatur	q-SDH
$\mathbb{G}_p$	$\mathfrak{g}_0, \mathfrak{g}_1$	Seriennummern und Sicherheitstags	q-DDHI

Tabelle 5.1: Öffentliche Parameter von KG-I

KGen wählt bei Eingabe der Systemparameter sp zufällig eine Zahl $u \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Kontonummer $I = \mathfrak{g}_0^u$. Die Ausgabe ist das Schlüsselpaar

$$(\mathsf{pk}_{\mathcal{K}}, \mathsf{sk}_{\mathcal{K}}) = (I, u) \leftarrow \mathsf{KGen}\,(\mathsf{sp})\,.$$

Withdraw: Damit ein Kunde $\mathcal{K}$ eine elektronische Geldbörse mit K Münzen bei der Bank $\mathcal{B}$ abheben kann, führen $\mathcal{K}$ und $\mathcal{B}$ das Abhebeprotokoll Withdraw durch, welches in Abbildung 5.1 dargestellt ist und im Wesentlichen dem Signaturerstellungsprotokoll Issue des BBS+-Signaturverfahrens und somit dem Abhebeprotokoll von [ASM07, Au09] entspricht. Zuvor authentifizieren sich jedoch beide Parteien gegenseitig.

Schritt 1: Der Kunde $\mathcal{K}$ wählt zufällig drei Zahlen $s', t, r \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Pedersen-Commitment $C = g_0^{s'} g_1^u g_2^t g_3^r$, wobei durch die Zufallszahl r die Zahlen u, s', t perfekt verborgen sind. Dann sendet $\mathcal{K}$ das Commitment C an die Bank $\mathcal{B}$ und erstellt die folgende Signature-of-Knowledge:

$$SoK_{\mathsf{W}}\left[(s', t, r, u) : C = g_0^{s'} g_1^u g_2^t g_3^r \ \wedge\ I = \mathfrak{g}_0^u\right](K)\,.$$

Durch SoK_{W} beweist $\mathcal{K}$ in zero-knowledge, dass er eine Darstellung von C bzgl. des Generatortupels (g_0, g_1, g_2, g_3) kennt und der Exponent von g_1 der private Schlüssel u des Kunden $\mathcal{K}$ ist.

Schritt 2: Die Bank $\mathcal{B}$ verifiziert SoK_{W}, wählt zufällig zwei Zahlen $e, s'' \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Element $\Sigma = \left(g g_0^{s''} C\right)^{\frac{1}{x+e}}$. Dann sendet $\mathcal{B}$ das Tripel (Σ, e, s'') an $\mathcal{K}$ und bucht dem Kunden $\mathcal{K}$ den Betrag K vom Konto mit der Kontonummer I ab. Anschließend speichert $\mathcal{B}$ die Abhebeinformationen $\mathbb{T} = (I)$ in ihrer Datenbank ab.

Schritt 3: Der Kunde $\mathcal{K}$ berechnet $s = s' + s'' \mod p$, verifiziert die Signatur durch Überprüfung der Gleichung $\hat{e}(\Sigma, Xh^e) \stackrel{?}{=} \hat{e}\left(gg_0^{s''}C, h\right)$ und speichert die Geldbörse $\mathbb{W} = (\Sigma, e, s, t, r, J)$ ab, wobei der Münzzähler $J = 1$ gesetzt wird.

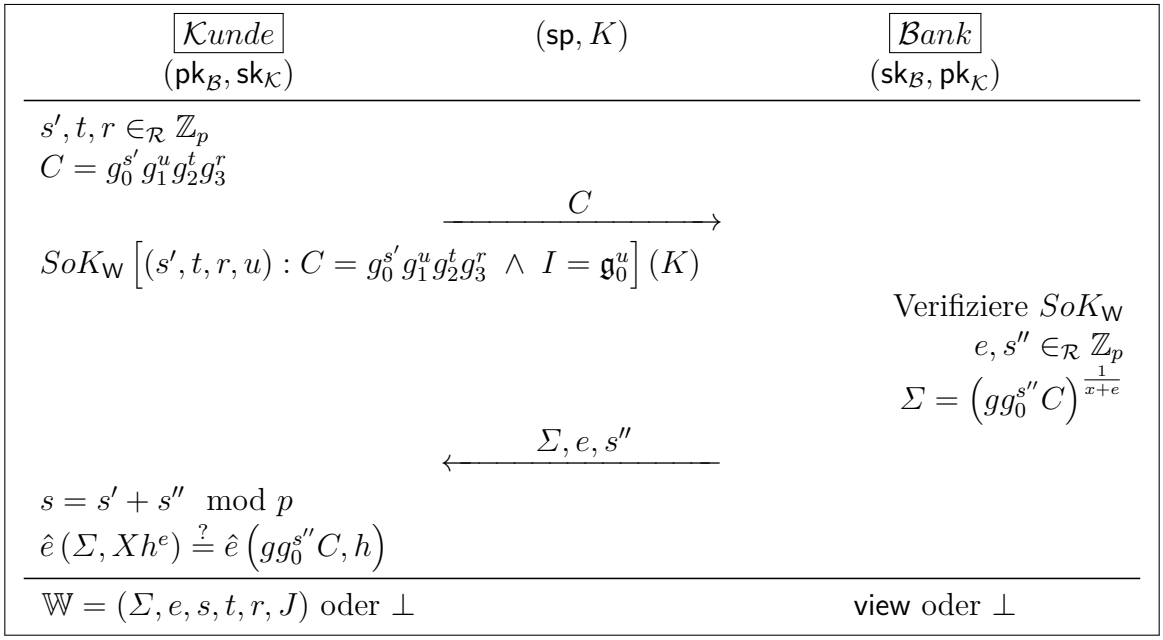

Abbildung 5.1: Abhebeprotokoll Withdraw von KG-I

Spend: Damit ein Kunde $\mathcal{K}$, der im Besitz der Geldbörse $\mathbb{W} = (\Sigma, e, s, t, r, J)$ mit $J \leq K$ ist, bei einem Händler $\mathcal{H}$ mit einer elektronischen Münze coin bezahlen kann, führen $\mathcal{K}$ und $\mathcal{H}$ das Bezahlprotokoll Spend durch, welches in Abbildung 5.2 dargestellt ist. Zuvor authentifiziert sich der Händler $\mathcal{H}$ gegenüber dem Kunden $\mathcal{K}$ und beide Parteien gleichen den aktuellen Zeitpunkt ts ab.

Schritt 1: Der Händler $\mathcal{H}$ und der Kunde $\mathcal{K}$ berechnen beide den Hashwert $R = \mathsf{H}(\mathsf{pk}_\mathcal{H} || \mathsf{ts})$, wobei $\mathsf{pk}_\mathcal{H}$ der öffentliche Schlüssel des Händlers ist.

Schritt 2: Der Kunde $\mathcal{K}$ wählt zufällig fünf Zahlen $a_1, a_2, b_1, b_2, c_1 \in_\mathcal{R} \mathbb{Z}_p$ und berechnet die Seriennummer $S = \mathfrak{g}_1^{\frac{1}{s+J+1}}$, das Sicherheitstag $T = \mathfrak{g}_0^u \mathfrak{g}_1^{\frac{R}{t+J+1}}$ sowie die Elemente $A_1 = g_1^{a_1} g_2^{a_2}, A_2 = \hat{W}_J g_2^{a_1}, B_1 = g_1^{b_1} g_2^{b_2}, B_2 = \Sigma g_2^{b_1}$ und $C_1 = g_1^{J+t} g_2^{c_1}$, um die Elemente $\hat{W}_J$ und Σ perfekt zu verbergen. Anschließend sendet $\mathcal{K}$ die Elemente $S, T, A_1, A_2, B_1, B_2, C_1$ an $\mathcal{H}$ und erstellt die Signature-of-Knowledge SoK, wobei $\delta_1 = a_1 J, \delta_2 = a_2 J, \beta_1 = b_1 e, \beta_2 = b_2 e, \gamma_1 = c_1 u$ und $\gamma_2 = (J + t)\, u$ ist:

$$SoK\Big[(a_1, a_2, \delta_1, \delta_2, b_1, b_2, \beta_1, \beta_2, c_1, \gamma_1, \gamma_2, e, s, t, u, r, J) : A_1 = g_1^{a_1} g_2^{a_2} \ \wedge$$

$$1 = A_1^J g_1^{-\delta_1} g_2^{-\delta_2} \ \wedge \ \hat{e}(V,h)\,\hat{e}(A_2,h_1)^{-1} = \hat{e}\left(g_2^{-\delta_1} A_2^J, h\right)\hat{e}\left(g_2^{-a_1}, h_1\right) \ \wedge$$
$$B_1 = g_1^{b_1} g_2^{b_2} \ \wedge \ 1 = B_1^e g_1^{-\beta_1} g_2^{-\beta_2} \ \wedge \ C_1 = g_1^{J+t} g_2^{c_1} \ \wedge \ 1 = C_1^u g_1^{-\gamma_2} g_2^{-\gamma_1} \ \wedge$$
$$\hat{e}(B_2,X)\,\hat{e}(g,h)^{-1} = \hat{e}\left(g_2^{b_1}, X\right)\hat{e}\left(g_0^s g_1^u g_2^{t+\beta_1} g_3^r B_2^{-e}, h\right) \ \wedge$$
$$\mathfrak{g}_1 S^{-1} = S^{J+s} \ \wedge \ \mathfrak{g}_1^R T^{-1} = T^{J+t}\mathfrak{g}_0^{-\gamma_2-u}\Big](R)\,.$$

Durch SoK beweist $\mathcal{K}$ in zero-knowledge, dass sich der Münzzähler J im Akkumulator V befindet (und damit indirekt, dass $1 \leq J \leq K$ ist), er im Besitz einer BBS+-Signatur $\sigma = (\Sigma, e, s)$ auf die Nachricht $m = (u, t, r)$ ist und die Seriennummer S sowie das Sicherheitstag T korrekt berechnet wurden. Schließlich aktualisiert $\mathcal{K}$ seine Geldbörse zu $\mathbb{W}' = (\Sigma, e, s, t, r, J+1)$.

Schritt 3: Der Händler $\mathcal{H}$ verifiziert SoK und speichert die Münze $\mathsf{coin} = (S, T, R, \Phi = (A_1, A_2, B_1, B_2, C_1, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$ ab.

$\boxed{\mathcal{K}unde}$ $(\mathsf{pk}_{\mathcal{H}}, \mathsf{sk}_{\mathcal{K}}, \mathbb{W})$	$(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{ts})$	$\boxed{\mathcal{H}ändler}$ $(\mathsf{sk}_{\mathcal{H}})$
$R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}} \| \mathsf{ts})$		$R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}} \| \mathsf{ts})$
$a_1, a_2, b_1, b_2, c_1 \in_{\mathcal{R}} \mathbb{Z}_p$		
$S = \mathfrak{g}_1^{\frac{1}{s+J+1}}$		
$T = \mathfrak{g}_0^u \mathfrak{g}_1^{\frac{R}{t+J+1}}$		
$A_1 = g_1^{a_1} g_2^{a_2}$		
$A_2 = \hat{W}_J g_2^{a_1}$		
$B_1 = g_1^{b_1} g_2^{b_2}$		
$B_2 = \Sigma g_2^{b_1}$		
$C_1 = g_1^{J+t} g_2^{c_1}$		
	$S, T, A_1, A_2, B_1, B_2, C_1 \longrightarrow$	
$SoK\Big[(a_1, a_2, \delta_1, \delta_2, b_1, b_2, \beta_1, \beta_2, c_1, \gamma_1, \gamma_2, e, s, t, u, r, J) : A_1 = g_1^{a_1} g_2^{a_2} \ \wedge$		
$1 = A_1^J g_1^{-\delta_1} g_2^{-\delta_2} \ \wedge \ \hat{e}(V,h)\,\hat{e}(A_2,h_1)^{-1} = \hat{e}\left(g_2^{-\delta_1} A_2^J, h\right)\hat{e}\left(g_2^{-a_1}, h_1\right) \ \wedge$		
$B_1 = g_1^{b_1} g_2^{b_2} \ \wedge \ 1 = B_1^e g_1^{-\beta_1} g_2^{-\beta_2} \ \wedge \ C_1 = g_1^{J+t} g_2^{c_1} \ \wedge \ 1 = C_1^u g_1^{-\gamma_2} g_2^{-\gamma_1} \ \wedge$		
$\hat{e}(B_2,X)\,\hat{e}(g,h)^{-1} = \hat{e}\left(g_2^{b_1}, X\right)\hat{e}\left(g_0^s g_1^u g_2^{t+\beta_1} g_3^r B_2^{-e}, h\right) \ \wedge$		
$\mathfrak{g}_1 S^{-1} = S^{J+s} \ \wedge \ \mathfrak{g}_1^R T^{-1} = T^{J+t}\mathfrak{g}_0^{-\gamma_2-u}\Big](R)$		
		Verifiziere SoK
$\mathbb{W}'$ oder $\perp$		coin oder $\perp$

Abbildung 5.2: Bezahlprotokoll Spend von KG-I

Deposit: Damit ein Händler $\mathcal{H}$ eine Münze coin bei der Bank $\mathcal{B}$ einlösen kann, führen

beide Parteien das Einlöseprotokoll Deposit durch (Abbildung 5.3), nachdem sich beide Parteien gegenseitig authentifiziert haben.

Schritt 1: Der Händler $\mathcal{H}$ sendet die Münze $\mathsf{coin} = (S, T, R, \Phi = (A_1, A_2, B_1, B_2, C_1, SoK, \mathsf{ts}, \mathsf{pk}_\mathcal{H}))$ an die Bank $\mathcal{B}$. (Die Kontonummer $\mathsf{pk}_\mathcal{H}$ muss dabei nicht erneut gesendet werden, da $\mathcal{B}$ diese bereits kennt.)

Schritt 2: Die Bank $\mathcal{B}$ überprüft $R \stackrel{?}{=} \mathsf{H}(\mathsf{pk}_\mathcal{H}||\mathsf{ts})$ um sicherzustellen, dass mit der Münze coin tatsächlich beim Händler $\mathcal{H}$ bezahlt wurde und nicht bspw. ein betrügerischer Händler die Münze abgefangen hat. Dann verifiziert $\mathcal{B}$ die Signature-of-Knowledge SoK. Falls SoK abgelehnt wird, sendet $\mathcal{B}$ eine Fehlermeldung $\perp$ an $\mathcal{H}$. Wenn SoK akzeptiert wird, sucht die Bank $\mathcal{B}$ in ihrer Datenbank $\mathbb{D}_C$, ob bereits eine Münze mit der Seriennummer S gespeichert wurde. Wenn nicht, speichert $\mathcal{B}$ die Münze coin in der Datenbank $\mathbb{D}_C$. Falls bereits eine Münze mit der Seriennummer S in der Datenbank $\mathbb{D}_C$ vorhanden ist, führt $\mathcal{B}$ den Algorithmus Identify aus, um den Double-Spender zu identifizieren.

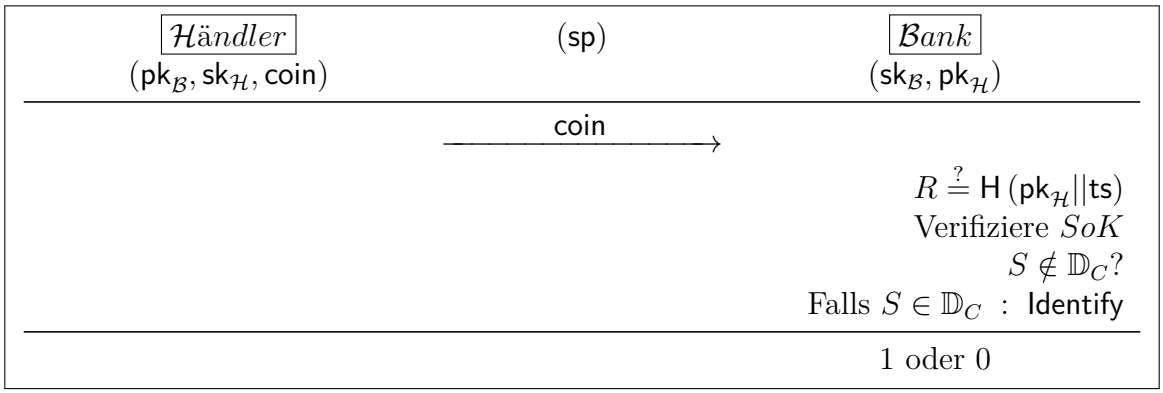

Abbildung 5.3: Einlöseprotokoll Deposit von KG-I

Identify überprüft bei Eingabe der Systemparameter sp und zweier Münzen $\mathsf{coin} = (S, T, R, \Phi)$ sowie $\mathsf{coin}' = (S, T', R', \Phi')$, ob $R \stackrel{?}{=} R'$ ist. Wenn $R = R'$ gilt, versucht der Händler eine bereits eingelöste Münze erneut einzulösen. Für $R \neq R'$ wird die Kontonummer des Double-Spenders folgendermaßen berechnet:

$$I = \left(\frac{T'^R}{T^{R'}}\right)^{(R-R')^{-1}}.$$

Die Ausgabe ist entweder die Kontonummer $I \in \mathbb{D}_W$ mit einem Beweis $\Pi_G = (\mathsf{coin}, \mathsf{coin}')$ oder das Symbol $\perp$, falls $I \notin \mathbb{D}_W$ ist.

Falls die Kontonummer I nicht in der Datenbank $\mathbb{D}_W$ existiert, haben zwei verschiedene Kunden $\mathcal{K}_0$ und $\mathcal{K}_1$ mit verschiedenen Zahlen $s_0, s_1 \in \mathbb{Z}_p$ für verschiedene Münzzähler $J_0, J_1 \in [1, K]$ dieselbe Seriennummer $S = \mathfrak{g}_1^{\frac{1}{s_0+J_0+1}} = \mathfrak{g}_1^{\frac{1}{s_1+J_1+1}}$

berechnet. Da die Zahlen s_0 und s_1 allerdings durch die Kunden und die Bank $\mathcal{B}$ gemeinsam zufällig gewählt werden, ist dieser Fall vernachlässigbar (vgl. auch [CHL05, ASM07]).

VerifyGuilt verifiziert bei Eingabe der Systemparameter sp, eines öffentlichen Schlüssels $\mathsf{pk}_{\mathcal{K}}$ und eines Beweises Π_G, der zwei Münzen $\mathsf{coin} = (S, T, R, \Phi)$ und $\mathsf{coin}' = (S, T', R', \Phi')$ enthält, die beiden Signatures-of-Knowledge SoK sowie SoK' und führt den Algorithmus $\mathsf{Identify}$ aus, um eine Kontonummer I zu berechnen. Somit kann jeder verifizieren, ob $\mathsf{pk}_{\mathcal{K}} \stackrel{?}{=} I$ ist und der Kunde $\mathcal{K}$ mit der Kontonummer I eine Seriennummer mehrfach ausgegeben hat.

5.1.1 Sicherheitsanalyse von KG-I

Die Sicherheit des kompakten elektronischen Geldsystems KG-I folgt direkt aus der Sicherheit des kompakten Geldsystems aus [ASM07, Abschnitt 4.2] und der Kollisionsfreiheit des Nguyen-Akkumulator-Schemas. (Die Beschränktheit des Nguyen-Akkumulator-Schemas wird nicht benötigt.) Denn die einzig wesentliche Veränderung zum Geldsystem [ASM07, Abschnitt 4.2] ist, dass der Kunde im Bezahlprotokoll Spend nun nicht in zero-knowledge beweist, dass er die Signatur σ_J auf die Zahl J kennt, sondern, dass sich der Münzzähler J im öffentlichen Akkumulator V befindet.

Wie bereits erwähnt, werden nur die in [ASM07] definierten Sicherheitseigenschaften *Balance*, *Anonymity* und *Exculpability* (vgl. Abschnitt 4.4 und Unterabschnitt 4.5.1) unter der Annahme bewiesen, dass das Geldsystem aus [ASM07, Abschnitt 4.2] diese Eigenschaften erfüllt. Aus diesem Grund orientiert sich der Sicherheitsbeweis sehr stark an [ASM07].

Satz 5.1.1. *Das kompakte elektronische Geldsystem KG-I erfüllt im Random-Oracle-Modell die in [ASM07] definierten Sicherheitseigenschaften, falls das kompakte elektronische Geldsystem aus [ASM07, Abschnitt 4.2] diese Sicherheitseigenschaften erfüllt und die q-SDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gilt.*

Beweis. Die in [ASM07] definierten Sicherheitseigenschaften Balance, Anonymity und Exculpability folgen direkt aus den entsprechenden Beweisen von [ASM07].

Balance: Sei $\mathcal{A}$ ein polynomieller Angreifer, der q_W Abhebe-Anfragen und q_S Bezahl-Anfragen stellt. Diese Anfragen können von einem Angreifer $\mathcal{B}$ (bzw. einem Simulator) wie in [ASM07] simuliert werden. Der Angreifer $\mathcal{A}$ kann nun wie in [ASM07] die Sicherheitseigenschaft Balance brechen, wenn er $Kq_W + q_S + 1$ Einlöseprotokolle $\mathsf{Deposit}$ durchführt, indem entweder (1) alle $Kq_W + q_S + 1$ Seriennummern verschieden oder (2) einige Seriennummern identisch sind, aber der Algorithmus $\mathsf{Identify}$ keine Kontonummer eines korrupten Kunden ausgibt.

Wie in [ASM07] beschrieben, kann der Angreifer den Fall (1) nur gewinnen, falls er einen falschen Beweis als Teil der Signature-of-Knowledge SoK erstellt, so dass mindestens einer der folgenden Beweise eine Fälschung ist:

1. Besitz einer BBS+-Signatur $\sigma = (\Sigma, e, s)$ auf die Nachricht $m = (u, t, r)$.
2. $S = \mathfrak{g}_1^{\frac{1}{s+J+1}}$.
3. Besitz eines Zeugen $\hat{W}_J$ für den Wert J und den Akkumulator V (statt: Besitz einer Signatur σ_J auf die Zahl J).

Da die ersten beiden Punkte aus [ASM07] übernommen sind, sind diese wie in [ASM07] vernachlässigbar. Daher muss nur der dritte Punkt analysiert werden:

Ein falscher Beweis für die Kenntnis eines Zeugen $\hat{W}_J$ für einen Wert $J \notin \{1, \ldots, K\}$ ist unter der q-SDH-Annahme vernachlässigbar. Denn dann kann ein polynomieller Angreifer $\mathcal{B}$ den Akkumulator V, die Wertemenge $X = \{1, \ldots, K\}$ und das Werte-Zeugen-Paar $\left(J, \hat{W}_J\right)$ mit $J \notin X$ ausgeben und somit die Kollisionsfreiheit des Nguyen-Akkumulator-Schemas brechen.

Der Fall (2) ist wie in [ASM07] ebenfalls vernachlässigbar. Daher ist analog zu [ASM07] die gesamte Erfolgswahrscheinlichkeit des Angreifers $\mathcal{A}$ vernachlässigbar.

Anonymity: Die Sicherheitseigenschaft Anonymity folgt ebenfalls aus der Anonymity von [ASM07]. Denn der im Bezahlprotokoll Spend verwendete Zeuge $\hat{W}_J$ ist perfekt verborgen. Somit werden im Vergleich zu [ASM07] keine weiteren Informationen preisgegeben.

Exculpability: Da die Algorithmen Identify und VerifyGuilt mit den Algorithmen von [ASM07] identisch sind, folgt die Sicherheitseigenschaft Exculpability ebenfalls direkt aus [ASM07]. □

5.2 Das kompakte elektronische Geldsystem KG-II mit beliebigem Geldbörsenwert

In diesem Abschnitt wird das kompakte Geldsystem KG-I erweitert, so dass jeder Kunde im Abhebeprotokoll Withdraw eine Geldbörse mit einem beliebigen Wert $k \leq K$ abheben kann und beim Bezahlen einer Münze nicht nachvollziehbar ist, welchen Wert die zugehörige Geldbörse hat. Im Unterschied zu [ASM07, Au09] ist die Komplexität der Bezahl- und Einlöseprotokolle allerdings unabhängig vom Wert der Geldbörse.

Bevor das kompakte Geldsystem KG-II im Detail beschrieben wird, werden zunächst die wesentlichen Veränderungen im Vergleich zum kompakten Geldsystem KG-I vorgestellt:

- Es werden anstelle eines Akkumulators V insgesamt K Akkumulatoren $V_1, \ldots, V_K$ berechnet, in denen jeweils die Werte $1, \ldots, k$ für $1 \leq k \leq K$ akkumuliert sind (somit ist $V = V_K$). Jeder dieser Akkumulatoren wird durch $V_k = u_0^{\prod_{j=1}^{k}(\alpha+j)}$ für $1 \leq k \leq K$ berechnet. Diese K Akkumulatoren müssen allerdings nicht veröffentlicht werden, sondern können von der Bank $\mathcal{B}$ entweder gespeichert oder immer wieder neu berechnet werden. Allerdings müssen im Gegensatz zum kompakten Geldsystem KG-I die Generatoren $u_0, \ldots u_K \in \mathbb{G}_1$ veröffentlicht werden, damit die benötigten Zeugen berechnet werden können. Somit bleibt die Größe des öffentlichen Schlüssels im Vergleich zum Geldsystem KG-I unverändert.
- Die Bank $\mathcal{B}$ berechnet im Abhebeprotokoll $\mathsf{Withdraw}$ anstelle einer BBS+-Signatur eine BBS+A-Signatur $\sigma = (\Sigma, e, s)$ auf die Nachricht $m = (u, t, r, \phi_k(\mathsf{x}))$ für $V_k = u_0^{\phi_k(\alpha)} = u_0^{\prod_{j=1}^{k}(\alpha+j)}$ mit $\Sigma = (g g_0^s g_1^u g_2^t g_3^r V_k)^{\frac{1}{x+e}}$.
- Der Kunde $\mathcal{K}$ beweist im Bezahlprotokoll Spend anstelle der Kenntnis einer BBS+-Signatur auf eine Nachricht (u, t, r) die Kenntnis einer BBS+A-Signatur auf eine Nachricht $(u, t, r, \phi_k(\mathsf{x}))$. Denn im Gegensatz zum kompakten Geldsystem KG-I muss der verwendete Akkumulator V_k geheim bleiben, damit bei einer Münze nicht nachvollzogen werden kann, welchen Wert die zugehörige Geldbörse hat.

Im Folgenden werden die Algorithmen und Protokolle ausführlich dargestellt.

Setup führt bei Eingabe des Sicherheitsparameters 1^λ und der Zahl $K \in \mathbb{N}$ den Algorithmus $\mathsf{Setup}_{\mathsf{Bil}}$ aus und erhält die Systemparameter $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, h) \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right)$, wobei $\mathbb{G}_1 = \langle u_0 \rangle, \mathbb{G}_2 = \langle h \rangle$ und $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ eine bilineare Abbildung ist. Dann wählt Setup eine kryptografische Hashfunktion H mit $\mathsf{H}\colon \{0,1\}^* \to \mathbb{Z}_p^*$, führt den Algorithmus $\mathsf{Setup}_{\mathsf{G}}$ aus und erhält die Systemparameter $\mathsf{sp}_{\mathsf{G}} = \left(p, \mathbb{G}_p', \mathfrak{g}_0\right) \leftarrow \mathsf{Setup}_{\mathsf{G}}\left(1^\lambda, p\right)$, wobei $\mathbb{G}_p = \langle \mathfrak{g}_0 \rangle$ eine weitere zyklische Gruppe der Ordnung p ist. Anschließend wählt Setup zufällig fünf Generatoren $g, g_0, g_1, g_2, g_3 \in_{\mathcal{R}} \mathbb{G}_1$, einen Generator $\mathfrak{g}_1 \in_{\mathcal{R}} \mathbb{G}_p$ sowie eine Zahl $\alpha \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Elemente $h_1 = h^\alpha$ sowie $u_i = u_0^{\alpha^i}$ für $1 \leq i \leq K$. Da die Zahl α nicht weiter benötigt wird, wird α wieder gelöscht, um zu verhindern, dass sie kompromittiert wird. Die Ausgabe sind die Systemparameter

$$\mathsf{sp} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \mathbb{G}_p, \hat{e}, g, g_0, g_1, g_2, g_3, u_0, \ldots, u_K, h, h_1, \mathfrak{g}_0, \mathfrak{g}_1, \mathsf{H}) \leftarrow \mathsf{Setup}\left(1^\lambda, K\right).$$

BGen wählt bei Eingabe der Systemparameter sp zufällig eine Zahl $x \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet das Element $X = h^x$. Die Ausgabe ist das Schlüsselpaar

$$(\mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{B}}) = (X, x) \leftarrow \mathsf{BGen}(\mathsf{sp}).$$

Weiter legt $\mathcal{B}$ zwei Datenbanken $\mathbb{D}_W$ und $\mathbb{D}_C$ an, in denen die Abhebeinformationen bzw. die eingelösten Münzen gespeichert werden.

In der folgenden Tabelle 5.2 werden die benötigten Gruppen, deren öffentliche Generatoren, deren Verwendung und die zugrunde liegenden, kryptografischen Annahmen zusammengefasst.

Gruppen	Generatoren	Verwendung	Annahmen
$\mathbb{G}_1$	$g, g_0, g_1, g_2, g_3,$ $u_0, \ldots, u_K$		
$\mathbb{G}_2$	h, h_1, X		
$(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$		Nguyen-Akkumulator und BBS+A-Signatur	q-SDH und q-polyDH
$\mathbb{G}_p$	$\mathfrak{g}_0, \mathfrak{g}_1$	Seriennummern und Sicherheitstags	q-DDHI

Tabelle 5.2: Öffentliche Parameter von KG-II

KGen wählt bei Eingabe der Systemparameter sp zufällig eine Zahl $u \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Kontonummer $I = \mathfrak{g}_0^u$. Die Ausgabe ist das Schlüsselpaar

$$(\mathsf{pk}_{\mathcal{K}}, \mathsf{sk}_{\mathcal{K}}) = (I, u) \leftarrow \mathsf{KGen}\,(\mathsf{sp})\,.$$

Withdraw: Damit ein Kunde $\mathcal{K}$ eine elektronische Geldbörse mit $k \leq K$ Münzen bei der Bank $\mathcal{B}$ abheben kann, führen $\mathcal{K}$ und $\mathcal{B}$ das Abhebeprotokoll Withdraw durch, welches in Abbildung 5.4 dargestellt ist und im Wesentlichen dem Abhebeprotokoll des Geldsystems KG-I entspricht. Zuvor authentifizieren sich jedoch beide Parteien gegenseitig.

Schritt 1: Der Kunde $\mathcal{K}$ wählt zufällig drei Zahlen $s', t, r \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Pedersen-Commitment $C = g_0^{s'} g_1^u g_2^t g_3^r$, wobei durch die Zufallszahl r die Zahlen u, s', t perfekt verborgen sind. Dann sendet $\mathcal{K}$ das Commitment C an die Bank $\mathcal{B}$ und erstellt die folgende Signature-of-Knowledge:

$$SoK_{\mathsf{W}}\left[(s', t, r, u) : C = g_0^{s'} g_1^u g_2^t g_3^r \ \wedge \ I = \mathfrak{g}_0^u\right](k)\,.$$

Durch SoK_{W} beweist $\mathcal{K}$ in zero-knowledge, dass er eine Darstellung von C bzgl. des Generatortupels (g_0, g_1, g_2, g_3) kennt und der Exponent von g_1 der private Schlüssel u des Kunden $\mathcal{K}$ ist.

Somit ist dieser Schritt derselbe wie im kompakten Geldsystem KG-I.

Schritt 2: Die Bank $\mathcal{B}$ verifiziert SoK_{W}, wählt zufällig zwei Zahlen $e, s'' \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Element $\Sigma = \left(g g_0^{s''} C V_k\right)^{\frac{1}{x+e}}$, wobei V_k der Akkumulator

der Werte $1, \ldots, k$ ist, der von der Bank abgespeichert wurde oder neu berechnet wird. Dann sendet $\mathcal{B}$ das Tripel (Σ, e, s'') an $\mathcal{K}$ und bucht dem Kunden $\mathcal{K}$ den Betrag k vom Konto mit der Kontonummer I ab. Anschließend speichert $\mathcal{B}$ die Abhebeinformationen $\mathbb{T} = (I, k)$ in ihrer Datenbank ab.

Schritt 3: Der Kunde $\mathcal{K}$ berechnet $s = s' + s'' \mod p$, verifiziert die Signatur durch Überprüfung der Gleichung $\hat{e}(\Sigma, Xh^e) \stackrel{?}{=} \hat{e}\left(gg_0^{s''} CV_k, h\right)$ und speichert die Geldbörse $\mathbb{W} = (k, \Sigma, e, s, t, r, J)$ ab, wobei der Münzzähler $J = 1$ gesetzt wird.

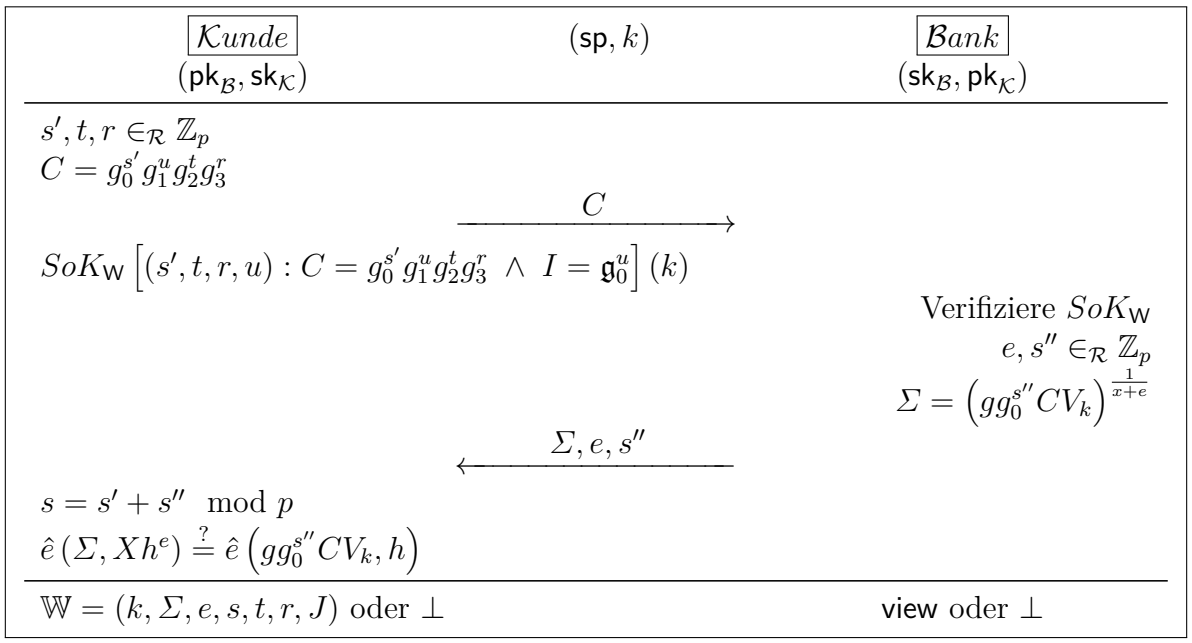

Abbildung 5.4: Abhebeprotokoll Withdraw von KG-II

Spend: Damit ein Kunde $\mathcal{K}$, der im Besitz der Geldbörse $\mathbb{W} = (k, \Sigma, e, s, t, r, J)$ mit $J \leq k \leq K$ ist, bei einem Händler $\mathcal{H}$ mit einer elektronischen Münze coin bezahlen kann, führen $\mathcal{K}$ und $\mathcal{H}$ das Bezahlprotokoll Spend durch, welches in Abbildung 5.5 dargestellt ist und im Wesentlich dem Signaturbeweisprotokoll Prove des BBS+A-Signaturverfahrens entspricht (siehe Unterabschnitt 3.2.2). Zuvor authentifiziert sich der Händler $\mathcal{H}$ gegenüber dem Kunden $\mathcal{K}$ und beide Parteien gleichen den aktuellen Zeitpunkt ts ab.

Schritt 1: Der Händler $\mathcal{H}$ und der Kunde $\mathcal{K}$ berechnen beide den Hashwert $R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}} || \mathsf{ts})$, wobei $\mathsf{pk}_{\mathcal{H}}$ der öffentliche Schlüssel des Händlers ist.

Schritt 2: Der Kunde $\mathcal{K}$ berechnet die Seriennummer $S = \mathfrak{g}_1^{\frac{1}{s+J+1}}$ sowie das Sicherheitstag $T = \mathfrak{g}_0^u \mathfrak{g}_1^{\frac{R}{t+J+1}}$ der Münze. Dann wählt $\mathcal{K}$ zufällig vier Zahlen $r_\sigma, b_1, b_2, c_1 \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Elemente $B_1 = g_0^{b_1} g_1^{b_2}, B_2 = \Sigma g_1^{b_1}$

sowie $C_1 = g_0^{J+t} g_1^{c_1}$, um das Element Σ perfekt zu verbergen. Weiter berechnet $\mathcal{K}$ das Pedersen-Commitment $D_\sigma = g_0^{r_\sigma} g_1^e g_2^s g_3^r$, um sich an die Zahlen e, s, r zu binden, da dies für die Sicherheit des BBS+A-Signaturverfahrens benötigt wird (vgl. Abschnitt 3.2). Anschließend berechnet $\mathcal{K}$ den Hashwert $l = \mathsf{H}(S||T||C_1||D_\sigma||B_1||B_2)$. Sei $\phi_{k,J}(\mathsf{x}) = \prod_{i=0,i\neq J}^{k-1}(\mathsf{x}+i) \in \mathbb{Z}_p[\mathsf{x}]$ das Polynom mit $\phi_k(\mathsf{x}) = \phi_{k,J}(\mathsf{x})(\mathsf{x}+J)$ dann berechnet der Kunde $\mathcal{K}$ die Zahl $\phi_{k,J}(l) \in \mathbb{Z}_p$ und das Polynom $\phi_{k,J,l}(\mathsf{x}) = (\phi_{k,J}(\mathsf{x}) - \phi_{k,J}(l))(\mathsf{x}-l)^{-1}$ sowie den Zeugen $W_{k,J,l} = u_0^{\phi_{k,J,l}(\alpha)}$. Weiter wählt $\mathcal{K}$ zufällig zwei Zahlen $a_1, r_a \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Elemente $A_2 = g_0^{r_a} g_1^{a_1} g_2^{\phi_{k,J}(l)}$ und $A_1 = W_{k,J,l} u_0^{a_1}$, damit der Zeuge $W_{k,J,l}$ perfekt verborgen ist. Anschließend sendet $\mathcal{K}$ die Elemente $S, T, A_1, A_2, B_1, B_2, C_1, D_\sigma$ an $\mathcal{H}$ und erstellt die Signature-of-Knowledge SoK, wobei $\delta_1 = a_1 J, \delta_2 = \phi_{k,J}(l) J, \delta_r = r_a J, \beta_1 = b_1 e, \beta_2 = b_2 e, \gamma_1 = c_1 u$ und $\gamma_2 = (J+t)u$ ist:

$$
\begin{aligned}
SoK\Big[(r_\sigma, r_a, a_1, \delta_1, \delta_2, \delta_r, b_1, b_2, \beta_1, \beta_2, c_1, \gamma_1, \gamma_2, e, s, t, u, r, J, \phi_{k,J}(l)) :\\
A_2 = g_0^{r_a} g_1^{a_1} g_2^{\phi_{k,J}(l)} \ \wedge\ 1 = A_2^J g_0^{-\delta_r} g_1^{-\delta_1} g_2^{-\delta_2} \ \wedge\ B_1 = g_0^{b_1} g_1^{b_2} \ \wedge\\
1 = B_1^e g_0^{-\beta_1} g_1^{-\beta_2} \ \wedge\ C_1 = g_0^{J+t} g_1^{c_1} \ \wedge\ 1 = C_1^u g_0^{-\gamma_2} g_1^{-\gamma_1} \ \wedge\\
\mathfrak{g}_1 S^{-1} = S^{J+s} \ \wedge\ \mathfrak{g}_1^R T^{-1} = T^{J+t} \mathfrak{g}_0^{-\gamma_2 - u} \ \wedge\ D_\sigma = g_0^{r_\sigma} g_1^e g_2^s g_3^r \ \wedge\\
\frac{\hat{e}(B_2, X)}{\hat{e}(g,h)\,\hat{e}\left(A_1, h_2 h_1^{-l}\right)} = \hat{e}\left(g_1^{b_1}, X\right)\hat{e}\left(A_1^J, h_1\right)\\
\hat{e}\left(g_0^s g_1^{u+\beta_1} g_2^t g_3^r u_0^{l\delta_1+\delta_2} u_1^{la_1-\delta_1+\phi_{k,J}(l)} u_2^{-a_1} A_1^{-lJ} B_2^{-e}, h\right)\Big](R).
\end{aligned}
$$

Durch SoK beweist $\mathcal{K}$ in zero-knowledge, dass sich der Münzzähler J im geheimen Akkumulator V_k befindet (und damit indirekt, dass $1 \leq J \leq k \leq K$ ist), er im Besitz einer BBS+A-Signatur $\sigma = (\Sigma, e, s)$ auf die Nachricht $m = (u, t, r, \phi_k(\mathsf{x}))$ ist und die Seriennummer S sowie das Sicherheitstag T korrekt berechnet wurden. Schließlich aktualisiert $\mathcal{K}$ seine Geldbörse zu $\mathbb{W}' = (k, \Sigma, e, s, t, r, J+1)$.

Schritt 3: Der Händler $\mathcal{H}$ berechnet den Hashwert $l = \mathsf{H}(S||T||C_1||D_\sigma||B_1||B_2)$, verifiziert SoK und speichert die Münze $\mathsf{coin} = (S, T, R, \Phi = (A_1, A_2, B_1, B_2, C_1, D_\sigma, SoK, \mathsf{ts}, \mathsf{pk}_\mathcal{H}))$ ab.

$\boxed{\mathcal{K}unde}$ $(\mathsf{pk}_{\mathcal{H}}, \mathsf{sk}_{\mathcal{K}}, \mathbb{W})$	$(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{ts})$	$\boxed{\mathcal{H}ändler}$ $(\mathsf{sk}_{\mathcal{H}})$
$R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}} \| \| \mathsf{ts})$		$R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}} \| \| \mathsf{ts})$
$a_1, b_1, b_2, c_1, r_a, r_\sigma \in_{\mathcal{R}} \mathbb{Z}_p$		
$S = \mathfrak{g}_1^{\frac{1}{s+J+1}}$		
$T = \mathfrak{g}_0^{u} \mathfrak{g}_1^{\frac{R}{t+J+1}}$		
$D_\sigma = g_0^{r_\sigma} g_1^{e} g_2^{s} g_3^{r}$		
$B_1 = g_0^{b_1} g_1^{b_2}$		
$B_2 = \Sigma g_1^{b_1}$		
$C_1 = g_1^{J+t} g_2^{c_1}$		
$l = \mathsf{H}(S \| \| T \| \| C_1 \| \| D_\sigma \| \| B_1 \| \| B_2)$		
$A_2 = g_0^{r_a} g_1^{a_1} g_2^{\phi_{k,J}(l)}$		
$A_1 = W_{k,j,l} u_0^{a_1}$		
	$S, T, A_1, A_2, B_1, B_2, C_1, D_\sigma \longrightarrow$	
$SoK\Big[(r_\sigma, r_a, a_1, \delta_1, \delta_2, \delta_r, b_1, b_2, \beta_1, \beta_2, c_1, \gamma_1, \gamma_2, e, s, t, u, r, J, \phi_{k,J}(l)) :$		
$A_2 = g_0^{r_a} g_1^{a_1} g_2^{\phi_{k,J}(l)} \ \wedge\ 1 = A_2^J g_0^{-\delta_r} g_1^{-\delta_1} g_2^{-\delta_2} \ \wedge\ B_1 = g_0^{b_1} g_1^{b_2} \ \wedge$		
$1 = B_1^e g_0^{-\beta_1} g_1^{-\beta_2} \ \wedge\ C_1 = g_0^{J+t} g_1^{c_1} \ \wedge\ 1 = C_1^u g_1^{-\gamma_2} g_2^{-\gamma_1} \ \wedge$		
$\mathfrak{g}_1 S^{-1} = S^{J+s} \ \wedge\ \mathfrak{g}_1^R T^{-1} = T^{J+t} \mathfrak{g}_0^{-\gamma_2 - u} \ \wedge D_\sigma = g_0^{r_\sigma} g_1^{e} g_2^{s} g_3^{r} \ \wedge$		
$\frac{\hat{e}(B_2, X)}{\hat{e}(g,h)\,\hat{e}\left(A_1, h_2 h_1^{-l}\right)} = \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(A_1^J, h_1\right)$		
$\hat{e}\left(g_0^{s} g_1^{u+\beta_1} g_2^{t} g_3^{r} u_0^{l\delta_1+\delta_2} u_1^{la_1-\delta_1+\phi_{k,J}(l)} u_2^{-a_1} A_1^{-lJ} B_2^{-e}, h\right)\Big]\,(R)$		
		$l = \mathsf{H}(S \| \| T \| \| C_1 \| \| D_\sigma \| \| B_1 \| \| B_2)$
		Verifiziere SoK
$\mathbb{W}'$ oder $\perp$		coin oder $\perp$

Abbildung 5.5: Bezahlprotokoll Spend von KG-II

Deposit: Damit ein Händler $\mathcal{H}$ eine Münze coin bei der Bank $\mathcal{B}$ einlösen kann, führen beide Parteien das Einlöseprotokoll $\mathsf{Deposit}$ durch (Abbildung 5.6), nachdem sich beide Parteien gegenseitig authentifiziert haben.

Schritt 1: Der Händler $\mathcal{H}$ sendet die Münze $\mathsf{coin} = (S, T, R, \Phi = (A_1, A_2, B_1, B_2, C_1, D_\sigma, SoK, \mathsf{ts}, \mathsf{pk}_\mathcal{H}))$ an die Bank $\mathcal{B}$. (Die Kontonummer $\mathsf{pk}_\mathcal{H}$ muss dabei nicht erneut gesendet werden, da $\mathcal{B}$ diese bereits kennt.)

Schritt 2: Die Bank $\mathcal{B}$ überprüft $R \stackrel{?}{=} \mathsf{H}(\mathsf{pk}_\mathcal{H}||\mathsf{ts})$ um sicherzustellen, dass mit der Münze coin tatsächlich beim Händler $\mathcal{H}$ bezahlt wurde und nicht bspw. ein betrügerischer Händler die Münze abgefangen hat. Dann berechnet $\mathcal{B}$ den Hashwert $l = \mathsf{H}(S||T||C_1||D_\sigma||B_1||B_2)$ und verifiziert die Signature-of-Knowledge SoK. Falls SoK abgelehnt wird, sendet $\mathcal{B}$ eine Fehlermeldung $\perp$ an $\mathcal{H}$. Wenn SoK akzeptiert wird, sucht die Bank $\mathcal{B}$ in ihrer Datenbank $\mathbb{D}_C$, ob bereits eine Münze mit der Seriennummer S gespeichert wurde. Wenn nicht, speichert $\mathcal{B}$ die Münze coin in ihrer Datenbank $\mathbb{D}_C$. Falls bereits eine Münze mit der Seriennummer S in der Datenbank $\mathbb{D}_C$ vorhanden ist, führt $\mathcal{B}$ den Algorithmus $\mathsf{Identify}$ aus, um den Double-Spender zu identifizieren.

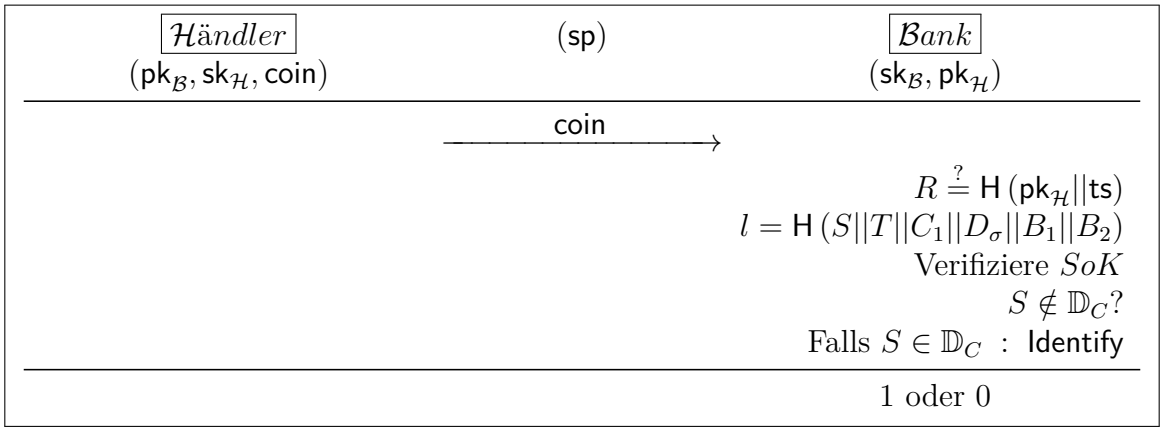

Abbildung 5.6: Einlöseprotokoll $\mathsf{Deposit}$ von KG-II

Die Algorithmen **Identify** und **VerifyGuilt** bleiben unverändert.

5.2.1 Sicherheitsanalyse von KG-II

Die Sicherheit des kompakten Geldsystems KG-II folgt direkt aus der Sicherheit des in Abschnitt 5.1 vorgestellten kompakten Geldsystems KG-I und der Sicherheit der BBS+A-Signaturverfahrens.

Satz 5.2.1. *Das kompakte elektronische Geldsystem KG-II erfüllt im Random-Oracle-Modell die in [ASM07] definierten Sicherheitseigenschaften, falls das kompakte elektronische Geldsystem aus [ASM07, Abschnitt 4.2] diese Sicherheitseigenschaften erfüllt und die q-SDH-Annahme sowie die q-polyDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gelten.*

Beweis. Die in [ASM07] definierten Sicherheitseigenschaften Balance, Anonymity und Exculpability folgen direkt aus den entsprechenden Beweisen von KG-I aus Abschnitt 5.1.

Balance: Sei $\mathcal{A}$ ein polynomieller Angreifer, der q_W Abhebe-Anfragen bzgl. Geldbörsen mit $k_1, \ldots, k_{q_W}$ Münzen und q_S Bezahl-Anfragen stellt. Analog zum kompakten Geldsystem KG-I, kann der Angreifer $\mathcal{A}$ die Sicherheitseigenschaft Balance brechen, wenn er $\sum_{i=1}^{q_W} k_i + q_S + 1$ Einlöseprotokolle Deposit durchführt, indem entweder (1) alle $\sum_{i=1}^{q_W} k_i + q_S + 1$ Seriennummern verschieden oder (2) einige Seriennummern identisch sind, aber der Algorithmus Identify keine Kontonummer eines korrupten Kunden ausgibt.

Der Angreifer $\mathcal{A}$ kann Fall (1) nur gewinnen, falls er einen falschen Beweis als Teil der Signature-of-Knowledge SoK erstellt, so dass mindestens einer der folgenden Beweise eine Fälschung ist:

1. Besitz einer BBS+A-Signatur $\sigma = (\Sigma, e, s)$ auf die Nachricht $m = (u, t, r, \phi_k(\mathsf{x}))$.
2. $S = \mathfrak{g}_1^{\frac{1}{s+J+1}}$.

Der zweite Punkt ist wie beim kompakten Geldsystem KG-II vernachlässigbar. Daher muss nur der erste Punkt analysiert werden:

Ein falscher Beweis für die Kenntnis einer BBS+A-Signatur ist unter der q-SDH-Annahme und q-polyDH-Annahme vernachlässigbar, da der Angreifer ansonsten die Unfälschbarkeit des BBS+A-Signaturverfahrens gebrochen hätte.

Der Fall (2) ist wie beim kompakten Geldsystem KG-I vernachlässigbar. Somit ist analog zum kompakten Geldsystem KG-I die gesamte Erfolgswahrscheinlichkeit des Angreifers $\mathcal{A}$ vernachlässigbar.

Anonymity: Die Sicherheitseigenschaft Anonymity folgt ebenfalls direkt aus der Anonymity des kompakten Geldsystems KG-I. Denn der im Bezahlprotokoll Spend verwendete Zeuge $W_{k,J,l}$ ist perfekt verborgen. Zudem sind die verwendeten Pedersen-Commitments A_2 und D_σ durch die Zufallszahlen r_a bzw. r_σ perfekt verbergend. Somit werden im Vergleich zum kompakten Geldsystem KG-I keine weiteren Informationen preisgegeben.

Exculpability: Da die Algorithmen Identify und VerifyGuilt mit denen des kompakten Geldsystems KG-I identisch sind, folgt die Sicherheitseigenschaft Exculpability ebenfalls direkt aus dem kompakten Geldsystem KG-I. □

KAPITEL 6

TEILBARE ELEKTRONISCHE GELDSYSTEME

In diesem Kapitel werden zwei teilbare elektronische Geldsysteme vorgestellt, welche die in Kapitel 4 beschriebenen Sicherheitsanforderungen erfüllen. Das erste teilbare Geldsystem TG-I (Abschnitt 6.5) ist in allen Pairing Typen sicher und bildet die Grundlage für alle weiteren teilbaren Geldsysteme in den nächsten Kapiteln. Das zweite Geldsystem TG-II (Abschnitt 6.6) ist eine spezielle Modifizierung des ersten Geldsystems, wodurch es zwar nicht weiter im Typ-1-Pairing einsetzbar ist, da die Anonymität dann leicht gebrochen werden kann, dafür aber effizienter ist. Ein weiterer Vorteil des Geldsystems TG-II ist, dass aus diesem ein faires Geldsystem (FTG-0) entsteht, wenn der Deanonymisierer $\mathcal{D}$ einen einzigen diskreten Logarithmus kennt, ohne dass die bestehenden Algorithmen und Protokolle verändert werden müssen (siehe Abschnitt 7.2).

Da das teilbare elektronische Geldsystem TG-I zentrale Elemente der beiden teilbaren elektronischen Geldsysteme aus [ASM08] und [CG10] kombiniert, werden diese zunächst im Allgemeinen beschrieben. Dazu wird jedoch zuerst die Konstruktion von teilbaren Münzen mit Hilfe von Binär-Bäumen vorgestellt.

6.1 Teilbare Münzen und Binär-Bäume

Ein wesentlicher Bestandteil teilbarer elektronischer Geldsysteme sind die sogenannten *Binär-Bäume*, wie sie in der folgenden Abbildung 6.1 dargestellt sind.

Dieser Binär-Baum besitzt drei Zeilen und sieben Knoten, die mit den Indizes $(0,0)$ bis $(2,3)$ nummeriert werden. Im Allgemeinen besitzt jeder Knoten einen Index (i,j) mit $0 \leq i \leq L$ und $0 \leq j \leq 2^i - 1$, wobei $L+1$ die Anzahl der Zeilen ist und die oberste Zeile der nullten Zeile entspricht. Jeder Binär-Baum besitzt für $0 \leq i \leq L$ in der i-ten

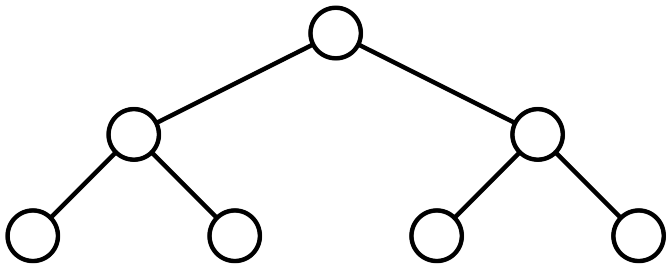

Abbildung 6.1: Darstellung eines Binär-Baums

Zeile 2^i Knoten, wobei jeder dieser Knoten einem *Schlüssel* entspricht, mit dem eine Münze erzeugt werden kann. Dabei ist der Wert der Münze von der Lage des Knotens abhängig. Üblicherweise besitzt ein Schlüssel in der i-ten Zeile einen Geldwert von 2^{L-i} für $0 \leq i \leq L$, wobei 2^L der Wert der Geldbörse ist.

Möchte nun ein Kunde mit einer Münze mit dem Wert $2^\ell \leq 2^L$ bezahlen, verwendet dieser einen unbenutzten Schlüssel aus der $(L-\ell)$-ten Zeile des Binär-Baums um eine Münze zu erzeugen. Danach gelten dieser Schlüssel sowie alle Nachfolger und Vorgänger als benutzt und dürfen anschließend nicht verwendet werden. Wird beispielsweise der Schlüssel $(1,0)$ verwendet (in der folgenden Abbildung 6.2 schwarz gefärbt), dürfen dieser, die Nachfolger $(2,0)$ und $(2,1)$ sowie der Vorgänger $(0,0)$ (in der folgenden Abbildung 6.2 grau gefärbt) nicht mehr benutzt werden. Dies veranschaulicht Abbildung 6.2:

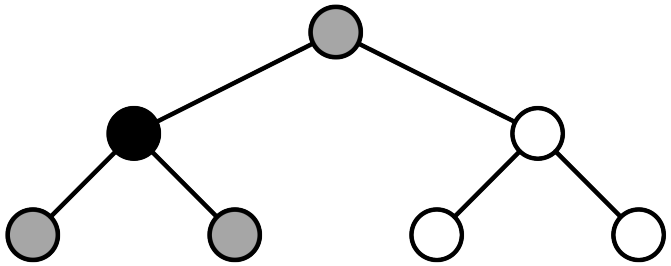

Abbildung 6.2: Verwendung eines Binär-Baums

6.2 Das teilbare elektronische Geldsystem von Au *et al.* (ASM08)

Das teilbare elektronische Geldsystem in [ASM08] (siehe auch [Au09, Abschnitt 5.5]) arbeitet auf einem Typ-3-Pairing $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ und verwendet als wesentliche Bausteine das Nguyen-Akkumulator-Schema (vgl. Unterabschnitt 2.11.1) und das ESS+-Signaturverfahren (vgl. Unterabschnitt 2.12.2). Jeder Kunde besitzt ein Schlüsselpaar $(\mathsf{pk}_\mathcal{K}, \mathsf{sk}_\mathcal{K}) \in \mathbb{G}_1 \times \mathbb{Z}_p^*$ und kann im Abhebeprotokoll eine Geldbörse im Wert von $K =$

2^L abheben. Dazu wählt der Kunde zufällig einen *Master-Schlüssel* $\kappa_{0,0} \in_{\mathcal{R}} \mathbb{Z}_p^*$ und erzeugt einen Binär-Baum mit $L+1$ Zeilen. Die Schlüssel des Binär-Baums werden durch $\kappa_{i+1,2j} = \mathsf{H}_0\left(g^{\kappa_{i,j}}\right)$ sowie $\kappa_{i+1,2j+1} = \mathsf{H}_1\left(g^{\kappa_{i,j}}\right)$ für $0 \leq i \leq L-1$ und $0 \leq j \leq 2^i - 1$ berechnet, wobei g ein zufälliger Generator der Gruppe $\mathbb{G}_1$ ist und H_0 sowie H_1 zwei Hashfunktionen mit $\mathsf{H}_0, \mathsf{H}_1 \colon \{0,1\}^* \to \mathbb{Z}_p^*$ sind.

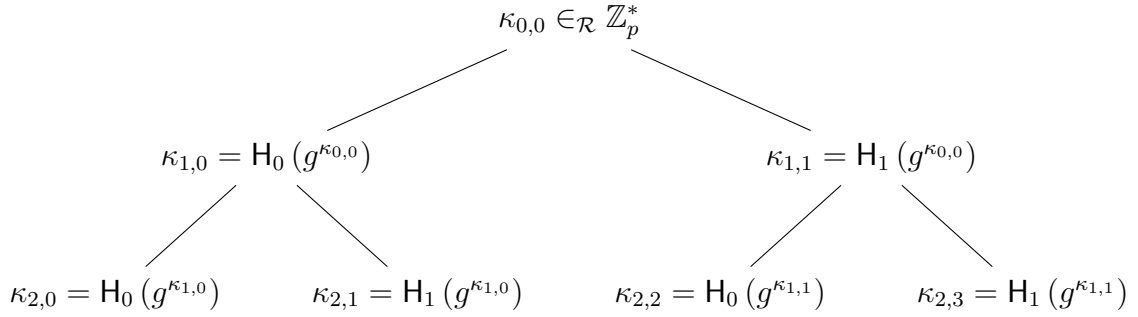

Abbildung 6.3: Berechnung des Binär-Baums in [ASM08] für $L = 2$

Das Abhebeprotokoll entspricht im Wesentlichen der $(L+1)$-fachen Durchführung des Issue-Protokolls des ESS+-Signaturverfahrens für die Nachrichten $m_i = (V_i, \mathsf{sk}_{\mathcal{K}}) \in \mathbb{G}_1 \times \mathbb{Z}_p$, wobei $V_i = \mathsf{Acc}\left(\left\{\kappa_{i,0}, \ldots, \kappa_{i,2^i-1}\right\}\right)$ der Akkumulator der 2^i Schlüssel der i-ten Zeile des Binär-Baums für $0 \leq i \leq L$ ist. Jedoch erhält der Kunde nur in 50 Prozent der Fälle die entsprechenden $L+1$ ESS+-Signaturen $\sigma_0, \ldots, \sigma_L$ auf die Nachrichten $m_0, \ldots, m_L$. In den anderen Fällen muss der Kunde den Binär-Baum und die verwendeten Zufallswerte an die Bank senden, damit diese die korrekte Berechnung der Akkumulatoren überprüfen kann. Im Anschluss muss das Abhebeprotokoll erneut durchgeführt werden.

Möchte der Kunde nun mit einer Münze mit dem Wert $2^\ell \leq 2^L$ bezahlen, wählt dieser einen unbenutzten Schlüssel $\kappa_{L-\ell,j}$ aus der $(L-\ell)$-ten Zeile des Binär-Baums und berechnet die Seriennummer $S = g_S^{\kappa_{L-\ell,j}}$ sowie das Sicherheitstag $T = \mathsf{pk}_{\mathcal{K}} g_T^{R\kappa_{L-\ell,j}}$, wobei g_S und g_T zufällige Generatoren der Gruppe $\mathbb{G}_1$ sind und $R \in \mathbb{Z}_p^*$ ein Hashwert ist. Das Bezahlprotokoll entspricht nun dem Prove-Protokoll des ESS+-Signaturverfahrens, wobei der geheime Wert n_i dem Schlüssel $\kappa_{L-\ell,j}$, die Seriennummer S dem Commitment D_n und das Sicherheitstag T dem Commitment D_m entsprechen.

Im Einlöseprotokoll sendet der Händler die erhaltene Münze an die Bank, die aus der Seriennummer S alle nachfolgenden Schlüssel des Binär-Baums berechnet und somit ein mögliches Double-Spending aufdeckt. Falls ein Double-Spending vorliegt, berechnet die Bank aus den beiden Sicherheitstags den öffentlichen Schlüssel des Double-Spenders.

Der entscheidende Nachteil dieses Geldsystems ist, dass die Bank im Abhebeprotokoll nur mit Wahrscheinlichkeit 1/2 die korrekte Berechnung der Akkumulatoren überprüft und somit ein korrupter Kunde mit Wahrscheinlichkeit 1/2 die Unfälschbarkeit brechen kann. Daher gewährleistet das System nur eine *statistische Unfälschbarkeit* (siehe [ASM08]).

6.3 Das teilbare elektronische Geldsystem von Canard und Gouget (CG10)

Das teilbare elektronische Geldsystem in [CG10] setzt ebenfalls als grundlegende Komponenten das Nguyen-Akkumulator-Schema und das ESS+-Signaturverfahren ein. In [CG10] arbeiten diese Bausteine jedoch auf der eindeutig bestimmten zyklischen Untergruppe $\mathbb{G}_p$ der Ordnung p von $\mathbb{Z}_P^*$, wobei p und P Primzahlen mit $P = 2p + 1$ sind. Da für diese Verfahren allerdings ein Pairing benötigt wird, sollte aus praktischer Sicht anstelle der Gruppe $\mathbb{G}_p$ eine Gruppe $\mathbb{G}_1$ der Ordnung p verwendet werden, wobei $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ eine bilineare Abbildung ist. Die Sicherheit des Geldsystems wird dabei nicht verletzt ([Can13]).

Seien nun $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ eine bilineare Abbildung vom Typ-3 und $\mathbb{G}_q$ die eindeutig bestimmte zyklische Untergruppe der Ordnung q von $\mathbb{Z}_p^*$, wobei q eine Primzahl mit $q \mid (p-1)$ ist.

Wie in [ASM08] besitzt jeder Kunde ein Schlüsselpaar $(\mathsf{pk}_{\mathcal{K}}, \mathsf{sk}_{\mathcal{K}}) \in \mathbb{G}_1 \times \mathbb{Z}_p^*$ und kann im Abhebeprotokoll eine Geldbörse im Wert von $K = 2^L$ abheben. Dazu wählt der Kunde zufällig einen Master-Schlüssel $\kappa_{0,0} \in_{\mathcal{R}} \mathbb{Z}_p^*$ und erzeugt einen Binär-Baum mit $L + 2$ Zeilen, wobei die Schlüssel der $(L+1)$-ten Zeile keinen Geldwert besitzen. Die Schlüssel des Binär-Baums werden durch $\kappa_{i+1,2j} = \mathfrak{g}_0^{\kappa_{i,j} \pmod q}$ sowie $\kappa_{i+1,2j+1} = \mathfrak{g}_1^{\kappa_{i,j} \pmod q}$ für $0 \leq i \leq L$ und $0 \leq j \leq 2^i - 1$ berechnet, wobei $\mathfrak{g}_0$ und $\mathfrak{g}_1$ zufällige Generatoren der Gruppe $\mathbb{G}_q$ sind.

Das Abhebeprotokoll entspricht im Wesentlichen der $(L+2)$-fachen Durchführung des Issue-Protokolls des ESS+-Signaturverfahrens für die Nachrichten $m_0 = (V, \mathsf{sk}_{\mathcal{K}}, s) \in \mathbb{G}_1 \times \mathbb{Z}_p^2$ und $m_i = (V_i, s, i) \in \mathbb{G}_1 \times \mathbb{Z}_p^2$ für $1 \leq i \leq L+1$, wobei $V = \mathsf{Acc}(\{\kappa_{1,0}, \ldots, \kappa_{L+1,2^{L+1}-1}\})$ der Akkumulator aller Schlüssel des Binär-Baums außer $\kappa_{0,0}$ und $V_i = \mathsf{Acc}\left(\left\{\kappa_{i,0}, \ldots, \kappa_{i,2^i-1}\right\}\right)$ der Akkumulator der 2^i Schlüssel der i-ten Zeile des Binär-Baums für $1 \leq i \leq L+1$ ist.

Möchte der Kunde nun mit einer Münze mit dem Wert $2^\ell \leq 2^L$ bezahlen, wählt dieser einen unbenutzten Schlüssel $\kappa_{L-\ell,j}$ aus der $(L-\ell)$-ten Zeile des Binär-Baums und berechnet die Seriennummer $S = \kappa_{L-\ell+1,2j} || \kappa_{L-\ell+1,2j+1} = \mathfrak{g}_0^{\kappa_{L-\ell,j}} || \mathfrak{g}_1^{\kappa_{L-\ell,j}}$ sowie das Sicherheitstag $T = \mathsf{pk}_{\mathcal{K}} g_T^{R\kappa_{L-\ell,j}}$, wobei g_T ein zufälliger Generator der Gruppe $\mathbb{G}_1$ und $R \in \mathbb{Z}_p^*$ ein Hashwert ist. Das Bezahlprotokoll entspricht nun im Wesentlichen der zweifachen Durchführung des Prove-Protokolls des ESS+-Signaturverfahrens, wobei mit den Erweiterungen von Canard und Gouget aus Unterunterabschnitt 2.11.1.1 und 2.11.1.2 in zero-knowledge bewiesen wird, dass sich die Schlüssel $\kappa_{L-\ell+1,2j}$ und $\kappa_{L-\ell+1,2j+1}$ im Akkumulator $V_{L-\ell+1}$ und sich diese Schlüssel $\kappa_{L-\ell+1,2j}$ und $\kappa_{L-\ell+1,2j+1}$ sowie alle nachfolgenden Schlüssel im Akkumulator V befinden. Dazu muss der Händler im Gegensatz zu [ASM08] aus der Seriennummer S alle nachfolgenden Schlüssel des Binär-Baums berechnen. Weiter muss der Kunde beweisen, dass er die beiden ESS+-Signaturen auf die Nachrichten $m_{L-\ell+1} = (V_{L-\ell+1}, s, L-\ell+1)$ und $m_0 = (V, \mathsf{sk}_{\mathcal{K}}, s)$ kennt.

Wie bereits erwähnt, ist dieses Geldsystem allerdings nicht sicher, da die Unfälschbar-

keit effizient gebrochen werden kann. Im folgenden Unterabschnitt 6.3.1 wird ein Angriff für $L \geq 2$ beschrieben, mit dem eine Geldbörse im Wert von $1,5 \cdot 2^L$ abgehoben werden kann.

6.3.1 Brechen der Unfälschbarkeit

In der folgenden Abbildung 6.4 ist ein Binär-Baum für $L = 2$ mit den entsprechenden Akkumulatoren schematisch für das Geldsystem aus [CG10] dargestellt:

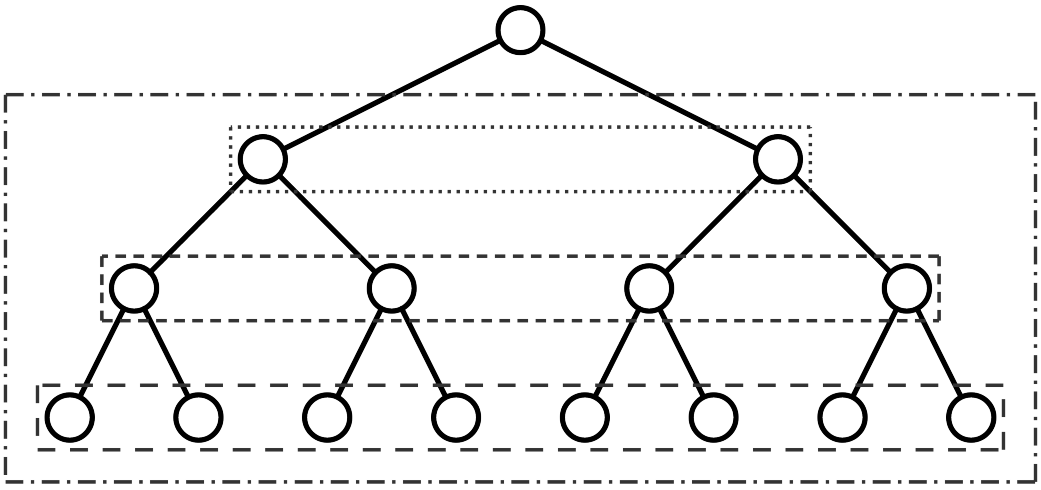

Abbildung 6.4: Anwendung der Akkumulatoren in [CG10] für $L = 2$

In den Akkumulatoren V_1, V_2 bzw. V_3 werden die jeweiligen Schlüssel der ersten, zweiten bzw. dritten Zeile und zusätzlich werden im Akkumulator V alle Schlüssel der ersten, zweiten und dritten Zeile akkumuliert. (Beachte, dass die oberste Zeile der nullten Zeile entspricht.) Gemäß [CG10] sendet der Kunde im Abhebeprotokoll anschließend die Akkumulatoren V, V_1, V_2 und V_3 an die Bank, welche dann jeweils ein Commitment und eine Signatur gemäß dem ESS+-Signaturverfahren erstellt. Offensichtlich kann in diesem Fall die Anonymität jedoch leicht gebrochen werden, da die Bank die Akkumulatoren im Klartext erhält. Denn nachdem eine Geldbörse vollständig ausgegeben wurde, kann die Bank den Akkumulator V_{L+1} berechnen und mit den Abhebeprotokollen vergleichen. Somit kann die Bank jede Münze mit dem Wert 2^L direkt dem entsprechenden Abhebeprotokoll zuordnen. Aus diesem Grund muss der Kunde, wie in [ASM08], das jeweilige Commitment erstellen und dieses an die Bank senden, die anschließend eine Signatur generiert (vgl. auch Ablauf des Issue-Protokolls des ESS+-Signaturverfahrens in Unterabschnitt 2.12.2). Durch die Verwendung des Commitment-Schemas für das ESS+-Signaturverfahren ist daher jeder Akkumulator perfekt verborgen. Somit kann die Bank nicht nachvollziehen, ob die Akkumulatoren korrekt gemäß dem Abhebeprotokoll berechnet wurden.

Nun wird ein Angriff für $L \geq 2$ vorgestellt, so dass ein korrupter Kunde eine Geldbörse im Wert von $1,5 \cdot 2^L$ anstatt 2^L abheben kann. Dazu erzeugt der Kunde zunächst zwei unabhängige Binär-Bäume mit jeweils $L+2$ Zeilen, indem er zufällig die beiden Master-Schlüssel $\kappa_{0,0}, \kappa'_{0,0} \in_{\mathcal{R}} \mathbb{Z}_p^*$ wählt. Die entsprechenden Schlüssel des ersten Binär-Baums seien $\kappa_{i,j}$ und die des zweiten Binär-Baums seien $\kappa'_{i,j}$ für $0 \leq i \leq L+1$ und $0 \leq j \leq 2^i - 1$. Dann werden die Akkumulatoren $V_{L+1} = \mathsf{Acc}\left(\left\{\kappa_{L+1,0}, \ldots, \kappa_{L+1,2^{L+1}-1}\right\}\right)$ und $V_i = \mathsf{Acc}\left(\left\{\kappa'_{i,0}, \ldots, \kappa'_{i,2^i-1}\right\}\right)$ für $1 \leq i \leq L$ berechnet. Insbesondere befinden sich somit die beiden Schlüssel $\kappa'_{2,0}$ und $\kappa'_{2,1}$ in V_2. Die Akkumulatoren $V_1, V_3, \ldots, V_L$ werden zur Erzeugung der Münzen nicht benötigt und könnten daher auch beliebig berechnet werden. Anschließend werden im Akkumulator V die 2^{L+1} Schlüssel $\kappa_{L+1,0}, \ldots, \kappa_{L+1,2^{L+1}-1}$ und alle $2^{L+1} - 2$ Nachfolger von $\kappa'_{1,0}$ akkumuliert:

$$V = \mathsf{Acc}\left(\left\{\kappa_{L+1,0}, \ldots, \kappa_{L+1,2^{L+1}-1}\right\} \cup \bigcup_{i=2}^{L+1} \left\{\kappa'_{i,0}, \ldots, \kappa'_{i,2^{i-1}-1}\right\}\right).$$

Da jeder Akkumulator durch das verwendete Commitment-Schema perfekt verborgen ist, wird der Betrug nicht bemerkt. Die Erzeugung der Akkumulatoren ist schematisch in der folgenden Abbildung 6.5 für $L = 2$ dargestellt, wobei der Akkumulator V entsprechend in zwei Rechtecke unterteilt ist.

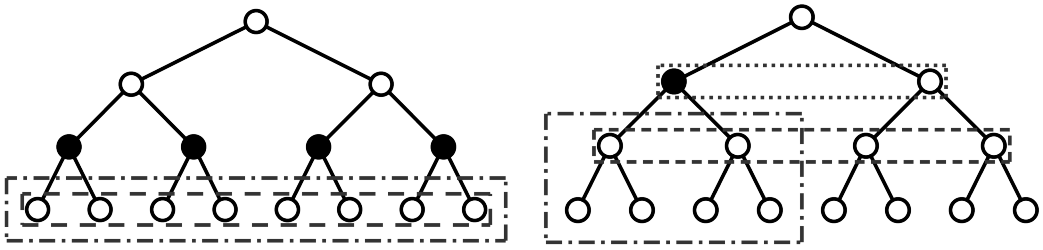

Abbildung 6.5: Angriff auf das Geldsystem in [CG10] für $L = 2$

Die in dieser Abbildung schwarz gefärbten Knoten entsprechenden denjenigen Schlüsseln, mit denen der Kunde gültige Münzen erzeugen kann. Im Allgemeinen können die 2^L Schlüssel $\kappa_{L,0}, \ldots, \kappa_{L,2^L-1}$ mit dem Wert 1 und der Schlüssel $\kappa'_{1,0}$ mit dem Wert $2^{L-1} = 0,5 \cdot 2^L$ verwendet werden, um gültige Münzen zu erstellen. Denn für jeden Schlüssel $\kappa_{L,j}$ für $0 \leq j \leq 2^L - 1$ sind die beiden Nachfolger $\kappa_{L+1,2j}$ und $\kappa_{L+1,2j+1}$ im Akkumulator V_{L+1} sowie im Akkumulator V akkumuliert. Weiter sind für den Schlüssel $\kappa'_{1,0}$ die beiden direkten Nachfolger $\kappa'_{2,0}$ und $\kappa'_{2,1}$ im Akkumulator V_2 und sämtliche Nachfolger im Akkumulator V akkumuliert. Folglich können insgesamt Münzen im Wert von $1,5 \cdot 2^L$ erzeugt und ausgegeben werden.

Verhindern ließe sich dieser Angriff analog zu [ASM08], indem die Bank zu 50 Prozent die Berechnung der Akkumulatoren überprüft. Dadurch wäre allerdings wie in [ASM08] nur noch eine statistische Unfälschbarkeit gewährleistet. Zudem muss der maximale Geldbetrag ermittelt werden, der durch einen Betrug abgehoben werden kann.

Ein weiterer Ansatz wäre, dass für alle Nachfolger der Seriennummer S bewiesen wird, dass sie sich in den jeweiligen Akkumulatoren befinden und diese von der Bank signiert wurden. Dadurch würde sich jedoch die Komplexität des Datentransfers beim Bezahlvorgang von $O(\lambda)$ auf $O(\lambda \cdot \ell)$ erhöhen und wäre damit abhängig vom Wert der Münze.

6.4 Allgemeine Konstruktion der neuen teilbaren elektronischen Geldsysteme

Bevor das teilbare elektronische Geldsystem TG-I im nächsten Abschnitt 6.5 detailliert beschrieben wird, werden in diesem Abschnitt zunächst die zentralen Eigenschaften dargelegt. Ein wesentlicher Unterschied im Vergleich zu den beiden Geldsystemen in [ASM08] und [CG10] ist, dass das effizientere BBS+A-Signaturverfahren anstelle des ESS+-Signaturverfahrens verwendet wird. Zudem ist dieses Verfahren in allen Pairing Typen stark existentiell unfälschbar unter einem Angriff mit gewählten Nachrichten. Aus diesem Grund kann das Geldsystem TG-I in allen Pairing Typen eingesetzt und in Abschnitt 6.6 zum teilbaren elektronischen Geldsystem TG-II modifiziert werden. Der bedeutendste Vorteil des Geldsystems TG-I liegt allerdings darin, dass die Komplexität des Datentransfers im Abhebeprotokoll im Vergleich zu [ASM08] und [CG10] von $O(\lambda \cdot \log(K))$ auf $O(\lambda)$ reduziert wird und damit unabhängig vom Wert der Geldbörse ist. Dies wird dadurch realisiert, dass der Kunde im Abhebeprotokoll nur einen einzigen Akkumulator berechnen und die Bank somit nur eine Signatur erstellen muss. Daher wird ebenfalls die Berechnungskomplexität der Bank von $O(\lambda \cdot K)$ in [ASM08] bzw. $O(\lambda \cdot \log(K))$ in [CG10] auf $O(\lambda)$ minimiert.

Die Berechnungen des Schlüsselpaares der Kunden, des Binär-Baums und der Seriennummern sowie Sicherheitstags werden dabei im Grunde wie in [ASM08] durchgeführt. Zur Konstruktion des Bezahlprotokolls wird dann die Erweiterung von [CG10] verwendet, mit der in zero-knowledge bewiesen werden kann, dass sich mehrere bekannte Werte in einem geheimen Akkumulator befinden.

Das Geldsystem arbeitet mit den Parametern $(p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, h) \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right)$ und $\left(p, \mathbb{G}_p', \mathfrak{g}\right) \leftarrow \mathsf{Setup}_{\mathsf{G}}\left(1^\lambda, p\right)$, wobei die q-SDH-Annahme sowie die q-polyDH-Annahme für $\mathsf{Setup}_{\mathsf{Bil}}$ und die DDH-Annahme für $\mathsf{Setup}_{\mathsf{G}}$ gelten müssen. Jeder Kunde besitzt ein Schlüsselpaar $(\mathsf{pk}_{\mathcal{K}}, \mathsf{sk}_{\mathcal{K}}) \in \mathbb{G}_p \times \mathbb{Z}_p^*$ und kann im Abhebeprotokoll eine Geldbörse im Wert von $K = 2^L$ abheben. Dazu generiert der Kunde zunächst einen Binär-Baum mit $L + 1$ Zeilen, der dem Binär-Baum aus [ASM08] entspricht. Allerdings werden die beiden Hashfunktionen H_0 und H_1 durch $\mathsf{H}_0(m) := \mathsf{H}(m||0)$ und $\mathsf{H}_1(m) := \mathsf{H}(m||1)$ für $m \in \{0,1\}^*$ definiert, wobei $\mathsf{H}\colon \{0,1\}^* \to \mathbb{Z}_p^*$ eine kryptografische Hashfunktion ist. Des Weiteren werden die Seriennummern nicht in der Gruppe $\mathbb{G}_1$, sondern in der Gruppe $\mathbb{G}_p$ durch einen zufälligen Generator $\mathfrak{g}$ erzeugt. Der wesentliche Unterschied liegt allerdings darin, dass der Binär-Baum eine weitere Zeile besitzt, damit im Bezahlprotokoll mit

jedem Betrag 2^ℓ für $0 \leq \ell \leq L$ bezahlt werden kann.

Zur Erzeugung des Binär-Baums wählt der Kunde zufällig einen Master-Schlüssel $\kappa_{0,0} \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Schlüssel des Binär-Baums durch $\kappa_{i+1,2j} = \mathsf{H}_0\left(\mathfrak{g}^{\kappa_{i,j}}\right)$ sowie $\kappa_{i+1,2j+1} = \mathsf{H}_1\left(\mathfrak{g}^{\kappa_{i,j}}\right)$ für $0 \leq i \leq L-1$ und $0 \leq j \leq 2^i - 1$. Dann berechnet der Kunde die K sogenannten *Serienschlüssel* $\kappa_j := \kappa_{L+1,j} = \mathsf{H}_0\left(\mathfrak{g}^{\kappa_{L,j}}\right)$ für $0 \leq j \leq K-1$. In folgender Abbildung 6.6 ist die Erzeugung des Binär-Baums exemplarisch für $L = 2$ dargestellt:

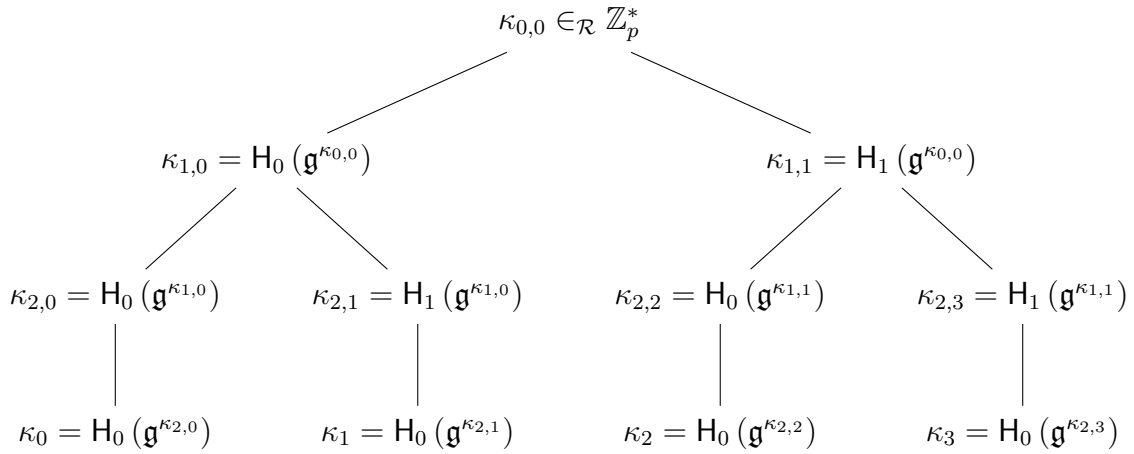

Abbildung 6.6: Berechnung des Binär-Baums für $L = 2$

Nach der Generierung des Binär-Baums werden die K Serienschlüssel $\kappa_0, \ldots, \kappa_{K-1} \in \mathbb{Z}_p^*$ zu $V = \mathsf{Acc}(\{\kappa_0, \ldots, \kappa_{K-1}\})$ akkumuliert. Nur diese Serienschlüssel besitzen jeweils einen Geldwert von 1, so dass der Geldwert der Geldbörse insgesamt K beträgt. Die übrigen Schlüssel des Binär-Baums können als *Master-Seriennummern* betrachtet werden, die keinen Geldwert besitzen und lediglich dazu benötigt werden, um mehrere Serienschlüssel zu generieren.

Das Abhebeprotokoll des teilbaren Geldsystems TG-I entspricht nun im Wesentlichen dem Signaturerstellungsprotokoll Issue des BBS+A-Signaturverfahrens aus Abschnitt 3.2 für die Nachricht $m = (\mathsf{sk}_{\mathcal{K}}, \phi(\mathsf{x})) \in \mathbb{Z}_p \times \mathbb{Z}_p\,[\mathsf{x}]$ mit $V = u_0^{\phi(\alpha)}$. Insbesondere werden somit nur die Schlüssel der untersten Zeile des Binär-Baums zu einem Akkumulator akkumuliert. Da folglich nur ein Akkumulator benötigt wird, wird einerseits die Komplexität des Datentransfers im Abhebeprotokoll im Vergleich zu [CG10] auf $O(\lambda)$ reduziert und andererseits der in Unterabschnitt 6.3.1 beschriebene Angriff auf das Sicherheitsziel Unfälschbarkeit verhindert.

Möchte der Kunde nun mit einer Münze mit dem Wert $k = 2^\ell \leq 2^L$ bezahlen, wählt dieser analog zu [ASM08] einen unbenutzten Schlüssel $\kappa_{L-\ell,j}$ aus der $(L-\ell)$-ten Zeile des Binär-Baums und berechnet die Seriennummer $S = \mathfrak{g}^{\kappa_{L-\ell,j}}$ sowie das Sicherheitstag $T = \mathsf{pk}_{\mathcal{K}} \mathfrak{g}_0^{R\kappa_{L-\ell,j}}$, wobei $\mathfrak{g}_0$ ein zufälliger Generator der Gruppe $\mathbb{G}_p$ und $R \in \mathbb{Z}_p^*$ ein Hashwert ist. Das Bezahlprotokoll entspricht nun hauptsächlich dem Signaturbeweisprotokoll

Prove^*-k des BBS+A-Signaturverfahrens, wobei analog zu [CG10] in zero-knowledge bewiesen wird, dass sich alle k Serienschlüssel im Akkumulator $V = u_0^{\phi(\alpha)}$ befinden und der Kunde eine BBS+A-Signatur auf die Nachricht $m = (\mathsf{sk}_{\mathcal{K}}, \phi(\mathsf{x}))$ kennt. Dazu muss der Händler wie in [CG10] alle nachfolgenden Schlüssel des Binär-Baums aus der Seriennummer S berechnen. Dies ist in folgender Abbildung 6.7 für $\ell = 2$ dargestellt:

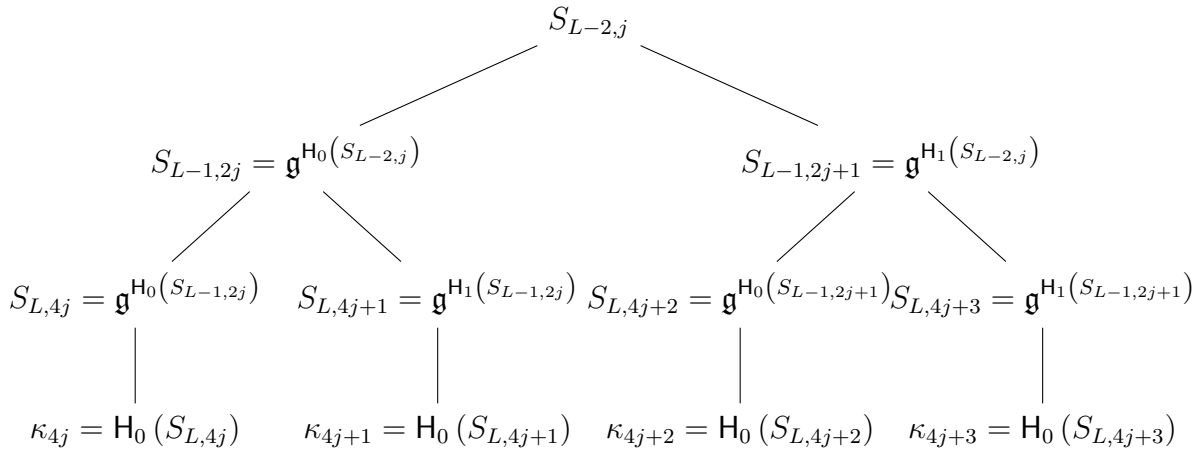

Abbildung 6.7: Berechnung der Seriennummern und Serienschlüssel für $\ell = 2$

Aus der Sicherheit des BBS+A-Signaturverfahrens folgt, dass ein Kunde keine Serienschlüssel zum Bezahlen verwenden kann, die nicht im Abhebeprotokoll von der Bank signiert wurden. Allerdings kann ein korrupter Kunde einfach mit einigen Serienschlüsseln mehrmals bezahlen, ohne dass die Händler dies bemerken. Indem die Bank eine Liste mit allen eingelösten Serienschlüsseln verwaltet, wird dieser Betrug allerdings nach der Tat aufgedeckt und durch die verwendeten Sicherheitstags die Identität des Double-Spenders analog zu [CG10] ermittelt.

6.5 Das teilbare elektronische Geldsystem TG-I

Im Folgenden werden die Algorithmen und Protokolle des teilbaren Geldsystems TG-I ausführlich dargestellt. Da das Bezahlprotokoll Spend analog zum Signaturbeweisprotokoll Prove der BBS+A-Signatur (siehe Unterabschnitt 3.2.2) abhängig vom Wert der Münze unterschiedlich ausgeführt werden muss, werden das Bezahlprotokoll Spend, bei dem der Kunde $\mathcal{K}$ mit einer Münze vom Wert $k = 2^\ell$ mit $1 \leq k < K$ bezahlen kann, sowie das Bezahlprotokoll Spend-K separat beschrieben.

Setup führt bei Eingabe des Sicherheitsparameters 1^λ und der Zahl $K \in \mathbb{N}$ den Algorithmus $\mathsf{Setup}_{\mathsf{Bil}}$ aus und erhält die Systemparameter $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, h) \leftarrow$

$\mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right)$, wobei $\mathbb{G}_1 = \langle u_0 \rangle, \mathbb{G}_2 = \langle h \rangle$ und $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ eine bilineare Abbildung ist. Dann wählt Setup eine kryptografische Hashfunktion H mit $\mathsf{H}\colon \{0,1\}^* \to \mathbb{Z}_p^*$, führt den Algorithmus $\mathsf{Setup}_{\mathsf{G}}$ aus und erhält die Systemparameter $\mathsf{sp}_{\mathsf{G}} = \left(p, \mathbb{G}_p', \mathfrak{g}\right) \leftarrow \mathsf{Setup}_{\mathsf{G}}\left(1^\lambda, p\right)$, wobei $\mathbb{G}_p = \langle \mathfrak{g} \rangle$ eine weitere zyklische Gruppe der Ordnung p ist. Anschließend wählt Setup zufällig drei Generatoren $g, g_0, g_1 \in_{\mathcal{R}} \mathbb{G}_1$, einen Generator $\mathfrak{g}_0 \in_{\mathcal{R}} \mathbb{G}_p$ sowie eine Zahl $\alpha \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Elemente $u_i = u_0^{\alpha^i}$ sowie $h_i = h^{\alpha^i}$ für $1 \leq i \leq K$. Schließlich definiert der Algorithmus die Hashfunktionen H_0 und H_1 mit $\mathsf{H}_0(m) := \mathsf{H}(m||0)$ sowie $\mathsf{H}_1(m) := \mathsf{H}(m||1)$ für alle $m \in \{0,1\}^*$. Da die Zahl α nicht weiter benötigt wird, wird α wieder gelöscht, um zu verhindern, dass sie kompromittiert wird. Die Ausgabe sind die Systemparameter

$$\mathsf{sp} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \mathbb{G}_p', \hat{e}, g, g_0, g_1, u_0, \ldots, u_K, h, h_1, \ldots, h_K, \mathfrak{g}, \mathfrak{g}_0, \mathsf{H}) \leftarrow \mathsf{Setup}\left(1^\lambda, K\right).$$

Unter der XDH-Annahme ist auch $\mathbb{G}_p = \mathbb{G}_1$ mit $\mathfrak{g} = g$ und $\mathfrak{g}_0 = g_0$ möglich.

BGen wählt bei Eingabe der Systemparameter sp zufällig einen geheimen Signaturschlüssel $x \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet den öffentlichen Schlüssel $X = h^x$. Die Ausgabe ist das Schlüsselpaar

$$(\mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{B}}) = (X, x) \leftarrow \mathsf{BGen}\left(\mathsf{sp}\right).$$

Weiter legt $\mathcal{B}$ drei Datenbanken $\mathbb{D}_{\mathcal{K}}, \mathbb{D}_W$ und $\mathbb{D}_C$ an, in denen die Informationen über Kontoinhaber, Abhebeinformationen bzw. die eingelösten Münzen gespeichert werden.

In der folgenden Tabelle 6.1 werden die benötigten Gruppen, deren öffentliche Generatoren, deren Verwendung und die zugrunde liegenden, kryptografischen Annahmen zusammengefasst.

Gruppen	**Generatoren**	**Verwendung**	**Annahmen**
$\mathbb{G}_1$	$g, g_0, g_1, u_0, \ldots, u_K$		
$\mathbb{G}_2$	$h, h_1, \ldots, h_K, X$		
$(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$		Nguyen-Akkumulator und BBS+A-Signatur	q-SDH und q-polyDH
$\mathbb{G}_p$	$\mathfrak{g}, \mathfrak{g}_0$	Seriennummern und Sicherheitstags	DDH

Tabelle 6.1: Öffentliche Parameter von TG-I

Account: Damit ein Kunde $\mathcal{K}$ dem System beitreten und ein Konto bei der Bank $\mathcal{B}$ eröffnen kann, führen $\mathcal{K}$ und $\mathcal{B}$ das Kontoeröffnungsprotokoll Account durch, welches in Abbildung 6.8 dargestellt ist. Zuvor muss sich der Kunde $\mathcal{K}$ gegenüber $\mathcal{B}$ ausweisen, beispielsweise mit einem Personalausweis.

Schritt 1: Der Kunde $\mathcal{K}$ erzeugt sein Schlüsselpaar $(\mathsf{pk}_\mathcal{K}, \mathsf{sk}_\mathcal{K}) = (I, u) \in \mathbb{G}_p \times \mathbb{Z}_p^*$. Dazu wählt $\mathcal{K}$ zufällig seinen geheimen Schlüssel $u \in_\mathcal{R} \mathbb{Z}_p^*$, berechnet seinen öffentlichen Schlüssel $I = \mathfrak{g}^u$ und sendet den öffentlichen Schlüssel $\mathsf{pk}_\mathcal{K}$ an die Bank. Anschließend speichert $\mathcal{K}$ sein Schlüsselpaar $(\mathsf{pk}_\mathcal{K}, \mathsf{sk}_\mathcal{K})$ ab.

Schritt 2: Die Bank $\mathcal{B}$ speichert den öffentlichen Schlüssel I zusammen mit persönlichen Daten von $\mathcal{K}$ in ihrer Datenbank $\mathbb{D}_\mathcal{K}$ ab und legt ein Konto für $\mathcal{K}$ an. Anhand von I kann $\mathcal{K}$ eindeutig identifiziert werden, weshalb I auch als *Kontonummer* oder *Identität* von $\mathcal{K}$ bezeichnet wird.

Falls die Kontonummer I bereits in der Datenbank $\mathbb{D}_\mathcal{K}$ gespeichert ist, muss dieses Protokoll erneut durchgeführt werden. Zudem muss der Kunde mit Kontonummer I ein neues Konto eröffnen, da die Bank nun die Kontonummer I sperren muss. Dies ist allerdings nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

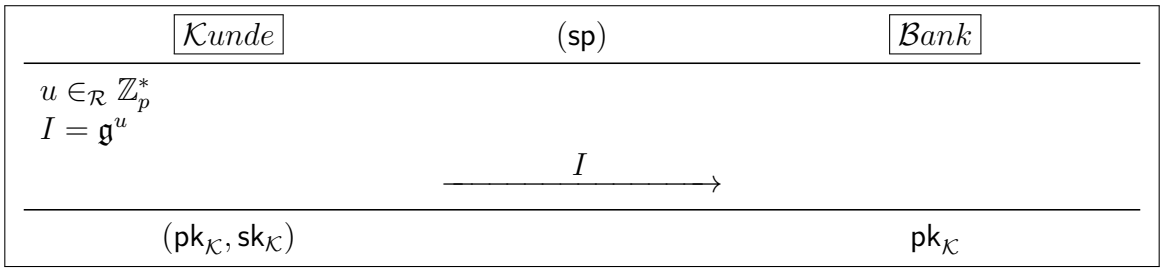

Abbildung 6.8: Kontoeröffnungsprotokoll Account von TG-I

Withdraw: Damit ein Kunde $\mathcal{K}$ eine Geldbörse mit dem Wert $K = 2^L$ bei der Bank $\mathcal{B}$ abheben kann, führen $\mathcal{K}$ und $\mathcal{B}$ das Abhebeprotokoll Withdraw durch, welches in Abbildung 6.9 dargestellt ist und im Wesentlichen dem Signaturerstellungsprotokoll Issue des BBS+A-Signaturverfahrens entspricht (siehe Unterabschnitt 3.2.2). Zuvor authentifizieren sich jedoch beide Parteien gegenseitig und gleichen den aktuellen Zeitpunkt ts ab.

Schritt 1: Der Kunde $\mathcal{K}$ wählt zufällig einen Master-Schlüssel $\kappa_{0,0} \in_\mathcal{R} \mathbb{Z}_p^*$, berechnet mit diesem alle Schlüssel $\mathbb{K}_0 = \{\kappa_{i,j} | 0 \leq i \leq L, 0 \leq j \leq 2^i - 1\}$ sowie die K Serienschlüssel $\kappa_j := \kappa_{L+1,j} = \mathsf{H}_0\left(\mathfrak{g}^{\kappa_{L,j}}\right)$ für $0 \leq j \leq K - 1$ wie in Abschnitt 6.4 beschrieben und speichert diese in einer Liste $\mathbb{K}_1 = \{\kappa_{L+1,j} | 0 \leq j \leq K - 1\}$ ab. Dann berechnet der Kunde $\mathcal{K}$ das Polynom

$\phi(\mathsf{x}) = \prod_{j=0}^{K-1}(\mathsf{x} + \kappa_{L+1,j})$. Mit einem Schlüssel $\kappa_{L-\ell,j_0} \in \mathbb{K}_0$ berechnet $\mathcal{K}$ im Bezahlprotokoll Spend die entsprechende Seriennummer $S_{L-\ell,j_0} = \mathfrak{g}^{\kappa_{L-\ell,j_0}} \in \mathbb{G}_p$ der Münze coin mit dem Wert $k = 2^\ell \leq K$, so dass ein Händler aus $S_{L-\ell,j_0}$ die dazu gehörenden 2^ℓ Serienschlüssel berechnen kann. (Die Berechnung des Binär-Baums kann auch bereits vor der Durchführung des Abhebeprotokolls durchgeführt werden. Somit können zu beliebigen Zeitpunkten bereits mehrere Binär-Bäume generiert und gespeichert werden.)

Das Withdraw-Protokoll entspricht im Wesentlichen dem Issue-Protokoll des BBS+A-Signaturverfahrens bzgl. der Nachricht $(u, \phi(\mathsf{x})) \in \mathbb{Z}_p^* \times \mathbb{Z}_p[\mathsf{x}]$. Somit wählt $\mathcal{K}$ zufällig eine Zahl $s' \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das perfekt verbergende Commitment $C = g_0^{s'} g_1^u u_0^{\phi(\alpha)}$. Dann berechnet der Kunde $\mathcal{K}$ den Hashwert $l = \mathsf{H}(C)$, die Zahl $\phi(l) \in \mathbb{Z}_p$, das Polynom $\phi_l(\mathsf{x}) = (\phi(\mathsf{x}) - \phi(l))(\mathsf{x} - l)^{-1}$ und den Zeugen $W_l = u_0^{\phi_l(\alpha)}$. Der Kunde $\mathcal{K}$ wählt zufällig eine Zahl $a_0 \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Element $A_0 = W_l u_0^{a_0}$, wodurch der Zeuge W_l perfekt verborgen ist. Anschließend sendet $\mathcal{K}$ die Elemente C, A_0 an $\mathcal{B}$ und erstellt die folgende Signature-of-Knowledge:

$$SoK_{\mathsf{W}}\Big[(a_0, s', u, \phi(l)) : I = \mathfrak{g}^u \wedge \hat{e}(C, h)\,\hat{e}\left(A_0, h_1 h^{-l}\right)^{-1} = \hat{e}\left(g_0^{s'} g_1^u u_0^{\phi(l)+a_0 l} u_1^{-a_0}, h\right)\Big](\mathsf{ts}||K).$$

Durch SoK_{W} beweist $\mathcal{K}$ in zero-knowledge, dass das Commitment C korrekt berechnet wurde und der Exponent von g_1 der private Schlüssel u des Kunden $\mathcal{K}$ ist.

Schritt 2: Die Bank $\mathcal{B}$ berechnet den Hashwert $l = \mathsf{H}(C)$, verifiziert SoK_{W}, wählt zufällig zwei Zahlen $e, s'' \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Element $\Sigma = \left(g g_0^{s''} C\right)^{\frac{1}{x+e}}$. Dann sendet $\mathcal{B}$ das Tripel (Σ, e, s'') an $\mathcal{K}$ und bucht dem Kunden $\mathcal{K}$ den Betrag K vom Konto mit der Kontonummer I ab. Anschließend speichert $\mathcal{B}$ die Abhebeinformationen $\mathbb{T} = (I, C, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K)$ in ihrer Datenbank $\mathbb{D}_W$ ab.

Schritt 3: Der Kunde $\mathcal{K}$ berechnet $s = s' + s'' \mod p$, verifiziert die Signatur durch Überprüfung der Gleichung $\hat{e}(\Sigma, X h^e) \stackrel{?}{=} \hat{e}\left(g g_0^{s''} C, h\right)$ und speichert die Geldbörse $\mathbb{W} = (\Sigma, e, s, \mathsf{info})$ ab, wobei info Informationen über die gültigen bzw. bereits verwendeten Serienschlüssel enthält. Nachstehend werden verschiedene Möglichkeiten angegeben.

- Es werden die Listen $\mathbb{K}_0$ sowie $\mathbb{K}_1$ gespeichert. Alle Vorgänger und Nachfolger eines bereits verwendeten Schlüssels werden in der Liste $\mathbb{K}_0$ gelöscht bzw. gleich 0 gesetzt. Somit besteht info aus $3K - 1$ Zahlen in $\mathbb{Z}_p^*$ und die Komplexität der Münze beträgt $O(\lambda \cdot K)$.

- Alternativ dazu kann der Kunde anstelle der Schlüssel $k_{i,j} \in \mathbb{Z}_p^*$ für $0 \leq i \leq L$ und $1 \leq j \leq 2^i - 1$ auch die entsprechenden Seriennummern $S_{i,j} = \mathfrak{g}^{\kappa_{i,j}} \in \mathbb{G}_p$ zusammen mit dem Master-Schlüssel $\kappa_{0,0} \in \mathbb{Z}_p^*$ in der Liste $\mathbb{K}_0$ abspeichern. In diesem Fall muss der Kunde die entsprechende Seriennummer $S_{i,j}$ nicht erneut während des Bezahlprotokolls berechnen, sondern kann mit Hilfe des Vorgängers von $S_{i,j}$ den zugehörigen Schlüssel $\kappa_{i,j}$ durch die entsprechende Hashfunktion H_0 bzw. H_1 erzeugen.
- Es werden nur der Master-Schlüssel $\kappa_{0,0}$ sowie ein K-Bit String 1^K gespeichert, der die K Serienschlüssel der $(L+1)$-ten Zeile des Binär-Baums repräsentiert. Bereits ausgegebene Serienschlüssel werden dann in der entsprechenden Position mit 0 markiert. Im Bezahlprotokoll muss der Kunde dann allerdings den benötigten Schlüssel $\kappa_{L-\ell,j_0}$ sowie alle Serienschlüssel, die kein Nachfolger von $\kappa_{L-\ell,j_0}$ sind, mit Hilfe von $\kappa_{0,0}$ berechnen. Somit besteht info aus einer Zahl in $\mathbb{Z}_p^*$ sowie K Bits und die Komplexität der Münze beträgt $O(\lambda + K)$.

$\boxed{\mathcal{K}unde}$ $(\mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{K})$	$(\mathsf{sp}, K, \mathsf{ts})$	$\boxed{\mathcal{B}ank}$ $(\mathsf{sk}_\mathcal{B}, \mathsf{pk}_\mathcal{K})$
$\kappa_{0,0} \in_\mathcal{R} \mathbb{Z}_p^*, s', a_0 \in_\mathcal{R} \mathbb{Z}_p$ Generiere $\mathbb{K}_0, \mathbb{K}_1$ $\phi(\mathsf{x}) = \prod_{j=0}^{K-1} (\mathsf{x} + \kappa_{L+1,j})$ $C = g_0^{s'} g_1^u u_0^{\phi(\alpha)}$ $l = \mathsf{H}(C)$ $A_0 = W_l u_0^{a_0}$		
	$\xrightarrow{\quad C, A_0 \quad}$	
$SoK_\mathsf{W}\Big[(a_0, s', u, \phi(l)) : I = \mathfrak{g}^u \wedge$ $\hat{e}(C, h)\, \hat{e}\left(A_0, h_1 h^{-l}\right)^{-1} = \hat{e}\left(g_0^{s'} g_1^u u_0^{\phi(l)+a_0 l} u_1^{-a_0}, h\right)\Big] (\mathsf{ts}\|K)$		
		$l = \mathsf{H}(C)$ Verifiziere SoK_W $e, s'' \in_\mathcal{R} \mathbb{Z}_p$ $\Sigma = \left(g g_0^{s''} C\right)^{\frac{1}{x+e}}$
	$\xleftarrow{\quad \Sigma, e, s'' \quad}$	
$s = s' + s'' \mod p$ $\hat{e}(\Sigma, X h^e) \stackrel{?}{=} \hat{e}\left(g g_0^{s''} C, h\right)$		
$\mathbb{W} = (\Sigma, e, s, \mathsf{info})$ oder $\perp$		view oder $\perp$

Abbildung 6.9: Abhebeprotokoll $\mathsf{Withdraw}$ von TG-I

Spend: Damit ein Kunde $\mathcal{K}$, der im Besitz der Geldbörse $\mathbb{W} = (\Sigma, e, s, \mathsf{info})$ ist, bei einem Händler $\mathcal{H}$ mit einer elektronischen Münze coin mit dem Wert $k = 2^\ell < 2^L = K$ bezahlen kann, führen $\mathcal{K}$ und $\mathcal{H}$ das Bezahlprotokoll Spend durch, welches in Abbildung 6.10 dargestellt ist und im Wesentlichen dem Prove*-k-Protokoll des BBS+A-Signaturverfahrens entspricht. Zuvor authentifiziert sich der Händler $\mathcal{H}$ gegenüber dem Kunden $\mathcal{K}$ und beide Parteien gleichen den Zeitpunkt ts ab.

Schritt 1: Der Händler $\mathcal{H}$ und der Kunde $\mathcal{K}$ berechnen beide den Hashwert $R = \mathsf{H}(\mathsf{pk}_\mathcal{H}||\mathsf{ts}||k)$, wobei $\mathsf{pk}_\mathcal{H}$ der öffentliche Schlüssel des Händlers und ts der aktuelle Zeitpunkt ist.

Schritt 2: Der Kunde $\mathcal{K}$ wählt den unbenutzten Schlüssel $\kappa_{L-\ell,j_0} := \kappa \in \mathbb{K}_0$ der $(L-\ell)$-ten Zeile mit dem kleinsten Wert j_0 und berechnet die Seriennummer $S = \mathfrak{g}^\kappa$ sowie das Sicherheitstag $T = \mathfrak{g}^u \mathfrak{g}_0^{R\kappa}$. Danach wählt $\mathcal{K}$, wie beim Signaturbeweisprotokoll Prove*-k der BBS+A-Signatur (siehe Unterabschnitt 3.2.2), zufällig zwei Zahlen $b_1, b_2 \in_\mathcal{R} \mathbb{Z}_p$ und berechnet die Elemente $B_1 = g_0^{b_1} g_1^{b_2}$ und $B_2 = \Sigma g_1^{b_1}$ sowie den Hashwert $l = \mathsf{H}(S||T||B_1||B_2)$. Da aus der Seriennummer S die Serienschlüssel $\kappa_{L+1,kj_0}, \ldots, \kappa_{L+1,kj_0+k-1}$ berechnet werden, sei $I = \{kj_0, \ldots, kj_0 + k - 1\}$ und $\phi_I(\mathsf{x}) = \prod_{j=0, j\notin I}^{K-1} (\mathsf{x} + \kappa_{L+1,j}) \in \mathbb{Z}_p[\mathsf{x}]$ das Polynom mit $\phi(\mathsf{x}) = \phi_I(\mathsf{x}) \prod_{j\in I} (\mathsf{x} + \kappa_{L+1,j})$. Dann berechnet $\mathcal{K}$ die Zahl $\phi_I(l) \in \mathbb{Z}_p$, das Polynom $\phi_{I,l}(\mathsf{x}) = (\phi_I(\mathsf{x}) - \phi_I(l))(\mathsf{x} - l)^{-1}$ und den Zeugen $W_{I,l} = u_0^{\phi_{I,l}(\alpha)}$. Weiter wählt $\mathcal{K}$ zufällig eine Zahl $a_1 \in_\mathcal{R} \mathbb{Z}_p$ und berechnet das Element $A_1 = W_{I,l} u_0^{a_1}$, um den Zeugen $W_{I,l} \in \mathbb{G}_1$ perfekt zu verbergen. Anschließend sendet $\mathcal{K}$ die Elemente S, T, A_1, B_1, B_2 an $\mathcal{H}$ und erstellt die Signature-of-Knowledge SoK, wobei $\beta_1 = b_1 e, \beta_2 = b_2 e, u_I = u_0^{\prod_{j\in I}(\alpha+\kappa_{L+1,j})}, u_{l,I} = u_0^{(\alpha-l)\prod_{j\in I}(\alpha+\kappa_{L+1,j})}$ und $h_{l,I} = h^{(\alpha-l)\prod_{j\in I}(\alpha+\kappa_{L+1,j})}$ ist:

$$SoK\Bigg[(a_1, b_1, b_2, \beta_1, \beta_2, e, s, \kappa, u, \phi_I(l)) : B_1 = g_0^{b_1} g_1^{b_2} \ \wedge\ 1 = B_1^e g_0^{-\beta_1} g_1^{-\beta_2} \ \wedge$$
$$S = \mathfrak{g}^\kappa \ \wedge\ T = \mathfrak{g}^u \mathfrak{g}_0^{R\kappa} \ \wedge$$
$$\frac{\hat{e}(B_2, X)}{\hat{e}(g, h)\,\hat{e}(A_1, h_{l,I})} = \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(g_0^s g_1^{u+\beta_1} u_{l,I}^{-a_1} u_I^{\phi_I(l)} B_2^{-e}, h\right)\Bigg](R).$$

Durch SoK beweist $\mathcal{K}$ in zero-knowledge, dass er im Besitz einer BBS+A-Signatur (Σ, e, s) auf die Nachricht $(u, \phi(\mathsf{x}))$ ist und die Seriennummer S sowie das Sicherheitstag T korrekt berechnet wurden. Schließlich aktualisiert $\mathcal{K}$ seine Geldbörse zu $\mathbb{W}' = (\Sigma, e, s, \mathsf{info}')$.

Schritt 3: Der Händler $\mathcal{H}$ berechnet aus der Seriennummer S alle k Serienschlüssel $\kappa_{L+1,kj_0}, \ldots, \kappa_{L+1,kj_0+k-1}$ (dies sind also alle $\kappa_{L+1,j}$ mit $j \in I$). Die k Serienschlüssel berechnet $\mathcal{H}$ wie in Abschnitt 6.4 beschrieben. Dann berechnet $\mathcal{H}$ den Hashwert $l = \mathsf{H}(S||T||B_1||B_2)$, verifiziert SoK und speichert die Münze $\mathsf{coin} = (k, S, T, R, \Phi = (A_1, B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_\mathcal{H}))$ ab.

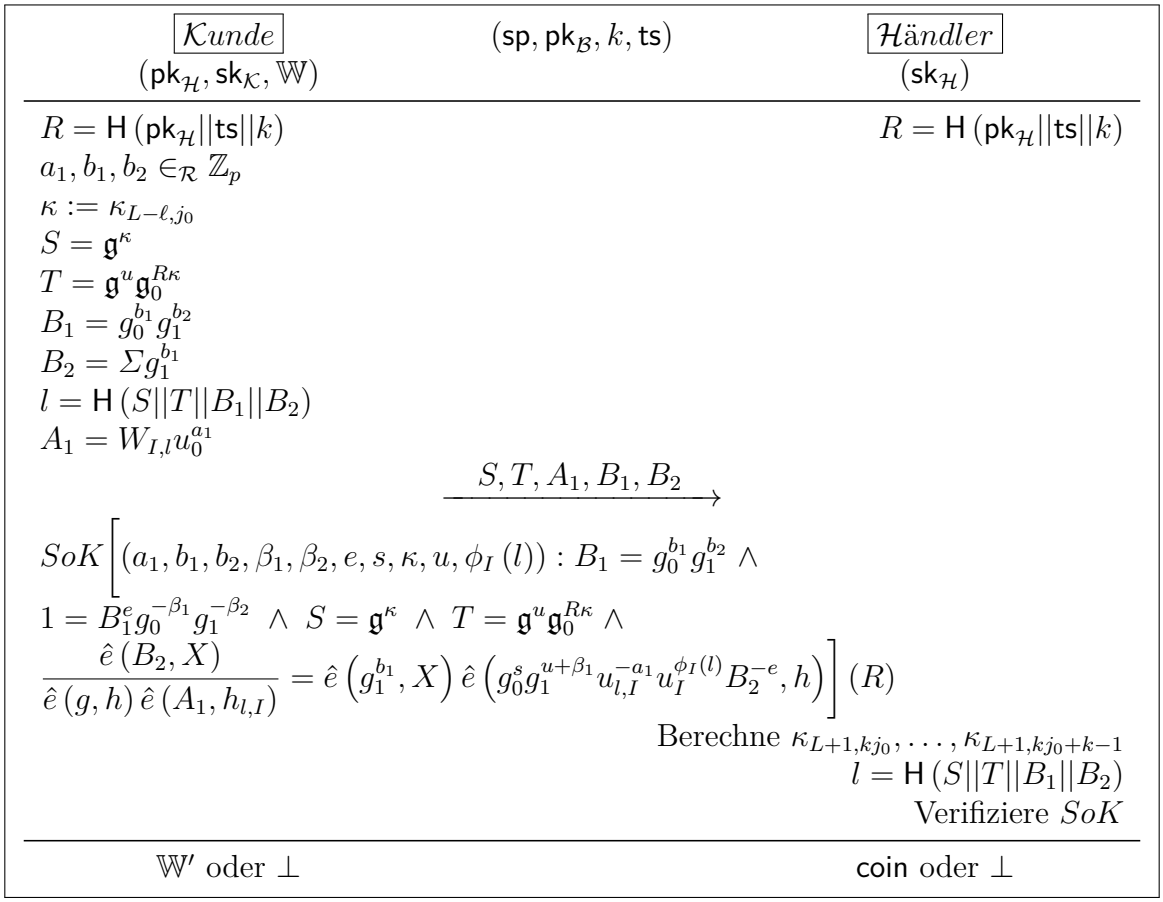

Abbildung 6.10: Bezahlprotokoll Spend von TG-I

Spend-K: Damit ein Kunde $\mathcal{K}$, der im Besitz der Geldbörse $\mathbb{W} = (\Sigma, e, s, \mathsf{info})$ ist, bei einem Händler $\mathcal{H}$ mit einer elektronischen Münze coin mit dem Wert $K = 2^L$ bezahlen kann, führen $\mathcal{K}$ und $\mathcal{H}$ das Bezahlprotokoll Spend-K durch, welches in Abbildung 6.11 dargestellt ist und im Wesentlichem dem Prove-K-Protokoll des BBS+A-Signaturverfahrens entspricht. Zuvor authentifiziert sich der Händler $\mathcal{H}$ gegenüber dem Kunden $\mathcal{K}$ und beide Parteien gleichen den Zeitpunkt ts ab.

Schritt 1: Der Händler $\mathcal{H}$ und der Kunde $\mathcal{K}$ berechnen beide den Hashwert $R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}}||\mathsf{ts}||K)$, wobei $\mathsf{pk}_{\mathcal{H}}$ der öffentliche Schlüssel des Händlers und ts der aktuelle Zeitpunkt ist.

Schritt 2: Der Kunde $\mathcal{K}$ verwendet den Master-Schlüssel $\kappa_{0,0} := \kappa \in \mathbb{K}_0$ und berechnet die Seriennummer $S = \mathfrak{g}^{\kappa}$ sowie das Sicherheitstag $T = \mathfrak{g}^{u}\mathfrak{g}_0^{R\kappa}$. Danach wählt $\mathcal{K}$, wie beim Signaturbeweisprotokoll Prove-K der BBS+A-Signatur (siehe Unterabschnitt 3.2.2), zufällig zwei Zahlen $b_1, b_2 \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Elemente $B_1 = g_0^{b_1} g_1^{b_2}$ und $B_2 = \Sigma g_1^{b_1}$. Anschließend sendet $\mathcal{K}$ die Elemente S, T, B_1, B_2 an $\mathcal{H}$ und erstellt die Signature-of-Knowledge SoK,

wobei $\beta_1 = b_1 e, \beta_2 = b_2 e$ und $V = u_0^{\prod_{j=0}^{K-1}(\alpha+\kappa_{L+1,j})}$ ist:

$$SoK\Bigg[(b_1, b_2, \beta_1, \beta_2, e, s, \kappa, u) : B_1 = g_0^{b_1} g_1^{b_2} \ \wedge \ 1 = B_1^e g_0^{-\beta_1} g_1^{-\beta_2} \ \wedge$$
$$S = \mathfrak{g}^\kappa \ \wedge \ T = \mathfrak{g}^u \mathfrak{g}_0^{R\kappa} \ \wedge \ \frac{\hat{e}(B_2, X)}{\hat{e}(gV, h)} = \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(g_0^s g_1^{u+\beta_1} B_2^{-e}, h\right)\Bigg](R)\,.$$

Schließlich aktualisiert $\mathcal{K}$ seine Geldbörse zu $\mathbb{W}' = (\Sigma, e, s, \mathsf{info}')$.

Schritt 3: Der Händler $\mathcal{H}$ berechnet aus der Seriennummer S alle K Seriennschlüssel $\kappa_{L+1,0}, \ldots, \kappa_{L+1,K-1}$ sowie den Akkumulator V, verifiziert SoK und speichert die Münze $\mathsf{coin} = (K, S, T, R, \Phi = (B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_\mathcal{H}))$ ab.

$\boxed{\mathcal{K}unde}$ $(\mathsf{pk}_\mathcal{H}, \mathsf{sk}_\mathcal{K}, \mathbb{W})$	$(\mathsf{sp}, \mathsf{pk}_\mathcal{B}, K, \mathsf{ts})$	$\boxed{\mathcal{H}ändler}$ $(\mathsf{sk}_\mathcal{H})$
$R = \mathsf{H}(\mathsf{pk}_\mathcal{H} \| \mathsf{ts} \| K)$		$R = \mathsf{H}(\mathsf{pk}_\mathcal{H} \| \mathsf{ts} \| K)$
$b_1, b_2 \in_\mathcal{R} \mathbb{Z}_p$		
$\kappa := \kappa_{0,0}$		
$S = \mathfrak{g}^\kappa$		
$T = \mathfrak{g}^u \mathfrak{g}_0^{R\kappa}$		
$B_1 = g_0^{b_1} g_1^{b_2}$		
$B_2 = \Sigma g_1^{b_1}$		
	$\xrightarrow{S, T, B_1, B_2}$	
$SoK\Big[(b_1, b_2, \beta_1, \beta_2, e, s, \kappa, u) : B_1 = g_0^{b_1} g_1^{b_2} \ \wedge \ 1 = B_1^e g_0^{-\beta_1} g_1^{-\beta_2} \ \wedge$		
$S = \mathfrak{g}^\kappa \ \wedge \ T = \mathfrak{g}^u \mathfrak{g}_0^{R\kappa} \ \wedge \ \frac{\hat{e}(B_2, X)}{\hat{e}(gV, h)} = \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(g_0^s g_1^{u+\beta_1} B_2^{-e}, h\right)\Big](R)$		
		Berechne $\kappa_{L+1,0}, \ldots, \kappa_{L+1,K-1}$
		Verifiziere SoK
$\mathbb{W}'$ oder $\perp$		coin oder $\perp$

Abbildung 6.11: Bezahlprotokoll Spend-K von TG-I

Deposit: Damit ein Händler $\mathcal{H}$ eine Münze coin bei der Bank $\mathcal{B}$ einlösen kann, führen beide Parteien das Einlöseprotokoll Deposit durch (Abbildung 6.12), nachdem sich beide Parteien gegenseitig authentifiziert haben.

Schritt 1: Der Händler $\mathcal{H}$ sendet die Münze $\mathsf{coin} = (k, S, T, R, \Phi = (A_1, B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_\mathcal{H}))$ an die Bank $\mathcal{B}$. (Die Kontonummer $\mathsf{pk}_\mathcal{H}$ muss dabei nicht erneut gesendet werden, da $\mathcal{B}$ diese bereits kennt.)

Schritt 2: Die Bank $\mathcal{B}$ verifiziert die Münze coin. Dazu überprüft $\mathcal{B}$ die Gleichung $R \stackrel{?}{=} \mathsf{H}(\mathsf{pk}_{\mathcal{H}}||\mathsf{ts}||k)$ um sicherzustellen, dass mit der Münze coin tatsächlich beim Händler $\mathcal{H}$ bezahlt wurde. Dann berechnet $\mathcal{B}$ aus der Seriennummer S die k Serienschlüssel $\kappa_{L+1,kj_0}, \ldots, \kappa_{L+1,kj_0+k-1}$ sowie den Hashwert $l = \mathsf{H}(S||T||B_1||B_2)$ und verifiziert SoK.

- Falls $R \neq \mathsf{H}(\mathsf{pk}_{\mathcal{H}}||\mathsf{ts}||k)$ ist oder SoK abgelehnt wird, ist die Münze ungültig und $\mathcal{B}$ sendet eine Fehlermeldung $\perp$ an $\mathcal{H}$.
- Wenn die Münze coin akzeptiert wird, überprüft die Bank $\mathcal{B}$, ob das Paar $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts})$ bereits in der Datenbank $\mathbb{D}_C$ gespeichert ist. Falls bereits eine Münze $\mathsf{coin}' = (\cdots, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}})$ mit $\mathsf{coin}' \in \mathbb{D}_C$ existiert, sendet $\mathcal{B}$ eine Fehlermeldung $\perp$ an $\mathcal{H}$. Wenn $(\cdots, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}) \notin \mathbb{D}_C$ ist, speichert $\mathcal{B}$ die Münze coin mit den dazu gehörenden k Serienschlüsseln in ihrer Datenbank $\mathbb{D}_C$ ab und schreibt dem Händler $\mathcal{H}$ das Geld gut.
- Anschließend überprüft die Bank $\mathcal{B}$, ob bereits einer der Serienschlüssel $\kappa_{L+1,kj_0}, \ldots, \kappa_{L+1,kj_0+k-1}$ in der Liste $\mathbb{D}_C$ gespeichert wurde. Falls ja, führt $\mathcal{B}$ den Algorithmus $\mathsf{Identify}$ aus, um den Double-Spender zu identifizieren.

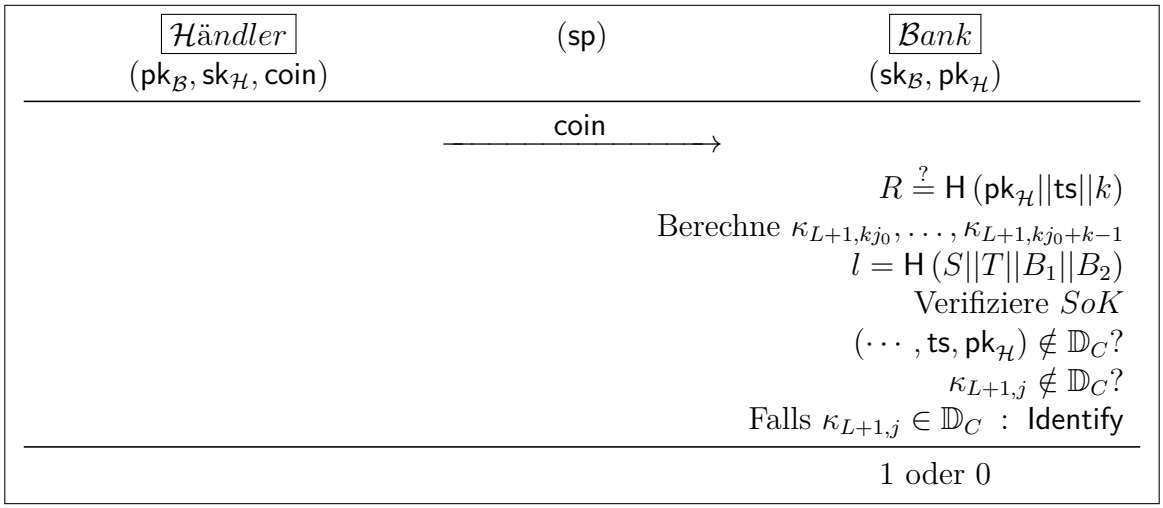

Abbildung 6.12: Einlöseprotokoll $\mathsf{Deposit}$ von TG-I

Identify überprüft bei Eingabe der Systemparameter sp, des öffentlichen Schlüssels $\mathsf{pk}_{\mathcal{B}}$ und zweier Münzen $\mathsf{coin} = (k, S, T, R, \Phi)$ sowie $\mathsf{coin}' = (k', S', T', R', \Phi')$, ob $R \stackrel{?}{=} R'$ ist. Falls $R = R'$ gilt, ist die Ausgabe das Symbol $\perp$. Für $R \neq R'$ wird die Kontonummer des Double-Spenders folgendermaßen berechnet:

1. Falls $S = S'$ ist, wurde derselbe Schlüssel κ in T und T' verwendet. Daher

wird die Kontonummer analog zu Abschnitt 5.1 berechnet:

$$I = \left(\frac{T'^R}{T^{R'}}\right)^{(R-R')^{-1}}.$$

2. Falls $S \neq S'$, sei o.B.d.A. $k > k'$. Dann kann aus der Seriennummer S der zu S' gehörende Schlüssel κ' mit $S' = \mathfrak{g}^{\kappa'}$ sowie $T' = I\mathfrak{g}_0^{R'\kappa'}$ wie in Abschnitt 6.4 berechnet werden. Dann wird die Kontonummer folgendermaßen berechnet (vgl. auch [CG10]):

$$I = \frac{T'}{\mathfrak{g}_0^{R'\kappa'}}.$$

Die Ausgabe ist entweder ein öffentlicher Schlüssel $I \in \mathbb{D}_{\mathcal{K}}$ mit einem Beweis $\Pi_G = (\mathsf{coin}, \mathsf{coin}')$ oder das Symbol $\perp$.

VerifyGuilt verifiziert bei Eingabe der Systemparameter sp, des öffentlichen Schlüssels $\mathsf{pk}_{\mathcal{B}}$, eines öffentlichen Schlüssels $\mathsf{pk}_{\mathcal{K}}$ und eines Beweises Π_G, der zwei Münzen coin und coin' enthält, die beiden Münzen und führt den Algorithmus $\mathsf{Identify}$ aus, um eine Kontonummer I zu berechnen. Somit kann jeder verifizieren, ob $\mathsf{pk}_{\mathcal{K}} \stackrel{?}{=} I$ ist und der Kunde mit der Kontonummer I einen Serienschlüssel mehrfach ausgegeben hat.

VerifyWithdraw verifiziert bei Eingabe der Systemparameter sp, des öffentlichen Schlüssels $\mathsf{pk}_{\mathcal{B}}$, eines öffentlichen Schlüssels $\mathsf{pk}_{\mathcal{K}}$ und einer Protokollansicht $\mathsf{view} = (\mathbb{T} = (I, C, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s''))$ die Signature-of-Knowledge SoK_{W} und die Signatur durch Überprüfung der Gleichung $\hat{e}\,(\Sigma, Xh^e) \stackrel{?}{=} \hat{e}\left(gg_0^{s''}C, h\right)$. Somit kann eine dritte Partei verifizieren, ob der Kunde mit der Kontonummer I dieses Abhebeprotokoll durchgeführt hat. Das Tripel (Σ, e, s'') wird dabei von der Bank gemäß dem Abhebeprotokoll $\mathsf{Withdraw}$ erzeugt.

6.5.1 Sicherheitsanalyse von TG-I

Satz 6.5.1. *Das teilbare elektronische Geldsystem TG-I ist im Random-Oracle-Modell ein sicheres elektronisches Geldsystem, falls die q-SDH-Annahme und die q-polyDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gelten und die DDH-Annahme für* $\mathsf{Setup}_{\mathsf{G}}$ *gilt.*

Bevor der Satz bewiesen wird, halten wir zunächst die folgende Bemerkung fest, die für sämtliche Sicherheitseigenschaften dieses Geldsystems und analog aller weiteren Geldsysteme gilt.

Bemerkung 6.5.1. *Wie bei den vorherigen Beweisen auch, konstruieren wir stets einen polynomiellen Angreifer $\mathcal{B}$, der den polynomiellen Angreifer $\mathcal{A}$ dazu verwendet, ein bestimmtes kryptografisches Problem zu lösen. Da der Angreifer $\mathcal{B}$ oft die Zahlen aus einer*

Signature-of-Knowledge extrahieren muss und dies nur kann, wenn die Signature-of-Knowledge von $\mathcal{A}$ erstellt wurde, halten wir vorerst fest:

Falls eine Münze $\mathsf{coin} = (k, S, T, R, \Phi = (A_1, B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$ *für* $1 \leq k \leq K$ *(wobei* A_1 *auch ein leerer String sein kann) nicht von* $\mathcal{B}$ *zum Bezahlen verwendet und somit nicht von* $\mathcal{B}$ *erstellt wurde, wurde die Signature-of-Knowledge* SoK *nur mit vernachlässigbarer Wahrscheinlichkeit von* $\mathcal{B}$ *(und somit nicht von* $\mathcal{A}$*) erstellt.*

Beweis. Denn angenommen, $\mathcal{B}$ hätte eine andere Münze $\mathsf{coin}' = (k', S', T', R', \Phi' = (A'_1, B'_1, B'_2, SoK', \mathsf{ts}', \mathsf{pk}'_{\mathcal{H}}))$ für $1 \leq k' \leq K$ (wobei A'_1 auch ein leerer String sein kann) mit

$$(k, S, T, R, A_1, B_1, B_2, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}) \neq (k', S', T', R', A'_1, B'_1, B'_2, \mathsf{ts}', \mathsf{pk}'_{\mathcal{H}})$$
$$\text{und}$$
$$SoK = SoK'$$

erstellt. Dann gilt insbesondere für die beiden Challenges

$$\begin{aligned} c &= \mathsf{H}(S||T||A_1||B_1||B_2||\cdots||\mathsf{H}(\mathsf{pk}_{\mathcal{H}}||\mathsf{ts}||k)) \\ &= \mathsf{H}(S'||T'||A'_1||B'_1||B'_2||\cdots||\mathsf{H}(\mathsf{pk}'_{\mathcal{H}}||\mathsf{ts}'||k')) = c'. \end{aligned}$$

Dies ist jedoch im Random-Oracle-Modell nur mit vernachlässigbarer Wahrscheinlichkeit möglich. □

Dies gilt analog für die Signatures-of-Knowledge in den anderen Protokollen.

Des Weiteren sei vermerkt, dass $\mathcal{B}$ *abbricht, falls die Zahlen aus einer Signature-of-Knowledge nicht extrahiert werden können. Dies ist allerdings wegen der Proof-of-Knowledge-Eigenschaft nur mit vernachlässigbarer Wahrscheinlichkeit möglich. Zur Vereinfachung der Sicherheitsbeweise wird dies nicht immer explizit an jeder Stelle angegeben.*

Der Angreifer $\mathcal{B}$ *verwaltet zur Simulation der Orakel-Anfragen mehrere Listen. Wird bspw. ein Eintrag* (A, B, C, D) *in einer Liste* $\mathbb{L}$ *gespeichert, gilt* $(A, B, C, D) \in \mathbb{L}$*. Die Schreibweise* $A \in \mathbb{L}$ *ist dann eine verkürzte Schreibweise für* $\exists A$ *mit* $(A, \cdot, \cdot, \cdot) \in \mathbb{L}$*, die an einigen Stellen verwendet wird.*

Beweis. Wir beweisen Unfälschbarkeit, Anonymität, Double-Spending-Beschuldigung und Abhebe-Beschuldigung.

Unfälschbarkeit: Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Unfälschbarkeit des Systems bricht. Seien $q_{W,1}$ die Anzahl der Anfragen an das Orakel $\mathcal{O}^{\mathsf{W}}_{\mathcal{B}}$, $q_{W,2}$ die Anzahl der Anfragen an das Orakel $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$ und q_V die Anzahl der Anfragen an das Orakel $\mathcal{O}^{\mathsf{VW}}_{\mathcal{B}}$. Sei k_1 der Wert aller Münzen, die $\mathcal{A}$ in den $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$-Anfragen erhält. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das q-SDH-Problem für $\mathsf{Setup}_{\mathsf{Bil}}$ mit $q \leq q_{W,1} + q_{W,2} + q_V$ zu lösen (bzw. $q \leq q_{W,1} + q_{W,2} + q_V + 1$ für ein Typ-1- und Typ-2-Pairing, vgl. Unterabschnitt 3.2.3).

Am Ende hat $\mathcal{A}$ mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ durch die Orakel $\mathcal{O}^{\mathsf{S}}_{\mathcal{H}}$ und $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$ Münzen im Wert von $\$^* = \$ + k^*$ für $k^* \geq 1$ ausgegeben oder eingelöst, ohne dass der $\mathsf{Identify}$-Algorithmus eine Kontonummer $\mathsf{pk} \in \mathcal{U}_{\mathcal{A}}$ ausgibt. Da der Angreifer $\mathcal{A}$ durch die $\mathcal{O}^{\mathsf{W}}_{\mathcal{B}}$-Anfragen $q_{W,1} \cdot K$ sowie durch die $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$-Anfragen k_1 Serienschlüssel erhält, ist $\$ = q_{W,1} \cdot K + k_1$.

Unabhängig davon, wie $\mathcal{B}$ die Orakel-Anfragen simuliert, werden zunächst die verschiedenen Fälle aufgelistet, in denen der Angreifer $\mathcal{A}$ das Spiel gewinnt.

Zunächst wird gezeigt, dass $\mathcal{A}$ nur mit vernachlässigbarer Wahrscheinlichkeit mit einer Münze $\mathsf{coin}' = (k', S, T, R, A_1, B_1, B_2, SoK, \mathsf{ts}', \mathsf{pk}'_{\mathcal{H}})$ bezahlen kann, wobei $\mathsf{coin} = (k, S, T, R, A_1, B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}})$ mit $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(\mathsf{pk}'_{\mathcal{H}}, \mathsf{ts}', k') \neq (\mathsf{pk}_{\mathcal{H}}, \mathsf{ts}, k)$ gilt. Somit kann $\mathcal{A}$ nur mit vernachlässigbarer Wahrscheinlichkeit eine Münze zum Bezahlen verwenden, die von $\mathcal{B}$ erstellt wurde.

Fall A: Falls $\mathcal{A}$ mit einer aus den $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$-Anfragen bzgl. $\mathsf{pk}_{\mathcal{H}} \in \mathcal{U}_{\mathcal{A}}, \mathsf{ts}, k$ erhaltenen Münze coin mit $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}}$ durch eine $\mathcal{O}^{\mathsf{S}}_{\mathcal{H}}$-Anfrage bzgl. $\mathsf{pk}'_{\mathcal{H}} \in \mathcal{U}_H, \mathsf{ts}', k'$ bezahlen will, muss $R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}} || \mathsf{ts} || k) = \mathsf{H}(\mathsf{pk}'_{\mathcal{H}} || \mathsf{ts}' || k')$ mit $\mathsf{pk}_{\mathcal{H}} \neq \mathsf{pk}'_{\mathcal{H}}$ gelten. Dies ist jedoch nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Fall B: Falls $\mathcal{A}$ mit einer aus den $\mathcal{O}^{\mathsf{S}}_{\mathcal{K},\mathcal{H}}$-Anfragen bzgl. $\mathsf{pk}_{\mathcal{H}} \in \mathcal{U}_H, \mathsf{ts}, k$ erhaltenen Münze coin mit $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_H$ durch eine $\mathcal{O}^{\mathsf{S}}_{\mathcal{H}}$-Anfrage bzgl. $\mathsf{pk}'_{\mathcal{H}} \in \mathcal{U}_H, \mathsf{ts}', k'$ bezahlen will, muss $R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}} || \mathsf{ts} || k) = \mathsf{H}(\mathsf{pk}'_{\mathcal{H}} || \mathsf{ts}' || k')$ mit $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts}) \neq (\mathsf{pk}'_{\mathcal{H}}, \mathsf{ts}')$ gelten, da der Zeitpunkt ts' ansonsten ungültig wäre. Dies ist jedoch nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Fall C: Falls $\mathcal{A}$ mit einer aus den $\mathcal{O}^{\mathsf{D}}_{\mathcal{H},\mathcal{B}}$-Anfragen erhaltenen Münze coin mit $(\cdot, \mathsf{coin}) \in \mathbb{D}_H$ durch eine $\mathcal{O}^{\mathsf{S}}_{\mathcal{H}}$-Anfrage bezahlen will, wurde die Münze coin entweder bereits bei einer $\mathcal{O}^{\mathsf{S}}_{\mathcal{K},\mathcal{H}}$-Anfrage verwendet, was nach Fall B nur mit vernachlässigbarer Wahrscheinlichkeit möglich ist, oder bereits bei einer $\mathcal{O}^{\mathsf{S}}_{\mathcal{H}}$-Anfrage vom Angreifer $\mathcal{A}$ selbst erstellt.

Folglich wurde jede Münze coin (und insbesondere SoK), mit der der Angreifer $\mathcal{A}$ durch eine $\mathcal{O}^{\mathsf{S}}_{\mathcal{H}}$-Anfrage bezahlt, außer mit vernachlässigbarer Wahrscheinlichkeit, von $\mathcal{A}$ erstellt. Daher kann $\mathcal{B}$, wie unten bei den Simulationen beschrieben, bei jeder Münze, mit der der Angreifer $\mathcal{A}$ durch eine $\mathcal{O}^{\mathsf{S}}_{\mathcal{H}}$-Anfrage bezahlt, das Nachrichten-Signatur-Paar extrahieren.

Schließlich wird gezeigt, dass $\mathcal{A}$ nur mit vernachlässigbarer Wahrscheinlichkeit eine Münze $\mathsf{coin}' = (k', S, T, R, A_1, B_1, B_2, SoK, \mathsf{ts}', \mathsf{pk}'_{\mathcal{H}})$ einlösen kann, wobei $\mathsf{coin} = (k, S, T, R, A_1, B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}})$ mit $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(\mathsf{pk}'_{\mathcal{H}}, \mathsf{ts}', k') \neq (\mathsf{pk}_{\mathcal{H}}, \mathsf{ts}, k)$ gilt. Somit kann $\mathcal{A}$ nur mit vernachlässigbarer Wahrscheinlichkeit eine Münze einlösen, die von $\mathcal{B}$ erstellt wurde, außer die Münzen coin mit $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}}$.

Fall D: Falls $\mathcal{A}$ eine aus den $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$-Anfragen bzgl. $\mathsf{pk}_{\mathcal{H}} \in \mathcal{U}_{\mathcal{A}}, \mathsf{ts}, k$ erhaltene Münze coin mit $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}}$ durch eine $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$-Anfrage bzgl. $\mathsf{pk}'_{\mathcal{H}} \in \mathcal{U}_{\mathcal{A}}, \mathsf{ts}', k'$ mit

$(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts}, k) \neq (\mathsf{pk}'_{\mathcal{H}}, \mathsf{ts}', k')$ einlösen will, muss $R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}}||\mathsf{ts}||k) = \mathsf{H}(\mathsf{pk}'_{\mathcal{H}}||\mathsf{ts}'||k')$ gelten. Dies ist jedoch nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Fall E: Falls $\mathcal{A}$ eine aus den $\mathcal{O}^{\mathsf{S}}_{\mathcal{K},\mathcal{H}}$-Anfragen bzgl. $\mathsf{pk}_{\mathcal{H}} \in \mathcal{U}_H, \mathsf{ts}, k$ erhaltene Münze coin mit $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_H$ durch eine $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$-Anfrage bzgl. $\mathsf{pk}'_{\mathcal{H}} \in \mathcal{U}_{\mathcal{A}}, \mathsf{ts}', k'$ einlösen will, muss $R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}}||\ \mathsf{ts}||k) = \mathsf{H}(\mathsf{pk}'_{\mathcal{H}}||\mathsf{ts}'||k')$ mit $\mathsf{pk}_{\mathcal{H}} \neq \mathsf{pk}'_{\mathcal{H}}$ gelten. Dies ist jedoch nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Fall F: Falls $\mathcal{A}$ eine aus den $\mathcal{O}^{\mathsf{D}}_{\mathcal{H},\mathcal{B}}$-Anfragen erhaltenen Münze coin mit $(\cdot, \mathsf{coin}) \in \mathbb{D}_H$ durch eine $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$-Anfrage bzgl. $\mathsf{pk}'_{\mathcal{H}} \in \mathcal{U}_{\mathcal{A}}, \mathsf{ts}', k'$ einlösen will, wurde die Münze coin entweder bereits bei einer $\mathcal{O}^{\mathsf{S}}_{\mathcal{K},\mathcal{H}}$-Anfrage verwendet, was nach Fall E nur mit vernachlässigbarer Wahrscheinlichkeit möglich ist, oder bereits bei einer $\mathcal{O}^{\mathsf{S}}_{\mathcal{H}}$-Anfrage vom Angreifer $\mathcal{A}$ selbst erstellt.

Folglich wurde jede Münze coin mit $(\cdot, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}}$ (und insbesondere SoK), die der Angreifer $\mathcal{A}$ durch eine $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$-Anfrage einlöst, außer mit vernachlässigbarer Wahrscheinlichkeit, von $\mathcal{A}$ erstellt. Daher kann $\mathcal{B}$, wie unten bei den Simulationen beschrieben, bei jeder Münze coin mit $(\cdot, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}}$, die der Angreifer $\mathcal{A}$ durch eine $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$-Anfrage einlöst, das Nachrichten-Signatur-Paar extrahieren.

Insgesamt wurden durch den Angreifer $\mathcal{A}$ Münzen im Wert von $\$^* > q_{W,1} \cdot K + k_1$ ausgegeben oder eingelöst. Dann gewinnt $\mathcal{A}$ das Spiel entweder, wenn (1) alle $\$^* = q_{W,1} \cdot K + k_1 + k^*$ Serienschlüssel für $k^* \geq 1$ verschieden sind, oder (2) einige Serienschlüssel mehrmals ausgeben oder eingelöst werden aber $\mathsf{Identify}$ keine Kontonummer $\mathsf{pk} \in \mathcal{U}_{\mathcal{A}}$ ausgibt (was bei einigen Autoren auch als Identifikation eines Double-Spenders bezeichnet wird, siehe [CHL05, CG10]).

Wir betrachten zunächst den Fall, dass alle Serienschlüssel verschieden sind. Da $\mathcal{A}$ aus den $\mathcal{O}^{\mathsf{W}}_{\mathcal{B}}$-Anfragen maximal zu $q_{W,1}$ verschiedenen Polynomen Signaturen erhält und alle Polynome maximal den Grad K haben (ansonsten würde $\mathcal{B}$, wie unten bei den Simulationen beschrieben, abbrechen), können aus diesen Polynomen maximal $q_{W,1} \cdot K$ Serienschlüssel generiert werden. Denn durch maximal K Serienschlüssel wird das jeweilige Polynom stets eindeutig festgelegt. Da allerdings neben den k_1 Serienschlüsseln durch die Münzen coin mit $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}}$ weitere $q_{W,1} \cdot K + k^*$ Serienschlüssel ausgegeben oder eingelöst werden, kann $\mathcal{B}$ mindestens $q_{W,1} + 1$ verschiedene Polynome extrahieren. Denn seien o.B.d.A. $\kappa_{j,0}, \ldots, \kappa_{j,K-1}$ für $1 \leq j \leq q_{W,1}$ die $q_{W,1} \cdot K$ Serienschlüssel, mit denen die $q_{W,1}$ Polynome $\phi_j(\mathsf{x}) = \gamma_j \prod_{i=0}^{K-1}(\mathsf{x} + \kappa_{j,i}) \in \mathbb{Z}_p[\mathsf{x}]$ mit $\gamma_j \in \mathbb{Z}_p$ berechnet werden. Da insgesamt allerdings $q_{W,1} \cdot K + k^*$ Serienschlüssel mit $k^* \geq 1$ ausgegeben oder eingelöst werden, kann aus diesen k^* weiteren Serienschlüsseln mindestens ein weiteres Polynom berechnet werden. (Beachte: Es ist durchaus möglich, dass $\mathcal{B}$ maximal $q_{W,1}$ verschiedene Akkumulatoren extrahiert. Dennoch wird die Beschränktheit des Akkumulator-Schemas im Gegensatz zu [CG10] nicht benötigt, da $\mathcal{B}$ in jedem Fall mindestens $q_{W,1}+1$ verschiedene Polynome extrahiert.) Somit extrahiert $\mathcal{B}$ mindestens eine Signatur $(\Sigma^\star, e^\star, s^\star)$ und die dazugehörende Nachricht $(u^\star, \phi^\star(\mathsf{x}))$, zu der $\mathcal{A}$ keine $\mathcal{O}^{\mathsf{W}}_{\mathcal{B}}$-Anfrage gestellt hat (und somit auch keine $\mathcal{O}^{\mathsf{VW}}_{\mathcal{B}}$-Anfrage). Also wurde das Nachrichten-Signatur-

Paar $((u^\star, \phi^\star(\mathsf{x})), (\Sigma^\star, e^\star, s^\star))$ nie bei einer $\mathcal{O}_{\mathcal{B}}^{\mathsf{W}}$-Anfrage und somit auch nie bei einer $\mathcal{O}_{\mathcal{B}}^{\mathsf{VW}}$-Anfrage erzeugt. Nun müssen wir die Fälle unterscheiden, dass das Nachrichten-Signatur-Paar $((u^\star, \phi^\star(\mathsf{x})), (\Sigma^\star, e^\star, s^\star))$ entweder (1.1) nie oder (1.2) mindestens einmal bei der Simulation einer $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{W}}$-Anfrage erzeugt wurde.

Fall (1.1): Der Angreifer $\mathcal{B}$ erhält den öffentlichen Schlüssel $(p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_0, g_1, u_0, \ldots, u_K, h, h_1, \ldots, h_K, X)$ des BBS+A-Signaturverfahrens und darf q Anfragen an das Signatur-Orakel stellen.

Simulation der Initialisierung: Der Angreifer $\mathcal{B}$ generiert eine zyklische Gruppe $\mathbb{G}_p$ der Ordnung p, wählt zufällig eine Zahl $y \in_{\mathcal{R}} \mathbb{Z}_p^*$ sowie einen Generator $\mathfrak{g} \in_{\mathcal{R}} \mathbb{G}_p$ und berechnet den Generator $\mathfrak{g}_0 = \mathfrak{g}^y$. Dann sendet $\mathcal{B}$ die perfekt simulierten Systemparameter $\mathsf{sp} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \mathbb{G}_p', \hat{e}, g, g_0, g_1, u_0, \ldots, u_K, h, h_1, \ldots, h_K, \mathfrak{g}, \mathfrak{g}_0)$ und den perfekt simulierten öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}} = X$ an den Angreifer $\mathcal{A}$.

Simulation von $\mathcal{O}^{\mathsf{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}_{\mathcal{B}}^{\mathsf{A}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{A}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}_{\mathcal{B}}^{\mathsf{W}}$: Der Angreifer $\mathcal{B}$ simuliert jede $\mathcal{O}_{\mathcal{B}}^{\mathsf{W}}$-Anfrage analog zum Issue-Protokoll des BBS+A-Signaturverfahrens wie in Unterabschnitt 3.2.3 beschrieben. Dann speichert $\mathcal{B}$ den Eintrag $(i_{\mathcal{A}}, \mathsf{view} = (\mathbb{T} = (I, C, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s'')), (s', u, \phi(\mathsf{x})))$ in einer Liste $\mathsf{view}_{\mathcal{A}}$ sowie das Nachrichten-Signatur-Paar $((u, \phi(\mathsf{x})), (\Sigma, e, s))$ mit $\deg(\phi) \leq K$ in einer Liste $\sigma_{\mathcal{A}}$, erhöht den Wert \$ um K und den Zähler $i_{\mathcal{A}}$ um 1. Falls $\mathcal{B}$ eine Anfrage nicht simulieren kann, bricht $\mathcal{B}$ ab. Dies passiert aber nur mit vernachlässigbarer Wahrscheinlichkeit. Der Angreifer $\mathcal{B}$ benötigt insgesamt $q_{W,1}$ Anfragen an das Signatur-Orakel.

Simulation von $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{W}}$: Der Angreifer $\mathcal{B}$ verhält sich wie der ehrliche Kunde und simuliert die Bank perfekt mit Zugriff auf das Signatur-Orakel, da $\mathcal{B}$ die zu signierende Nachricht selbst wählt. Dann speichert $\mathcal{B}$ den Eintrag $(i, \mathbb{W}_i = (\Sigma, e, s, \mathsf{info}), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$, einen Eintrag $(i, \mathsf{view} = (\mathbb{T} = (I, C, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s'')))$ in einer Liste view_H sowie das Nachrichten-Signatur-Paar $((u, \phi(\mathsf{x})), (\Sigma, e, s))$ in einer Liste σ_H ab und erhöht den Zähler i um 1. Der Angreifer $\mathcal{B}$ benötigt insgesamt $q_{W,2}$ Anfragen an das Signatur-Orakel.

Simulation von $\mathcal{O}_{\mathcal{H}}^{\mathsf{S}}$: Nach den Fällen A bis C wurde die Münze coin nur mit vernachlässigbarer Wahrscheinlichkeit nicht von $\mathcal{A}$ erstellt. Somit verhält sich der Angreifer $\mathcal{B}$ wie der ehrliche Händler, außer dass $\mathcal{B}$ den Angreifer $\mathcal{A}$ während der Durchführung von SoK zurückspult, um das Nachrichten-Signatur-Paar $((u, \phi(\mathsf{x})), (\Sigma, e, s))$ analog zum Prove*-k bzw. Prove-K-Protokoll des BBS+A-Signaturverfahrens (Unterabschnitt 3.2.3) zu extrahieren und in einer Liste $\mathbb{D}_\sigma$ abzuspeichern. Falls das

Nachrichten-Signatur-Paar nicht extrahiert werden kann, bricht $\mathcal{B}$ ab. Dies passiert aber nur mit vernachlässigbarer Wahrscheinlichkeit. Dann erhöht $\mathcal{B}$ nach erfolgreicher Durchführung den Wert $\$^*$ um k, speichert einen Eintrag (j, coin) in einer Liste $\mathbb{C}_{\mathcal{A}}$ sowie einen Eintrag $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts})$ in einer Liste $\mathbb{S}$ ab und erhöht den Zähler j um 1.

Simulation von $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird der Wert $\$$ um k erhöht und ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_{H,\mathcal{A}}$ gespeichert.

Simulation von $\mathcal{O}^{\mathsf{S}}_{\mathcal{K},\mathcal{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_H$, ein Eintrag (j, coin) in einer Liste $\mathbb{C}$ und ein Eintrag $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts})$ in der Liste $\mathbb{S}$ gespeichert sowie der Zähler j um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$: Falls $\mathsf{coin} \in \mathbb{D}_{\mathcal{A}}$ ist, verhält sich $\mathcal{B}$ wie das Orakel und gibt eine Fehlermeldung $\perp$ aus.

Falls $\mathsf{coin} \in \mathbb{C}_{H,\mathcal{A}} \setminus \mathbb{D}_{\mathcal{A}}$ ist, verhält sich $\mathcal{B}$ wie das Orakel. Somit wird nach erfolgreicher Durchführung der Wert $\$^*$ um k erhöht und die Münze coin in einer Liste $\mathbb{D}_{\mathcal{A}}$ gespeichert.

Falls $\mathsf{coin} \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{D}_{\mathcal{A}}$ ist, wurde die Münze coin nach den Fällen D bis F nur mit vernachlässigbarer Wahrscheinlichkeit nicht von $\mathcal{A}$ erstellt. Somit verhält sich der Angreifer $\mathcal{B}$ wie die ehrliche Bank, außer dass $\mathcal{B}$ den Angreifer $\mathcal{A}$ während der Durchführung von SoK zurückspult, um das Nachrichten-Signatur-Paar $((u, \phi(\mathsf{x})), (\Sigma, e, s))$ analog zum Prove^*-k bzw. Prove-K-Protokoll des BBS+A-Signaturverfahrens zu extrahieren und in einer Liste $\mathbb{D}_\sigma$ abzuspeichern. Falls das Nachrichten-Signatur-Paar nicht extrahiert werden kann, bricht $\mathcal{B}$ ab. Dies passiert aber nur mit vernachlässigbarer Wahrscheinlichkeit. Anschließend wird nach erfolgreicher Durchführung der Wert $\$^*$ um k erhöht und die Münze coin in einer Liste $\mathbb{D}_{\mathcal{A}}$ gespeichert.

Simulation von $\mathcal{O}^{\mathsf{D}}_{\mathcal{H},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird ein Eintrag (j, coin) in einer Liste $\mathbb{D}_H$ gespeichert.

Simulation von $\mathcal{O}^{\mathsf{VW}}_{\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verwendet die List $\mathsf{view}_{\mathcal{A}}$ und simuliert die Signatur perfekt mit Zugriff auf das Signatur-Orakel. Der Angreifer $\mathcal{B}$ benötigt insgesamt q_V Anfragen an das Signatur-Orakel.

Spielende: Der Angreifer $\mathcal{B}$ löst jede Münze coin mit $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{\mathcal{A}} \setminus \mathbb{D}_H$ ein. Mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ gilt $\$^* > \$$.

Nach Voraussetzung extrahiert $\mathcal{B}$ bei einer $\mathcal{O}^{\mathsf{S}}_{\mathcal{H}}$- oder $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$ -Anfrage ein Nachrichten-Signatur-Paar $((u^\star, \phi^\star(\mathsf{x})), (\Sigma^\star, e^\star, s^\star))$, das nie bei der Simulation einer Orakel-Anfrage verwendet wurde. Somit ist $((u^\star, \phi^\star(\mathsf{x})), (\Sigma^\star, e^\star, s^\star)) \in \mathbb{D}_\sigma \setminus (\sigma_H \cup \sigma_{\mathcal{A}})$.

Der Angreifer $\mathcal{B}$ sendet $((u^\star, \phi^\star(\mathsf{x})), (\Sigma^\star, e^\star, s^\star))$ an das Signatur-Orakel und bricht damit die starke existentielle Unfälschbarkeit der BBS+A-Signatur (und löst somit das

q-SDH-Problem) mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist.

Fall (1.2): Der Angreifer $\mathcal{B}$ erhält eine zufällige 2-SDH-Probleminstanz $(p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, u_0^a, h, h^a)$.

Simulation der Initialisierung: Der Angreifer $\mathcal{B}$ generiert eine zyklische Gruppe $\mathbb{G}_p$ der Ordnung p, wählt zufällig vier Zahlen $\alpha, \beta, x, y \in_{\mathcal{R}} \mathbb{Z}_p^*$, einen Generator $\mathfrak{g} \in_{\mathcal{R}} \mathbb{G}_p$ sowie zwei Generatoren $g, g_1 \in_{\mathcal{R}} \mathbb{G}_1$. Dann berechnet $\mathcal{B}$ die Generatoren $\mathfrak{g}_0 = \mathfrak{g}^y, g_0 = u_0^\beta, X = h^x$ sowie $u_i = u_0^{\alpha^i}$ und $h_i = h^{\alpha^i}$ für $1 \leq i \leq K$. Anschließend sendet $\mathcal{B}$ die perfekt simulierten Systemparameter $\mathsf{sp} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \mathbb{G}_p', \hat{e}, g, g_0, g_1, u_0, \ldots, u_K, h, h_1, \ldots, h_K, \mathfrak{g}, \mathfrak{g}_0)$ sowie den perfekt simulierten Schlüssel $\mathsf{pk}_{\mathcal{B}} = X$ an den Angreifer $\mathcal{A}$.

Simulation von $\mathcal{O}^{\mathsf{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}_{\mathcal{B}}^{\mathsf{A}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{A}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}_{\mathcal{B}}^{\mathsf{W}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den privaten Schlüssel $x \in \mathbb{Z}_p^*$ kennt. Somit wird der Wert \$ um K erhöht, ein Eintrag $(i_{\mathcal{A}}, \mathsf{view})$ in einer Liste $\mathsf{view}_{\mathcal{A}}$ gespeichert und der Zähler $i_{\mathcal{A}}$ um 1 erhöht.

Simulation von $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{W}}$: Der Angreifer $\mathcal{B}$ verhält sich wie der ehrliche Kunde, außer bei der Berechnung von C und der Erstellung von SoK_{W}. Hier wählt $\mathcal{B}$ zufällig eine Zahl $\delta \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Commitment $C = (u_0^a)^\beta g_0^\delta g_1^u u_0^{\phi(\alpha)} = g_0^a g_0^\delta g_1^u u_0^{\phi(\alpha)} = g_0^{s'} g_1^u u_0^{\phi(\alpha)}$ für $s' := a + \delta \mod p$ und simuliert SoK_{W}. Da $\mathcal{B}$ den privaten Schlüssel x kennt, kann $\mathcal{B}$ eine gültige Signatur erstellen. Insgesamt hat $\mathcal{B}$ somit die Anfrage perfekt simuliert. Dann wird ein Eintrag $(i, \mathbb{W}_i = (\Sigma, e, \delta, s'', \mathsf{info}), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$, ein Eintrag $(i, \mathsf{view} = (\mathbb{T} = (I, C, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s'')))$ in einer Liste view_H und das Nachrichten-Signatur-Paar $((u, \phi(\mathsf{x})), (\Sigma, e, \delta, s''))$ in einer Liste σ_H gespeichert und der Zähler i um 1 erhöht.

Simulation von $\mathcal{O}_{\mathcal{H}}^{\mathsf{S}}$: Nach den Fällen A bis C wurde die Münze coin nur mit vernachlässigbarer Wahrscheinlichkeit nicht von $\mathcal{A}$ erstellt. Somit verhält sich der Angreifer $\mathcal{B}$ wie der ehrliche Händler, außer dass $\mathcal{B}$ den Angreifer $\mathcal{A}$ während der Durchführung von SoK zurückspult, um das Nachrichten-Signatur-Paar $((u, \phi(\mathsf{x})), (\Sigma, e, s))$ analog zum Prove^*-k bzw. Prove-K-Protokoll des BBS+A-Signaturverfahrens (Unterabschnitt 3.2.3) zu extrahieren und in einer Liste $\mathbb{D}_\sigma$ abzuspeichern. Falls das Nachrichten-Signatur-Paar nicht extrahiert werden kann, bricht $\mathcal{B}$ ab. Dies passiert aber nur mit vernachlässigbarer Wahrscheinlichkeit. Dann erhöht $\mathcal{B}$ nach erfolgreicher Durchführung den Wert \$* um k, speichert einen Eintrag (j, coin) in einer

Liste $\mathbb{C}_{\mathcal{A}}$ sowie einen Eintrag $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts})$ in einer Liste $\mathbb{S}$ ab und erhöht den Zähler j um 1.

Simulation von $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$: Der Angreifer $\mathcal{B}$ verhält sich wie der ehrliche Kunde mit Geldbörse $\mathbb{W}_i$, außer dass $\mathcal{B}$ die Signature-of-Knowledge SoK perfekt simulieren muss, da $\mathcal{B}$ die Signatur (die Zahl $s = a + \delta + s'' \mod p$) nicht kennt. Anschließend wird der Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i)$ zu $(i, \mathbb{W}'_i, \mathsf{pk}_{\mathcal{K}}, \$_i - k)$ aktualisiert, der Wert \$ um k erhöht und ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_{H,\mathcal{A}}$ gespeichert.

Simulation von $\mathcal{O}^{\mathsf{S}}_{\mathcal{K},\mathcal{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie der ehrliche Kunde mit Geldbörse $\mathbb{W}_i$, außer dass $\mathcal{B}$ die Signature-of-Knowledge SoK perfekt simulieren muss, da $\mathcal{B}$ die Signatur (die Zahl $s = a + \delta + s'' \mod p$) nicht kennt. Weiter verhält sich $\mathcal{B}$ wie der ehrliche Händler. Anschließend wird der Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i)$ zu $(i, \mathbb{W}'_i, \mathsf{pk}_{\mathcal{K}}, \$_i - k)$ aktualisiert, ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_H$, ein Eintrag (j, coin) in einer Liste $\mathbb{C}$ und ein Eintrag $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts})$ in einer Liste $\mathbb{S}$ gespeichert sowie der Zähler j um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$: Falls $\mathsf{coin} \in \mathbb{D}_{\mathcal{A}}$ ist, verhält sich $\mathcal{B}$ wie das Orakel und gibt eine Fehlermeldung $\perp$ aus.

Falls $\mathsf{coin} \in \mathbb{C}_{H,\mathcal{A}} \setminus \mathbb{D}_{\mathcal{A}}$ ist, verhält sich $\mathcal{B}$ wie das Orakel. Somit wird nach erfolgreicher Durchführung der Wert $\$^*$ um k erhöht und die Münze coin in einer Liste $\mathbb{D}_{\mathcal{A}}$ gespeichert.

Falls $\mathsf{coin} \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{D}_{\mathcal{A}}$ ist, wurde die Münze coin nach den Fällen D bis F nur mit vernachlässigbarer Wahrscheinlichkeit nicht von $\mathcal{A}$ erstellt. Somit verhält sich der Angreifer $\mathcal{B}$ wie die ehrliche Bank, außer dass $\mathcal{B}$ den Angreifer $\mathcal{A}$ während der Durchführung von SoK zurückspult, um das Nachrichten-Signatur-Paar $((u, \phi(\mathsf{x})), (\Sigma, e, s))$ analog zum Prove^*-k bzw. Prove-K-Protokoll des BBS+A-Signaturverfahrens zu extrahieren und in einer Liste $\mathbb{D}_\sigma$ abzuspeichern. Falls das Nachrichten-Signatur-Paar nicht extrahiert werden kann, bricht $\mathcal{B}$ ab. Dies passiert aber nur mit vernachlässigbarer Wahrscheinlichkeit. Anschließend wird nach erfolgreicher Durchführung der Wert $\$^*$ um k erhöht und die Münze coin in einer Liste $\mathbb{D}_{\mathcal{A}}$ gespeichert.

Simulation von $\mathcal{O}^{\mathsf{D}}_{\mathcal{H},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird ein Eintrag (j, coin) in einer Liste $\mathbb{D}_H$ gespeichert.

Simulation von $\mathcal{O}^{\mathsf{VW}}_{\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den privaten Schlüssel $x \in \mathbb{Z}_p^*$ kennt und die entsprechenden Einträge in der Liste $\mathsf{view}_{\mathcal{A}}$ gespeichert hat.

Spielende: Der Angreifer $\mathcal{B}$ löst jede Münze coin mit $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{\mathcal{A}} \setminus \mathbb{D}_H$ ein. Mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ gilt $\$^* > \$$.

Nach Voraussetzung extrahiert $\mathcal{B}$ bei einer $\mathcal{O}^{\mathsf{S}}_{\mathcal{H}}$- oder $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$-Anfrage ein Nachrichten-Signatur-Paar $((u^\star, \phi^\star(\mathsf{x})), (\Sigma^\star, e^\star, s^\star))$, wobei $\mathcal{A}$ zur Nachricht $(u^\star, \phi^\star(\mathsf{x}))$ keine $\mathcal{O}^{\mathsf{W}}_{\mathcal{B}}$- und somit auch keine $\mathcal{O}^{\mathsf{VW}}_{\mathcal{B}}$-Anfrage gestellt hat. Allerdings wurde nach Voraussetzung das Nachrichten-Signatur-Paar $((u^\star, \phi^\star(\mathsf{x})), (\Sigma^\star, e^\star, s^\star))$ bei der Simulation einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$-Anfrage erstellt. Somit existiert ein Eintrag $((u^\star, \phi^\star(\mathsf{x})), (\Sigma^\star, e^\star, \delta^\star, s''^\star)) \in \sigma_H$.

Der Angreifer $\mathcal{B}$ sucht diesen Eintrag und berechnet den diskreten Logarithmus $a = s^\star - s''^\star - \delta^\star \mod p$ und löst damit das 2-SDH-Problem mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist.

Fall (2): Wir betrachten nun den Fall, dass einige Serienschlüssel mehrmals ausgegeben oder eingelöst werden, ohne dass ein betrügerischer Kunde $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_{\mathcal{A}}$ identifiziert werden kann. Seien $\mathsf{coin} = (k, S, T, R, \Phi = (A_1, B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$ und $\mathsf{coin}' = (k', S', T', R', \Phi' = (A'_1, B'_1, B'_2, SoK', \mathsf{ts}', \mathsf{pk}'_{\mathcal{H}}))$ die beiden Münzen, bei denen ein Double-Spending auftrat. Da $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts}) \neq (\mathsf{pk}'_{\mathcal{H}}, \mathsf{ts}')$ gelten muss, ist nur mit vernachlässigbarer Wahrscheinlichkeit $R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}} || \mathsf{ts} || k) = \mathsf{H}(\mathsf{pk}'_{\mathcal{H}} || \mathsf{ts}' || k') = R'$. Sei $\mathsf{pk}^\star_{\mathcal{K}} \notin \mathcal{U}_{\mathcal{A}}$ das Ergebnis der Berechnung des $\mathsf{Identify}$-Algorithmus.

Durch die obigen sechs Fälle A bis F wurde bereits gezeigt, dass nur mit vernachlässigbarer Wahrscheinlichkeit $(\cdot, \cdot, \mathsf{coin}), (\cdot, \cdot, \mathsf{coin}') \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ gilt. Dann gibt es die beiden Fälle, dass (2.1) keine oder (2.2) genau eine der Münzen coin und coin' von $\mathcal{B}$ zum Bezahlen verwendet wurde.

Fall (2.1): $(\cdot, \cdot, \mathsf{coin}), (\cdot, \cdot, \mathsf{coin}') \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$:

In diesem Fall wurden SoK und SoK' vom Angreifer $\mathcal{A}$ erstellt. Daher kann $\mathcal{B}$ die beiden Zahlen $\kappa, \kappa' \in \mathbb{Z}_p$ sowie die beiden Nachrichten-Signatur-Paare $((u, \phi(\mathsf{x})), (\Sigma, e, s))$ und $((u', \phi'(\mathsf{x})), (\Sigma', e', s'))$ mit $S = \mathfrak{g}^\kappa, S' = \mathfrak{g}^{\kappa'}, T = \mathfrak{g}^u \mathfrak{g}_0^{R\kappa}$ und $T' = \mathfrak{g}^{u'} \mathfrak{g}_0^{R'\kappa'}$ aus SoK und SoK' extrahieren.

- Falls $k = k'$ und $S = S'$ ist, folgt $\kappa = \kappa'$ und mit dem $\mathsf{Identify}$-Algorithmus:

$$\mathsf{pk}^\star_{\mathcal{K}} = \left(\frac{T'^R}{T^{R'}}\right)^{(R-R')^{-1}} = \left(\frac{\mathfrak{g}^{Ru'} \mathfrak{g}_0^{RR'\kappa'}}{\mathfrak{g}^{R'u} \mathfrak{g}_0^{RR'\kappa}}\right)^{(R-R')^{-1}} = \mathfrak{g}^{\frac{Ru'-R'u}{R-R'}} \notin \mathcal{U}_{\mathcal{A}}.$$

  - Falls mindestens eines der beiden Nachrichten-Signatur-Paare niemals bei einer Orakel-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.1) vor.
  - Falls mindestens eines der beiden Nachrichten-Signatur-Paare bei einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.2) vor.
  - Falls obige Fälle nicht eintreten, wurden beide Nachrichten-Signatur-Paare bei jeweils einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{B}}$ bzw. $\mathcal{O}^{\mathsf{VW}}_{\mathcal{B}}$-Anfrage erstellt. Aus $\mathfrak{g}^{\frac{Ru'-R'u}{R-R'}} \notin \mathcal{U}_{\mathcal{A}}$ folgt $u' \neq u$. Somit handelt es sich nicht um ein Double-Spending, da zwei verschiedene, korrupte Kunden mit $\mathsf{pk}_{\mathcal{K}} = \mathfrak{g}^u, \mathsf{pk}'_{\mathcal{K}} = \mathfrak{g}^{u'} \in \mathcal{U}_{\mathcal{A}}$ zwei Geldbörsen abgehoben haben und somit auch mit den entsprechenden Münzen bezahlen dürfen. Insbesondere kann $\mathcal{A}$ in diesem Fall aber nicht

Münzen im Wert von mehr als \$ ausgeben oder einlösen und somit das Spiel nicht gewinnen.

- Falls $k \neq k'$ ist, sei. o.B.d.A. $k > k'$. Falls der zu S' gehörende Schlüssel $\kappa' \in \mathbb{Z}_p$ mit $S' = \mathfrak{g}^{\kappa'}$ aus der Seriennummer S berechnet werden kann, folgt mit dem $\mathsf{Identify}$-Algorithmus:

$$\mathsf{pk}_{\mathcal{K}}^{\star} = \frac{T'}{\mathfrak{g}_0^{R'\kappa'}} = \frac{\mathfrak{g}^{u'}\mathfrak{g}_0^{R'\kappa'}}{\mathfrak{g}_0^{R'\kappa'}} = \mathfrak{g}^{u'} \notin \mathcal{U}_{\mathcal{A}}.$$

  Da $\mathfrak{g}^{u'} \notin \mathcal{U}_{\mathcal{A}}$ ist, wurde das Nachrichten-Signatur-Paar $((u', \phi'(\mathsf{x})), (\Sigma', e', s'))$ niemals bei einer $\mathcal{O}_{\mathcal{B}}^{\mathsf{W}}$- bzw. $\mathcal{O}_{\mathcal{B}}^{\mathsf{VW}}$-Anfrage erstellt.

  - Falls das Nachrichten-Signatur-Paar $((u', \phi'(\mathsf{x})), (\Sigma', e', s'))$ niemals bei einer Orakel-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.1) vor.
  - Falls das Nachrichten-Signatur-Paar $((u', \phi'(\mathsf{x})), (\Sigma', e', s'))$ bei einer $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{W}}$-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.2) vor.

- Falls $k = k'$ ist, aber $S \neq S'$ gilt, oder o.B.d.A. $k > k'$ ist, aber der zu S' gehörende Schlüssel $\kappa' \in \mathbb{Z}_p$ mit $S' = \mathfrak{g}^{\kappa'}$ nicht aus der Seriennummer S berechnet werden kann, muss aus beiden Seriennummern S und S' mindestens einmal derselbe Serienschlüssel $\kappa_j \in \mathbb{Z}_p^*$ berechnet werden (denn es ist ja ein Double-Spending aufgetreten). Somit gibt es einen Nachfolger S^* von S sowie einen Nachfolger S'^* von S' (wobei auch $S^* = S$ bzw. $S'^* = S'$ erlaubt ist) und eine Zahl $\kappa^* \in \mathbb{Z}_p^*$ mit $\kappa^* = \mathsf{H}(S^*||b) = \mathsf{H}(S'^*||b')$ für $b, b' \in \{0, 1\}$. Dies ist allerdings nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Fall (2.2): Sei o.B.d.A. $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(\cdot, \cdot, \mathsf{coin}') \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$:

Falls $\mathcal{B}$ genau eine der beiden Münzen zum Bezahlen verwendet hat, sei dies o.B.d.A. die Münze coin. Dann wurde SoK' vom Angreifer $\mathcal{A}$ erstellt und daher kann $\mathcal{B}$ die Zahl $\kappa' \in \mathbb{Z}_p$ sowie das Nachrichten-Signatur-Paar $((u', \phi'(\mathsf{x})), (\Sigma', e', s'))$ mit $S' = \mathfrak{g}^{\kappa'}$ und $T' = \mathfrak{g}^{u'}\mathfrak{g}_0^{R'\kappa'}$ aus SoK' extrahieren.

Sei $\kappa \in \mathbb{Z}_p^*$ und $((u, \phi(\mathsf{x})), (\Sigma, e, s))$ das Nachrichten-Signatur-Paar der Münze coin mit $S = \mathfrak{g}^{\kappa}$ und $T = \mathfrak{g}^{u}\mathfrak{g}_0^{R\kappa}$.

- Falls $k = k'$ und $S = S'$ ist, folgt $\kappa = \kappa'$ und mit dem $\mathsf{Identify}$-Algorithmus:

$$\mathsf{pk}_{\mathcal{K}}^{\star} = \left(\frac{T'^R}{T^{R'}}\right)^{(R-R')^{-1}} = \left(\frac{\mathfrak{g}^{Ru'}\mathfrak{g}_0^{RR'\kappa'}}{\mathfrak{g}^{R'u}\mathfrak{g}_0^{RR'\kappa}}\right)^{(R-R')^{-1}} = \mathfrak{g}^{\frac{Ru'-R'u}{R-R'}} \notin \mathcal{U}_{\mathcal{A}}.$$

  - Falls das Nachrichten-Signatur-Paar $((u', \phi'(\mathsf{x})), (\Sigma', e', s'))$ niemals bei einer Orakel-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.1) vor.
  - Falls das Nachrichten-Signatur-Paar $((u', \phi'(\mathsf{x})), (\Sigma', e', s'))$ bei einer $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{W}}$-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.2) vor.

  - Falls obige Fälle nicht eintreten, wurde das Nachrichten-Signatur-Paar $((u', \phi'(\mathsf{x})), (\Sigma', e', s'))$ bei einer $\mathcal{O}_{\mathcal{B}}^{\mathsf{W}}$- bzw. $\mathcal{O}_{\mathcal{B}}^{\mathsf{VW}}$-Anfrage erstellt. Somit handelt es sich nicht um ein Double-Spending, da zwei verschiedene Kunden mit $\mathsf{pk}_{\mathcal{K}} = \mathfrak{g}^u \in \mathcal{U}_H$ und $\mathsf{pk}'_{\mathcal{K}} = \mathfrak{g}^{u'} \in \mathcal{U}_{\mathcal{A}}$ zwei Geldbörsen abgehoben haben. Dazu müsste $\mathcal{A}$ entweder denselben Master-Schlüssel $\kappa_{0,0}$ wie $\mathsf{pk}_{\mathcal{K}}$ wählen oder es tritt eine Kollision bei der Simulation des Random-Oracles auf, was beides nur mit vernachlässigbarer Wahrscheinlichkeit möglich ist. Insbesondere kann $\mathcal{A}$ in diesem Fall aber nicht Münzen im Wert von mehr als \$ ausgeben oder einlösen und somit das Spiel nicht gewinnen.

- Falls $k > k'$ ist und der zu S' gehörende Schlüssel $\kappa' \in \mathbb{Z}_p$ mit $S' = \mathfrak{g}^{\kappa'}$ aus der Seriennummer S berechnet werden kann, folgt mit dem $\mathsf{Identify}$-Algorithmus:

$$\mathsf{pk}_{\mathcal{K}}^{\star} = \frac{T'}{\mathfrak{g}_0^{R'\kappa'}} = \frac{\mathfrak{g}^{u'}\mathfrak{g}_0^{R'\kappa'}}{\mathfrak{g}_0^{R'\kappa'}} = \mathfrak{g}^{u'} \notin \mathcal{U}_{\mathcal{A}}.$$

  Da $\mathfrak{g}^{u'} \notin \mathcal{U}_{\mathcal{A}}$ ist, wurde das Nachrichten-Signatur-Paar $((u', \phi'(\mathsf{x})), (\Sigma', e', s'))$ niemals bei einer $\mathcal{O}_{\mathcal{B}}^{\mathsf{W}}$- bzw. $\mathcal{O}_{\mathcal{B}}^{\mathsf{VW}}$-Anfrage erstellt.

  - Falls das Nachrichten-Signatur-Paar $((u', \phi'(\mathsf{x})), (\Sigma', e', s'))$ niemals bei einer Orakel-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.1) vor.
  - Falls das Nachrichten-Signatur-Paar $((u', \phi'(\mathsf{x})), (\Sigma', e', s'))$ bei einer $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{W}}$-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.2) vor.

- Falls $k < k'$ ist und der zu S gehörende Schlüssel $\kappa \in \mathbb{Z}_p$ mit $S = \mathfrak{g}^{\kappa}$ aus der Seriennummer S' berechnet werden kann, folgt mit dem $\mathsf{Identify}$-Algorithmus:

$$\mathsf{pk}_{\mathcal{K}}^{\star} = \frac{T}{\mathfrak{g}_0^{R\kappa}} = \frac{\mathfrak{g}^{u}\mathfrak{g}_0^{R\kappa}}{\mathfrak{g}_0^{R\kappa}} = \mathfrak{g}^{u} \notin \mathcal{U}_{\mathcal{A}}.$$

  - Falls das Nachrichten-Signatur-Paar $((u', \phi'(\mathsf{x})), (\Sigma', e', s'))$ niemals bei einer Orakel-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.1) vor.
  - Falls das Nachrichten-Signatur-Paar $((u', \phi'(\mathsf{x})), (\Sigma', e', s'))$ bei einer $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{W}}$-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.2) vor.
  - Falls obige Fälle nicht eintreten, wurde das Nachrichten-Signatur-Paar $((u', \phi'(\mathsf{x})), (\Sigma', e', s'))$ bei einer $\mathcal{O}_{\mathcal{B}}^{\mathsf{W}}$- bzw. $\mathcal{O}_{\mathcal{B}}^{\mathsf{VW}}$-Anfrage erstellt. Somit handelt es sich nicht um ein Double-Spending, da zwei verschiedene Kunden mit $\mathsf{pk}_{\mathcal{K}} = \mathfrak{g}^u \in \mathcal{U}_H$ und $\mathsf{pk}'_{\mathcal{K}} = \mathfrak{g}^{u'} \in \mathcal{U}_{\mathcal{A}}$ zwei Geldbörsen abgehoben haben. Dazu müsste $\mathcal{A}$ entweder denselben Master-Schlüssel $\kappa_{0,0}$ wie $\mathsf{pk}_{\mathcal{K}}$ wählen oder es tritt eine Kollision bei der Simulation des Random-Oracles auf, was beides nur mit vernachlässigbarer Wahrscheinlichkeit möglich ist. Insbesondere kann $\mathcal{A}$ in diesem Fall aber nicht Münzen im Wert von mehr als \$ ausgeben oder einlösen und somit das Spiel nicht gewinnen.

- Falls $k = k'$ ist, aber $S \neq S'$ gilt, oder $k > k'$ ist, aber der zu S' gehörende Schlüssel $\kappa' \in \mathbb{Z}_p$ mit $S' = g^{\kappa'}$ nicht aus der Seriennummer S berechnet werden kann, oder $k < k'$ ist, aber der zu S gehörende Schlüssel $\kappa \in \mathbb{Z}_p$ mit $S = g^{\kappa}$ nicht aus der Seriennummer S' berechnet werden kann, muss aus beiden Seriennummern S und S' mindestens einmal derselbe Serienschlüssel $\kappa_j \in \mathbb{Z}_p^*$ berechnet werden (denn es ist ja ein Double-Spending aufgetreten). Somit gibt es einen Nachfolger S^* von S sowie einen Nachfolger S'^* von S' (wobei auch $S^* = S$ bzw. $S'^* = S'$ erlaubt ist) und eine Zahl $\kappa^* \in \mathbb{Z}_p^*$ mit $\kappa^* = \mathsf{H}(S^*||b) = \mathsf{H}(S'^*||b')$ für $b, b' \in \{0, 1\}$. Dies ist allerdings nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Somit ist der Fall, dass einige Serienschlüssel mehrmals eingelöst werden, ohne dass ein betrügerischer Kunde $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_{\mathcal{A}}$ eindeutig identifiziert werden kann, vernachlässigbar.

Daher bricht der Angreifer $\mathcal{A}$ die Unfälschbarkeit nur mit vernachlässigbarer Wahrscheinlichkeit.

Bemerkung 6.5.2. *Für die Unfälschbarkeit des Geldsystems ist es also wichtig, dass die Hashfunktion* H *stark kollisionsresistent ist.*

Anonymität: Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Anonymität des Systems bricht. Wir konstruieren einen probabilistischen polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, die semantische Sicherheit des ElGamal-Verschlüsselungsverfahrens in der Gruppe $\mathbb{G}_p$ mit $\left(p, \mathbb{G}'_p, \mathfrak{g}\right) \leftarrow \mathsf{Setup}_{\mathsf{G}}\left(1^{\lambda}\right)$ zu brechen, was ein Widerspruch zur DDH-Annahme für $\mathsf{Setup}_{\mathsf{G}}$ ist.

Der Angreifer $\mathcal{B}$ erhält den öffentlichen Schlüssel $\left(p, \mathbb{G}'_p, \mathfrak{g}, \mathfrak{g}_0\right)$ des ElGamal-Verschlüsselungsverfahrens.

Simulation der Initialisierung: Der Angreifer $\mathcal{B}$ generiert die Parameter $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \hat{e}, u_0, h)$, wobei $\mathbb{G}_1 = \langle u_0 \rangle, \mathbb{G}_2 = \langle h \rangle$ und $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ eine bilineare Abbildung ist. Dann wählt $\mathcal{B}$ zufällig eine Zahl $\alpha \in_{\mathcal{R}} \mathbb{Z}_p^*$ sowie drei Generatoren $g, g_0, g_1 \in_{\mathcal{R}} \mathbb{G}_1$. Danach berechnet $\mathcal{B}$ die Elemente $u_i = u_0^{\alpha^i}$ sowie $h_i = h^{\alpha^i}$ für $1 \leq i \leq K$. (Falls $\mathbb{G}_p = \mathbb{G}_1$ ist, werden die Generatoren $g, g_0 \in \mathbb{G}_1$ nicht zufällig gewählt, sondern $g := \mathfrak{g}$ und $g_0 := \mathfrak{g}_0$ gesetzt.) Anschließend sendet $\mathcal{B}$ die perfekt simulierten Systemparameter $\mathsf{sp} = (p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \mathbb{G}'_p, \hat{e}, g, g_0, g_1, u_0, \ldots, u_K, h, h_1, \ldots, h_K, \mathfrak{g}, \mathfrak{g}_0)$ an den Angreifer $\mathcal{A}$.

Der Angreifer $\mathcal{A}$ generiert den öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}} = X$ der Bank und sendet diesen an $\mathcal{B}$.

Simulation der Probe-Phase: Der Angreifer $\mathcal{B}$ verhält sich bei allen Orakel-Anfragen wie das jeweilige Orakel und kann somit alle Anfragen perfekt simulieren.

Simulation der Challenge-Phase: Der Angreifer $\mathcal{A}$ sendet zwei verschiedene Indizes i_0, i_1 mit $(i_0, \mathbb{W}_0, I_0, \$_0), (i_1, \mathbb{W}_1, I_1, \$_1) \in \mathbb{W}_H$, zwei öffentliche Schlüssel $\mathsf{pk}_b, \mathsf{pk}_{\bar{b}}$,

zwei Zeitpunkte $\mathsf{ts}_b, \mathsf{ts}_{\bar{b}}$ (wobei sogar $(\mathsf{pk}_b, \mathsf{ts}_b) = (\mathsf{pk}_{\bar{b}}, \mathsf{ts}_{\bar{b}})$ erlaubt ist) und zwei gültige Werte $k_b, k_{\bar{b}}$ mit $\$_0, \$_1 \geq \max\{k_b, k_{\bar{b}}\}$ (wobei auch $k_b = k_{\bar{b}}$ erlaubt ist). Der Angreifer $\mathcal{B}$ simuliert die Hashwerte $R_b, R_{\bar{b}}$ gemäß dem Random-Orakel $\mathcal{O}^{\mathsf{H}}$. Wir betrachten zunächst den Fall $I_0 \neq I_1$.

Der Angreifer $\mathcal{B}$ sendet die beiden Kontonummern I_0, I_1 an das ElGamal-Verschlüsselungs-Orakel und erhält die beiden Geheimtexte $c_b = (c_{1,b}, c_{2,b}) = (I_b \mathfrak{g}_0^{r_b}, \mathfrak{g}^{r_b})$ sowie $c_{\bar{b}} = (c_{1,\bar{b}}, c_{2,\bar{b}}) = \left(I_{\bar{b}} \mathfrak{g}_0^{r_{\bar{b}}}, \mathfrak{g}^{r_{\bar{b}}}\right)$ mit $r_b, r_{\bar{b}} \in_{\mathcal{R}} \mathbb{Z}_p$ und $b, \bar{b} \in \{0,1\}$ mit $b \neq \bar{b}$.

Dann definiert $\mathcal{B}$ die Seriennummern $S_b := c_{2,b}^{R_b^{-1}}$ und $S_{\bar{b}} := c_{2,\bar{b}}^{R_{\bar{b}}^{-1}}$ sowie die Sicherheitstags $T_b := c_{1,b}$ und $T_{\bar{b}} := c_{1,\bar{b}}$.

Da die Seriennummern S_b und $S_{\bar{b}}$ also nicht mit den korrekten Geldbörsen $\mathbb{W}_b$ bzw. $\mathbb{W}_{\bar{b}}$ berechnet wurden, sind beide Seriennummern falsch berechnet. Aber seien $\kappa_b := r_b \cdot R_b^{-1} \mod p$ sowie $\kappa_{\bar{b}} := r_{\bar{b}} \cdot R_{\bar{b}}^{-1} \mod p$, dann sind die beiden Seriennummern $S_b = \mathfrak{g}^{\kappa_b}$ und $S_{\bar{b}} = \mathfrak{g}^{\kappa_{\bar{b}}}$ mit $\kappa_b, \kappa_{\bar{b}} \in_{\mathcal{R}} \mathbb{Z}_p^*$ im Random-Oracle-Modell perfekt simuliert. (Falls $\kappa_b = 0$ bzw. $\kappa_{\bar{b}} = 0$ wäre, folgt direkt $r_b = 0$ bzw. $r_{\bar{b}} = 0$ und in diesem Fall kann $\mathcal{B}$ die semantische Sicherheit sofort brechen, da dann $I_b \mathfrak{g}_0^{r_b}$ bzw. $I_{\bar{b}} \mathfrak{g}_0^{r_{\bar{b}}}$ eine der beiden Kontonummern ist.) Somit sind $T_b = I_b \mathfrak{g}_0^{R_b \kappa_b}$ und $T_{\bar{b}} = I_{\bar{b}} \mathfrak{g}_0^{R_{\bar{b}} \kappa_{\bar{b}}}$ im Random-Oracle-Modell ebenfalls perfekt simuliert.

Weiter wählt $\mathcal{B}$ zufällig die Elemente $A_{1,b}, B_{1,b}, B_{2,b}, A_{1,\bar{b}}, B_{1,\bar{b}}, B_{2,\bar{b}} \in_{\mathcal{R}} \mathbb{G}_1$, wodurch diese perfekt simuliert sind, und simuliert SoK_b und $SoK_{\bar{b}}$ perfekt. Schließlich sendet $\mathcal{B}$ die beiden Münzen $\mathsf{coin}_b = (k_b, S_b, T_b, R_b, A_{1,b}, B_{1,b}, B_{2,b}, SoK_b, \mathsf{ts}_b, \mathsf{pk}_b)$ und $\mathsf{coin}_{\bar{b}} = \left(k_{\bar{b}}, S_{\bar{b}}, T_{\bar{b}}, R_{\bar{b}}, A_{1,\bar{b}}, B_{1,\bar{b}}, B_{2,\bar{b}}, SoK_{\bar{b}}, \mathsf{ts}_{\bar{b}}, \mathsf{pk}_{\bar{b}}\right)$ an den Angreifer $\mathcal{A}$.

Der Angreifer $\mathcal{A}$ kann alle k_b Serienschlüssel der Münze coin_b aus der Seriennummer S_b berechnen. Da diese Seriennummer falsch berechnet wurde, sind auch die Serienschlüssel falsch, denn sie stammen nicht aus der Geldbörse $\mathbb{W}_b$. Da jedoch der Akkumulator der Geldbörse $\mathbb{W}_b$ perfekt verborgen ist und auch jeweils der Zeuge $W_{I,l}$ sowie die Berechnung des Polynoms $\phi_I(l)$ in den Bezahl- und Einlöseprotokollen perfekt verborgen sind, sind die k_b Serienschlüssel im Random-Oracle-Modell perfekt simuliert. Analog sind die $k_{\bar{b}}$ Serienschlüssel der Münze $\mathsf{coin}_{\bar{b}}$ perfekt simuliert.

Es ist allerdings zu beachten, dass $\mathcal{B}$ die beiden Geldbörsen $\mathbb{W}_0$ und $\mathbb{W}_1$ nicht korrekt aktualisieren kann, da $\mathcal{B}$ nicht weiß, welcher der Geldbeträge k_b bzw. $k_{\bar{b}}$ zur welcher Geldbörse gehört. Somit aktualisiert $\mathcal{B}$ die beiden Geldbörsen $\mathbb{W}_0$ und $\mathbb{W}_1$ jeweils so, als würde bei beiden mit dem Geldbetrag $\max\{k_b, k_{\bar{b}}\}$ bezahlt. Daher gilt für die Einträge nun $(i_0, \mathbb{W}'_0, I_0, \$'_0 = \$_0 - \max\{k_b, k_{\bar{b}}\})$ bzw. $(i_1, \mathbb{W}'_1, I_1, \$'_1 = \$_1 - \max\{k_b, k_{\bar{b}}\})$ und somit wurde ein Eintrag korrekt und der andere falsch aktualisiert.

Simulation der Post-Challenge-Phase: Der Angreifer $\mathcal{B}$ verhält sich bei allen Orakel-Anfragen, außer bei $\mathcal{O}_{\mathcal{K}}^{\mathsf{S}^\star}$ und $\mathcal{O}_{\mathcal{K},\mathcal{H}}^{\mathsf{S}^\star}$, wie das jeweilige Orakel und kann somit

diese Anfragen perfekt simulieren.

Simulation von $\mathcal{O}_{\mathcal{K}}^{\mathsf{S}^\star}$ und $\mathcal{O}_{\mathcal{K},\mathcal{H}}^{\mathsf{S}^\star}$: Für jeden Index $i \notin \{i_0, i_1\}$ verhält sich der Angreifer $\mathcal{B}$ wie das Orakel.

Falls der Index $i \in \{i_0, i_1\}$ ist, kann sich der Angreifer $\mathcal{B}$ nicht genau wie das Orakel verhalten, da $\mathcal{B}$ die beiden Geldbörsen $\mathbb{W}_0$ und $\mathbb{W}_1$ wie oben beschrieben anders als die Orakel aktualisiert. Der Angreifer $\mathcal{B}$ verhält sich aber wie der entsprechende ehrliche Kunde mit aktualisierter Geldbörse, wodurch die Anfrage im Random-Oracle-Modell perfekt simuliert ist, da der Angreifer $\mathcal{A}$ in dieser Phase bzgl. $\hat{b} \in \{0,1\}$ nur Münzen im Gesamtwert von $\$_{\hat{b}} - \max\{k_b, k_{\bar{b}}\}$ anfragen darf.

Spielende: Der Angreifer $\mathcal{A}$ gibt mit Wahrscheinlichkeit $1/2 + \epsilon(\lambda)$ das korrekte Bit b aus, wobei ϵ eine nicht vernachlässigbare Funktion ist. Der Angreifer $\mathcal{B}$ leitet das Bit b weiter an das ElGamal-Verschlüsselungs-Orakel und bricht somit die semantische Sicherheit des ElGamal-Verschlüsselungsverfahrens mit derselben Wahrscheinlichkeit $1/2 + \epsilon(\lambda)$.

Für den Fall $I_0 = I_1$ kann der Angreifer $\mathcal{B}$ nicht das ElGamal-Verschlüsselungs-Orakel befragen. Allerdings folgt direkt aus dem obigen Beweis, dass der Angreifer $\mathcal{A}$ in diesem Fall nur raten kann, da die beiden Münzen coin_b und $\mathsf{coin}_{\bar{b}}$ dann für beide Indizes perfekt simuliert und perfekt ununterscheidbar sind.

Bemerkung 6.5.3. *Für die Anonymität des Geldsystems ist es also wichtig, dass die Hashfunktion* H *eine Einwegfunktion mit zufälliger, gleichverteilter Ausgabe ist.*

Double-Spending-Beschuldigung: Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Double-Spending-Beschuldigung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das DLog-Problem in der Gruppe $\mathbb{G}_p$ zu lösen.

Der Angreifer $\mathcal{B}$ erhält die DLog-Probleminstanz $\left(p, \mathbb{G}_p', \mathfrak{g}, \mathfrak{g}^a\right)$ und soll den diskreten Logarithmus $a \in \mathbb{Z}_p^*$ berechnen.

Simulation der Initialisierung: Der Angreifer $\mathcal{B}$ generiert die Parameter $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, h)$, wobei $\mathbb{G}_1 = \langle u_0 \rangle$, $\mathbb{G}_2 = \langle h \rangle$ und $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ eine bilineare Abbildung ist. Dann wählt $\mathcal{B}$ zufällig zwei Zahlen $\alpha, y \in_{\mathcal{R}} \mathbb{Z}_p^*$ und drei Generatoren $g, g_0, g_1 \in_{\mathcal{R}} \mathbb{G}_1$. Danach berechnet $\mathcal{B}$ die Generatoren $\mathfrak{g}_0 = \mathfrak{g}^y, u_i = u_0^{\alpha^i}$ sowie $h_i = h^{\alpha^i}$ für $1 \leq i \leq K$. Anschließend sendet $\mathcal{B}$ die perfekt simulierten Systemparameter $\mathsf{sp} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \mathbb{G}_p', \hat{e}, g, g_0, g_1, u_0, \ldots, u_K, h, h_1, \ldots, h_K, \mathfrak{g}, \mathfrak{g}_0)$ an den Angreifer $\mathcal{A}$.

Der Angreifer $\mathcal{A}$ generiert den öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}} = X$ der Bank und sendet diesen an $\mathcal{B}$.

Simulation von $\mathcal{O}^{\mathsf{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{A}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{A}}_{\mathcal{K}}$: Der Angreifer $\mathcal{B}$ wählt zufällig eine Zahl $\gamma \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die perfekt simulierte Kontonummer $I = \mathfrak{g}^a\mathfrak{g}^\gamma = \mathfrak{g}^{a+\gamma} = \mathfrak{g}^u$ für $u := a + \gamma \mod p$. Dann speichert $\mathcal{B}$ den Eintrag (I, γ) in einer Liste $\mathcal{U}_H$.

Simulation von $\mathcal{O}^{\mathsf{W}}_{\mathcal{K}}$: Der Angreifer $\mathcal{B}$ wählt zufällig die Zahlen $\kappa_{0,0} \in_{\mathcal{R}} \mathbb{Z}_p^*, a_0 \in_{\mathcal{R}} \mathbb{Z}_p$ sowie das Element $C \in_{\mathcal{R}} \mathbb{G}_1$ und berechnet das Element A_0 gemäß Protokoll. Somit sind die Elemente C, A_0 perfekt simuliert. Anschließend simuliert $\mathcal{B}$ die Signature-of-Knowledge SoK_{W} perfekt. Nach erfolgreicher Durchführung wird ein Eintrag $(i, \mathbb{W}_i = (\Sigma, e, s'', \mathsf{info}), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$ sowie ein Eintrag $(i, \mathbb{T} = (I, C, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K))$ in einer Liste $\mathbb{T}_H$ gespeichert und der Zähler i um 1 erhöht.

Falls die Anfrage nicht erfolgreich durchgeführt wurde, wird nur ein Eintrag $(i, \mathbb{T} = (I, C, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K))$ in einer Liste $\mathbb{T}_H$ gespeichert und der Zähler i um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{S}^\star}_{\mathcal{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird u. a. ein Eintrag (j, coin) in einer Liste $\mathbb{C}_{\mathcal{A}}$ gespeichert und der Zähler j um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{S}^\star}_{\mathcal{K}}$: Der Angreifer $\mathcal{B}$ verwendet den Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i) \in \mathbb{W}_H$, berechnet die Elemente S, A_1, B_1, B_2 gemäß Protokoll und berechnet das korrekte Sicherheitstag $T = I\mathfrak{g}_0^{R\kappa}$ ohne Kenntnis des privaten Schlüssels u, wobei $\kappa \in \mathbb{Z}_p^*$ der Schlüssel mit $S = \mathfrak{g}^\kappa$ ist. Dann simuliert $\mathcal{B}$ die Signature-of-Knowledge SoK perfekt. Anschließend wird der Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i)$ zu $(i, \mathbb{W}'_i, \mathsf{pk}_{\mathcal{K}}, \$_i - k)$ aktualisiert und ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_{H,\mathcal{A}}$ gespeichert.

Simulation von $\mathcal{O}^{\mathsf{S}^\star}_{\mathcal{K},\mathcal{H}}$: Der Angreifer $\mathcal{B}$ verwendet den Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i) \in \mathbb{W}_H$, berechnet die Elemente S, A_1, B_1, B_2 gemäß Protokoll und berechnet das korrekte Sicherheitstag $T = I\mathfrak{g}_0^{R\kappa}$ ohne Kenntnis des privaten Schlüssels u, wobei $\kappa \in \mathbb{Z}_p^*$ der Schlüssel mit $S = \mathfrak{g}^\kappa$ ist. Dann simuliert $\mathcal{B}$ die Signature-of-Knowledge SoK perfekt. Anschließend wird der Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i)$ zu $(i, \mathbb{W}'_i, \mathsf{pk}_{\mathcal{K}}, \$_i - k)$ aktualisiert und ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_H$ sowie ein Eintrag (j, coin) in einer Liste $\mathbb{C}$ gespeichert und der Zähler j um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{D}}_{\mathcal{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Spielende: Der Angreifer $\mathcal{A}$ gibt nun mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ einen öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H$ sowie einen Beweis Π_G, der zwei Münzen $\mathsf{coin} = (k, S, T, R, A_1, B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}})$ und $\mathsf{coin}' = (k', S', T', R', A'_1, B'_1, B'_2, SoK', \mathsf{ts}', \mathsf{pk}'_{\mathcal{H}})$ enthält, aus, so dass $\mathsf{VerifyGuilt}(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{K}}, \Pi_G) = 1$ gilt, obwohl nicht $(\cdot, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin}), (\cdot, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin}') \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ ist.

Zunächst wird gezeigt, dass $\mathcal{A}$ eine aus den $\mathcal{O}^{\mathsf{S}^\star}_{\mathcal{K}}$- oder $\mathcal{O}^{\mathsf{S}^\star}_{\mathcal{K},\mathcal{H}}$-Anfragen erhaltenen Münze $\mathsf{coin}^\star = (k^\star, S, T, R, A_1, B_1, B_2, SoK, \mathsf{ts}^\star, \mathsf{pk}^\star_{\mathcal{H}})$ mit $(\cdot, \cdot, \mathsf{coin}^\star) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ nur mit vernachlässigbarer Wahrscheinlichkeit als veränderte Münze $\mathsf{coin} = (k, S, T, R, A_1, B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}})$ mit $(\mathsf{pk}^\star_{\mathcal{H}}, \mathsf{ts}^\star, k^\star) \neq (\mathsf{pk}_{\mathcal{H}}, \mathsf{ts}, k)$ ausgeben kann.

Fall A: Falls $\mathcal{A}$ eine aus den $\mathcal{O}_{\mathcal{K}}^{\mathsf{S}^\star}$- oder $\mathcal{O}_{\mathcal{K},\mathcal{H}}^{\mathsf{S}^\star}$-Anfragen bzgl. $(\mathsf{pk}_{\mathcal{H}}^\star, \mathsf{ts}^\star, k^\star)$ erhaltene Münze $\mathsf{coin}^\star$ mit $(\mathsf{pk}_{\mathcal{H}}^\star, \mathsf{ts}^\star, k^\star) \neq (\mathsf{pk}_{\mathcal{H}}, \mathsf{ts}, k)$ ausgibt, muss $R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}}^\star || \mathsf{ts}^\star || k^\star) = \mathsf{H}(\mathsf{pk}_{\mathcal{H}} || \mathsf{ts} || k)$ gelten. Dies ist jedoch nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Analog gilt dies für die Münze coin'.

Dann gibt es die drei Möglichkeiten, dass $\mathcal{B}$ beide Münzen bzgl. verschiedener Kunden zum Bezahlen verwendet hat (Fall B), oder keine (Fall C) oder genau eine (Fall D) der Münzen zum Bezahlen verwendet hat. Sei $(\mathsf{pk}_{\mathcal{K}}, \gamma_i) \in \mathcal{U}_H$ der entsprechende Eintrag mit $\mathsf{pk}_{\mathcal{K}} = I = \mathfrak{g}^{u_i} = \mathfrak{g}^{a+\gamma_i}$.

Fall B: $(\cdot, \mathsf{pk}'_{\mathcal{K}}, \mathsf{coin}), (\cdot, \mathsf{pk}''_{\mathcal{K}}, \mathsf{coin}') \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ mit $\mathsf{pk}'_{\mathcal{K}} \neq \mathsf{pk}''_{\mathcal{K}}$:

Die Wahrscheinlichkeit, dass $\mathcal{B}$ beide Münzen bzgl. verschiedener Kunden zum Bezahlen verwendet hat, ist vernachlässigbar, da $\mathcal{B}$ dazu entweder denselben Master-Schlüssel gewählt haben oder eine Kollision bei der Simulation des Random-Oracles auftreten müsste.

Fall C: $(\cdot, \cdot, \mathsf{coin}), (\cdot, \cdot, \mathsf{coin}') \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$:

In diesem Fall wurden SoK und SoK' vom Angreifer $\mathcal{A}$ erstellt. Daher kann $\mathcal{B}$ die Zahlen $u, u', \kappa, \kappa' \in \mathbb{Z}_p$ mit $S = \mathfrak{g}^\kappa, S' = \mathfrak{g}^{\kappa'}, T = \mathfrak{g}^u \mathfrak{g}_0^{R\kappa}$ und $T' = \mathfrak{g}^{u'} \mathfrak{g}_0^{R'\kappa'}$ aus SoK und SoK' extrahieren.

- Falls $k = k'$ ist, folgt $\kappa = \kappa'$ und weiter mit dem $\mathsf{Identify}$-Algorithmus:

$$\mathsf{pk}_{\mathcal{K}} = \left(\frac{T'^R}{T^{R'}}\right)^{(R-R')^{-1}} = \left(\frac{\mathfrak{g}^{Ru'} \mathfrak{g}_0^{RR'\kappa'}}{\mathfrak{g}^{R'u} \mathfrak{g}_0^{RR'\kappa}}\right)^{(R-R')^{-1}} = \mathfrak{g}^{\frac{Ru'-R'u}{R-R'}}.$$

  Dann ist $u_i = \frac{Ru'-R'u}{R-R'} \mod p$ und somit berechnet $\mathcal{B}$ den diskreten Logarithmus $a = \frac{Ru'-R'u}{R-R'} - \gamma_i \mod p$, da $R \neq R'$ gelten muss.
- Falls $k \neq k'$ ist, sei o.B.d.A. $k > k'$. Dann folgt mit dem $\mathsf{Identify}$-Algorithmus:

$$\mathsf{pk}_{\mathcal{K}} = \frac{T'}{\mathfrak{g}_0^{R'\kappa'}} = \frac{\mathfrak{g}^{u'} \mathfrak{g}_0^{R'\kappa'}}{\mathfrak{g}_0^{R'\kappa'}} = \mathfrak{g}^{u'}.$$

  Somit ist $u_i = u'$ und $\mathcal{B}$ berechnet den diskreten Logarithmus $a = u' - \gamma_i \mod p$.

Fall D: Sei o.B.d.A. $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(\cdot, \cdot, \mathsf{coin}') \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$:

Falls $\mathcal{B}$ genau eine der beiden Münzen zum Bezahlen verwendet hat, sei dies o.B.d.A. die Münze coin. Dann wurde SoK' vom Angreifer $\mathcal{A}$ erstellt und daher kann $\mathcal{B}$ die Zahlen $u', \kappa' \in \mathbb{Z}_p$ mit $S' = \mathfrak{g}^{\kappa'}$ und $T' = \mathfrak{g}^{u'} \mathfrak{g}_0^{R\kappa'}$ aus SoK' extrahieren. Sei $\mathsf{pk}_{\mathcal{K}_j}$ der Kunde mit $\left(\cdot, \mathsf{pk}_{\mathcal{K}_j}, \mathsf{coin}\right) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $\left(\mathsf{pk}_{\mathcal{K}_j}, \gamma_j\right) \in \mathcal{U}_H$. Somit ist $\mathsf{pk}_{\mathcal{K}_j} = \mathfrak{g}^{a+\gamma_j} =: \mathfrak{g}^{u_j}$.

- Falls $k = k'$ ist, folgt $\kappa = \kappa'$ und weiter mit dem Identify-Algorithmus:

$$\mathsf{pk}_{\mathcal{K}} = \left(\frac{T'^R}{T^{R'}}\right)^{(R-R')^{-1}} = \left(\frac{\mathfrak{g}^{Ru'}\mathfrak{g}_0^{RR'\kappa'}}{\mathfrak{g}^{R'u_j}\mathfrak{g}_0^{RR'\kappa}}\right)^{(R-R')^{-1}} = \mathfrak{g}^{\frac{Ru'-R'u_j}{R-R'}}.$$

  Dann gilt $u_i = \frac{Ru'-R'u_j}{R-R'} \mod p$ und somit $a + \gamma_i = \frac{Ru'-R'(a+\gamma_j)}{R-R'} \mod p$. Daher berechnet $\mathcal{B}$ den diskreten Logarithmus $a = \frac{Ru'-R'\gamma_j-\gamma_i(R-R')}{R} \mod p$, da $R \neq 0$ ist.

- Falls $k \neq k'$ ist, müssen wir die beiden Fälle $k > k'$ sowie $k < k'$ unterscheiden.
  - Sei $k > k'$. Dann folgt mit dem Identify-Algorithmus erneut $\mathsf{pk}_{\mathcal{K}} = \frac{T'}{\mathfrak{g}_0^{R'\kappa'}} = \mathfrak{g}^{u'}$. Somit ist $u_i = u'$ und $\mathcal{B}$ berechnet den diskreten Logarithmus $a = u' - \gamma_i \mod p$.
  - Sei $k < k'$. Dann kann aus der Seriennummer S' die Seriennummer S berechnet werden, obwohl der Angreifer $\mathcal{A}$ das Element S' nicht auf eine Orakel-Anfrage erhalten hat (sonst hätte der entsprechende Kunde ja die beiden Seriennummern S und S' mehrfach eingelöst und es würde $(\cdot, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin}), (\cdot, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin}') \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ gelten). Dies ist aber im Random-Oracle-Modell nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Somit löst $\mathcal{B}$ das DLog-Problem in $\mathbb{G}_p$ mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist.

Bemerkung 6.5.4. *Für die Double-Spending-Beschuldigung des Geldsystems ist es also wichtig, dass die Hashfunktion* H *eine Einwegfunktion und stark kollisionsresistent ist.*

Abhebe-Beschuldigung: Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Abhebe-Beschuldigung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das DLog-Problem in der Gruppe $\mathbb{G}_p$ zu lösen.

Der Angreifer $\mathcal{B}$ erhält die DLog-Probleminstanz $\left(p, \mathbb{G}'_p, \mathfrak{g}, \mathfrak{g}^a\right)$ und soll den diskreten Logarithmus $a \in \mathbb{Z}_p^*$ berechnen.

Simulation: Der Angreifer $\mathcal{B}$ simuliert die Systemparameter sowie die Orakel-Anfragen wie bei der Double-Spending-Beschuldigung.

Spielende: Der Angreifer $\mathcal{A}$ gibt nun mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ eine Protokollansicht $\mathsf{view} = (\mathbb{T} = (\mathsf{pk}_{\mathcal{K}}, C, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s''))$ mit $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H$ aus, so dass $\mathsf{VerifyWithdraw}(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{K}}, \mathsf{view}) = 1$ gilt, obwohl $(\cdot, \mathbb{T}) \notin \mathbb{T}_H$ ist. Somit wurde SoK_{W}, außer mit vernachlässigbarer Wahrscheinlichkeit, von $\mathcal{A}$ erstellt und $\mathcal{B}$ kann den privaten Schlüssel $u \in \mathbb{Z}_p$ mit $\mathsf{pk}_{\mathcal{K}} = \mathfrak{g}^u$ extrahieren und somit den diskreten Logarithmus $a = u - \gamma \mod p$ berechnen.

Somit löst $\mathcal{B}$ das DLog-Problem in $\mathbb{G}_p$ mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist. □

Bemerkung 6.5.5. *Für die Abhebe-Beschuldigung des Geldsystems ist es also wichtig, dass die Hashfunktion* H *stark kollisionsresistent ist.*

Bemerkung 6.5.6. *Die Sicherheitsziele Anonymität, Double-Spending-Beschuldigung und Abhebe-Beschuldigung sind auch dann noch gewährleistet, wenn der Angreifer* $\mathcal{A}$ *die Zahl* $\alpha \in \mathbb{Z}_p^*$ *wählt (und diese somit insbesondere kennt) und die Elemente* $u_1, \ldots, u_K \in \mathbb{G}_1$ *sowie* $h_1, \ldots, h_K \in \mathbb{G}_2$ *berechnet (vgl. [ASM08]). Denn der Angreifer* $\mathcal{B}$ *kann alle Simulationen ohne die Kenntnis der Zahl* α *durchführen. Die korrekte Berechnung der Elemente kann dann durch Überprüfung der Gleichungen* $\hat{e}(u_K, h) \stackrel{?}{=} \hat{e}(u_{K-1}, h_1) \stackrel{?}{=} \cdots \stackrel{?}{=} \hat{e}(u_1, h_{K-1}) \stackrel{?}{=} \hat{e}(u_0, h_K)$ *verifiziert werden. Des Weiteren erfüllt das Geldsystem auch die in [Tro06] definierte Anonymität, bei der der Angreifer* $\mathcal{A}$ *in der Challenge-Phase zusätzlich alle privaten Schlüssel der ehrlichen Kunden erhält.*

6.6 Das teilbare elektronische Geldsystem TG-II

In diesem Abschnitt wird das oben entwickelte teilbare elektronische Geldsystem TG-I aus Abschnitt 6.5 modifiziert, um ein effizienteres Bezahlprotokoll Spend zu erhalten. Denn da die Bank $\mathcal{B}$ im Gegensatz zu Geldsystemen, die eine blinde Signatur verwenden (wie bspw. [CFN89, Bra93, Bra94, FTY98, Sch09]), einen Teil der BBS+A-Signatur kennt und da das BBS+A-Signaturverfahren *stark* existentiell unfälschbar unter einem adaptiven Angriff mit gewählten Nachrichten ist, kann das Element Σ anstelle der Kontonummer I in die Münze coin eingebaut werden, um einen Double-Spender zu identifizieren. Dies ist jedoch nur dann möglich, wenn die Gruppe $\mathbb{G}_p := \mathbb{G}_1$ mit $\mathfrak{g} := g$ und $\mathfrak{g}_0 := g_0$ definiert wird. Im Vergleich zu dem obigen Geldsystem TG-I wird das Sicherheitstag T für eine Seriennummer $S = g^\kappa$ nun durch $T = \Sigma g_0^{R\kappa}$ berechnet, wodurch die Elemente B_1 und B_2 nicht mehr benötigt werden. Dadurch ist das Element Σ zwar nicht mehr perfekt verborgen, allerdings ist es rechnerisch verborgen, wenn für $\mathsf{Setup}_{\mathsf{Bil}}$ die XDH-Annahme gilt, da dann das DDH-Problem in der Gruppe $\mathbb{G}_1$ nur schwer gelöst werden kann. Daher ist das folgende teilbare elektronische Geldsystem TG-II nur in den Pairing Typen 2 und 3 sicher. Denn im Typ-1-Pairing könnte die Bank durch Überprüfung der Gleichung $\hat{e}(T/\Sigma, g) \stackrel{?}{=} \hat{e}\left(S^R, g_0\right)$ feststellen, ob eine Münze $\mathsf{coin} = (k, S, T, R, \Phi)$ zu dem (eindeutigen) Abhebeprotokoll bzgl. des Elements Σ gehört. Damit der Verifikationsalgorithmus $\mathsf{VerifyGuilt}$ korrekt ausgeführt werden kann, muss die Bank zusätzlich das Tripel $\sigma' = (\Sigma, e, s'')$ in ihrer Datenbank $\mathbb{D}_W$ abspeichern. Somit speichert $\mathcal{B}$ die vollständige Protokollansicht view eines Abhebeprotokolls ab.

Ein weiterer Unterschied zu dem obigen Geldsystem TG-I ist, dass die Bank $\mathcal{B}$ die folgende Signature-of-Knowledge $SoK_{\mathcal{B}}$ über die Kenntnis des geheimen Signaturschlüssels $x \in \mathbb{Z}_p^*$ erstellen und veröffentlichen muss: $SoK_{\mathcal{B}}[(x) : X = h^x]$. Alternativ könnte die

Signature-of-Knowledge auch als interaktiver Honest-Verifier-Zero-Knowledge-Proof-of-Knowledge $PoK\left[(x) : X = h^x\right]$ mit einer Trusted-Third-Party durchgeführt werden.

Bemerkung 6.6.1. *In [AWSM07] wurde bewiesen, dass das dort entwickelte spezielle Signaturverfahren existentiell unfälschbar unter einem adaptiven Angriff mit gewählten Nachrichten ist. Da das ESS+-Signaturverfahren aus Unterabschnitt 2.12.2 eine Erweiterung dieses speziellen Signaturverfahrens darstellt (vgl. [AWSM07, ASM08, Au09]), kann nicht davon ausgegangen werden, dass das ESS+-Signaturverfahren* stark *existentiell unfälschbar unter einem adaptiven Angriff mit gewählten Nachrichten ist. Es ist aber möglich, dass diese Sicherheitseigenschaft dennoch erfüllt ist.*

Wenn die ESS+-Signatur allerdings nicht stark existentiell unfälschbar unter einem adaptiven Angriff mit gewählten Nachrichten ist, ist eine entsprechende Version des folgenden Geldsystems TG-II, in der das ESS+-Signaturverfahren anstelle des BBS+A-Signaturverfahrens verwendet wird, nicht sicher.

Im Folgenden werden lediglich die Veränderungen der Algorithmen und Protokolle im Vergleich zum teilbaren Geldsystem TG-I dargestellt. Die Protokolle Account und Deposit werden analog zum teilbaren Geldsystem TG-I ausgeführt und daher nicht beschrieben.

Setup geht wie beim teilbaren Geldsystem TG-I vor, außer dass $\mathbb{G}_p := \mathbb{G}_1$ mit $\mathfrak{g} := g$ und $\mathfrak{g}_0 := g_0$ definiert werden und der Generator $g_1 \in \mathbb{G}_1$ nicht benötigt wird. Die Ausgabe sind die Systemparameter

$$\mathsf{sp} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_0, u_0, \ldots, u_K, h, h_1, \ldots, h_K, \mathsf{H}) \leftarrow \mathsf{Setup}\left(1^\lambda, K\right).$$

BGen geht wie beim teilbaren Geldsystem TG-I vor. Im Anschluss an die Schlüsselgenerierung erstellt die Bank $\mathcal{B}$ die Signature-of-Knowledge $SoK_{\mathcal{B}}\left[(x) : X = h^x\right]$ und fügt $SoK_{\mathcal{B}}$, also das Paar $(z, c) \in \mathbb{Z}_p^2$ mit $\widetilde{T} = h^z X^{-c}$ und $c = \mathsf{H}\left(X, \widetilde{T}\right)$, dem öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}}$ hinzu.

In folgender Tabelle 6.2 werden die benötigten Gruppen, deren öffentliche Generatoren, deren Verwendung und die nötigen, kryptografischen Annahmen zusammengefasst.

Gruppen	**Generatoren**	**Verwendung**	**Annahmen**
$\mathbb{G}_1$	$g, g_0, u_0, \ldots, u_K$	Seriennummern und Sicherheitstags	
$\mathbb{G}_2$	$h, h_1, \ldots, h_K, X$		
$(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$		Nguyen-Akkumulator und BBS+A-Signatur	XDH, q-SDH und q-polyDH

Tabelle 6.2: Öffentliche Parameter von TG-II

Withdraw: Der Kunde $\mathcal{K}$ berechnet das Commitment $C = g_0^{s'} u_0^{\phi(\alpha)}$ und erstellt entsprechend die Signature-of-Knowledge:

$$SoK_{\mathsf{W}}\Big[(a_0, s', u, \phi(l)) : I = g^u \wedge \hat{e}(C,h)\,\hat{e}\left(A_0, h_1 h^{-l}\right)^{-1} = \hat{e}\left(g_0^{s'} u_0^{\phi(l)+a_0 l} u_1^{-a_0}, h\right)\Big](\mathsf{ts}||K).$$

Die Bank $\mathcal{B}$ speichert zusätzlich die bekannten Werte der Signatur $\sigma' = (\Sigma, e, s'')$. Somit speichert $\mathcal{B}$ die gesamte Protokollansicht $\mathsf{view} = (\mathbb{T}, \sigma')$ in ihrer Datenbank $\mathbb{D}_W$ ab. Falls das Element Σ bereits in der Datenbank gespeichert ist, generiert die Bank eine neue Signatur, damit ein Double-Spender eindeutig identifiziert werden kann (dieser Fall ist jedoch vernachlässigbar, siehe Unterabschnitt 6.6.1).

Für die Sicherheitsziele Double-Spending-Beschuldigung und Abhebe-Beschuldigung ist es wichtig, dass $\mathcal{K}$ durch SoK_{W} auch die Gleichung $I = g^u$ beweist, obwohl die Zahl $u \in \mathbb{Z}_p^*$ nicht signiert und nicht für die Bezahlprotokolle verwendet wird.

Spend: Das Spend-Protokoll verläuft mit den oben erwähnten Veränderungen wie in Abbildung 6.13 beschrieben ab, wobei $\beta = e\kappa$ ist:

$\boxed{\mathcal{K}unde}$ $(\mathsf{pk}_{\mathcal{H}}, \mathsf{sk}_{\mathcal{K}}, \mathbb{W})$	$(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, k, \mathsf{ts})$	$\boxed{\mathcal{H}ändler}$ $(\mathsf{sk}_{\mathcal{H}})$
$R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}}\|\|\mathsf{ts}\|\|k)$		$R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}}\|\|\mathsf{ts}\|\|k)$
$a_1 \in_{\mathcal{R}} \mathbb{Z}_p$		
$\kappa := \kappa_{L-\ell,j_0}$		
$S = g^{\kappa}$		
$T = \Sigma g_0^{R\kappa}$		
$l = \mathsf{H}(S\|\|T)$		
$A_1 = W_{I,l} u_0^{a_1}$		
	$\xrightarrow{S, T, A_1}$	
$SoK\Big[(a_1, \beta, e, s, \kappa, \phi_I(l)) : S = g^{\kappa} \wedge 1 = S^e g^{-\beta} \wedge \frac{\hat{e}(T,X)}{\hat{e}(g,h)\,\hat{e}(A_1, h_{l,I})} = \hat{e}\left(g_0^{R\kappa}, X\right)\hat{e}\left(g_0^{s+R\beta} u_{l,I}^{-a_1} u_I^{\phi_I(l)} T^{-e}, h\right)\Big](R)$		
		Berechne $\kappa_{L+1,kj_0}, \ldots, \kappa_{L+1,kj_0+k-1}$
		$l = \mathsf{H}(S\|\|T)$
		Verifiziere SoK
$\mathbb{W}'$ oder $\perp$		coin oder $\perp$

Abbildung 6.13: Bezahlprotokoll Spend von TG-II

Spend-K: Analog wird das Bezahlprotokoll Spend-K wie in Abbildung 6.14 beschrieben durchgeführt:

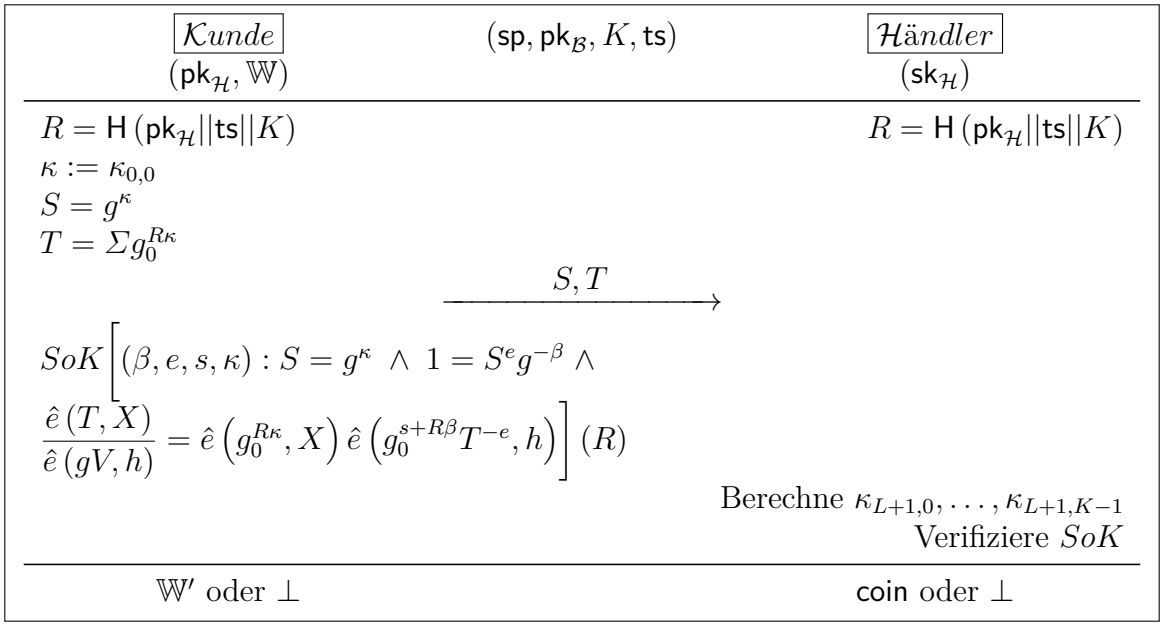

Abbildung 6.14: Bezahlprotokoll Spend-K von TG-II

Identify geht analog zum teilbaren Geldsystem TG-I vor. Allerdings ist das Ergebnis der Berechnung nun das Element Σ anstelle der Kontonummer I. Dann durchsucht $\mathcal{B}$ die Datenbank $\mathbb{D}_W$ nach der Protokollansicht $\mathsf{view} = (\mathsf{pk}_{\mathcal{K}}, \cdots, \Sigma, \cdots)$. Die Ausgabe ist entweder der öffentliche Schlüssel $\mathsf{pk}_{\mathcal{K}} \in \mathbb{D}_{\mathcal{K}}$ mit einem Beweis $\Pi_G = (\mathsf{view}, \mathsf{coin}, \mathsf{coin}')$ oder das Symbol $\perp$.

VerifyGuilt verifiziert bei Eingabe der Systemparameter sp, des öffentlichen Schlüssels $\mathsf{pk}_{\mathcal{B}}$, eines öffentlichen Schlüssels $\mathsf{pk}_{\mathcal{K}}$ und eines Beweises Π_G, der eine Protokollansicht $\mathsf{view} = (I, C, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K, \Sigma, e, s'')$ sowie zwei Münzen coin und coin' enthält, die beiden Münzen, die Signature-of-Knowledge SoK_{W} und die Signatur. Dann wird der Algorithmus Identify ausgeführt, um das Element Σ zu berechnen. Somit kann jeder verifizieren, ob der Kunde mit der Kontonummer $\mathsf{pk}_{\mathcal{K}} = I$ das Abhebeprotokoll durchgeführt und einen Serienschlüssel mehrfach ausgegeben hat.

VerifyWithdraw geht analog zum teilbaren Geldsystem TG-I vor, jedoch wird das Tripel (Σ, e, s'') nicht erneut von der Bank erzeugt, da es bereits in der Protokollansicht view gespeichert ist.

6.6.1 Sicherheitsanalyse von TG-II

Satz 6.6.1. *Das teilbare elektronische Geldsystem TG-II ist im Random-Oracle-Modell ein sicheres elektronisches Geldsystem, falls die XDH-Annahme, die q-SDH-Annahme und die q-polyDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gelten.*

Beweis. Wir beweisen Unfälschbarkeit, Anonymität, Double-Spending-Beschuldigung und Abhebe-Beschuldigung, wobei wir uns im Wesentlichen auf die Änderungen im Vergleich zu TG-I beschränken.

Unfälschbarkeit: Die Unfälschbarkeit des Geldsystems TG-II folgt analog zur Unfälschbarkeit des Geldsystems TG-I. Die Fälle A bis F sind genau wie beim vorherigen Geldsystem TG-I vernachlässigbar. Es ist lediglich zu beachten, dass eine Münze coin nun nicht mehr die Elemente B_1 und B_2 beinhaltet.

Fall (1.1): Der Beweis von Fall (1.1) läuft mit den folgenden Änderungen analog zu TG-I ab. Bei der Initialisierung wird die Gruppe $\mathbb{G}_p$ und damit die Zahl $y \in_{\mathcal{R}} \mathbb{Z}_p^*$ sowie die Generatoren $\mathfrak{g}, \mathfrak{g}_0$ nicht benötigt. Zudem wird der Generator g_1 ebenfalls nicht gebraucht. Der Angreifer $\mathcal{B}$ muss die Signature-of-Knowledge $SoK_{\mathcal{B}}$ perfekt simulieren und dem öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}}$ hinzufügen. Weiter besitzen die Nachrichten-Signatur-Paare nun die Form $(\phi(\mathsf{x}), (\Sigma, e, s))$ und $\mathcal{B}$ stellt bei der Simulation der $\mathcal{O}_{\mathcal{B}}^{\mathsf{W}}$- und $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{W}}$-Anfragen eine neue Signatur-Anfrage, falls sich das Element Σ bereits in der Liste $\mathsf{view}_{\mathcal{A}}$ oder view_H befindet. Dieser Fall ist allerdings vernachlässigbar (vgl. Double-Spending-Beschuldigung). Weiter gibt $\mathcal{B}$ bei jeder $\mathcal{O}_{\mathcal{B}}^{\mathsf{VW}}$ das in der Liste $\mathsf{view}_{\mathcal{A}}$ gespeicherte Tripel $\sigma' = (\Sigma, e, s'')$ aus.

Fall (1.2): Der Beweis von Fall (1.2) läuft mit den folgenden Änderungen ebenfalls analog zu TG-I ab. Bei der Initialisierung wird die Gruppe $\mathbb{G}_p$, die Generatoren $\mathfrak{g}, \mathfrak{g}_0$ und der Generator g_1 nicht benötigt. Der Angreifer $\mathcal{B}$ generiert die Generatoren $g = u_0^{\beta}$ sowie $g_0 = g^y$ und erstellt die Signature-of-Knowledge $SoK_{\mathcal{B}}$. Entsprechend wird bei jeder $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{W}}$-Anfrage das Commitment $C = (u_0^a)^{\beta y} g_0^{\delta} u_0^{\phi(\alpha)} = g_0^a g_0^{\delta} u_0^{\phi(\alpha)} = g_0^{s'} u_0^{\phi(\alpha)}$ für $s' := a + \delta \mod p$ berechnet. Weiter besitzen die Nachrichten-Signatur-Paare nun die Form $(\phi(\mathsf{x}), (\Sigma, e, s))$ und $\mathcal{B}$ erstellt bei der Simulation der $\mathcal{O}_{\mathcal{B}}^{\mathsf{W}}$- und $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{W}}$-Anfragen eine neue Signatur, falls sich das Element Σ bereits in der Liste $\mathsf{view}_{\mathcal{A}}$ oder view_H befindet. Dieser Fall ist allerdings vernachlässigbar. Weiter gibt $\mathcal{B}$ bei jeder $\mathcal{O}_{\mathcal{B}}^{\mathsf{VW}}$ das in der Liste $\mathsf{view}_{\mathcal{A}}$ gespeicherte Tripel $\sigma' = (\Sigma, e, s'')$ aus.

Fall (2): Wir betrachten nun den Fall, dass einige Serienschlüssel mehrmals ausgegeben oder eingelöst werden, ohne dass ein betrügerischer Kunde $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_{\mathcal{A}}$ identifiziert werden kann. Seien $\mathsf{coin} = (k, S, T, R, \Phi = (A_1, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$ und $\mathsf{coin}' =$

$(k', S', T', R', \Phi' = (A'_1, SoK', \mathsf{ts}', \mathsf{pk}'_{\mathcal{H}}))$ die beiden Münzen, bei denen ein Double-Spending auftrat. Da $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts}) \neq (\mathsf{pk}'_{\mathcal{H}}, \mathsf{ts}')$ gelten muss, ist nur mit vernachlässigbarer Wahrscheinlichkeit $R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}}||\mathsf{ts}||k) = \mathsf{H}(\mathsf{pk}'_{\mathcal{H}}||\mathsf{ts}'||k') = R'$. Sei $\Sigma^\star \notin \mathsf{view}_{\mathcal{A}}$ das Ergebnis der Berechnung des Identify-Algorithmus.

Durch die obigen sechs Fälle A bis F wurde bereits gezeigt, dass nur mit vernachlässigbarer Wahrscheinlichkeit $(\cdot, \cdot, \mathsf{coin}), (\cdot, \cdot, \mathsf{coin}') \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ gilt. Dann gibt es die beiden Fälle, dass (2.1) keine oder (2.2) genau eine der Münzen coin und coin′ von $\mathcal{B}$ zum Bezahlen verwendet wurde.

Fall (2.1): $(\cdot, \cdot, \mathsf{coin}), (\cdot, \cdot, \mathsf{coin}') \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$:

In diesem Fall wurden SoK und SoK' vom Angreifer $\mathcal{A}$ erstellt. Daher kann $\mathcal{B}$ die beiden Zahlen $\kappa, \kappa' \in \mathbb{Z}_p$ sowie die beiden Nachrichten-Signatur-Paare $(\phi(\mathsf{x}), (\Sigma, e, s))$ und $(\phi'(\mathsf{x}), (\Sigma', e', s'))$ mit $S = g^\kappa, S' = g^{\kappa'}, T = \Sigma g_0^{R\kappa}$ und $T' = \Sigma' g_0^{R'\kappa'}$ aus SoK und SoK' extrahieren.

- Falls $k = k'$ und $S = S'$ ist, folgt $\kappa = \kappa'$ und mit dem Identify-Algorithmus:

$$\Sigma^\star = \left(\frac{T'^R}{T^{R'}}\right)^{(R-R')^{-1}} = \left(\frac{\Sigma'^R g_0^{RR'\kappa'}}{\Sigma^{R'} g_0^{RR'\kappa}}\right)^{(R-R')^{-1}} = \left(\frac{\Sigma'^R}{\Sigma^{R'}}\right)^{(R-R')^{-1}} \notin \mathsf{view}_{\mathcal{A}}.$$

  - Falls mindestens eines der beiden Nachrichten-Signatur-Paare niemals bei einer Orakel-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.1) vor.
  - Falls mindestens eines der beiden Nachrichten-Signatur-Paare bei einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.2) vor.
  - Falls obige Fälle nicht eintreten, wurden beide Nachrichten-Signatur-Paare bei jeweils einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{B}}$-Anfrage erstellt. Aus $\Sigma^\star \notin \mathsf{view}_{\mathcal{A}}$ folgt $\Sigma' \neq \Sigma$. Somit handelt es sich nicht um ein Double-Spending, da entweder von einem oder von zwei korrupten Kunden zwei verschiedene Geldbörsen $\mathbb{W} = (\Sigma, \cdots)$ und $\mathbb{W}' = (\Sigma', \cdots)$ abgehoben wurden. Insbesondere kann $\mathcal{A}$ in diesem Fall aber nicht Münzen im Wert von mehr als \$ ausgeben oder einlösen und somit das Spiel nicht gewinnen.

- Falls $k \neq k'$, sei. o.B.d.A. $k > k'$. Falls der zu S' gehörende Schlüssel $\kappa' \in \mathbb{Z}_p$ mit $S' = \mathfrak{g}^{\kappa'}$ aus der Seriennummer S berechnet werden kann, folgt mit dem Identify-Algorithmus:

$$\Sigma^\star = \frac{T'}{g_0^{R'\kappa'}} = \frac{\Sigma' g_0^{R'\kappa'}}{g_0^{R'\kappa'}} = \Sigma' \notin \mathsf{view}_{\mathcal{A}}.$$

  Da $\Sigma' \notin \mathsf{view}_{\mathcal{A}}$ ist, wurde das Nachrichten-Signatur-Paar $(\phi'(\mathsf{x}), (\Sigma', e', s'))$ niemals bei einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{B}}$-Anfrage erstellt.

  - Falls das Nachrichten-Signatur-Paar $(\phi'(\mathsf{x}), (\Sigma', e', s'))$ niemals bei einer Orakel-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.1) vor.

  - Falls das Nachrichten-Signatur-Paar $(\phi'(\mathsf{x}), (\Sigma', e', s'))$ bei einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.2) vor.

- Die Fälle, dass $k = k'$ ist, aber $S \neq S'$ gilt, oder o.B.d.A. $k > k'$ ist, aber der zu S' gehörende Schlüssel $\kappa' \in \mathbb{Z}_p$ mit $S' = \mathfrak{g}^{\kappa'}$ nicht aus der Seriennummer S berechnet werden kann, sind genau wie beim Geldsystem TG-I nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Fall (2.2): Sei o.B.d.A. $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(\cdot, \cdot, \mathsf{coin}') \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$:

Falls $\mathcal{B}$ genau eine der beiden Münzen zum Bezahlen verwendet hat, sei dies o.B.d.A. die Münze coin. Dann wurde SoK' vom Angreifer $\mathcal{A}$ erstellt und daher kann $\mathcal{B}$ die Zahl $\kappa' \in \mathbb{Z}_p$ sowie das Nachrichten-Signatur-Paar $(\phi'(\mathsf{x}), (\Sigma', e', s'))$ mit $S' = g^{\kappa'}$ und $T' = \Sigma' g_0^{R'\kappa'}$ aus SoK' extrahieren.

Sei $\kappa \in \mathbb{Z}_p^*$ und $(\phi(\mathsf{x}), (\Sigma, e, s))$ das Nachrichten-Signatur-Paar der Münze coin mit $S = g^{\kappa}$ und $T = \Sigma g_0^{R\kappa}$.

- Falls $k = k'$ und $S = S'$ ist, folgt $\kappa = \kappa'$ und mit dem $\mathsf{Identify}$-Algorithmus:

$$\Sigma^\star = \left(\frac{T'^R}{T^{R'}}\right)^{(R-R')^{-1}} = \left(\frac{\Sigma'^R g_0^{RR'\kappa'}}{\Sigma^{R'} g_0^{RR'\kappa}}\right)^{(R-R')^{-1}} = \left(\frac{\Sigma'^R}{\Sigma^{R'}}\right)^{(R-R')^{-1}} \notin \mathsf{view}_{\mathcal{A}}.$$

  - Falls das Nachrichten-Signatur-Paar $(\phi'(\mathsf{x}), (\Sigma', e', s'))$ niemals bei einer Orakel-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.1) vor.
  - Falls das Nachrichten-Signatur-Paar $(\phi'(\mathsf{x}), (\Sigma', e', s'))$ bei einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.2) vor.
  - Falls obige Fälle nicht eintreten, wurde das Nachrichten-Signatur-Paar $(\phi'(\mathsf{x}), (\Sigma', e', s'))$ bei einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{B}}$-Anfrage erstellt. Somit handelt es sich nicht um ein Double-Spending, da zwei Kunden mit $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H$ und $\mathsf{pk}'_{\mathcal{K}} \in \mathcal{U}_{\mathcal{A}}$ zwei Geldbörsen $\mathbb{W} = (\Sigma, \cdots)$ und $\mathbb{W}' = (\Sigma', \cdots)$ abgehoben haben. Dazu müsste $\mathcal{A}$ entweder denselben Master-Schlüssel $\kappa_{0,0}$ wie $\mathsf{pk}_{\mathcal{K}}$ wählen oder es tritt eine Kollision bei der Simulation des Random-Oracles auf, was beides nur mit vernachlässigbarer Wahrscheinlichkeit möglich ist. Insbesondere kann $\mathcal{A}$ in diesem Fall aber nicht Münzen im Wert von mehr als \$ ausgeben oder einlösen und somit das Spiel nicht gewinnen.

- Falls $k > k'$ ist und der zu S' gehörende Schlüssel $\kappa' \in \mathbb{Z}_p$ mit $S' = g^{\kappa'}$ aus der Seriennummer S berechnet werden kann, folgt mit dem $\mathsf{Identify}$-Algorithmus:

$$\Sigma^\star = \frac{T'}{g_0^{R'\kappa'}} = \frac{\Sigma' g_0^{R'\kappa'}}{g_0^{R'\kappa'}} = \Sigma' \notin \mathsf{view}_{\mathcal{A}}.$$

  Da $\Sigma^\star \notin \mathsf{view}_{\mathcal{A}}$ ist, wurde das Nachrichten-Signatur-Paar $(\phi'(\mathsf{x}), (\Sigma', e', s'))$ niemals bei einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{B}}$-Anfrage erstellt.

  - Falls das Nachrichten-Signatur-Paar $(\phi'(\mathsf{x}), (\Sigma', e', s'))$ niemals bei einer Orakel-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.1) vor.
  - Falls das Nachrichten-Signatur-Paar $(\phi'(\mathsf{x}), (\Sigma', e', s'))$ bei einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.2) vor.

- Falls $k < k'$ ist und der zu S gehörende Schlüssel $\kappa \in \mathbb{Z}_p$ mit $S = g^{\kappa}$ aus der Seriennummer S' berechnet werden kann, folgt mit dem $\mathsf{Identify}$-Algorithmus:

$$\Sigma^{\star} = \frac{T}{g_0^{R\kappa}} = \frac{\Sigma g_0^{R\kappa}}{g_0^{R\kappa}} = \Sigma \notin \mathsf{view}_{\mathcal{A}}.$$

  - Falls das Nachrichten-Signatur-Paar $(\phi'(\mathsf{x}), (\Sigma', e', s'))$ niemals bei einer Orakel-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.1) vor.
  - Falls das Nachrichten-Signatur-Paar $(\phi'(\mathsf{x}), (\Sigma', e', s'))$ bei einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.2) vor.
  - Falls obige Fälle nicht eintreten, wurde das Nachrichten-Signatur-Paar $(\phi'(\mathsf{x}), (\Sigma', e', s'))$ bei einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{B}}$-Anfrage erstellt. Somit handelt es sich, wie bereits im obigen Fall, nicht um ein Double-Spending, da zwei Kunden mit $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H$ und $\mathsf{pk}'_{\mathcal{K}} \in \mathcal{U}_{\mathcal{A}}$ zwei Geldbörsen abgehoben haben.

- Die Fälle, dass $k = k'$ ist, aber $S \neq S'$ gilt, oder $k > k'$ ist, aber der zu S' gehörende Schlüssel $\kappa' \in \mathbb{Z}_p$ mit $S' = g^{\kappa'}$ nicht aus der Seriennummer S berechnet werden kann, oder $k < k'$ ist, aber der zu S gehörende Schlüssel $\kappa \in \mathbb{Z}_p$ mit $S = g^{\kappa}$ nicht aus der Seriennummer S' berechnet werden kann, sind wie beim Geldsystem TG-I nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Somit ist der Fall, dass einige Serienschlüssel mehrmals eingelöst werden, ohne dass ein betrügerischer Kunde $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_{\mathcal{A}}$ eindeutig identifiziert werden kann, vernachlässigbar.

Daher bricht der Angreifer $\mathcal{A}$ die Unfälschbarkeit nur mit vernachlässigbarer Wahrscheinlichkeit.

Anonymität: Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Anonymität des Systems bricht. Wir konstruieren einen probabilistischen, polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, die semantische Sicherheit des ElGamal-Verschlüsselungsverfahrens in der Gruppe $\mathbb{G}_1$ zu brechen, was ein Widerspruch zur XDH-Annahme für $\mathsf{Setup}_{\mathsf{Bil}}$ ist.

Der Angreifer $\mathcal{B}$ erhält die Parameter $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \hat{e}, u_0, h)$ sowie das Paar $(g, g_0) \in \mathbb{G}_1^2$, wobei $(p, \mathbb{G}'_1, g, g_0)$ der öffentliche Schlüssel des ElGamal-Verschlüsselungsverfahrens ist.

Der Anonymitätsbeweis verläuft analog zu dem des teilbaren Geldsystems TG-I aus dem obigen Abschnitt 6.5, wobei $\mathbb{G}_p = \mathbb{G}_1, \mathfrak{g} = g$ und $\mathfrak{g}_0 = g_0$ gelten. Der Angreifer $\mathcal{B}$

sendet in der Challenge-Phase jedoch nicht die beiden Kontonummern $I_0, I_1 \in \mathbb{G}_p$, sondern die zu den beiden Indizes i_0 und i_1 gehörenden Elemente $\Sigma_0, \Sigma_1 \in \mathbb{G}_1$ mit $(i_0, \mathbb{W}_0 = (\Sigma_0, \cdots), I_0, \cdot), (i_1, \mathbb{W}_1 = (\Sigma_1, \cdots), I_1, \cdot) \in \mathbb{W}_H$ an das ElGamal-Verschlüsselungs-Orakel. Nach dem Ablauf des Abhebeprotokolls gilt $\Sigma_0 \neq \Sigma_1$. Entsprechend erhält $\mathcal{B}$ die beiden Geheimtexte $c_b = (c_{1,b}, c_{2,b}) = (\Sigma_b g_0^{r_b}, g^{r_b})$ sowie $c_{\bar{b}} = (c_{1,\bar{b}}, c_{2,\bar{b}}) = \left(\Sigma_{\bar{b}} g_0^{r_{\bar{b}}}, g^{r_{\bar{b}}}\right)$ mit $r_b, r_{\bar{b}} \in_{\mathcal{R}} \mathbb{Z}_p$ und $b, \bar{b} \in \{0,1\}$ mit $b \neq \bar{b}$. Des Weiteren fallen die Elemente $B_{1,b}, B_{2,b}, B_{1,\bar{b}}$, $B_{2,\bar{b}} \in \mathbb{G}_1$ bei den beiden Münzen coin_b und $\mathsf{coin}_{\bar{b}}$ weg. Der verbleibende Beweis verläuft analog zu dem des teilbaren Geldsystems TG-I.

Es ist lediglich zu beachten, dass nun in der Gruppe $\mathbb{G}_1$ das DDH-Problem schwer sein muss, also die XDH-Annahme für $\mathsf{Setup}_{\mathsf{Bil}}$ gelten muss.

Double-Spending-Beschuldigung: Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Double-Spending-Beschuldigung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das DLog-Problem in der Gruppe $\mathbb{G}_1$ zu lösen.

Der Angreifer $\mathcal{B}$ erhält die Parameter $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, h)$ sowie das Paar $(g, g_0) \in \mathbb{G}_1^2$ und soll das Darstellungsproblem bzgl. des Generatortupels (g, g_0, u_0) lösen. Seien $a, b \in \mathbb{Z}_p^*$ die diskreten Logarithmen mit $g_0 = g^a$ und $u_0 = g_0^b$.

Vereinfacht formuliert, kennt $\mathcal{B}$ nach der Ausgabe von $\mathcal{A}$ sechs Zahlen $a_1, a_2, a_3, a_1', a_2'$, $a_3' \in \mathbb{Z}_p$ mit $g^{a_1} g_0^{a_2} u_0^{a_3} = g^{a_1'} g_0^{a_2'} u_0^{a_3'}$. Daher müssen wir die beiden Fälle (1) $a_1 \neq a_1' \vee a_2 \neq a_2'$ und (2) $a_1 = a_1' \wedge a_2 = a_2'$ unterscheiden. Im Folgenden wird die Konstruktion des Angreifers $\mathcal{B}$ detailliert beschrieben und an den nötigen Stellen die beiden Fälle einzeln behandelt.

Simulation der Initialisierung: Der Angreifer $\mathcal{B}$ wählt zufällig eine Zahl $\alpha \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Elemente $u_i = u_0^{\alpha^i}$ sowie $h_i = h^{\alpha^i}$ für $1 \leq i \leq K$. Anschließend sendet $\mathcal{B}$ die perfekt simulierten Systemparameter $\mathsf{sp} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_0, u_0, \ldots, u_K, h, h_1, \ldots, h_K)$ an den Angreifer $\mathcal{A}$.

Der Angreifer $\mathcal{A}$ generiert den öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}} = (X, SoK_{\mathcal{B}})$ der Bank und sendet diesen an $\mathcal{B}$, wobei $\mathcal{B}$ die Zahl $x \in \mathbb{Z}_p^*$ mit $X = h^x$ aus $SoK_{\mathcal{B}}$ extrahiert.

Simulation von $\mathcal{O}^{\mathsf{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{A}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}_{\mathcal{K}}^{\mathsf{A}}$: Der Angreifer $\mathcal{B}$ wählt zufällig eine Zahl $\gamma \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die perfekt simulierte Kontonummer $I = g_0 g^{\gamma} = g^{a+\gamma} = g^u$ für $u := a + \gamma \mod p$. Dann speichert $\mathcal{B}$ den Eintrag (I, γ) in einer Liste $\mathcal{U}_H$.

Simulation von $\mathcal{O}_{\mathcal{K}}^{\mathsf{W}}$: Wir unterscheiden die beiden Fälle:

Fall 1: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, außer dass $\mathcal{B}$ die Signature-of-Knowledge SoK_{W} perfekt simuliert (da $\mathcal{B}$ die Zahl $u = a + \gamma \mod p$

nicht kennt). Nach erfolgreicher Durchführung wird ein Eintrag $(i, \mathbb{W}_i = (s', \Sigma, e, s'', \mathsf{info}, \phi(\mathsf{x})), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$ sowie ein Eintrag $(i, \mathbb{T} = (I, C, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K))$ in einer Liste $\mathbb{T}_H$ gespeichert und der Zähler i um 1 erhöht.

Falls die Anfrage nicht erfolgreich durchgeführt wurde, wird ein Eintrag $(i, \mathbb{W}_i = (s', \perp, \perp, \perp, \mathsf{info}, \phi(\mathsf{x})), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$ sowie ein Eintrag $(i, \mathbb{T} = (I, C, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K))$ in einer Liste $\mathbb{T}_H$ gespeichert und der Zähler i um 1 erhöht.

Fall 2: Der Angreifer $\mathcal{B}$ wählt zufällig die Zahlen $\kappa_{0,0} \in_{\mathcal{R}} \mathbb{Z}_p^*$ sowie $a_0, \delta \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Commitment $C = u_0 g_0^{\delta} u_0^{\phi(\alpha)} = g_0^{b+\delta} u_0^{\phi(\alpha)} = g_0^{s'} u_0^{\phi(\alpha)}$ für $s' := b + \delta \mod p$ und das Element A_0 gemäß Protokoll. Anschließend simuliert $\mathcal{B}$ die Signature-of-Knowledge SoK_{W} perfekt, wodurch das gesamte Protokoll perfekt simuliert ist. Nach erfolgreicher Durchführung wird ein Eintrag $(i, \mathbb{W}_i = (\delta, \Sigma, e, s'', \mathsf{info}, \phi(\mathsf{x})), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$ sowie ein Eintrag $(i, \mathbb{T} = (I, C, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K))$ in einer Liste $\mathbb{T}_H$ gespeichert und der Zähler i um 1 erhöht.

Falls die Anfrage nicht erfolgreich durchgeführt wurde, wird ein Eintrag $(i, \mathbb{W}_i = (\delta, \perp, \perp, \perp, \mathsf{info}, \phi(\mathsf{x})), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$ sowie ein Eintrag $(i, \mathbb{T} = (I, C, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K))$ in einer Liste $\mathbb{T}_H$ gespeichert und der Zähler i um 1 erhöht.

Simulation von $\mathcal{O}_{\mathcal{H}}^{\mathsf{S}^\star}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird u. a. ein Eintrag (j, coin) in einer Liste $\mathbb{C}_{\mathcal{A}}$ gespeichert und der Zähler j um 1 erhöht.

Simulation von $\mathcal{O}_{\mathcal{K}}^{\mathsf{S}^\star}$: Wir unterscheiden die beiden Fälle:

Fall 1: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird u. a. ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_{H,\mathcal{A}}$ gespeichert.

Fall 2: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, außer bei der Signature-of-Knowledge SoK, die $\mathcal{B}$ perfekt simuliert (da $\mathcal{B}$ die Zahl $s = b + \delta + s'' \mod p$ nicht kennt). Somit wird u. a. ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_{H,\mathcal{A}}$ gespeichert.

Simulation von $\mathcal{O}_{\mathcal{K},\mathcal{H}}^{\mathsf{S}^\star}$: Wir unterscheiden die beiden Fälle:

Fall 1: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird u. a. ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_H$ sowie ein Eintrag (j, coin) in einer Liste $\mathbb{C}$ gespeichert und der Zähler j um 1 erhöht.

Fall 2: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, außer bei der Signature-of-Knowledge SoK, die $\mathcal{B}$ perfekt simuliert (da $\mathcal{B}$ die Zahl $s = b + \delta + s'' \mod p$ nicht kennt). Somit wird u. a. ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_H$ sowie ein Eintrag (j, coin) in einer Liste $\mathbb{C}$ gespeichert und der Zähler j um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{D}}_{\mathcal{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird ein Eintrag (j, coin) in einer Liste $\mathbb{D}_H$ gespeichert.

Spielende: Der Angreifer $\mathcal{A}$ gibt mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ eine Kontonummer $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H$ sowie einen Beweis Π_G, der eine Protokollansicht $\mathsf{view} = (\mathbb{T} = (\mathsf{pk}_{\mathcal{K}}, C, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s''))$ sowie zwei Münzen $\mathsf{coin} = (k, S, T, R, A_1, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}})$ und $\mathsf{coin}' = (k', S', T', R', A_1', SoK', \mathsf{ts}', \mathsf{pk}'_{\mathcal{H}})$ enthält, aus, so dass $\mathsf{VerifyGuilt}(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{K}}, \Pi_G) = 1$ gilt, obwohl nicht $(\cdot, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin}), (\cdot, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin}') \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ ist.

Der Fall A ist wie bei TG-I vernachlässigbar. Somit gibt es wie bei TG-I die drei Möglichkeiten, dass $\mathcal{B}$ beide Münzen bzgl. verschiedener Kunden (Fall B) oder keine (Fall C) oder genau eine (Fall D) der Münzen zum Bezahlen verwendet hat. Zusätzlich gibt es die vierte Möglichkeit, dass $\mathcal{B}$ die Abhebeanfrage nicht durchgeführt hat (Fall E). Sei $(\mathsf{pk}_{\mathcal{K}}, \gamma) \in \mathcal{U}_H$ der entsprechende Eintrag mit $\mathsf{pk}_{\mathcal{K}} = I = g^u = g^{a+\gamma}$.

Fall E: $(\cdot, \mathbb{T}) \notin \mathbb{T}_H$:

In diesem Fall wurde SoK_{W} vom Angreifer $\mathcal{A}$ erstellt. Daher kann $\mathcal{B}$ den privaten Schlüssel $u \in \mathbb{Z}_p$ mit $I = g^u$ extrahieren und den diskreten Logarithmus $a = u - \gamma \mod p$ berechnen.

Daher nehmen wir an, dass im Folgenden die Signature-of-Knowledge SoK_{W} stets von $\mathcal{B}$ durchgeführt wurde und somit $(\cdot, \mathbb{T}) \in \mathbb{T}_H$ ist. Dann gilt je nach Fall:

Fall 1: Es gibt einen Eintrag $(i, \mathbb{T}) \in \mathbb{T}_H$ und einen Eintrag $(i, \mathbb{W}_i = (s', \cdot, \cdot, \cdot, \mathsf{info}, \phi(\mathsf{x})), \mathsf{pk}_{\mathcal{K}}, \$_i) \in \mathbb{W}_H$. Dieser Eintrag wird zu $(i, \mathbb{W}_i = (s', \Sigma, e, s'', \mathsf{info}, \phi(\mathsf{x})), \mathsf{pk}_{\mathcal{K}}, \$_i)$ überschrieben, wobei $\sigma' = (\Sigma, e, s'')$ in der Ausgabe des Angreifers $\mathcal{A}$ enthalten ist. Sei $s := s' + s'' \mod p$, dann kennt $\mathcal{B}$ das Nachrichten-Signatur-Paar $(\phi(\mathsf{x}), (\Sigma, e, s))$.

Fall 2: Es gibt einen Eintrag $(i, \mathbb{T}) \in \mathbb{T}_H$ und einen Eintrag $(i, \mathbb{W}_i = (\delta, \cdot, \cdot, \cdot, \mathsf{info}, \phi(\mathsf{x})), \mathsf{pk}_{\mathcal{K}}, \$_i) \in \mathbb{W}_H$. Dieser Eintrag wird zu $(i, \mathbb{W}_i = (\delta, \Sigma, e, s'', \mathsf{info}, \phi(\mathsf{x})), \mathsf{pk}_{\mathcal{K}}, \$_i)$ überschrieben, wobei $\sigma' = (\Sigma, e, s'')$ in der Ausgabe des Angreifers $\mathcal{A}$ enthalten ist. Sei $s := b + \delta + s'' \mod p$ und $(\phi(\mathsf{x}), (\Sigma, e, s))$ das entsprechende Nachrichten-Signatur-Paar, welches $\mathcal{B}$ nicht kennt.

Fall B: $(\cdot, \mathsf{pk}'_{\mathcal{K}}, \mathsf{coin}), (\cdot, \mathsf{pk}''_{\mathcal{K}}, \mathsf{coin}') \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ mit $\mathsf{pk}'_{\mathcal{K}} \neq \mathsf{pk}''_{\mathcal{K}}$:

Dieser Fall ist genau wie bei TG-I vernachlässigbar.

Fall C: $(\cdot, \cdot, \mathsf{coin}), (\cdot, \cdot, \mathsf{coin}') \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$:

In diesem Fall wurden SoK und SoK' vom Angreifer $\mathcal{A}$ erstellt. Daher kann $\mathcal{B}$ die Zahlen $\kappa, \kappa' \in \mathbb{Z}_p$ mit $S = g^{\kappa}, S' = g^{\kappa'}, T = \Sigma_1 g_0^{R\kappa}$ und $T' = \Sigma_2 g_0^{R'\kappa'}$ aus SoK und SoK' extrahieren. Analog kann $\mathcal{B}$ die Nachrichten-Signatur-Paare $(\phi_1(\mathsf{x}), (\Sigma_1, e_1, s_1))$ und $(\phi_2(\mathsf{x}), (\Sigma_2, e_2, s_2))$ extrahieren.

- Falls $k = k'$ ist, folgt $\kappa = \kappa'$ und weiter mit dem Identify-Algorithmus:

$$\Sigma = \left(\frac{T'^R}{T^{R'}}\right)^{(R-R')^{-1}} = \left(\frac{\Sigma_2^R g_0^{RR'\kappa'}}{\Sigma_1^{R'} g_0^{RR'\kappa}}\right)^{(R-R')^{-1}} = \left(\Sigma_2^R \Sigma_1^{-R'}\right)^{(R-R')^{-1}}.$$

Damit folgt weiter

$$\left(g g_0^s u_0^{\phi(\alpha)}\right)^{\frac{1}{x+e}} = \left(\left(g g_0^{s_2} u_0^{\phi_2(\alpha)}\right)^{\frac{R}{x+e_2}} \left(g g_0^{s_1} u_0^{\phi_1(\alpha)}\right)^{\frac{-R'}{x+e_1}}\right)^{(R-R')^{-1}}$$

und schließlich

$$\begin{aligned} & g^{(x+e_1)(x+e_2)(R-R')} g_0^{s(x+e_1)(x+e_2)(R-R')} u_0^{\phi(\alpha)(x+e_1)(x+e_2)(R-R')} \\ =& g^{R(x+e)(x+e_1)-R'(x+e)(x+e_2)} g_0^{s_2 R(x+e)(x+e_1)-s_1 R'(x+e)(x+e_2)} \\ & u_0^{\phi_2(\alpha)R(x+e)(x+e_1)-\phi_1(\alpha)R'(x+e)(x+e_2)}. \end{aligned} \tag{6.1}$$

Nun müssen die folgenden Fälle betrachtet werden:

Fall 1: $s\,(x+e_1)\,(x+e_2)\,(R-R') \neq s_2 R\,(x+e)\,(x+e_1) - s_1 R'\,(x+e)\,(x+e_2)$. In diesem Fall kann $\mathcal{B}$ zwei verschiedene Darstellungen ausgeben.

Fall 2: $s\,(x+e_1)\,(x+e_2)\,(R-R') = s_2 R\,(x+e)\,(x+e_1) - s_1 R'\,(x+e)\,(x+e_2)$. In diesem Fall kann $\mathcal{B}$ die Zahl s (da $(x+e_1)\,(x+e_2)\,(R-R') \neq 0$ ist) und anschließend den diskreten Logarithmus $b = s - s'' - \delta \mod p$ berechnen.

- Falls $k \neq k'$ ist, sei o.B.d.A. $k > k'$. Dann folgt mit dem Identify-Algorithmus:

$$\Sigma = \frac{T'}{g_0^{R'\kappa'}} = \frac{\Sigma_2 g_0^{R'\kappa'}}{g_0^{R'\kappa'}} = \Sigma_2.$$

Daraus folgt weiter:

$$\begin{aligned} & \left(g g_0^s u_0^{\phi(\alpha)}\right)^{\frac{1}{x+e}} = \left(g g_0^{s_2} u_0^{\phi_2(\alpha)}\right)^{\frac{1}{x+e_2}} \\ \Leftrightarrow \qquad & g^{(x+e_2)} g_0^{s(x+e_2)} u_0^{\phi(\alpha)(x+e_2)} = g^{(x+e)} g_0^{s_2(x+e)} u_0^{\phi_2(\alpha)(x+e)}. \end{aligned} \tag{6.2}$$

Analog zum Fall $k = k'$ müssen nun die folgenden Fälle betrachtet werden:

Fall 1: $s\,(x+e_2) \neq s_2\,(x+e)$. In diesem Fall kann $\mathcal{B}$ zwei verschiedene Darstellungen ausgeben.

Fall 2: $s\,(x+e_2) = s_2\,(x+e)$. In diesem Fall kann $\mathcal{B}$ die Zahl s (da $x+e_2 \neq 0$ ist) und anschließend den diskreten Logarithmus $b = s - s'' - \delta \mod p$ berechnen.

Fall D: Sei o.B.d.A. $(\cdot,\cdot,\mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(\cdot,\cdot,\mathsf{coin}') \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$:

Falls $\mathcal{B}$ genau eine der beiden Münzen zum Bezahlen verwendet hat, sei dies o.B.d.A. die Münze coin. Dann wurde SoK' vom Angreifer $\mathcal{A}$ erstellt und daher kann $\mathcal{B}$ die Zahl $\kappa' \in \mathbb{Z}_p$ mit $S' = g^{\kappa'}$ und $T' = \Sigma_2 g_0^{R\kappa'}$ sowie das Nachrichten-Signatur-Paar $(\phi_2(\mathsf{x}), (\Sigma_2, e_2, s_2))$ aus SoK' extrahieren. Seien $\left(j, \mathsf{pk}_{\mathcal{K}_j}, \mathsf{coin}\right) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $\left(j, \mathbb{W}_j, \mathsf{pk}_{\mathcal{K}_j}, \$_j\right) \in \mathbb{W}_H$ die entsprechenden Einträge bzgl. der Münze coin. Je nach Fall müssen wir unterscheiden:

Fall 1: Es sei $\mathbb{W}_j = (s_1', \Sigma_1, e_1, s_1'', \cdots)$ und $s_1 := s_1' + s_1'' \mod p$.

Fall 2: Es sei $\mathbb{W}_j = (\delta_1, \Sigma_1, e_1, s_1'', \cdots)$ und $s_1 := b + \delta_1 + s_1'' \mod p$.

- Falls $k = k'$ ist, folgt $\kappa = \kappa'$ und weiter mit dem $\mathsf{Identify}$-Algorithmus erneut die Gleichung 6.1:

$$\begin{aligned} & g^{(x+e_1)(x+e_2)(R-R')} g_0^{s(x+e_1)(x+e_2)(R-R')} u_0^{\phi(\alpha)(x+e_1)(x+e_2)(R-R')} \\ =& g^{R(x+e)(x+e_1)-R'(x+e)(x+e_2)} g_0^{s_2R(x+e)(x+e_1)-s_1R'(x+e)(x+e_2)} \\ & u_0^{\phi_2(\alpha)R(x+e)(x+e_1)-\phi_1(\alpha)R'(x+e)(x+e_2)}. \end{aligned}$$

  Fall 1: $(x+e_1)(x+e_2)(R-R') \neq R(x+e)(x+e_1) - R'(x+e)(x+e_2)$ oder $s(x+e_1)(x+e_2)(R-R') \neq s_2R(x+e)(x+e_1) - s_1R'(x+e)(x+e_2)$. In diesem Fall kann $\mathcal{B}$ zwei verschiedene Darstellungen ausgeben.

  Fall 2: $(x+e_1)(x+e_2)(R-R') = R(x+e)(x+e_1) - R'(x+e)(x+e_2)$ und $s(x+e_1)(x+e_2)(R-R') = s_2R(x+e)(x+e_1) - s_1R'(x+e)(x+e_2)$. Somit folgt aus der zweiten Gleichung für $s = b + \delta + s'' \mod p$ und $s_1 = b + \delta_1 + s_1'' \mod p$:

$$\begin{aligned} &(b+\delta+s'')(x+e_1)(x+e_2)(R-R') = \\ &\qquad s_2R(x+e)(x+e_1) - (b+\delta_1+s_1'')R'(x+e)(x+e_2). \end{aligned}$$

  Die erste Gleichung in diese eingesetzt ergibt:

$$\begin{aligned} &(b+\delta+s'')(R(x+e)(x+e_1) - R'(x+e)(x+e_2)) = \\ &\qquad s_2R(x+e)(x+e_1) - (b+\delta_1+s_1'')R'(x+e)(x+e_2). \end{aligned}$$

  Daher kann $\mathcal{B}$ den diskreten Logarithmus

$$b = \frac{s_2R(x+e_1) - (\delta+s'')(R(x+e_1) - R'(x+e_2)) - (\delta_1+s_1'')R'(x+e_2)}{R(x+e_1)}$$

  in $\mathbb{Z}_p$ berechnen (da $R(x+e_1) \neq 0$ ist).

- Falls $k \neq k'$ ist, müssen wir die beiden Fälle $k > k'$ sowie $k < k'$ unterscheiden.

– Sei $k > k'$. Dann folgt mit dem Identify-Algorithmus erneut Gleichung 6.2:

$$g^{(x+e_2)} g_0^{s(x+e_2)} u_0^{\phi(\alpha)(x+e_2)} = g^{(x+e)} g_0^{s_2(x+e)} u_0^{\phi_2(\alpha)(x+e)}.$$

Wie oben müssen nun die folgenden Fälle betrachtet werden:

Fall 1: $s(x+e_2) \neq s_2(x+e)$. In diesem Fall kann $\mathcal{B}$ zwei verschiedene Darstellungen ausgeben.

Fall 2: $s(x+e_2) = s_2(x+e)$. In diesem Fall kann $\mathcal{B}$ die Zahl s (da $x + e_2 \neq 0$ ist) und anschließend den diskreten Logarithmus $b = s - s'' - \delta \mod p$ berechnen.

– Sei $k < k'$. Dann kann aus der Seriennummer S' die Seriennummer S berechnet werden, obwohl der Angreifer $\mathcal{A}$ das Element S' nicht auf eine Orakel-Anfrage erhalten hat (sonst hätte der entsprechende Kunde ja die beiden Seriennummern S und S' mehrfach eingelöst und es würde $(\cdot, \cdot, \mathsf{coin}), (\cdot, \cdot, \mathsf{coin}') \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ gelten). Dies ist aber im Random-Oracle-Modell nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Somit löst $\mathcal{B}$ das DLog-Problem in $\mathbb{G}_1$ mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)/2 - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist.

Abhebe-Beschuldigung: Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Abhebe-Beschuldigung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das DLog-Problem in der Gruppe $\mathbb{G}_1$ zu lösen.

Der Angreifer $\mathcal{B}$ erhält die Parameter $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, h)$ sowie die DLog-Probleminstanz $(g, g_0) \in \mathbb{G}_1^2$ und soll den diskreten Logarithmus $a \in \mathbb{Z}_p^*$ mit $g_0 = g^a$ berechnen.

Simulation: Der Angreifer $\mathcal{B}$ simuliert die Systemparameter sowie die Orakel-Anfragen wie bei der Double-Spending-Beschuldigung im Fall 1, außer, dass $\mathcal{B}$ den privaten Signaturschlüssel $x \in \mathbb{Z}_p^*$ nicht aus der Signature-of-Knowledge $SoK_{\mathcal{B}}$ extrahiert.

Spielende: Der Angreifer $\mathcal{A}$ gibt mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ eine Protokollansicht $\mathsf{view} = (\mathbb{T} = (\mathsf{pk}_{\mathcal{K}}, C, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s''))$ mit $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H$ aus, so dass $\mathsf{VerifyWithdraw}(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{K}}, \mathsf{view}) = 1$ gilt, obwohl $(\cdot, \mathbb{T}) \notin \mathbb{T}_H$ ist. Dann wurde die Signature-of-Knowledge SoK_{W} analog zu TG-I, außer mit vernachlässigbarer Wahrscheinlichkeit, von $\mathcal{A}$ erstellt und $\mathcal{B}$ kann den privaten Schlüssel $u \in \mathbb{Z}_p$ mit $\mathsf{pk}_{\mathcal{K}} = g^u$ extrahieren und somit den diskreten Logarithmus $a = u - \gamma \mod p$ berechnen.

Somit löst $\mathcal{B}$ das DLog-Problem in $\mathbb{G}_1$ mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist. □

Bemerkung 6.6.2. *Die Sicherheitsziele Anonymität, Double-Spending-Beschuldigung und Abhebe-Beschuldigung sind auch dann noch gewährleistet, wenn der Angreifer $\mathcal{A}$ die Zahl $\alpha \in \mathbb{Z}_p^*$ wählt (vgl. Bemerkung 6.5.6). Allerdings muss der Angreifer $\mathcal{B}$ diese Zahl kennen, da $\mathcal{B}$ bei der Double-Spending-Beschuldigung andernfalls keine verschiedenen Darstellungen ausgeben kann. Falls die Bank also die Zahl $\alpha \in \mathbb{Z}_p^*$ selbst wählt, muss die Signature-of-Knowledge $SoK_{\mathcal{B}} : [(\alpha, x) : u_1 = u_0^{\alpha} \wedge X = h^x]$ veröffentlicht werden. Dann kann der Angreifer $\mathcal{B}$ ebenfalls die Zahl $\alpha \in \mathbb{Z}_p^*$ extrahieren. Des Weiteren erfüllt das Geldsystem auch die in [Tro06] definierte Anonymität, bei der der Angreifer $\mathcal{A}$ in der Challenge-Phase alle privaten Schlüssel der ehrlichen Kunden erhält.*

6.7 Effizienzvergleich der teilbaren elektronischen Geldsysteme

In der folgenden Tabelle 6.3 werden die Unterschiede zwischen dem teilbaren Geldsystem TG-II aus Abschnitt 6.6 bzgl. des teilbaren Geldsystems TG-I aus Abschnitt 6.5 aufgelistet. Zur Simplifizierung wird $\mathbb{G}_p := \mathbb{G}_1$ gesetzt und das Typ-3-Pairing verwendet, wobei p eine 256-Bit Primzahl ist. Somit besitzt jedes Gruppenelement der Gruppe $\mathbb{G}_1$ eine Länge von 33 Bytes (vgl. Tabelle 2.3).

	Unterschied		
sp	$-\mathbb{G}_1^3$		$\Rightarrow -99$ Bytes
$\mathsf{pk}_{\mathcal{B}}$	$+\mathbb{Z}_p^2$		$\Rightarrow +64$ Bytes
$(\mathbb{T}, \sigma')$	$+\left(\mathbb{G}_1 \times \mathbb{Z}_p^2\right)$		$\Rightarrow +97$ Bytes
Spend bzw. Spend-K	$-\left(\mathbb{G}_1^2 \times \mathbb{Z}_p^4\right)$		$\Rightarrow -194$ Bytes
Kunde		-4 ME	
Händler		-2 ME	
Deposit	$-\left(\mathbb{G}_1^2 \times \mathbb{Z}_p^4\right)$		$\Rightarrow -194$ Bytes
Bank		-2 ME	
$\mathsf{coin} \in \mathbb{D}_C$	$-\left(\mathbb{G}_1^2 \times \mathbb{Z}_p^4\right)$		$\Rightarrow -194$ Bytes

Tabelle 6.3: Unterschied TG-II im Vergleich zu TG-I

Die folgende Tabelle 6.4 vergleicht die teilbaren Geldsysteme aus [ASM08] (bzw. [Au09, Abschnitt 5.5]), [CG10] und dem teilbaren Geldsystem TG-II aus Abschnitt 6.6. Dabei wird als Geldbörsengröße ein typischer Wert von $L = 10$ und somit $K = 2^{10} = 1.024$ gewählt (vgl. [Au09]). Für das Bezahlprotokoll wird als Münzwert exemplarisch $k = 2^7$ gewählt.

In der Liste tree wird der gesamte Binär-Baum, also die beiden Listen $\mathbb{K}_0$ sowie $\mathbb{K}_1$, außer des Master-Schlüssels $\kappa_{0,0}$ gespeichert. Die Liste tree muss nicht notwendigerweise gespeichert werden, da aus dem Master-Schlüssel $\kappa_{0,0}$ alle Schlüssel des Binär-Baums berechnet werden können und ein zusätzlicher K-Bit String ausreicht, um bezahlte Serienschlüssel an den entsprechenden Stellen zu markieren (vgl. Withdraw in Abschnitt 6.5). Der Master-Schlüssel $\kappa_{0,0}$ wird in der Geldbörse $\mathbb{W}$ gespeichert.

Um die Systeme sinnvoll miteinander vergleichen zu können, müssen alle auf einem Typ-3-Pairing arbeiten. Weiter wird jeder interaktive Proof-of-Knowledge als Signature-of-Knowledge behandelt und der jeweilige öffentliche Schlüssel $\mathsf{pk}_{\mathcal{H}}$ des Händlers ebenfalls von der Bank $\mathcal{B}$ in der Münze coin abgespeichert. Damit die Sicherheitseigenschaft Abhebe-Beschuldigung auch von den Geldsystemen [ASM08] und [CG10] garantiert wird, muss die Bank bestimmte Abhebeinformationen $\mathbb{T}$ abspeichern. (Im Vergleich zu dem teilbaren Geldsystem TG-II muss die Bank allerdings nicht die Daten σ' speichern.) Zudem muss der Kunde in [ASM08] zusätzlich die Kenntnis seines privaten Schlüssels in zero-knowledge beweisen. Des Weiteren werden keine Pairings vorab berechnet oder veröffentlicht. Aus diesem Grund weichen einige Angaben der Tabelle 6.4 von den Angaben in [ASM08] ab. Die Zeitpunkte ts und die Werte der Geldbörsen bzw. Münzen bleiben unberücksichtigt, da sie sich durch wenige Bits darstellen lassen.

Wir wählen p in unserem System als 256-Bit Primzahl, was aktuell und auch in naher Zukunft ein hohes Sicherheitsniveau garantiert. Um dasselbe Sicherheitsniveau in [CG10] zu gewährleisten, muss dort die Primzahl q ebenfalls als 256-Bit Primzahl und p als 3.072-Bit Primzahl gewählt werden (siehe [NSA09] und Tabelle 2.1), wobei $q \mid (p-1)$ gilt. Denn die Seriennummern werden in [CG10] in der eindeutig bestimmten zyklischen Untergruppe $\mathbb{G}_q$ der Ordnung q von $\mathbb{Z}_p^*$ berechnet und für die Sicherheit des Systems muss die DDH-Annahme für die Familie der eindeutig bestimmten zyklischen Untergruppen $\mathbb{G}_q$ gelten.

Wie bereits in Abschnitt 6.3 erwähnt, arbeiten in [CG10] das Nguyen-Akkumulator-Schema und das ESS+-Signaturverfahren auf der eindeutig bestimmten zyklischen Untergruppe $\mathbb{G}_p$ der Ordnung p von $\mathbb{Z}_P^*$, wobei P eine Primzahl mit $P = 2p+1$ ist. Da für diese Verfahren allerdings ein Pairing benötigt wird, welche momentan nur für elliptische Kurven bekannt sind, sollte aus praktischer Sicht anstelle der Gruppe $\mathbb{G}_p$ eine elliptische Kurve $\mathbb{G}_1$ der Ordnung p verwendet werden, wobei $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ eine bilineare Abbildung ist (vgl. Abschnitt 2.5). Die Sicherheit des Geldsystems wird dadurch nicht verletzt ([Can13]).

Es ist jedoch zu beachten, dass das Geldsystem [CG10] ohnehin nicht die Sicherheitseigenschaft Unfälschbarkeit erfüllt (siehe Unterabschnitt 6.3.1).

	ASM08	CG10	TG-II
Byte Längen	$\mathbb{G}_1 : 33,\ \mathbb{G}_2 : 65$ $\mathbb{G}_T : 128$	$\mathbb{G}_q : 384,\ \mathbb{G}_1 : 385$ $\mathbb{G}_2 : 769,\ \mathbb{G}_T : 1.536$	$\mathbb{G}_1 : 33,\ \mathbb{G}_2 : 65$ $\mathbb{G}_T : 128$
sp Bytes	$\mathbb{G}_1^{2K+11} \times \mathbb{G}_2^{L+4}$ 68.857	$\mathbb{G}_q^2 \times \mathbb{G}_1^{8K+3} \times \mathbb{G}_2^{4K+2L+3}$ 6.323.354	$\mathbb{G}_1^{K+3} \times \mathbb{G}_2^{K+1}$ 100.516
$\mathsf{pk}_{\mathcal{B}}$ Bytes	$\mathbb{G}_2 \times \mathbb{G}_T$ 193	$\mathbb{G}_2 \times \mathbb{G}_T$ 2.305	$\mathbb{G}_2 \times \mathbb{Z}_p^2$ 129
$\mathsf{sk}_{\mathcal{B}}$ Bytes	$\mathbb{G}_1 \times \mathbb{Z}_p$ 65	$\mathbb{G}_1 \times \mathbb{Z}_p$ 769	$\mathbb{Z}_p$ 32
Withdraw Bytes *Kunde* *Bank*	$\mathbb{G}_1^{6L+6} \times \mathbb{G}_2^{L+1} \times \mathbb{Z}_p^{5L+8}$ 4.749 $(2K + 8L + 7)$ ME + $(4L + 4)$ P $(K + 6L + 6)$ ME	$\mathbb{G}_1^{4L+8} \times \mathbb{G}_2^{L+2} \times \mathbb{Z}_p^{3L+10}$ 43.068 $(4K + 5L + 8)$ ME + $(4L + 8)$ P $(4L + 8)$ ME	$\mathbb{G}_1^3 \times \mathbb{Z}_p^7$ 323 $(2K + 5)$ ME + 3 P 4 ME + 2 P
tree Bytes	$\mathbb{Z}_p^{2K-2}$ 65.472	$\mathbb{Z}_p^{4K-2}$ 1.572.096	$\mathbb{Z}_p^{3K-2}$ 98.240
$\mathbb{W}$ Bytes	$\mathbb{G}_1^{3L+3} \times \mathbb{G}_2^{L+1} \times \mathbb{Z}_p^{2L+3}$ 2.540	$\mathbb{G}_1^{3L+6} \times \mathbb{G}_2^{L+2} \times \mathbb{Z}_p^{2L+6}$ 33.072	$\mathbb{G}_1 \times \mathbb{Z}_p^3$ 129
$(\mathbb{T}, \sigma')$ Bytes	$\mathbb{G}_1^{2L+3} \times \mathbb{Z}_p^{2L+4}$ 1.527	$\mathbb{G}_1^{2L+5} \times \mathbb{Z}_p^{2L+7}$ 19.993	$\mathbb{G}_1^4 \times \mathbb{Z}_p^7$ 356
Spend Bytes *Kunde* *Händler*	$\mathbb{G}_1^8 \times \mathbb{G}_2 \times \mathbb{Z}_p^{21}$ 1.001 25 ME + 8 P 16 ME + 8 P	$\mathbb{G}_q^2 \times \mathbb{G}_1^{11} \times \mathbb{G}_2^2 \times \mathbb{Z}_p^{32}$ 18.829 40 ME + 16 P $(4k + 23)$ ME + 16 P	$\mathbb{G}_1^3 \times \mathbb{Z}_p^7$ 323 7 ME + 2 P $(2k + 3)$ ME + 3 P
Deposit Bytes *Bank*	$\mathbb{G}_1^8 \times \mathbb{G}_2 \times \mathbb{Z}_p^{22}$ 1.033 $(2k + 14)$ ME + 8 P	$\mathbb{G}_q^2 \times \mathbb{G}_1^{11} \times \mathbb{G}_2^2 \times \mathbb{Z}_p^{33}$ 19.213 $(4k + 23)$ ME + 16 P	$\mathbb{G}_1^3 \times \mathbb{Z}_p^8$ 355 $(2k + 3)$ ME + 3 P
$\mathsf{coin} \in \mathbb{D}_C$ Bytes	$\mathbb{G}_1^{k+9} \times \mathbb{G}_2 \times \mathbb{Z}_p^{22}$ 5.290	$\mathbb{G}_q^{k+2} \times \mathbb{G}_1^{12} \times \mathbb{G}_2^2 \times \mathbb{Z}_p^{33}$ 68.750	$\mathbb{G}_1^4 \times \mathbb{Z}_p^{k+8}$ 4.484

Tabelle 6.4: Effizienzvergleich zwischen [ASM08], [CG10] und TG-II

6.8 Mögliche Varianten der teilbaren elektronischen Geldsysteme

In diesem Abschnitt werden mögliche Veränderungen der teilbaren elektronischen Geldsysteme TG-I und TG-II beschrieben, die auch auf die in den folgenden Kapiteln 7 bis 9 konstruierten elektronischen Geldsysteme angewandt werden können.

- Wie aus den Sicherheitsbeweisen entnommen werden kann, muss die bei den Bezahlprotokollen verwendete Zahl $R \in \mathbb{Z}_p^*$ nicht durch eine Hashfunktion generiert werden. Denn an keiner Stelle der Beweise wird benötigt, dass die Zahl R von einem Random-Oracle erzeugt werden muss. Es ist lediglich wichtig, dass diese Zahl bei jeder Münze, außer mit vernachlässigbarer Wahrscheinlichkeit, einzigartig ist. Somit kann die folgende alternative Möglichkeit zur Erzeugung der Zahl $R \in \mathbb{Z}_p^*$ verwendet werden:

  Zunächst halten wir fest, dass jeder Bezahlzeitpunkt ts und jeder Münzwert $k \leq K$ als Bitstring mit $\mathsf{ts} \in \{0,1\}^{n_1}$ und $k \in \{0,1\}^{n_2}$ für zwei Zahlen $n_1, n_2 \in \mathbb{N}$ darstellbar ist. Jeder Händler $\mathcal{H}$ besitzt nun neben seinem öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{H}}$ eine weitere, eindeutige, öffentliche Identifikationsnummer $\mathsf{ID} \in \{0,1\}^{n_3}$ für eine Zahl $n_3 \in \mathbb{N}$, so dass $\mathsf{ID}||1^{n_1+n_2} < p$ gilt, wobei $1^{n_1+n_2}$ den (n_1+n_2)-Bit String $1 \cdots 1$ symbolisiert. Die Zahl R ist nun bei jedem Bezahlprotokoll lediglich die Konkatenation $R = \mathsf{ID}||\mathsf{ts}||k$ mit $1 \leq R < p$. Somit sind die beiden Bedingungen $R \in \mathbb{Z}_p^*$ und $R = \mathsf{ID}||\mathsf{ts}||k \neq \mathsf{ID}'||\mathsf{ts}'||k' = R'$ für $(\mathsf{ID}, \mathsf{ts}) \neq (\mathsf{ID}', \mathsf{ts}')$ (und somit für $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts}) \neq (\mathsf{pk}'_{\mathcal{H}}, \mathsf{ts}')$) erfüllt.

  Daher müssen bei dieser Variante der öffentliche Schlüssel $\mathsf{pk}_{\mathcal{H}}$, der Bezahlzeitpunkt ts und der Münzwert k nicht in der Münze coin gespeichert werden, da diese Informationen bereits in R enthalten sind.

- Falls die Sicherheitseigenschaft Abhebe-Beschuldigung nicht für die Geldsysteme benötigt wird (diese ist in den meisten Geldsystemen in der Literatur nicht definiert), werden der Algorithmus $\mathsf{VerifyWithdraw}$ und der Zeitpunkt ts für das Abhebeprotokoll nicht benötigt. Weiter muss die Bank in diesem Fall für das teilbare Geldsystem TG-I nicht die Abhebeinformation $\mathbb{T} = (I, C, A_0, SoK_{\mathsf{W}}, K)$ abspeichern. Für das teilbare Geldsystem TG-II werden diese Daten allerdings für die Double-Spending-Detection benötigt.

- Da bei den Geldsystemen TG-I und TG-II jeweils eine Geldbörse vom festgelegten Wert K abgehoben wird, muss dieser Wert nicht zur Berechnung der Challenge von SoK_{W} verwendet und nicht von der Bank gespeichert werden. Des Weiteren wird bei jedem Bezahlprotokoll und jeder Münze lediglich der Wert ℓ mit $k = 2^{\ell} \leq K$ benötigt. Da allerdings bei den Modifizierungen in Kapitel 8 beliebige Geldbeträge $K' \leq K$ abgehoben und Münzen mit beliebigen Werten $k \leq K'$ ausgegeben werden können, ist es notwendig, dass der Wert K' zur Berechnung der Challenge von

SoK_{W} verwendet und von der Bank gespeichert wird. Weiter muss der Wert k in der Münze gespeichert werden. Zum besseren Verständnis wurden die Geldsysteme bereits in diesem Kapitel entsprechend konzipiert.

- Das Einlöseprotokoll der teilbaren elektronischen Geldsysteme TG-I und TG-II ist so konstruiert, dass die Bank abbricht, falls ein Händler eine Münze ein weiteres Mal einlösen möchte. Falls dies bereits als Betrugsversuch geahndet werden soll (vgl. [CG10]), ist es notwendig, dass die Bank die Einlöseanfragen einer dritten Partei nachweisen kann. In diesem Fall enthält das Einlöseprotokoll $\mathsf{Deposit}$ als gemeinsame Eingabe auch den aktuellen Zeitpunkt ts. Der Händler mit Schlüsselpaar $(\mathsf{pk}_{\mathcal{H}}, \mathsf{sk}_{\mathcal{H}}) = (I, u)$ erstellt dann die Signature-of-Knowledge

$$SoK_{\mathsf{D}}\left[(u) : I = \mathfrak{g}^{u}\right](R||\mathsf{ts})$$

und sendet SoK_{D} gemeinsam mit der Münze $\mathsf{coin} = (k, S, T, R, \Phi)$ an die Bank. Da jede Münze coin durch die Zahl R, außer mit vernachlässigbarer Wahrscheinlichkeit, eindeutig festgelegt ist, kann die Bank durch die Signature-of-Knowledge SoK_{D} beweisen, dass der Händler mit dem öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{H}}$ die Münze coin zum Zeitpunkt ts eingelöst hat.

KAPITEL 7

FAIRE TEILBARE ELEKTRONISCHE GELDSYSTEME

Die beiden im vorherigen Kapitel 6 vorgestellten teilbaren elektronischen Geldsysteme TG-I und TG-II bieten rechnerische Anonymität, die von niemandem aufgehoben werden kann. Doch Geldsysteme mit perfekter oder rechnerischer Anonymität ermöglichen den Kunden kriminelle Aktivitäten wie Geldwäsche oder Erpressung (vgl. [SN92, FTY96, FTY98]). Y. Frankel, Y. Tsiounis und M. Yung haben in [FTY96] ein *faires elektronisches Offline-Geldsystem* entwickelt, welches die Autoren in [FTY98] effizienter gestaltet haben. Die Hauptmerkmale dieser fairen Geldsysteme sind, dass die Bank $\mathcal{B}$ mit Hilfe einer vertrauenswürdigen dritten Partei, dem *Deanonymisierer* $\mathcal{D}$, eine *Kundenverfolgung* sowie eine *Münzverfolgung* durchführen kann, wobei der Deanonymisierer bei keinen weiteren Protokollen beteiligt sein muss und somit *offline* ist. Liegt bspw. ein begründeter Verdacht der Geldwäsche vor, kann durch eine Kundenverfolgung der Eigentümer einer bestimmten, eingelösten Münze aufgedeckt werden. Alle anderen Münzen bleiben weiterhin anonym. Durch eine Münzverfolgung können bspw. im Fall einer Erpressung alle zu einem bestimmten Abhebeprotokoll gehörenden Münzen verfolgt und gegebenenfalls gesperrt werden. Auch hier bleiben alle übrigen Münzen anonym.

In diesem Kapitel werden die beiden teilbaren elektronischen Geldsysteme aus Kapitel 6 zu fairen teilbaren elektronischen Geldsystemen erweitert.

In Abschnitt 7.2 wird gezeigt, dass das teilbare elektronische Geldsystem TG-II im Wesentlichen durch eine einzige Veränderung zu einem fairen elektronischen Geldsystem FTG-0 wird. Da in diesem System allerdings nur bereits eingelöste Münzen verfolgt werden können und somit das Sperren bestimmter Münzen nicht realisiert wird, wird in Abschnitt 7.3 das teilbare Geldsystem TG-I zu einem fairen teilbaren Geldsystem FTG-I

und in Abschnitt 7.4 das teilbare Geldsystem TG-II zu einem fairen teilbaren Geldsystem FTG-II modifiziert. Die beiden fairen Geldsysteme FTG-I und FTG-II ermöglichen auch die Verfolgung von Münzen, die erst zukünftig ausgegeben werden. Somit können diese Münzen gegebenenfalls auch direkt beim Bezahlvorgang gesperrt werden.

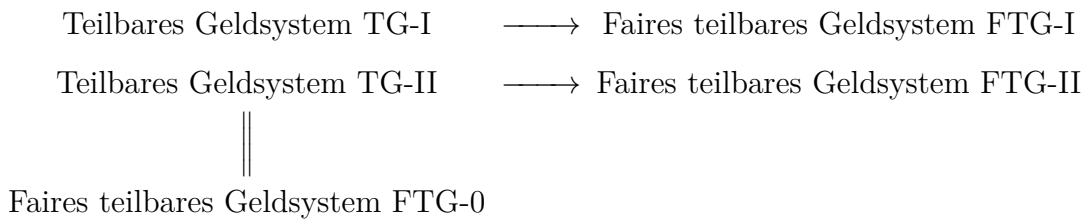

Abbildung 7.1: Zusammenhang der fairen teilbaren Geldsysteme

Anschließend werden die fairen teilbaren elektronischen Geldsysteme FTG-I und FTG-II in Abschnitt 7.5 hinsichtlich ihrer Effizienz miteinander verglichen. In Abschnitt 7.6 werden alternative Varianten zur Münzverfolgung angegeben, die in die Systeme FTG-I und FTG-II integriert werden können. Schließlich wird in Abschnitt 7.7 eine Möglichkeit thematisiert, ob und wie Kunden im Fall eines Verlustes oder Diebstahls der elektronischen Geldbörse mit Hilfe einer (möglicherweise weiteren) vertrauenswürdigen dritten Partei die noch nicht ausgegebenen Münzen von der Bank erstattet bekommen können.

Zunächst werden im folgenden Abschnitt 7.1 allerdings die zentralen Charakteristika und die allgemeinen Konstruktionen der folgenden fairen elektronischen Geldsysteme behandelt.

7.1 Allgemeine Konstruktion der fairen elektronischen Geldsysteme

Wie in [FTY96] und [FTY98] beschrieben, müssen zur Realisierung einer Kundenverfolgung lediglich bestimmte Informationen über den Kunden in die Münzen eingebettet werden, aus denen der Deanonymisierer die Identität des Kunden ermitteln kann. So wird im Bezahlprotokoll von [FTY96, FTY98] eine *verifizierbare* ElGamal-Verschlüsselung der Kontonummer des Kunden bzgl. des öffentlichen Schlüssels des Deanonymisierers erstellt. Um eine Münzverfolgung zu gewährleisten, enthält jede Münze einen sogenannten *Münz-Identifikator* (vgl. [Sch09]), wobei die Bank durch das Abhebeprotokoll Informationen über diesen Münz-Identifikator erhält. Diese Informationen ermöglichen dem Deanonymisierer anschließend eine Verfolgung der Münzen. So wird im Abhebeprotokoll von [FTY98] eine verifizierbare ElGamal-Verschlüsselung des Münz-Identifikators der Münze bzgl. des öffentlichen Schlüssels des Deanonymisierers erstellt und der Münz-Identifikator in die entsprechende Münze integriert.

Da bei kompakten und teilbaren Geldsystemen eine Geldbörse vom Wert K abgehoben wird, ist das in [FTY98] eingesetzte Verfahren allerdings nicht effizient übertragbar. Denn dazu müssten beim Abheben der Geldbörse K Münz-Identifikatoren generiert, verifizierbar verschlüsselt und anschließend in die einzelnen Münzen integriert werden.

Eine Methode zur Verfolgung von Münzen wurde für das kompakte elektronische Geldsystem in [CHL05, Abschnitt 4.2] entwickelt und wird in Unterabschnitt 7.6.2 näher erläutert.

Ein allgemeines Verfahren zur Realisierung einer Münzverfolgung bei teilbaren Geldsystemen wird von M. Au [Au09, Abschnitt 5.6] vorgeschlagen. Bei dieser Technik werden allerdings verifizierbare Verschlüsselungsverfahren eingesetzt, die nicht besonders effizient sind. Zudem wird, wie in Unterabschnitt 7.6.1 beschrieben, durch diese Methode nicht das Sicherheitsziel Münzverfolgung-Beschuldigung gewährleistet.

Das Hauptziel dieses Kapitels ist eine möglichst effiziente Integration einer Kunden- und Münzverfolgung in die vorhandenen teilbaren elektronischen Geldsysteme TG-I und TG-II. Denn es ist zu beachten, dass in jeder Münze eines Offline-Geldsystems gewisse Informationen über den Kunden eingebettet sind, um eine Double-Spending-Detection zu gewährleisten (vgl. auch [FTY96]). Die teilbaren elektronischen Geldsysteme TG-I und TG-II in Kapitel 6 sind so konzipiert, dass das für die Double-Spending-Detection benötigte Tripel (S, T, R) gleichzeitig eine ElGamal-Verschlüsselung darstellt, wobei $\left(T, S^R\right)$ der Geheimtext ist (vgl. Unterabschnitt 2.7.1). Der Klartext ist entsprechend die eindeutige Kontonummer $I \in \mathbb{G}_p$ bei TG-I bzw. das eindeutige Element $\Sigma \in \mathbb{G}_1$ bei TG-II. Um eine effiziente Kundenverfolgung zu gewährleisten, benötigt der Deanonymisierer $\mathcal{D}$ lediglich den diskreten Logarithmus $y \in \mathbb{Z}_p^*$ mit $\mathfrak{g}_0 = \mathfrak{g}^y$. Denn damit kann $\mathcal{D}$ im Verdachtsfall den Klartext TS^{-yR} berechnen und den Kunden bei TG-I anhand der Kontonummer I bzw. bei TG-II anhand des Elements Σ eindeutig identifizieren. Durch diese Methode wird im folgenden Abschnitt 7.2 das faire teilbare elektronische Geldsystem FTG-0 konstruiert.

7.2 Das faire teilbare elektronische Geldsystem FTG-0

In diesem Abschnitt wird die Konstruktion des fairen teilbaren Geldsystems FTG-0 beschrieben. Dabei entsteht das Geldsystem FTG-0 aus dem in Abschnitt 6.6 entwickelten teilbaren Geldsystem TG-II, indem der Deanonymisierer $\mathcal{D}$ den diskreten Logarithmus $y \in \mathbb{Z}_p^*$ mit $g_0 = g^y$ als geheimen Schlüssel $\mathsf{sk}_{\mathcal{D}}$ wählt. So kann $\mathcal{D}$ im Verdachtsfall das Element $\Sigma = TS^{-yR}$ berechnen, wodurch sowohl der entsprechende Kunde als auch das entsprechende Abhebeprotokoll (bzw. die entsprechende, abgehobene Geldbörse) eindeutig identifiziert werden und daher sowohl Kunden- als auch Münzverfolgung möglich sind. Dies ist allerdings nur dann möglich, wenn $\mathcal{D}$ zusätzlich Zugriff auf die Datenbank $\mathbb{D}_C$ der eingelösten Münzen hat. Daher können die bisherigen Algorithmen und

Protokolle aus dem teilbaren Geldsystem TG-II übernommen werden. Es ist lediglich zu beachten, dass der Generator $g_0 \in \mathbb{G}_1$ nun nicht Teil der Systemparameter sp, sondern der öffentliche Schlüssel des Deanonymisierers ist. Aus diesem Grund werden im Folgenden nur die Veränderungen detailliert beschrieben.

Setup geht wie beim teilbaren Geldsystem TG-II vor, außer dass der Generator $g_0 \in \mathbb{G}_1$ nicht zufällig gewählt wird. Die Ausgabe sind die Systemparameter

$$\mathsf{sp} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, u_0, \ldots, u_K, h, h_1, \ldots, h_K, \mathsf{H}) \leftarrow \mathsf{Setup}\left(1^\lambda, K\right).$$

DGen wählt bei Eingabe der Systemparameter sp zufällig eine Zahl $y \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet das Element $g_0 = g^y$. Die Ausgabe ist das Schlüsselpaar

$$(\mathsf{pk}_{\mathcal{D}}, \mathsf{sk}_{\mathcal{D}}) = (g_0, y) \leftarrow \mathsf{DGen}\,(\mathsf{sp})\,.$$

Zusätzlich muss der Deanonymisierer $\mathcal{D}$ Zugriff auf die Datenbank $\mathbb{D}_C$ der eingelösten Münzen haben.

In der folgenden Tabelle 7.1 werden die benötigten Gruppen, deren öffentliche Generatoren, deren Verwendung und die zugrunde liegenden, kryptografischen Annahmen zusammengefasst.

Gruppen	**Generatoren**	**Verwendung**	**Annahmen**
$\mathbb{G}_1$	$g, g_0, u_0, \ldots, u_K$	Seriennummern und Sicherheitstags	
$\mathbb{G}_2$	$h, h_1, \ldots, h_K, X$		
$(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$		Nguyen-Akkumulator und BBS+A-Signatur	XDH, q-SDH und q-polyDH

Tabelle 7.1: Öffentliche Parameter von FTG-0

UTrace: Möchte die Bank $\mathcal{B}$ gemeinsam mit dem Deanonymisierer $\mathcal{D}$ den Eigentümer einer eingelösten Münze bestimmen, führen $\mathcal{B}$ und $\mathcal{D}$ das Kundenverfolgungsprotokoll UTrace durch, welches in Abbildung 7.2 dargestellt ist. Zuvor authentifizieren sich jedoch beide Parteien gegenseitig.

Schritt 1: Die Bank $\mathcal{B}$ sendet eine Münze $\mathsf{coin} = (k, S, T, R, \Phi)$ an $\mathcal{D}$.

Schritt 2: Der Deanonymisierer $\mathcal{D}$ verifiziert die Münze coin und berechnet das Element

$$TS^{-yR} = \Sigma g_0^{R\kappa} g^{-yR\kappa} = \Sigma g_0^{R\kappa} g_0^{-R\kappa} = \Sigma.$$

Anschließend sendet $\mathcal{D}$ das Element Σ an $\mathcal{B}$.

Schritt 3: Die Bank durchsucht ihre Datenbank $\mathbb{D}_W$ und kann den Kunden $\mathcal{K}$ anhand von Σ eindeutig identifizieren. Dann sendet $\mathcal{B}$ die Protokollansicht des entsprechenden Abhebeprotokolls $\mathsf{view} = (\mathbb{T} = (I, C, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s''))$ an $\mathcal{D}$.

Schritt 4: Der Deanonymisierer $\mathcal{D}$ verifiziert SoK_{W} sowie die Signatur durch Überprüfung der Gleichung $\hat{e}(\Sigma, Xh^e) \stackrel{?}{=} \hat{e}\left(gg_0^{s''}C, h\right)$, was der Durchführung des Algorithmus VerifyWithdraw entspricht, und gibt den öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{K}} = I$ des Kunden aus.

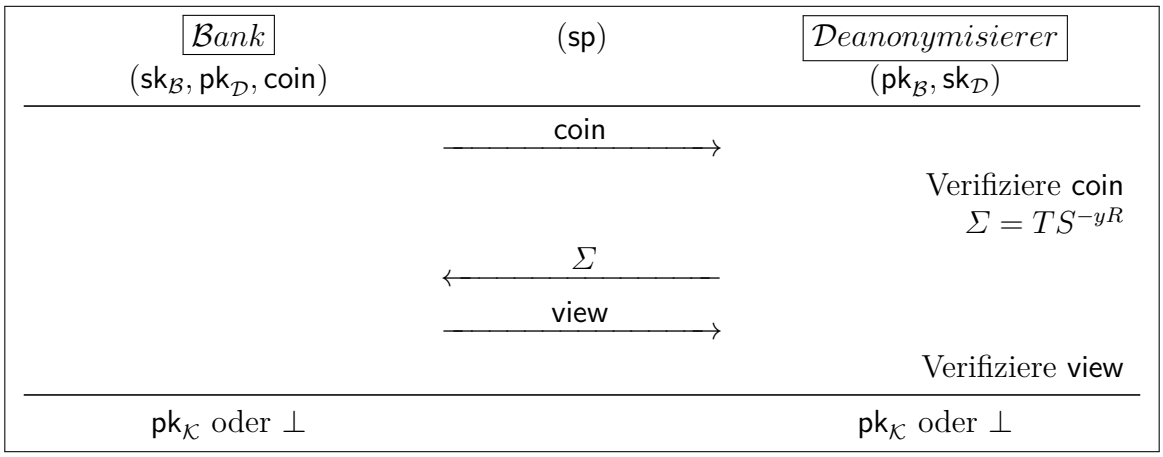

Abbildung 7.2: Kundenverfolgungsprotokoll UTrace von FTG-0

CTrace: Möchte die Bank $\mathcal{B}$ gemeinsam mit dem Deanonymisierer $\mathcal{D}$ alle bereits eingelösten Münzen einer zu einem bestimmten Abhebeprotokoll Withdraw gehörenden Geldbörse verfolgen, führen $\mathcal{B}$ und $\mathcal{D}$ das Münzverfolgungsprotokoll CTrace durch, welches in Abbildung 7.3 dargestellt ist. Zuvor authentifizieren sich jedoch beide Parteien gegenseitig.

Schritt 1: Die Bank $\mathcal{B}$ sendet eine Protokollansicht $\mathsf{view} = (\mathbb{T} = (I, C, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s''))$ eines bestimmten Abhebeprotokolls an $\mathcal{D}$.

Schritt 2: Der Deanonymisierer $\mathcal{D}$ verifiziert SoK_{W} sowie die Signatur durch Überprüfung der Gleichung $\hat{e}(\Sigma, Xh^e) \stackrel{?}{=} \hat{e}\left(gg_0^{s'}C, h\right)$, was der Durchführung des Algorithmus VerifyWithdraw entspricht. Weiter verifiziert $\mathcal{D}$ bei jeder in der Datenbank $\mathbb{D}_C$ gespeicherten Münze $\mathsf{coin}_i = (k_i, S_i, T_i, R_i, \Phi_i)$ die Münze coin_i (vgl. Deposit bei TG-I) und überprüft die Gleichung $\Sigma \stackrel{?}{=} T_i S_i^{-yR_i}$. Anschließend sendet $\mathcal{D}$ alle Münzen coin_i (bzw. jeweils die Zahl R_i, wodurch jede Münze eindeutig identifiziert wird), bei denen die Gleichung erfüllt ist, an $\mathcal{B}$.

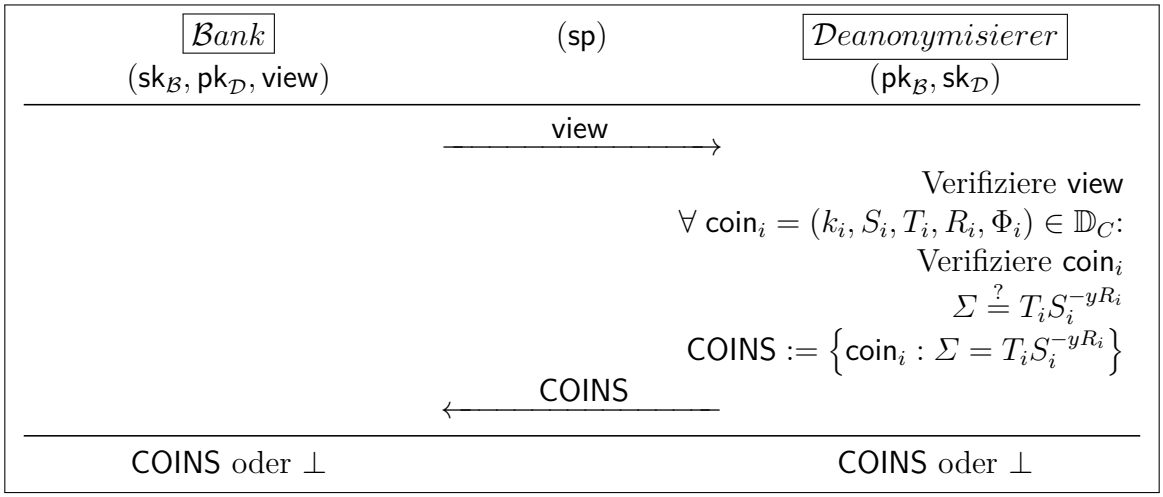

Abbildung 7.3: Münzverfolgungsprotokoll CTrace von FTG-0

7.2.1 Sicherheitsanalyse von FTG-0

Satz 7.2.1. *Das faire teilbare elektronische Geldsystem FTG-0 ist im Random-Oracle-Modell ein sicheres faires elektronisches Geldsystem, falls die XDH-Annahme, die q-SDH-Annahme und die q-polyDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gelten.*

Beweis. Die Sicherheitsziele Unfälschbarkeit, Anonymität, Double-Spending-Beschuldigung und Abhebe-Beschuldigung folgen im Wesentlichen direkt aus TG-II. Zusätzlich müssen anschließend die Sicherheitsziele für die Kunden- und Münzverfolgung bewiesen werden.

Unfälschbarkeit: Die Unfälschbarkeit des Geldsystems FTG-0 folgt direkt aus dem in Abschnitt 6.6 vorgestellten teilbaren Geldsystem TG-II, da sich aus Sicht der Kunden und Händler nichts verändert hat. Somit geht der Angreifer $\mathcal{B}$ wie in TG-II in Abschnitt 6.6 beschrieben vor, außer dass der Generator g_0 nun nicht Teil der Systemparameter sp, sondern der öffentliche Schlüssel $\mathsf{pk}_{\mathcal{D}}$ ist. Da der Angreifer $\mathcal{A}$ nun allerdings die beiden Orakel $\mathcal{O}^{\mathsf{UT}}_{\mathcal{B},\mathcal{D}}$ und $\mathcal{O}^{\mathsf{CT}}_{\mathcal{B},\mathcal{D}}$ befragen darf, wird im Folgenden detailliert beschrieben, wie $\mathcal{B}$ diese (im Fall (1.1)) ohne Kenntnis des geheimen Schlüssels $y \in \mathbb{Z}_p^*$ mit $g_0 = g^y$ simuliert.

Simulation von $\mathcal{O}^{\mathsf{UT}}_{\mathcal{B},\mathcal{D}}$: Der Angreifer $\mathcal{B}$ erhält eine Münze $\mathsf{coin} = (k, S, T, R, \Phi = (A_1, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$.

- Falls der Angreifer $\mathcal{B}$ die Münze coin zum Bezahlen verwendet hat, gibt es die Einträge $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i) \in \mathbb{W}_H$ mit $\mathbb{W}_i = (\Sigma, \cdots)$. Somit kennt $\mathcal{B}$ das Element Σ und gibt dieses aus, wodurch die Anfrage perfekt simuliert ist.

- Falls der Angreifer $\mathcal{B}$ die Münze coin nicht zum Bezahlen verwendet hat, wurde SoK vom Angreifer $\mathcal{A}$ erstellt. Somit kann $\mathcal{B}$ die Zahl $\kappa \in \mathbb{Z}_p$ mit $S = g^{\kappa}$ sowie $T = \Sigma g_0^{R\kappa}$ aus SoK extrahieren und das Element $\Sigma = T g_0^{-R\kappa}$ korrekt berechnen. Dann gibt $\mathcal{B}$ das Element Σ aus, wodurch die Anfrage perfekt simuliert ist. Falls $\mathcal{B}$ die Zahl κ nicht extrahieren kann, bricht $\mathcal{B}$ ab. Dies ist aber nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Simulation von $\mathcal{O}^{\mathsf{CT}}_{\mathcal{B},\mathcal{D}}$: Der Angreifer $\mathcal{B}$ erhält die Datenbank $\mathbb{D}_C$ mit allen eingelösten Münzen $\mathsf{coin}_i = \left(k_i, S_i, T_i, R_i, \Phi_i = \left(A_{1,i}, SoK_i, \mathsf{ts}_i, \mathsf{pk}_{\mathcal{H}_i}\right)\right)$ und eine Protokollansicht $\mathsf{view} = (\mathbb{T} = (I, C, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s''))$.

- Falls der Angreifer $\mathcal{B}$ eine Münze coin_i zum Bezahlen verwendet hat, gibt es die Einträge $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin}_i) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i) \in \mathbb{W}_H$ mit $\mathbb{W}_i = (\Sigma_i, \cdots)$ und somit kennt $\mathcal{B}$ das Element Σ_i. Daher kann $\mathcal{B}$ die Signature-of-Knowledge SoK_{W} verifizieren, die Gleichung $\Sigma \stackrel{?}{=} \Sigma_i$ überprüfen und die Anfrage perfekt simulieren.
- Falls der Angreifer $\mathcal{B}$ eine Münze coin_i nicht zum Bezahlen verwendet hat, wurde SoK_i vom Angreifer $\mathcal{A}$ erstellt. Somit kann $\mathcal{B}$ die Zahl $\kappa_i \in \mathbb{Z}_p$ mit $S_i = g^{\kappa_i}$ sowie $T_i = \Sigma_i g_0^{R_i \kappa_i}$ aus SoK_i extrahieren und das Element $\Sigma_i = T_i g_0^{-R_i \kappa_i}$ korrekt berechnen. Daher kann $\mathcal{B}$ die Signature-of-Knowledge SoK_{W} verifizieren, die Gleichung $\Sigma \stackrel{?}{=} \Sigma_i$ überprüfen und die Anfrage perfekt simulieren. Falls $\mathcal{B}$ die Zahl κ_i nicht extrahieren kann, bricht $\mathcal{B}$ ab. Dies ist aber nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Anonymität: Der Angreifer $\mathcal{B}$ geht wie in TG-II in Abschnitt 6.6 beschrieben vor, außer dass der Generator g_0 nun nicht Teil der Systemparameter sp, sondern der öffentliche Schlüssel $\mathsf{pk}_{\mathcal{D}}$ ist. Zusätzlich muss $\mathcal{B}$ jedoch die beiden Orakel $\mathcal{O}^{\mathsf{UT}}_{\mathcal{D}}$ und $\mathcal{O}^{\mathsf{CT}}_{\mathcal{D}}$ ohne Kenntnis des geheimen Schlüssels $y \in \mathbb{Z}_p^*$ mit $g_0 = g^y$ simulieren.

Simulation von $\mathcal{O}^{\mathsf{UT}}_{\mathcal{D}}$: Wie Simulation von $\mathcal{O}^{\mathsf{UT}}_{\mathcal{B},\mathcal{D}}$ bei der Unfälschbarkeit.

Simulation von $\mathcal{O}^{\mathsf{CT}}_{\mathcal{D}}$: Wie Simulation von $\mathcal{O}^{\mathsf{CT}}_{\mathcal{B},\mathcal{D}}$ bei der Unfälschbarkeit.

Folglich bricht $\mathcal{B}$ die semantische Sicherheit des ElGamal-Verschlüsselungsverfahrens in $\mathbb{G}_1$ mit Wahrscheinlichkeit $1/2 + \epsilon(\lambda) - \nu(\lambda)$, wobei ϵ eine nicht vernachlässigbare Funktion und ν eine vernachlässigbare Funktion ist.

Double-Spending-Beschuldigung und Abhebe-Beschuldigung: Der Angreifer $\mathcal{B}$ geht jeweils wie in TG-II in Abschnitt 6.6 beschrieben vor, außer dass der Generator g_0 nun nicht Teil der Systemparameter sp, sondern der öffentliche Schlüssel $\mathsf{pk}_{\mathcal{D}}$ ist. Zusätzlich muss $\mathcal{B}$ jedoch die beiden Orakel $\mathcal{O}^{\mathsf{UT}}_{\mathcal{D}}$ und $\mathcal{O}^{\mathsf{CT}}_{\mathcal{D}}$ ohne Kenntnis des geheimen Schlüssels $y = a$ mit $g_0 = g^a$ simulieren. Diese Simulationen werden wie bei der Anonymität durchgeführt.

Korrekte Kundenverfolgung: Dies folgt direkt aus der Unfälschbarkeit dieses Geldsystems. Sei $\mathcal{A}$ ein polynomieller Angreifer, der die korrekte Kundenverfolgung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das q-SDH-Problem zu lösen. Der Angreifer $\mathcal{A}$ gibt am Ende eine Münze $\mathsf{coin} = (k, S, T, R, \Phi = (A_1, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}})) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ aus.

Somit wurde SoK vom Angreifer $\mathcal{A}$ erstellt und daher kann $\mathcal{B}$ die Zahl $\kappa \in \mathbb{Z}_p$ und das Nachrichten-Signatur-Paar $(\phi(\mathsf{x}), (\Sigma, e, s))$ mit $S = g^{\kappa}$ und $T = \Sigma g_0^{R\kappa}$ aus SoK extrahieren. Da kein Kunde $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_{\mathcal{A}}$ eindeutig durch das Protokoll UTrace ausgegeben wird, wird entweder (1) kein Kunde $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_{\mathcal{A}}$ ausgegeben oder es werden (2) mindestens zwei verschiedene Kunden $\mathsf{pk}_{\mathcal{K}}, \mathsf{pk}'_{\mathcal{K}} \in \mathcal{U}_{\mathcal{A}} \cup \mathcal{U}_H$ ausgegeben. Da nach Protokollablauf allerdings jedes Element Σ einmalig berechnet wird, ist Fall (2) ausgeschlossen. Somit wird kein Kunde $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_{\mathcal{A}}$ ausgegeben und das Nachrichten-Signatur-Paar $(\phi(\mathsf{x}), (\Sigma, e, s))$ wurde folglich nie bei einer $\mathcal{O}_{\mathcal{B}}^{\mathsf{W}}$-Anfrage erstellt.

- Falls das obige Nachrichten-Signatur-Paar niemals bei einer Orakel-Anfrage erstellt wurde, geht $\mathcal{B}$ wie bei der Unfälschbarkeit im Fall (1.1) vor.
- Falls das obige Nachrichten-Signatur-Paar bei einer $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{W}}$-Anfrage erstellt wurde, geht $\mathcal{B}$ wie bei der Unfälschbarkeit im Fall (1.2) vor.

Kundenverfolgung-Beschuldigung: Dies folgt im Wesentlichen analog zur Double-Spending-Beschuldigung dieses Geldsystems. Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Kundenverfolgung-Beschuldigung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das DLog-Problem in $\mathbb{G}_1$ zu lösen.

Simulation: Der Angreifer $\mathcal{B}$ erhält dieselben Parameter wie bei der Double-Spending-Beschuldigung und simuliert die Initialisierung sowie die Orakel-Anfragen genauso.

Spielende: Der Angreifer $\mathcal{A}$ sendet mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ eine Münze $\mathsf{coin} = (k, S, T, R, \Phi = (A_1, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$ und nach Erhalt eines Elements Σ eine Protokollansicht $\mathsf{view} = (\mathbb{T} = (\mathsf{pk}_{\mathcal{K}}, C, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s''))$ an $\mathcal{B}$, wobei $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H$ oder $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ gilt, obwohl $(\cdot, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ ist.

Sei zunächst $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H$, dann gibt es die Möglichkeiten $(\cdot, \mathbb{T}) \notin \mathbb{T}_H$ (Fall A), oder $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ (Fall B) oder $(\cdot, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ (Fall C).

Fall A: $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H$ und $(\cdot, \mathbb{T}) \notin \mathbb{T}_H$:

In diesem Fall wurde SoK_{W} vom Angreifer $\mathcal{A}$ erstellt. Daher kann $\mathcal{B}$ (wie bei der Double-Spending-Beschuldigung) den privaten Schlüssel u aus SoK_{W} extrahieren und den diskreten Logarithmus $a = u - \gamma \mod p$ berechnen.

Daher nehmen wir nun an, dass für die folgenden beiden Fälle B und C die Signature-of-Knowledge SoK_{W} stets von $\mathcal{B}$ erstellt wurde und somit $(\cdot, \mathbb{T}) \in \mathbb{T}_H$ gilt.

Fall B: $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H, (i, \mathbb{T}) \in \mathbb{T}_H$ und $(j, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ mit $j \neq i$:

In diesem Fall hat $\mathcal{B}$ die Münze coin zum Bezahlen verwendet. Daher kennt $\mathcal{B}$ das Element $\Sigma \in \mathbb{T}_H$.

Sei im Folgenden $(\phi(\mathsf{x}), (\Sigma, e, s))$ das zur Geldbörse und $(\phi_1(\mathsf{x}), (\Sigma, e_1, s_1))$ das zur Münze gehörende Nachrichten-Signatur-Paar. Da die Münze nicht mit der Geldbörse bezahlt wurde und die beiden Polynome $\phi(\mathsf{x}), \phi_1(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ von $\mathcal{B}$ erstellt wurden, gilt nur mit vernachlässigbarer Wahrscheinlichkeit $\phi(\alpha) = \phi_1(\alpha)$. Somit gilt:

$$\begin{aligned} & \left(g g_0^{s} u_0^{\phi(\alpha)}\right)^{\frac{1}{x+e}} = \left(g g_0^{s_1} u_0^{\phi_1(\alpha)}\right)^{\frac{1}{x+e_1}} \\ \Leftrightarrow \quad & g^{x+e_1} g_0^{s(x+e_1)} u_0^{\phi(\alpha)(x+e_1)} = g^{x+e} g_0^{s_1(x+e)} u_0^{\phi_1(\alpha)(x+e)}. \end{aligned} \tag{7.1}$$

Nun müssen (analog zur Double-Spending-Beschuldigung) die folgenden Fälle betrachtet werden:

Fall 1: $s\,(x + e_1) \neq s_1\,(x + e)$. In diesem Fall kann $\mathcal{B}$ zwei verschiedene Darstellungen ausgeben.

Fall 2: $s\,(x + e_1) = s_1\,(x + e)$. Seien $s = b + \delta + s'' \mod p$ und $s_1 = b + \delta_1 + s_1'' \mod p$. Dann gilt

$$\begin{aligned} & (b + \delta + s'')\,(x + e_1) = (b + \delta_1 + s_1'')\,(x + e) \\ \Leftrightarrow \quad & b\,(e_1 - e) = (\delta_1 + s_1'')\,(x + e) - (\delta + s'')\,(x + e_1)\,. \end{aligned}$$

Da aus $e = e_1$ auch $\phi(\alpha) = \phi_1(\alpha)$ folgen würde, ist dies vernachlässigbar. Somit kann $\mathcal{B}$ den diskreten Logarithmus

$$b = \frac{(\delta_1 + s_1'')\,(x + e) - (\delta + s'')\,(x + e_1)}{e_1 - e} \mod p$$

berechnen.

Fall C: $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H, (\cdot, \mathbb{T}) \in \mathbb{T}_H$ und $(\cdot, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$:

In diesem Fall wurde SoK von $\mathcal{A}$ erstellt und daher kann $\mathcal{B}$ die Zahl $\kappa \in \mathbb{Z}_p$ mit $S = g^{\kappa}$ und $T = \Sigma g_0^{R\kappa}$ sowie das Nachrichten-Signatur-Paar $(\phi_1(\mathsf{x}), (\Sigma, e_1, s_1))$ aus SoK extrahieren. Falls $\mathcal{B}$ die Werte nicht extrahieren kann, bricht $\mathcal{B}$ ab. Dies tritt allerdings nur mit vernachlässigbarer Wahrscheinlichkeit ein. Dann gilt erneut Gleichung 7.1:

$$g^{x+e_1} g_0^{s(x+e_1)} u_0^{\phi(\alpha)(x+e_1)} = g^{x+e} g_0^{s_1(x+e)} u_0^{\phi_1(\alpha)(x+e)}.$$

Wieder unterscheiden wir die Fälle:

Fall 1: $s\,(x+e_1) \neq s_1\,(x+e)$. In diesem Fall kann $\mathcal{B}$ zwei verschiedene Darstellungen ausgeben.

Fall 2: $s\,(x+e_1) = s_1\,(x+e)$. Sei $s = b+\delta+s'' \mod p$. Dann gilt

$$\begin{aligned} & (b+\delta+s'')\,(x+e_1) = s_1\,(x+e) \\ \Leftrightarrow \quad & b\,(x+e_1) = s_1\,(x+e) - (\delta+s'')\,(x+e_1)\,. \end{aligned}$$

Da $x+e_1 \neq 0$ ist, kann $\mathcal{B}$ den diskreten Logarithmus

$$b = \frac{s_1\,(x+e)}{x+e_1} - (\delta+s'') \mod p$$

berechnen.

Sei nun $\mathsf{pk}_{\mathcal{K}} \notin \mathcal{U}_H$, dann gilt nach Voraussetzung $(\cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$.

Fall D: $\mathsf{pk}_{\mathcal{K}} \notin \mathcal{U}_H$ und $(\cdot,\cdot,\mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$:

In diesem Fall wurde SoK_{W} vom Angreifer $\mathcal{A}$ erstellt und $\mathcal{B}$ kann das Nachrichten-Signatur-Paar $(\phi(\mathsf{x}), (\Sigma, e, s))$ aus SoK_{W} extrahieren. Dann gilt erneut Gleichung 7.1:

$$g^{x+e_1} g_0^{s(x+e_1)} u_0^{\phi(\alpha)(x+e_1)} = g^{x+e} g_0^{s_1(x+e)} u_0^{\phi_1(\alpha)(x+e)}.$$

Wieder unterscheiden wir die Fälle:

Fall 1: $s\,(x+e_1) \neq s_1\,(x+e)$. In diesem Fall kann $\mathcal{B}$ zwei verschiedene Darstellungen ausgeben.

Fall 2: $s\,(x+e_1) = s_1\,(x+e)$. Sei $s_1 = b+\delta_1+s_1'' \mod p$. Dann gilt

$$\begin{aligned} & s\,(x+e_1) = (b+\delta_1+s_1'')\,(x+e) \\ \Leftrightarrow \quad & b\,(x+e) = s\,(x+e_1) - (\delta_1+s_1'')\,(x+e)\,. \end{aligned}$$

Da $x+e \neq 0$ ist, kann $\mathcal{B}$ den diskreten Logarithmus

$$b = \frac{s\,(x+e_1)}{x+e} - (\delta_1+s_1'') \mod p$$

berechnen.

Somit löst $\mathcal{B}$ das DLog-Problem in $\mathbb{G}_1$ mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)/2 - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist.

Korrekte Münzverfolgung: Dies folgt (ebenfalls wie die korrekte Kundenverfolgung) direkt aus der Unfälschbarkeit dieses Geldsystems. Sei $\mathcal{A}$ ein polynomieller Angreifer, der die korrekte Münzverfolgung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das q-SDH-Problem zu lösen. Der Angreifer $\mathcal{A}$ gibt eine Münze $\mathsf{coin} = (k, S, T, R, \Phi = (A_1, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}})) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ aus.

Somit wurde SoK vom Angreifer $\mathcal{A}$ erstellt und daher kann $\mathcal{B}$ die Zahl $\kappa \in \mathbb{Z}_p$ und das Nachrichten-Signatur-Paar $(\phi(\mathsf{x}), (\Sigma, e, s))$ mit $S = g^{\kappa}$ und $T = \Sigma g_0^{R\kappa}$ aus SoK extrahieren. Da die Münze coin nicht eindeutig durch das Protokoll CTrace verfolgt werden kann, folgt $\Sigma \notin \mathbb{W}_{\mathcal{A}}$, da jedes Element Σ einmalig ist (vgl. korrekte Kundenverfolgung). Somit wurde das Nachrichten-Signatur-Paar nie bei einer $\mathcal{O}_{\mathcal{B}}^{\mathsf{W}}$-Anfrage erstellt.

- Falls das obige Nachrichten-Signatur-Paar niemals bei einer Orakel-Anfrage erstellt wurde, geht $\mathcal{B}$ wie bei der Unfälschbarkeit im Fall (1.1) vor.
- Falls das obige Nachrichten-Signatur-Paar bei einer $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{W}}$-Anfrage erstellt wurde, geht $\mathcal{B}$ wie bei der Unfälschbarkeit im Fall (1.2) vor.

Münzverfolgung-Beschuldigung: Dies folgt im Wesentlichen aus der Sicherheitseigenschaft Kundenverfolgung-Beschuldigung dieses Geldsystems. Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Kundenverfolgung-Beschuldigung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das DLog-Problem in $\mathbb{G}_1$ zu lösen.

Simulation: Der Angreifer $\mathcal{B}$ erhält dieselben Parameter wie bei der Kundenverfolgung-Beschuldigung und simuliert die Initialisierung sowie die Orakel-Anfragen genauso.

Spielende: Der Angreifer $\mathcal{A}$ sendet mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ eine Protokollansicht $\mathsf{view} = (\mathbb{T} = (\mathsf{pk}_{\mathcal{K}}, C, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s''))$ an $\mathcal{B}$. Sei $\mathsf{coin} = (k, S, T, R, \Phi = (A_1, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$ eine Münze, die u. a. durch das Protokoll CTrace ausgegeben wird, obwohl

- $(i, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(j, \mathbb{T}) \in \mathbb{T}_H$ mit $j \neq i$ ist, oder
- $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(\cdot, \mathbb{T}) \notin \mathbb{T}_H$ ist, oder
- $(\cdot, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(\cdot, \mathbb{T}) \in \mathbb{T}_H$ ist.

- Im Fall $(i, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(j, \mathbb{T}) \in \mathbb{T}_H$ mit $j \neq i$ geht $\mathcal{B}$ genau wie bei der Kundenverfolgung-Beschuldigung Fall B vor.
- Der Fall $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(\cdot, \mathbb{T}) \notin \mathbb{T}_H$ kann unterteilt werden in $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H$ oder $\mathsf{pk}_{\mathcal{K}} \notin \mathcal{U}_H$.
  - Im Fall $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H$ geht $\mathcal{B}$ genau wie bei der Kundenverfolgung-Beschuldigung Fall A vor.

  - Im Fall $\mathsf{pk}_{\mathcal{K}} \notin \mathcal{U}_H$ geht $\mathcal{B}$ genau wie bei der Kundenverfolgung-Beschuldigung Fall D vor.

- Im Fall $(\cdot, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(\cdot, \mathbb{T}) \in \mathbb{T}_H$ geht $\mathcal{B}$ genau wie bei der Kundenverfolgung-Beschuldigung Fall C vor.

□

Bemerkung 7.2.1. *Analog zu Bemerkung 6.6.2 sind die Sicherheitsziele Anonymität, Double-Spending-Beschuldigung, Abhebe-Beschuldigung, Kundenverfolgung-Beschuldigung und Münzverfolgung-Beschuldigung auch dann noch gewährleistet, wenn der Angreifer $\mathcal{A}$ die Zahl $\alpha \in \mathbb{Z}_p^*$ wählt.*

7.3 Das faire teilbare elektronische Geldsystem FTG-I

Im vorherigen Abschnitt 7.2 wurde gezeigt, wie das teilbare Geldsystem TG-II äußerst effizient zu dem fairen teilbaren Geldsystem FTG-0 erweitert werden kann. Allerdings besitzt dieses faire Geldsystem folgende Nachteile:

1. Es besteht nicht die Möglichkeit ungültige Münzen bereits während des Bezahlvorgangs zu sperren.

2. Es besteht nicht die Möglichkeit Münzen zu verfolgen, die erst nach der Durchführung des Münzverfolgungsprotokolls eingelöst werden.

3. Der Deanonymisierer benötigt Zugriff auf die Datenbank $\mathbb{D}_C$.

4. Das Geldsystem ist in einem Typ-1-Pairing nicht sicher.

Deshalb wird in diesem Abschnitt ein faires teilbares Geldsystem konstruiert, das die genannten Nachteile behebt. Dazu wird jede Münze mit einem Münz-Identifikator ausgestattet und die Bank erhält durch das Abhebeprotokoll Informationen über den *Münz-Identifikationsschlüssel*, mit dem die Münz-Identifikatoren generiert werden. Diese Informationen reichen der Bank allerdings nicht aus, ohne Hilfe des Deanonymisierers eine Münze zu verfolgen.

Die Grundidee dieser Konstruktion ist, bei jeder Münze einen Teil eines DBDH Tupels $\left(g, g^a, g^b, g^c, h, h^b, h^c, \hat{e}\left(g,h\right)^{abc}\right) =: (u_0, Q, \cdot, \cdot, h, h^t, Z, T_C) \in \mathbb{G}_1^4 \times \mathbb{G}_2^3 \times \mathbb{G}_T$ einzubauen. Die Elemente u_0 und h sind öffentliche Generatoren mit $\mathbb{G}_1 = \langle u_0 \rangle$ sowie $\mathbb{G}_2 = \langle h \rangle$ und das Element $Z \in \mathbb{G}_2$ mit $z \in_{\mathcal{R}} \mathbb{Z}_p^*$ und $Z = h^z$ ist ein Teil des öffentlichen Schlüssels des Deanonymisierers.

Der Münz-Identifikationsschlüssel ist die Zufallszahl $t \in_{\mathcal{R}} \mathbb{Z}_p$, die gemeinsam vom Kunden und der Bank generiert wird, aber nur dem Kunden bekannt ist. Der Kunde

erhält nun im Abhebeprotokoll eine BBS+A-Signatur $\sigma = (\Sigma, e, s)$ auf die Nachricht $m = (u, t, \phi(\mathsf{x}))$ und die Bank erhält das Element $h^t \in \mathbb{G}_2$. Zusätzlich erhält die Bank bei einer Münze das zufällige Element $Q \in_{\mathcal{R}} \mathbb{G}_1$ sowie den zum Tripel (Q, h^t, Z) gehörenden Münz-Identifikator $T_C = \hat{e}\left(Q^t, Z\right) \in \mathbb{G}_T$.

Aus dem DBDH-Problem folgt, dass das Element T_C nicht effizient von einem zufälligen Gruppenelement unterschieden werden kann. Somit kann die Bank den Münz-Identifikator T_C nicht dem entsprechenden Element h^t zuordnen.

Möchte die Bank nun im Verdachtsfall alle zum Element h^t gehörenden Münzen verfolgen, erhält sie vom Deanonymisierer das Element $E^\star = \left(h^t\right)^z = Z^t$. Somit kann die Bank bei jeder Münze die Gleichung $T_C \stackrel{?}{=} \hat{e}\left(Q, E^\star\right)$ überprüfen, unabhängig davon, ob die Münze bereits eingelöst wurde oder nicht.

Wird das Element $E^\star$ auf einer *schwarzen Liste* veröffentlicht um diese Münzen zu sperren, überprüft der Händler bereits während des Bezahlprotokolls obige Gleichung und akzeptiert die Münze bei Gleichheit nicht. Da der Kunde wie beim teilbaren Geldsystem TG-I bei jeder Münze bereits ein zufälliges Element $B_1 \in \mathbb{G}_1$ erzeugt, kann $Q := B_1$ gewählt werden.

Insbesondere wird dieses Geldsystem so konstruiert, dass es im Gegensatz zu dem fairen teilbaren Geldsystem FTG-0 aus Abschnitt 7.2 in allen Pairing Typen sicher ist. Dazu wird das teilbare Geldsystem TG-I aus Abschnitt 6.5 zu dem fairen teilbaren Geldsystem FTG-I erweitert.

Aus diesem Grund werden im Folgenden hauptsächlich die Veränderungen im Vergleich zu TG-I detailliert dargestellt. Der Algorithmus BGen und das Protokoll $\mathsf{Account}$ werden wie beim teilbaren Geldsystem TG-I durchgeführt. Da sich die Abhebe- und Bezahlprotokolle $\mathsf{Withdraw}$ und Spend bzw. Spend-K verändern, werden die Algorithmen $\mathsf{Identify}, \mathsf{VerifyGuilt}$ und $\mathsf{VerifyWithdraw}$ sowie das Protokoll $\mathsf{Deposit}$ dann analog zum teilbaren Geldsystem TG-I durchgeführt und aus diesem Grund nicht beschrieben.

Setup wählt zusätzlich einen weiteren, zufälligen Generator $g_2 \in_{\mathcal{R}} \mathbb{G}_1$. Der Generator $\mathfrak{g}_0 \in \mathbb{G}_p$ wird nicht zufällig gewählt. Die Ausgabe sind die Systemparameter

$$\mathsf{sp} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \mathbb{G}_p', \hat{e}, g, g_0, g_1, g_2, u_0, \ldots, u_K, h, h_1, \ldots, h_K, \mathfrak{g}, \mathsf{H}) \leftarrow \mathsf{Setup}\left(1^\lambda, K\right).$$

DGen wählt bei Eingabe der Systemparameter sp zufällig zwei Zahlen $y, z \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Elemente $\mathfrak{g}_0 = \mathfrak{g}^y \in \mathbb{G}_p$ sowie $Z = h^z \in \mathbb{G}_2$. Die Zahl y wird für eine Kundenverfolgung und die Zahl z für eine Münzverfolgung verwendet. Die Ausgabe ist das Schlüsselpaar

$$(\mathsf{pk}_{\mathcal{D}}, \mathsf{sk}_{\mathcal{D}}) = ((\mathfrak{g}_0, Z), (y, z)) \leftarrow \mathsf{DGen}\left(\mathsf{sp}\right).$$

In der folgenden Tabelle 7.2 werden die benötigten Gruppen, deren öffentliche Generatoren, deren Verwendung und die zugrunde liegenden, kryptografischen Annahmen zusammengefasst.

Gruppen	Generatoren	Verwendung	Annahmen
$\mathbb{G}_1$	$g, g_0, g_1, g_2,$ $u_0, \ldots, u_K$		
$\mathbb{G}_2$	$h, h_1, \ldots, h_K, X, Z$		
$(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$		Nguyen-Akkumulator, BBS+A-Signatur und Münz-Identifikatoren	q-SDH, q-polyDH und DBDH
$\mathbb{G}_p$	$\mathfrak{g}, \mathfrak{g}_0$	Seriennummern und Sicherheitstags	DDH

Tabelle 7.2: Öffentliche Parameter von FTG-I

Withdraw: Damit ein Kunde $\mathcal{K}$ eine Geldbörse mit dem Wert $K = 2^L$ bei der Bank $\mathcal{B}$ abheben kann, führen $\mathcal{K}$ und $\mathcal{B}$ das Abhebeprotokoll Withdraw durch, welches in Abbildung 7.4 dargestellt ist. Zuvor authentifizieren sich jedoch beide Parteien gegenseitig und gleichen den aktuellen Zeitpunkt ts ab.

Schritt 1: Der Kunde $\mathcal{K}$ wählt zufällig zwei Zahlen $s', t' \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das perfekt verbergende Commitment $C = g_0^{s'} g_1^u g_2^{t'} u_0^{\phi(\alpha)}$ und das Element $E = h^{t'}$, wobei $\phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ wie beim teilbaren Geldsystem TG-I das Polynom $\phi(\mathsf{x}) = \prod_{j=0}^{K-1} (\mathsf{x} + \kappa_{L+1,j})$ und $\kappa_{L+1,0}, \ldots, \kappa_{L+1,K-1}$ die K Serienschlüssel sind. Dann erstellt der Kunde $\mathcal{K}$ die folgende Signature-of-Knowledge:

$$SoK_{\mathsf{W}}\Big[(a_0, s', t', u, \phi(l)) : I = \mathfrak{g}^u \ \wedge \ E = h^{t'} \ \wedge \\ \hat{e}(C, h)\, \hat{e}\left(A_0, h_1 h^{-l}\right)^{-1} = \hat{e}\left(g_0^{s'} g_1^u g_2^{t'} u_0^{\phi(l)+a_0 l} u_1^{-a_0}, h\right)\Big](\mathsf{ts}||K).$$

Schritt 2: Die Bank $\mathcal{B}$ berechnet den Hashwert $l = \mathsf{H}(C)$, verifiziert SoK_{W}, wählt zufällig drei Zahlen $e, s'', t'' \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Element $\Sigma = \left(g g_0^{s''} g_2^{t''} C\right)^{\frac{1}{x+e}}$. Dann sendet $\mathcal{B}$ das Tupel (Σ, e, s'', t'') an $\mathcal{K}$ und bucht dem Kunden $\mathcal{K}$ den Betrag K vom Konto mit der Kontonummer I ab. Anschließend speichert $\mathcal{B}$ die Informationen $(\mathbb{T} = (I, C, E, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = t'')$ in ihrer Datenbank $\mathbb{D}_W$ ab.

Schritt 3: Der Kunde $\mathcal{K}$ berechnet $s = s' + s'' \mod p$ sowie $t = t' + t'' \mod p$, verifiziert die Signatur durch Überprüfung der Gleichung $\hat{e}(\Sigma, X h^e) \stackrel{?}{=} \hat{e}\left(g g_0^{s''} g_2^{t''} C, h\right)$ und speichert $\mathbb{W} = (\Sigma, e, s, t, \mathsf{info})$ ab. Somit ist $\sigma = (\Sigma, e, s)$ eine BBS+A-Signatur auf die Nachricht $m = (u, t, \phi(\mathsf{x}))$.

$\boxed{\mathcal{K}unde}$ $(\mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{K}})$	$(\mathsf{sp}, K, \mathsf{ts})$	$\boxed{\mathcal{B}ank}$ $(\mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{K}})$
$\kappa_{0,0} \in_{\mathcal{R}} \mathbb{Z}_p^*, s', t', a_0 \in_{\mathcal{R}} \mathbb{Z}_p$		
Generiere $\mathbb{K}_0, \mathbb{K}_1$		
$\phi(\mathsf{x}) = \prod_{j=0}^{K-1} (\mathsf{x} + \kappa_{L+1,j})$		
$C = g_0^{s'} g_1^u g_2^{t'} u_0^{\phi(\alpha)}$		
$E = h^{t'}$		
$l = \mathsf{H}(C)$		
$A_0 = W_l u_0^{a_0}$		
	$\xrightarrow{C, E, A_0}$	
$SoK_{\mathsf{W}}\Big[(a_0, s', t', u, \phi(l)) : I = \mathfrak{g}^u \ \wedge \ E = h^{t'} \ \wedge$		
$\hat{e}(C, h)\, \hat{e}\left(A_0, h_1 h^{-l}\right)^{-1} = \hat{e}\left(g_0^{s'} g_1^u g_2^{t'} u_0^{\phi(l)+a_0 l} u_1^{-a_0}, h\right)\Big]\ (\mathsf{ts}\|K)$		
		$l = \mathsf{H}(C)$
		Verifiziere SoK_{W}
		$e, s'', t'' \in_{\mathcal{R}} \mathbb{Z}_p$
		$\Sigma = \left(g g_0^{s''} g_2^{t''} C\right)^{\frac{1}{x+e}}$
	$\xleftarrow{\Sigma, e, s'', t''}$	
$s = s' + s'' \mod p$		
$t = t' + t'' \mod p$		
$\hat{e}(\Sigma, X h^e) \stackrel{?}{=} \hat{e}\left(g g_0^{s''} g_2^{t''} C, h\right)$		
$\mathbb{W} = (\Sigma, e, s, t, \mathsf{info})$ oder $\perp$		view oder $\perp$

Abbildung 7.4: Abhebeprotokoll Withdraw von FTG-I

Spend: Damit ein Kunde $\mathcal{K}$, der im Besitz der Geldbörse $\mathbb{W} = (\Sigma, e, s, t, \mathsf{info})$ ist, bei einem Händler $\mathcal{H}$ mit einer elektronischen Münze coin mit dem Wert $k = 2^\ell < K$ bezahlen kann, führen $\mathcal{K}$ und $\mathcal{H}$ das Bezahlprotokoll Spend durch, welches in Abbildung 7.5 dargestellt ist. Zuvor authentifiziert sich der Händler $\mathcal{H}$ gegenüber dem Kunden $\mathcal{K}$ und beide Parteien gleichen den Zeitpunkt ts ab.

Schritt 1: Der Händler $\mathcal{H}$ und der Kunde $\mathcal{K}$ berechnen beide den Hashwert $R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}} \| \mathsf{ts} \| k)$.

Schritt 2: Der Kunde $\mathcal{K}$ geht wie im teilbaren Geldsystem TG-I vor und berechnet zusätzlich den Münz-Identifikator $T_C = \hat{e}(B_1^t, Z)$. Folglich berechnet $\mathcal{K}$ den Hashwert $l = \mathsf{H}(S \| T \| T_C \| B_1 \| B_2)$, sendet die Elemente S, T, T_C, A_1, B_1, B_2 an $\mathcal{H}$ und erstellt die folgende Signature-of-Knowledge SoK, wobei $\beta_1 = b_1 e, \beta_2 = b_2 e, u_I = u_0^{\prod_{j \in I} (\alpha + \kappa_{L+1,j})}, u_{l,I} = u_0^{(\alpha - l) \prod_{j \in I} (\alpha + \kappa_{L+1,j})}$ und

$h_{l,I} = h^{(\alpha-l)\prod_{j\in I}\left(\alpha+\kappa_{L+1,j}\right)}$ ist:

$$SoK\Bigg[(a_1,b_1,b_2,\beta_1,\beta_2,e,s,t,u,\kappa,\phi_I(l)) : B_1 = g_0^{b_1}g_1^{b_2} \wedge 1 = B_1^e g_0^{-\beta_1}g_1^{-\beta_2} \wedge S = \mathfrak{g}^\kappa \wedge T = \mathfrak{g}^u\mathfrak{g}_0^{R\kappa} \wedge T_C = \hat{e}\left(B_1^t, Z\right) \wedge \frac{\hat{e}(B_2,X)}{\hat{e}(g,h)\,\hat{e}(A_1,h_{l,I})} = \hat{e}\left(g_1^{b_1},X\right)\hat{e}\left(g_0^s g_1^{u+\beta_1} g_2^t u_{l,I}^{-a_1} u_I^{\phi_I(l)} B_2^{-e}, h\right)\Bigg](R).$$

Anschließend aktualisiert $\mathcal{K}$ seine Geldbörse zu $\mathbb{W}' = (\Sigma, e, s, t, \mathsf{info}')$.

Schritt 3: Der Händler $\mathcal{H}$ berechnet aus der Seriennummer S alle k Seriennschlüssel sowie den Hashwert $l = \mathsf{H}(S||T||T_C||B_1||B_2)$, verifiziert SoK und speichert die Münze $\mathsf{coin} = (k, S, T, T_C, R, \Phi = (A_1, B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_\mathcal{H}))$ ab.

$\boxed{\mathcal{K}unde}$ $(\mathsf{pk}_\mathcal{H}, \mathsf{sk}_\mathcal{K}, \mathbb{W})$	$(\mathsf{sp}, \mathsf{pk}_\mathcal{B}, \mathsf{pk}_\mathcal{D}, k, \mathsf{ts})$	$\boxed{\mathcal{H}ändler}$ $(\mathsf{sk}_\mathcal{H})$
$R = \mathsf{H}(\mathsf{pk}_\mathcal{H}\|\|\mathsf{ts}\|\|k)$		$R = \mathsf{H}(\mathsf{pk}_\mathcal{H}\|\|\mathsf{ts}\|\|k)$
$a_1, b_1, b_2 \in_\mathcal{R} \mathbb{Z}_p$		
$\kappa := \kappa_{L-\ell,j_0}$		
$S = \mathfrak{g}^\kappa$		
$T = \mathfrak{g}^u\mathfrak{g}_0^{R\kappa}$		
$B_1 = g_0^{b_1}g_1^{b_2}$		
$B_2 = \Sigma g_1^{b_1}$		
$T_C = \hat{e}(B_1^t, Z)$		
$l = \mathsf{H}(S\|\|T\|\|T_C\|\|B_1\|\|B_2)$		
$A_1 = W_{I,l}u_0^{a_1}$		
	$\xrightarrow{S,T,T_C,A_1,B_1,B_2}$	
$SoK\Big[(a_1,b_1,b_2,\beta_1,\beta_2,e,s,t,u,\kappa,\phi_I(l)) : B_1 = g_0^{b_1}g_1^{b_2} \wedge 1 = B_1^e g_0^{-\beta_1}g_1^{-\beta_2} \wedge S = \mathfrak{g}^\kappa \wedge T = \mathfrak{g}^u\mathfrak{g}_0^{R\kappa} \wedge T_C = \hat{e}(B_1^t,Z) \wedge \frac{\hat{e}(B_2,X)}{\hat{e}(g,h)\,\hat{e}(A_1,h_{l,I})} = \hat{e}\left(g_1^{b_1},X\right)\hat{e}\left(g_0^s g_1^{u+\beta_1} g_2^t u_{l,I}^{-a_1} u_I^{\phi_I(l)} B_2^{-e}, h\right)\Big](R)$		
		Berechne $\kappa_{L+1,0},\ldots,\kappa_{L+1,k-1}$
		$l = \mathsf{H}(S\|\|T\|\|T_C\|\|B_1\|\|B_2)$
		Verifiziere SoK
$\mathbb{W}'$ oder $\perp$		coin oder $\perp$

Abbildung 7.5: Bezahlprotokoll Spend von FTG-I

Spend-K: Damit ein Kunde $\mathcal{K}$, der im Besitz der Geldbörse $\mathbb{W} = (\Sigma, e, s, t, \mathsf{info})$ ist, bei einem Händler $\mathcal{H}$ mit einer elektronischen Münze coin mit dem Wert $K = 2^L$

bezahlen kann, führen $\mathcal{K}$ und $\mathcal{H}$ das Bezahlprotokoll Spend-K durch, welches in Abbildung 7.6 dargestellt ist. Zuvor authentifiziert sich der Händler $\mathcal{H}$ gegenüber dem Kunden $\mathcal{K}$ und beide Parteien gleichen den Zeitpunkt ts ab.

$\boxed{\mathcal{K}unde}$ $(\mathsf{pk}_{\mathcal{H}}, \mathsf{sk}_{\mathcal{K}}, \mathbb{W})$	$(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{D}}, K, \mathsf{ts})$	$\boxed{\mathcal{H}ändler}$ $(\mathsf{sk}_{\mathcal{H}})$
$R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}} \Vert \mathsf{ts} \Vert K)$		$R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}} \Vert \mathsf{ts} \Vert K)$
$b_1, b_2 \in_{\mathcal{R}} \mathbb{Z}_p$		
$\kappa := \kappa_{0,0}$		
$S = \mathfrak{g}^{\kappa}$		
$T = \mathfrak{g}^{u}\mathfrak{g}_0^{R\kappa}$		
$B_1 = g_0^{b_1} g_1^{b_2}$		
$B_2 = \Sigma g_1^{b_1}$		
$T_C = \hat{e}(B_1^t, Z)$		
	$\xrightarrow{S, T, T_C, B_1, B_2}$	
$SoK\Big[(b_1, b_2, \beta_1, \beta_2, e, s, t, u, \kappa) : B_1 = g_0^{b_1} g_1^{b_2} \ \wedge\ 1 = B_1^e g_0^{-\beta_1} g_1^{-\beta_2} \ \wedge\ S = \mathfrak{g}^{\kappa} \wedge$		
$T = \mathfrak{g}^{u}\mathfrak{g}_0^{R\kappa} \ \wedge\ T_C = \hat{e}(B_1^t, Z) \ \wedge\ \frac{\hat{e}(B_2, X)}{\hat{e}(gV, h)} = \hat{e}\left(g_1^{b_1}, X\right)\hat{e}\left(g_0^{s} g_1^{u+\beta_1} g_2^{t} B_2^{-e}, h\right)\Big](R)$		
		Berechne $\kappa_{L+1,0}, \ldots, \kappa_{L+1,K-1}$
		Verifiziere SoK
$\mathbb{W}'$ oder $\perp$		coin oder $\perp$

Abbildung 7.6: Bezahlprotokoll Spend-K von FTG-I

Schritt 1: Der Händler $\mathcal{H}$ und der Kunde $\mathcal{K}$ berechnen beide den Hashwert $R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}} \Vert \mathsf{ts} \Vert K)$.

Schritt 2: Der Kunde $\mathcal{K}$ geht wie im teilbaren Geldsystem TG-I vor und berechnet zusätzlich den Münz-Identifikator $T_C = \hat{e}(B_1^t, Z)$. Folglich sendet $\mathcal{K}$ die Elemente S, T, T_C, B_1, B_2 an $\mathcal{H}$ und erstellt die folgende Signature-of-Knowledge SoK, wobei $\beta_1 = b_1 e, \beta_2 = b_2 e$ und $V = u_0^{\prod_{j=0}^{K-1}(\alpha + \kappa_{L+1,j})}$ ist:

$$SoK\Big[(b_1, b_2, \beta_1, \beta_2, e, s, t, u, \kappa) : B_1 = g_1^{b_1} g_2^{b_2} \ \wedge\ 1 = B_1^e g_1^{-\beta_1} g_2^{-\beta_2} \ \wedge\ S = \mathfrak{g}^{\kappa} \wedge$$

$$T = \mathfrak{g}^{u}\mathfrak{g}_0^{R\kappa} \wedge T_C = \hat{e}\left(B_1^t, Z\right) \wedge \frac{\hat{e}(B_2, X)}{\hat{e}(gV, h)} = \hat{e}\left(g_1^{b_1}, X\right)\hat{e}\left(g_0^{s} g_1^{u+\beta_1} g_2^{t} B_2^{-e}, h\right)\Big](R)\,.$$

Anschließend aktualisiert $\mathcal{K}$ seine Geldbörse zu $\mathbb{W}' = (\Sigma, e, s, t, \mathsf{info}')$.

Schritt 3: Der Händler $\mathcal{H}$ berechnet aus der Seriennummer S alle K Serienschlüssel sowie den Akkumulator V, verifiziert SoK und speichert die Münze

$\mathsf{coin} = (K, S, T, T_C, R, \Phi = (B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$ ab.

UTrace: Möchte die Bank $\mathcal{B}$ gemeinsam mit dem Deanonymisierer $\mathcal{D}$ den Eigentümer einer eingelösten Münze bestimmen, führen $\mathcal{B}$ und $\mathcal{D}$ das Kundenverfolgungsprotokoll UTrace durch, welches in Abbildung 7.7 dargestellt ist. Zuvor authentifizieren sich jedoch beide Parteien gegenseitig.

Schritt 1: Die Bank $\mathcal{B}$ sendet eine Münze $\mathsf{coin} = (k, S, T, T_C, R, \Phi)$ an $\mathcal{D}$.

Schritt 2: Der Deanonymisierer $\mathcal{D}$ verifiziert die Münze coin und berechnet die Kontonummer

$$TS^{-yR} = I\mathfrak{g}_0^{R\kappa}\mathfrak{g}^{-yR\kappa} = I\mathfrak{g}_0^{R\kappa}\mathfrak{g}_0^{-R\kappa} = I.$$

Anschließend sendet $\mathcal{D}$ die Kontonummer $\mathsf{pk}_{\mathcal{K}} = I$ an $\mathcal{B}$.

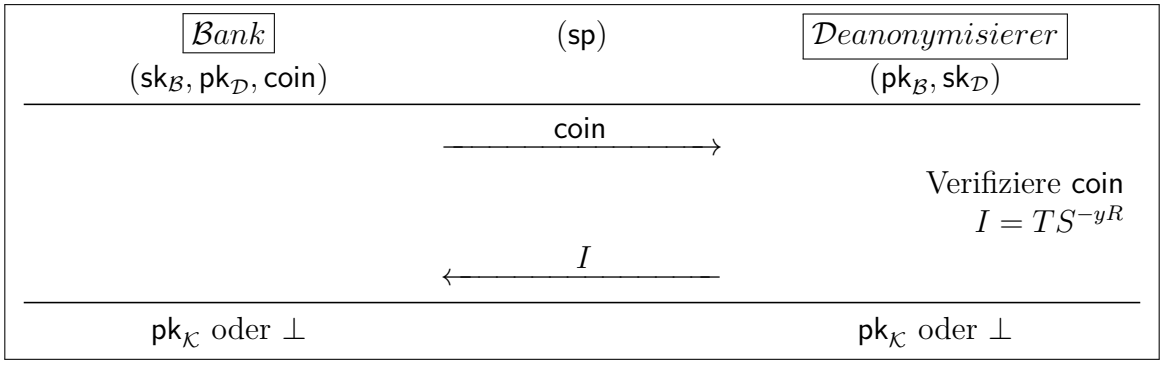

Abbildung 7.7: Kundenverfolgungsprotokoll UTrace von FTG-I

CTrace: Möchte die Bank $\mathcal{B}$ gemeinsam mit dem Deanonymisierer $\mathcal{D}$ alle Münzen einer zu einem bestimmten Abhebeprotokoll Withdraw gehörenden Geldbörse verfolgen, führen $\mathcal{B}$ und $\mathcal{D}$ das Münzverfolgungsprotokoll CTrace durch, welches in Abbildung 7.8 dargestellt ist. Zuvor authentifizieren sich jedoch beide Parteien gegenseitig.

Schritt 1: Die Bank $\mathcal{B}$ sendet die Informationen $(\mathbb{T} = (I, C, E, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = t'')$ eines bestimmten Abhebeprotokolls an $\mathcal{D}$.

Schritt 2: Der Deanonymisierer $\mathcal{D}$ verifiziert SoK_{W}, berechnet das Element $E^\star = E^z Z^{t''} = Z^{t'+t''} = Z^t$ für $t = t' + t'' \mod p$ und sendet $E^\star$ an $\mathcal{B}$.

Schritt 3: Die Bank $\mathcal{B}$ kann nun bei jeder Münze $\mathsf{coin}_i = (k_i, S_i, T_i, T_{C,i}, R_i, \Phi_i = (A_{1,i}, B_{1,i}, B_{2,i}, SoK_i, \mathsf{ts}_i, \mathsf{pk}_{\mathcal{H},i}))$ die Gleichung

$$T_{C,i} \stackrel{?}{=} \hat{e}\,(B_{1,i}, E^\star)$$

überprüfen, egal, ob diese bereits eingelöst wurde oder erst zukünftig verwendet wird. Die Bank kann das Element $E^\star$ auch auf einer schwarzen Liste veröffentlichen und so alle Münzen dieser Geldbörse sperren lassen.

Damit der Deanonymisierer eine mögliche Bestrafung überprüfen kann, sendet $\mathcal{B}$ alle Münzen coin_i mit $T_{C,i} = \hat{e}\,(B_{1,i}, E^\star)$ an $\mathcal{D}$. Es sei $\mathsf{COINS} := \{\mathsf{coin}_i : T_{C,i} = \hat{e}\,(B_{1,i}, E^\star)\}$ die Menge dieser Münzen.

Schritt 4: Der Deanonymisierer $\mathcal{D}$ verifiziert bei jeder Münze $\mathsf{coin}_i \in \mathsf{COINS}$ die Münze coin_i (vgl. Deposit bei TG-I) und die Gleichung $T_{C,i} \stackrel{?}{=} \hat{e}\,(B_{1,i}, E^\star)$.

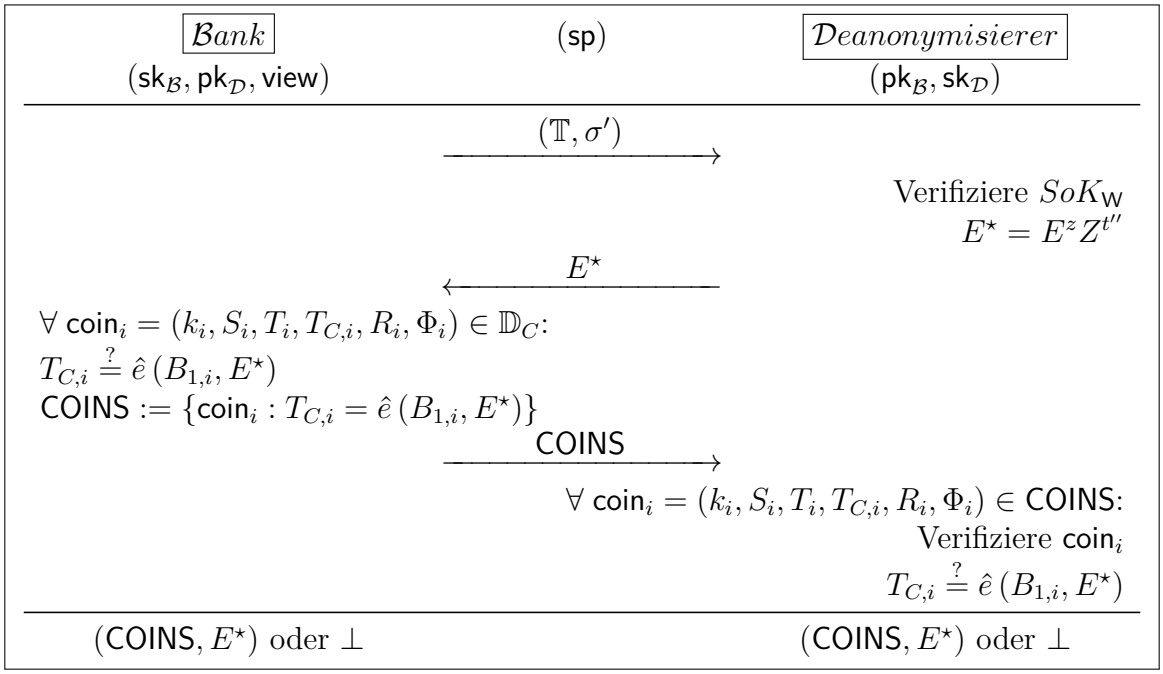

Abbildung 7.8: Münzverfolgungsprotokoll CTrace von FTG-I

7.3.1 Sicherheitsanalyse von FTG-I

Satz 7.3.1. *Das faire teilbare elektronische Geldsystem FTG-I ist im Random-Oracle-Modell ein sicheres faires elektronisches Geldsystem, falls die q-SDH-Annahme, die q-polyDH-Annahme und die DBDH-Annahme für* $\mathsf{Setup}_\mathsf{Bil}$ *gelten und die DDH-Annahme für* $\mathsf{Setup}_\mathsf{G}$ *gilt.*

Beweis. Wir beweisen Unfälschbarkeit, Anonymität, Double-Spending-Beschuldigung, Abhebe-Beschuldigung, korrekte Kundenverfolgung, Kundenverfolgung-Beschuldigung, korrekte Münzverfolgung und Münzverfolgung-Beschuldigung.

Unfälschbarkeit: Die Unfälschbarkeit des Geldsystems FTG-I folgt direkt aus der Unfälschbarkeit des teilbaren Geldsystems TG-I. Denn die Abhebe- bzw. Bezahlprotokolle bestehen im Wesentlichen aus denen des teilbaren Geldsystems TG-I und den zusätzlichen Elementen E bzw. T_C. Somit werden nun bei den Orakel-Anfragen entsprechend Nachrichten-Signatur-Paare $((u, t, \phi(\mathsf{x})), (\Sigma, e, s))$ erstellt bzw. extrahiert. Bei der Simulation der Initialisierung wählt $\mathcal{B}$ die privaten Schlüssel $y, z \in_{\mathcal{R}} \mathbb{Z}_p^*$ des Deanonymisierers. Das Element $\mathfrak{g}_0$ ist nun nicht Teil der Systemparameter sp sondern ein Teil des öffentlichen Schlüssels $\mathsf{pk}_{\mathcal{D}}$. Ansonsten geht $\mathcal{B}$ analog zu TG-I vor. Da der Angreifer $\mathcal{A}$ nun allerdings die beiden Orakel $\mathcal{O}^{\mathsf{UT}}_{\mathcal{B},\mathcal{D}}$ und $\mathcal{O}^{\mathsf{CT}}_{\mathcal{B},\mathcal{D}}$ befragen darf, wird beschrieben, wie $\mathcal{B}$ diese simuliert.

Simulation von $\mathcal{O}^{\mathsf{UT}}_{\mathcal{B},\mathcal{D}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den geheimen Schlüssel $y \in \mathbb{Z}_p^*$ kennt.

Simulation von $\mathcal{O}^{\mathsf{CT}}_{\mathcal{B},\mathcal{D}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den geheimen Schlüssel $z \in \mathbb{Z}_p^*$ kennt.

Vorbemerkungen zur Anonymität: Wir benötigen zunächst folgendes Lemma:

Lemma 7.3.1. *Angenommen, es gilt die DBDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$. *Seien* $q_0, q_1 \in \mathbb{N}$ *polynomiell beschränkt. Dann gibt es für jeden polynomiellen Angreifer* $\mathcal{A}$ *eine vernachlässigbare Funktion* ν, *so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\begin{aligned}
&\Big|\Pr\Big[\mathsf{sp}_{\mathsf{Bil}} \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right); g_0, g_1, g_{0,1}, \ldots, g_{0,q_0}, g_{1,1}, \ldots, g_{1,q_1} \in_{\mathcal{R}} \mathbb{G}_1; h_1 \in_{\mathcal{R}} \mathbb{G}_2;\\
&\qquad a_0, a_1 \in_{\mathcal{R}} \mathbb{Z}_p\colon\ 1 \leftarrow \mathcal{A}\Big(\mathsf{sp}_{\mathsf{Bil}}, g_0, g_1, \{g_{0,i}\}_{i=1}^{q_0}, \{g_{1,j}\}_{j=1}^{q_1}, h_1,\\
&\qquad \{\hat{e}\left(g_{0,i}, h_1\right)^{a_0}\}_{i=1}^{q_0}, \{\hat{e}\left(g_{1,j}, h_1\right)^{a_1}\}_{j=1}^{q_1}, h^{a_0}, h^{a_1}, \hat{e}\left(g_0, h_1\right)^{a_0}, \hat{e}\left(g_1, h_1\right)^{a_1}\Big)\Big]\\
&- \Pr\Big[\mathsf{sp}_{\mathsf{Bil}} \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right); g_0, g_1, g_{0,1}, \ldots, g_{0,q_0}, g_{1,1}, \ldots, g_{1,q_1} \in_{\mathcal{R}} \mathbb{G}_1; h_1 \in_{\mathcal{R}} \mathbb{G}_2;\\
&\qquad A, B \in_{\mathcal{R}} \mathbb{G}_T; a_0, a_1 \in_{\mathcal{R}} \mathbb{Z}_p\colon\ 1 \leftarrow \mathcal{A}\Big(\mathsf{sp}_{\mathsf{Bil}}, g_0, g_1, \{g_{0,i}\}_{i=1}^{q_0}, \{g_{1,j}\}_{j=1}^{q_1}, h_1,\\
&\qquad \{\hat{e}\left(g_{0,i}, h_1\right)^{a_0}\}_{i=1}^{q_0}, \{\hat{e}\left(g_{1,j}, h_1\right)^{a_1}\}_{j=1}^{q_1}, h^{a_0}, h^{a_1}, A, B\Big)\Big]\Big| \leq \nu(\lambda).
\end{aligned}$$

Beweis. Angenommen, es existiert ein polynomieller Angreifer $\mathcal{A}$, der mit Wahrscheinlichkeit $1/2 + \epsilon(\lambda)$ für eine nicht vernachlässigbare Funktion ϵ bei Eingabe des Tupels $\Big(\mathsf{sp}_{\mathsf{Bil}}, g_0, g_1, \{g_{0,i}\}_{i=1}^{q_0}, \{g_{1,j}\}_{j=1}^{q_1}, h_1, \{\hat{e}\left(g_{0,i}, h_1\right)^{a_0}\}_{i=1}^{q_0}, \{\hat{e}\left(g_{1,j}, h_1\right)^{a_1}\}_{j=1}^{q_1}, h^{a_0}, h^{a_1}, A, B\Big)$ entscheiden kann, ob $A, B \in_{\mathcal{R}} \mathbb{G}_T$ ist oder, ob $A = \hat{e}\left(g_0, h_1\right)^{a_0} \wedge B = \hat{e}\left(g_1, h_1\right)^{a_1}$ gilt.

Wir konstruieren einen polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das DBDH-Problem zu lösen.

Schlüsselgenerierung: Der Angreifer $\mathcal{B}$ erhält eine zufällige DBDH-Probleminstanz $(p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, h, g^a, g^b, g^c, h^b, h^c, \Gamma)$, wobei entweder $\Gamma = \hat{e}\left(g, h\right)^{abc}$ oder $\Gamma \in_{\mathcal{R}}$

$\mathbb{G}_T$ ist. Dann wählt $\mathcal{B}$ zufällig $q_0 + q_1 + 3$ Zahlen $\gamma, \gamma', \delta, \beta_{0,1}, \ldots, \beta_{0,q_0}, \beta_{1,1}, \ldots, \beta_{1,q_1} \in_{\mathcal{R}} \mathbb{Z}_p^*$ und definiert die Generatoren $g_0 := g^a, g_1 := (g^a)^\gamma g^{\gamma'}, g_{0,i} := g^{\beta_{0,i}}$, $g_{1,j} := g^{\beta_{1,j}}$ und $h_1 := h^c$ für $1 \leq i \leq q_0$ und $1 \leq j \leq q_1$. Weiter definiert $\mathcal{B}$ die Generatoren $\hat{e}(g_{0,i}, h_1)^{a_0} := \hat{e}\left(g^b, h_1^{\beta_{0,i}}\right), \hat{e}(g_{1,j}, h_1)^{a_1} := \hat{e}\left(g^b g^\delta, h_1^{\beta_{1,j}}\right), h^{a_0} := h^b$ und $h^{a_1} := h^b h^\delta$ für $1 \leq i \leq q_0$ und $1 \leq j \leq q_1$. Nach diesen Definitionen gilt $a_0 = b$ und $a_1 = b + \delta \mod p$. Anschließend definiert $\mathcal{B}$ die Elemente $A := \Gamma$ und $B := \Gamma^\gamma \hat{e}(g^a, h^c)^{\gamma\delta} \hat{e}\left(g^b, h^c\right)^{\gamma'} \hat{e}(g, h^c)^{\gamma'\delta}$.

Falls $\Gamma = \hat{e}(g, h)^{abc}$ ist, folgt $A = \hat{e}(g_0, h_1)^{a_0}$ sowie $B = \hat{e}(g, h)^{abc\gamma + ac\gamma\delta + bc\gamma' + c\gamma'\delta} = \hat{e}(g, h_1)^{(a\gamma+\gamma')(b+\delta)} = \hat{e}(g_1, h_1)^{a_1}$ wie gewünscht.

Falls $\Gamma \in_{\mathcal{R}} \mathbb{G}_T$ ist, sei $\Gamma = \hat{e}(g, h)^{abc+d}$ mit $d \in_{\mathcal{R}} \mathbb{Z}_p^*$. Dann folgt $A = \hat{e}(g_0, h_1)^{a_0} \hat{e}(g, h)^d$ und $B = \hat{e}(g_1, h_1)^{a_1} \hat{e}(g, h)^{d\gamma}$. Daher gilt $A, B \in_{\mathcal{R}} \mathbb{G}_T$ und somit sind die beiden Elemente perfekt simuliert.

Schließlich sendet $\mathcal{B}$ das perfekt simulierte Tupel $\Big(\mathsf{sp}_{\mathsf{Bil}}, g_0, g_1, \{g_{0,i}\}_{i=1}^{q_0}, \{g_{1,j}\}_{j=1}^{q_1}, h_1, \{\hat{e}(g_{0,i}, h_1)^{a_0}\}_{i=1}^{q_0}, \{\hat{e}(g_{1,j}, h_1)^{a_1}\}_{j=1}^{q_1}, h^{a_0}, h^{a_1}, A, B\Big)$ an den Angreifer $\mathcal{A}$, wobei $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, h)$ ist.

Ausgabe: Der Angreifer $\mathcal{A}$ gibt ein Bit $b \in \{0, 1\}$ aus, welches $\mathcal{B}$ ebenfalls ausgibt.

Somit löst der Angreifer $\mathcal{B}$ das DBDH-Problem mit derselben Wahrscheinlichkeit $1/2 + \epsilon(\lambda)$. □

Anonymität: Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Anonymität des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, die semantische Sicherheit des ElGamal-Verschlüsselungsverfahrens in der Gruppe $\mathbb{G}_p$ mit $\left(p, \mathbb{G}_p', \mathfrak{g}\right) \leftarrow \mathsf{Setup}_{\mathsf{G}}\left(1^\lambda\right)$ zu brechen, was ein Widerspruch zur DDH-Annahme für $\mathsf{Setup}_{\mathsf{G}}$ ist.

Der Angreifer $\mathcal{B}$ erhält den öffentlichen Schlüssel $\left(p, \mathbb{G}_p', \mathfrak{g}, \mathfrak{g}_0\right)$ des ElGamal-Verschlüsselungsverfahrens.

Simulation der Initialisierung: Der Angreifer $\mathcal{B}$ generiert die Parameter $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, h)$, wobei $\mathbb{G}_1 = \langle u_0 \rangle, \mathbb{G}_2 = \langle h \rangle$ und $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ eine bilineare Abbildung ist. Dann wählt $\mathcal{B}$ zufällig zwei Zahlen $\alpha, z \in_{\mathcal{R}} \mathbb{Z}_p^*$ und vier Generatoren $g, g_0, g_1, g_2 \in_{\mathcal{R}} \mathbb{G}_1$. Danach berechnet $\mathcal{B}$ die Elemente $Z = h^z, u_i = u_0^{\alpha^i}$ sowie $h_i = h^{\alpha^i}$ für $1 \leq i \leq K$. Anschließend sendet $\mathcal{B}$ die perfekt simulierten Systemparameter $\mathsf{sp} = \Big(p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \mathbb{G}_p', \hat{e}, g, g_0, g_1, g_2, u_0, \ldots, u_K, h, h_1, \ldots, h_K, \mathfrak{g}\Big)$ sowie den perfekt simulierten öffentlichen Schlüssel des Deanonymisierers $\mathsf{pk}_{\mathcal{D}} = (\mathfrak{g}_0, Z)$ an den Angreifer $\mathcal{A}$.

Der Angreifer $\mathcal{A}$ generiert den öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}} = X$ der Bank und sendet diesen an $\mathcal{B}$.

Simulation der Probe-Phase: Der Angreifer $\mathcal{B}$ verhält sich bei allen Orakel-Anfragen, außer bei $\mathcal{O}_{\mathcal{D}}^{\mathsf{UT}}$, wie das jeweilige Orakel und kann somit diese Anfragen perfekt simulieren.

Simulation von $\mathcal{O}_{\mathcal{D}}^{\mathsf{UT}}$: Der Angreifer $\mathcal{B}$ erhält eine Münze $\mathsf{coin} = (k, S, T, T_C, R, \Phi = (A_1, B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$.

- Falls der Angreifer $\mathcal{B}$ die Münze coin zum Bezahlen verwendet hat, gibt es einen Eintrag $(\cdot, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$. Somit kennt $\mathcal{B}$ die Kontonummer $I = \mathsf{pk}_{\mathcal{K}}$ und gibt diese aus, wodurch die Anfrage perfekt simuliert ist.
- Falls der Angreifer $\mathcal{B}$ die Münze coin nicht zum Bezahlen verwendet hat, wurde SoK vom Angreifer $\mathcal{A}$ erstellt. Daher kann $\mathcal{B}$ die Zahlen $u, \kappa \in \mathbb{Z}_p$ mit $S = \mathfrak{g}^{\kappa}$ sowie $T = \mathfrak{g}^{u}\mathfrak{g}_0^{R\kappa}$ aus SoK extrahieren und die Kontonummer $I = \mathfrak{g}^{u}$ korrekt berechnen. Dann gibt $\mathcal{B}$ die Kontonummer I aus, wodurch die Anfrage perfekt simuliert ist. Falls $\mathcal{B}$ die Zahlen u, κ nicht extrahieren kann, bricht $\mathcal{B}$ ab. Dies ist aber nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Simulation der Challenge-Phase: Der Angreifer $\mathcal{B}$ geht wie in TG-I in Abschnitt 6.5 beschrieben vor. Somit sind $B_{1,b}, B_{1,\bar{b}} \in_{\mathcal{R}} \mathbb{G}_1$ zufällige Generatoren. Zusätzlich wählt $\mathcal{B}$ zufällig zwei Münz-Identifikatoren $T_{C,b}, T_{C,\bar{b}} \in_{\mathcal{R}} \mathbb{G}_T$. Schließlich sendet $\mathcal{B}$ die beiden Münzen $\mathsf{coin}_b = (k_b, S_b, T_b, T_{C,b}, R_b, A_{1,b}, B_{1,b}, B_{2,b}, SoK_b, \mathsf{ts}_b, \mathsf{pk}_b)$ und $\mathsf{coin}_{\bar{b}} = \left(k_{\bar{b}}, S_{\bar{b}}, T_{\bar{b}}, T_{C,\bar{b}}, R_{\bar{b}}, A_{1,\bar{b}}, B_{1,\bar{b}}, B_{2,\bar{b}}, SoK_{\bar{b}}, \mathsf{ts}_{\bar{b}}, \mathsf{pk}_{\bar{b}}\right)$ an den Angreifer $\mathcal{A}$.

Nun müssen wir zeigen, dass beide Münzen rechnerisch ununterscheidbar simuliert sind. Im Gegensatz zum teilbaren Geldsystem TG-I kennt der Angreifer $\mathcal{A}$ zusätzlich pro Abhebeanfrage ein Paar $\left(E = h^{t'}, t''\right) \in \mathbb{G}_2 \times \mathbb{Z}_p$ und pro Bezahlanfrage ein Paar $(B, T_C) \in \mathbb{G}_1 \times \mathbb{G}_T$, mit der Eigenschaft $T_C = \hat{e}\,(B^t, Z)$ für $t = t' + t'' \mod p$, wobei zur Simplifizierung $B := B_1$ ist. Im Folgenden wird gezeigt, warum diese Werte dem Angreifer $\mathcal{A}$ unter der DBDH-Annahme nur mit vernachlässigbarer Wahrscheinlichkeit helfen, das Spiel zu gewinnen.

Seien $\mathsf{view}_0 = (I_0, \cdots, E_0, \cdots, t_0'')$ sowie $\mathsf{view}_1 = (I_1, \cdots, E_1, \cdots, t_1'')$ die beiden Protokollansichten und $\mathbb{W}_0$ sowie $\mathbb{W}_1$ die beiden Geldbörsen bzgl. der Indizes i_0 und i_1. Seien weiter $(B_{0,i}, T_{C,0,i})$ bzw. $(B_{1,j}, T_{C,1,j})$ für $1 \leq i \leq q_0$ bzw. $1 \leq j \leq q_1$ die Paare der entsprechenden Münzen, die mit $\mathbb{W}_0$ bzw. $\mathbb{W}_1$ bezahlt wurden.

Wir müssen zeigen, dass die Münz-Identifikatoren $T_{C,b}, T_{C,\bar{b}}$ rechnerisch ununterscheidbar simuliert sind. Da $\mathcal{B}$ die beiden Elemente $T_{C,b}, T_{C,\bar{b}} \in_{\mathcal{R}} \mathbb{G}_T$ zufällig wählt, sind diese nicht korrekt berechnet. Doch dies bemerkt $\mathcal{A}$ nach Lemma 7.3.1 nur mit vernachlässigbarer Wahrscheinlichkeit unter der DBDH-Annahme, denn das dem Angreifer $\mathcal{A}$ bekannte Tupel

$$(\mathsf{sp}_{\mathsf{Bil}}, B_b, B_{\bar{b}}, \{B_{0,i}\}_{i=1}^{q_0}, \{B_{1,j}\}_{j=1}^{q_1}, Z, \{T_{C,0,i}\}_{i=1}^{q_0}, \{T_{C,1,j}\}_{j=1}^{q_1}, E_0 h^{t_0''}, E_1 h^{t_1''}, T_{C,b}, T_{C,\bar{b}})$$

entspricht dem Tupel

$$(\mathsf{sp}_{\mathsf{Bil}}, g_0, g_1, \{g_{0,i}\}_{i=1}^{q_0}, \{g_{1,j}\}_{j=1}^{q_1}, h_1, \{\hat{e}\,(g_{0,i}, h_1)^{a_0}\}_{i=1}^{q_0}, \{\hat{e}\,(g_{1,j}, h_1)^{a_1}\}_{j=1}^{q_1}, h^{a_0}, h^{a_1}, A, B)$$

aus Lemma 7.3.1. Damit sind die beiden Münz-Identifikatoren rechnerisch ununterscheidbar simuliert.

Somit sind beide Münzen coin_b und $\mathsf{coin}_{\bar{b}}$ rechnerisch ununterscheidbar bzgl. der Kontonummern I_b und $I_{\bar{b}}$ simuliert.

Anschließend aktualisiert $\mathcal{B}$ die beiden Geldbörsen $\mathbb{W}_0$ und $\mathbb{W}_1$ jeweils so, als würde bei beiden mit dem Geldbetrag $\max\{k_b, k_{\bar{b}}\}$ bezahlt. Daher gilt für die Einträge nun $(i_0, \mathbb{W}'_0, I_0, \$'_0 = \$_0 - \max\{k_b, k_{\bar{b}}\})$ bzw. $(i_1, \mathbb{W}'_1, I_1, \$'_1 = \$_1 - \max\{k_b, k_{\bar{b}}\})$ und somit wurde ein Eintrag korrekt und der andere falsch aktualisiert.

Simulation der Post-Challenge-Phase: Der Angreifer $\mathcal{B}$ verhält sich bei allen Orakel-Anfragen, außer bei $\mathcal{O}_{\mathcal{K}}^{\mathsf{S}^\star}$ und $\mathcal{O}_{\mathcal{K},\mathcal{H}}^{\mathsf{S}^\star}$, wie bei der Simulation der Probe-Phase.

Simulation von $\mathcal{O}_{\mathcal{K}}^{\mathsf{S}^\star}$ und $\mathcal{O}_{\mathcal{K},\mathcal{H}}^{\mathsf{S}^\star}$: Für jeden Index $i \notin \{i_0, i_1\}$ verhält sich der Angreifer $\mathcal{B}$ wie das Orakel.

Falls der Index $i \in \{i_0, i_1\}$ ist, kann sich der Angreifer $\mathcal{B}$ nicht genau wie das Orakel verhalten, da $\mathcal{B}$ die beiden Geldbörsen $\mathbb{W}_0$ und $\mathbb{W}_1$ wie oben beschrieben anders als die Orakel aktualisiert. Der Angreifer $\mathcal{B}$ verhält sich aber wie der entsprechende ehrliche Kunde mit aktualisierter Geldbörse, wodurch die Anfrage im Random-Oracle-Modell perfekt simuliert ist, da der Angreifer $\mathcal{A}$ in dieser Phase bzgl. $\hat{b} \in \{0, 1\}$ nur Münzen im Gesamtwert von $\$_{\hat{b}} - \max\{k_b, k_{\bar{b}}\}$ anfragen darf.

Spielende: Der Angreifer $\mathcal{A}$ gibt mit Wahrscheinlichkeit $1/2 + \epsilon(\lambda)$ das korrekte Bit b aus, wobei ϵ eine nicht vernachlässigbare Funktion ist. Der Angreifer $\mathcal{B}$ leitet das Bit b an das ElGamal-Verschlüsselungs-Orakel weiter und bricht somit die semantische Sicherheit des ElGamal-Verschlüsselungsverfahrens mit der Wahrscheinlichkeit $1/2 + \epsilon(\lambda) - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist.

Double-Spending-Beschuldigung: Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Double-Spending-Beschuldigung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das DLog-Problem in der Gruppe $\mathbb{G}_p$ zu lösen.

Der Angreifer $\mathcal{B}$ erhält die DLog-Probleminstanz $\left(p, \mathbb{G}'_p, \mathfrak{g}, \mathfrak{g}^a\right)$ und soll den diskreten Logarithmus $a \in \mathbb{Z}_p^*$ berechnen.

Simulation der Initialisierung: Der Angreifer $\mathcal{B}$ generiert die Parameter $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \hat{e}, u_0, h)$, wobei $\mathbb{G}_1 = \langle u_0 \rangle, \mathbb{G}_2 = \langle h \rangle$ und $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ eine bilineare Abbildung ist. Dann wählt $\mathcal{B}$ zufällig drei Zahlen $\alpha, y, z \in_{\mathcal{R}} \mathbb{Z}_p^*$ und vier Generatoren $g, g_0, g_1, g_2 \in_{\mathcal{R}} \mathbb{G}_1$. Danach berechnet $\mathcal{B}$ die Generatoren $\mathfrak{g}_0 = \mathfrak{g}^y, Z = h^z, u_i = u_0^{\alpha^i}$ sowie $h_i = h^{\alpha^i}$ für $1 \leq i \leq K$. Anschließend sendet $\mathcal{B}$ die perfekt simulierten Systemparameter $\mathsf{sp} = (p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \mathbb{G}'_p, \hat{e}, g, g_0, g_1, g_2, u_0, \ldots, u_K, h, h_1, \ldots, h_K, \mathfrak{g})$ sowie den perfekt simulierten öffentlichen Schlüssel des Deanonymisierers $\mathsf{pk}_{\mathcal{D}} = (\mathfrak{g}_0, Z)$ an den Angreifer $\mathcal{A}$.

Der Angreifer $\mathcal{A}$ generiert den öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}} = X$ der Bank und sendet diesen an $\mathcal{B}$.

Simulation von $\mathcal{O}^{\mathsf{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{A}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{A}}_{\mathcal{K}}$: Der Angreifer $\mathcal{B}$ wählt zufällig eine Zahl $\gamma \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die perfekt simulierte Kontonummer $I = \mathfrak{g}^a\mathfrak{g}^\gamma = \mathfrak{g}^{a+\gamma} = \mathfrak{g}^u$ für $u := a + \gamma \mod p$. Dann speichert $\mathcal{B}$ den Eintrag (I, γ) in einer Liste $\mathcal{U}_H$.

Simulation von $\mathcal{O}^{\mathsf{W}}_{\mathcal{K}}$: Der Angreifer $\mathcal{B}$ wählt zufällig die drei Zahlen $\kappa_{0,0} \in_{\mathcal{R}} \mathbb{Z}_p^*$ und $t', a_0 \in_{\mathcal{R}} \mathbb{Z}_p$ sowie das Element $C \in_{\mathcal{R}} \mathbb{G}_1$ und berechnet die Elemente E, A_0 gemäß Protokoll. Somit sind die Elemente C, E, A_0 perfekt simuliert. Anschließend simuliert $\mathcal{B}$ die Signature-of-Knowledge SoK_{W} perfekt. Nach erfolgreicher Durchführung wird ein Eintrag $(i, \mathbb{W}_i = (\Sigma, e, s'', t', t'', \mathsf{info}), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$ sowie ein Eintrag $(i, \mathbb{T} = (I, C, E, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K))$ in einer Liste $\mathbb{T}_H$ gespeichert und der Zähler i um 1 erhöht.

Falls die Anfrage nicht erfolgreich durchgeführt wurde, wird nur ein Eintrag $(i, \mathbb{T} = (I, C, E, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K))$ in einer Liste $\mathbb{T}_H$ gespeichert und der Zähler i um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{S}^\star}_{\mathcal{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird u. a. ein Eintrag (j, coin) in einer Liste $\mathbb{C}_{\mathcal{A}}$ gespeichert und der Zähler j um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{S}^\star}_{\mathcal{K}}$: Der Angreifer $\mathcal{B}$ verwendet den Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i) \in \mathbb{W}_H$, berechnet die Elemente S, T_C, A_1, B_1, B_2 gemäß Protokoll und berechnet das korrekte Sicherheitstag $T = I\mathfrak{g}_0^{R\kappa}$ ohne Kenntnis des privaten Schlüssels u, wobei $\kappa \in \mathbb{Z}_p^*$ der Schlüssel mit $S = \mathfrak{g}^\kappa$ ist. Dann simuliert $\mathcal{B}$ die Signature-of-Knowledge SoK perfekt. Anschließend wird der Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i)$ zu $(i, \mathbb{W}'_i, \mathsf{pk}_{\mathcal{K}}, \$_i - k)$ aktualisiert und ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_{H,\mathcal{A}}$ gespeichert.

Simulation von $\mathcal{O}^{\mathsf{S}^\star}_{\mathcal{K},\mathcal{H}}$: Der Angreifer $\mathcal{B}$ verwendet den Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i) \in \mathbb{W}_H$, berechnet die Elemente S, T_C, A_1, B_1, B_2 gemäß Protokoll und berechnet das korrekte Sicherheitstag $T = I\mathfrak{g}_0^{R\kappa}$ ohne Kenntnis des privaten Schlüssels u, wobei $\kappa \in \mathbb{Z}_p^*$ der Schlüssel mit $S = \mathfrak{g}^\kappa$ ist. Dann simuliert $\mathcal{B}$ die Signature-of-Knowledge SoK perfekt. Anschließend wird der Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i)$ zu $(i, \mathbb{W}'_i, \mathsf{pk}_{\mathcal{K}}, \$_i - k)$ aktualisiert und ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_H$ sowie ein Eintrag (j, coin) in einer Liste $\mathbb{C}$ gespeichert und der Zähler j um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{D}}_{\mathcal{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{UT}}_{\mathcal{D}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den geheimen Schlüssel $y \in \mathbb{Z}_p^*$ kennt.

Simulation von $\mathcal{O}_D^{\mathsf{CT}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den geheimen Schlüssel $z \in \mathbb{Z}_p^*$ kennt.

Spielende: Das Spielende verläuft wie beim teilbaren Geldsystem TG-I, außer dass eine Münze coin nun zusätzlich das Element T_C enthält. Dies muss für den Beweis jedoch nicht berücksichtigt werden.

Somit löst $\mathcal{B}$ das DLog-Problem in $\mathbb{G}_p$ mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist.

Abhebe-Beschuldigung: Dies folgt direkt aus dem teilbaren Geldsystem TG-I.

Korrekte Kundenverfolgung: Dies folgt direkt aus der Unfälschbarkeit dieses Geldsystems. Sei $\mathcal{A}$ ein polynomieller Angreifer, der die korrekte Kundenverfolgung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das q-SDH-Problem zu lösen. Der Angreifer $\mathcal{A}$ gibt am Ende eine Münze $\mathsf{coin} = (k, S, T, T_C, R, \Phi = (A_1, B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}})) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ aus.

Somit wurde SoK vom Angreifer $\mathcal{A}$ erstellt und daher kann $\mathcal{B}$ die Zahl $\kappa \in \mathbb{Z}_p$ und das Nachrichten-Signatur-Paar $((u, t, \phi(\mathsf{x})), (\Sigma, e, s))$ mit $S = \mathfrak{g}^{\kappa}$ und $T = \mathfrak{g}^{u}\mathfrak{g}_0^{R\kappa}$ aus SoK extrahieren. Da kein Kunde $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_{\mathcal{A}}$ eindeutig durch das Protokoll UTrace ausgegeben wird, folgt $\mathfrak{g}^{u} \notin \mathcal{U}_{\mathcal{A}}$. Somit wurde das Nachrichten-Signatur-Paar $((u, t, \phi(\mathsf{x})), (\Sigma, e, s))$ nie bei einer $\mathcal{O}_{\mathcal{B}}^{\mathsf{W}}$-Anfrage erstellt.

- Falls das obige Nachrichten-Signatur-Paar niemals bei einer Orakel-Anfrage erstellt wurde, geht $\mathcal{B}$ wie bei der Unfälschbarkeit im Fall (1.1) vor.
- Falls das obige Nachrichten-Signatur-Paar bei einer $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{W}}$-Anfrage erstellt wurde, geht $\mathcal{B}$ wie bei der Unfälschbarkeit im Fall (1.2) vor.

Kundenverfolgung-Beschuldigung: Dies folgt im Wesentlichen analog zur Double-Spending-Beschuldigung dieses Geldsystems. Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Kundenverfolgung-Beschuldigung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das DLog-Problem in $\mathbb{G}_p$ zu lösen.

Simulation: Der Angreifer $\mathcal{B}$ erhält dieselben Parameter wie bei der Double-Spending-Beschuldigung und simuliert die Initialisierung sowie die Orakel-Anfragen genauso.

Spielende: Der Angreifer $\mathcal{A}$ gibt mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ eine Münze $\mathsf{coin} = (k, S, T, T_C, R, \Phi = (A_1, B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$ aus, so dass die Ausgabe des Kundenverfolgungsprotokolls UTrace eine Kontonummer $\mathsf{pk}_{\mathcal{K}}$ ist, wobei $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H$ oder $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ gilt, obwohl $(\cdot, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ ist.

Da der Angreifer $\mathcal{B}$ beim Protokoll UTrace für eine Münze coin mit $(\cdot, \mathsf{pk}'_{\mathcal{K}}, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ immer die korrekte (eindeutige) Kontonummer $\mathsf{pk}'_{\mathcal{K}}$ ausgibt, muss $(\cdot, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und damit $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H$ gelten. Somit wurde die Münze coin und die Signature-of-Knowledge SoK vom Angreifer $\mathcal{A}$ erstellt und daher kann $\mathcal{B}$ die Zahlen $u, \kappa \in \mathbb{Z}_p$ mit $S = \mathfrak{g}^{\kappa}$ und $T = \mathfrak{g}^{u}\mathfrak{g}_0^{R\kappa}$ aus SoK extrahieren. Der Angreifer $\mathcal{B}$ sucht das Paar $(I = \mathfrak{g}^{u}, \gamma) \in \mathcal{U}_H$ und berechnet den diskreten Logarithmus $a = u - \gamma \mod p$.

Somit löst $\mathcal{B}$ das DLog-Problem in $\mathbb{G}_p$ mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist.

Korrekte Münzverfolgung: Dies folgt (ebenfalls wie die korrekte Kundenverfolgung) direkt aus der Unfälschbarkeit dieses Geldsystems. Sei $\mathcal{A}$ ein polynomieller Angreifer, der die korrekte Kundenverfolgung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das q-SDH-Problem zu lösen. Der Angreifer $\mathcal{A}$ gibt eine Münze $\mathsf{coin} = (k, S, T, T_C, R, \Phi = (A_1, B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}})) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ aus.

Somit wurde SoK vom Angreifer $\mathcal{A}$ erstellt und daher kann $\mathcal{B}$ das Nachrichten-Signatur-Paar $((u, t, \phi(\mathsf{x})), (\Sigma, e, s))$ mit $T_C = \hat{e}(B_1^t, Z)$ aus SoK extrahieren. Da die Münze coin nicht eindeutig durch das Protokoll CTrace verfolgt werden kann, folgt entweder (1) $T_C \neq \hat{e}\left(B_1, E_i^z Z^{t''_i}\right) = \hat{e}\left(B_1^{t_i}, Z\right)$ für jede Protokollansicht $\mathsf{view}_i = (\cdots, E_i, \cdots, t''_i) \in \mathsf{view}_{\mathcal{A}}$ oder (2) es gibt zwei verschiedene Protokollansichten $\mathsf{view}_0 = (\cdots, E_0, \cdots, t''_0), \mathsf{view}_1 = (\cdots, E_1, \cdots, t''_1) \in \mathsf{view}_{\mathcal{A}} \cup \mathsf{view}_H$ mit $T_C = \hat{e}\left(B_1, E_0^z Z^{t''_0}\right) = \hat{e}\left(B_1, E_1^z Z^{t''_1}\right)$.

Wir zeigen zunächst, dass Fall (2) vernachlässigbar ist. Seien $t'_0, t'_1 \in \mathbb{Z}_p$ die während der Abhebeanfragen (von $\mathcal{A}$ oder $\mathcal{B}$) gewählten Zahlen mit $E_0 = h^{t'_0}, E_1 = h^{t'_1}$ und $t''_0, t''_1 \in_{\mathcal{R}} \mathbb{Z}_p$ die zugehörenden von $\mathcal{B}$ zufällig gewählten Zahlen. Dann folgt $E_0^z Z^{t''_0} = Z^{t'_0+t''_0} = Z^{t'_1+t''_1} = E_1^z Z^{t''_1}$ und somit $t''_1 = t'_0 + t''_0 - t'_1 \mod p$. Da allerdings $t''_1 \in_{\mathcal{R}} \mathbb{Z}_p$ zufällig von $\mathcal{B}$ gewählt wird, ist dies vernachlässigbar.

Im Fall (1) wurde das Nachrichten-Signatur-Paar nie bei einer $\mathcal{O}_{\mathcal{B}}^{\mathsf{W}}$-Anfrage erstellt.

- Falls das obige Nachrichten-Signatur-Paar niemals bei einer Orakel-Anfrage erstellt wurde, geht $\mathcal{B}$ wie bei der Unfälschbarkeit im Fall (1.1) vor.

- Falls das obige Nachrichten-Signatur-Paar bei einer $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{W}}$-Anfrage erstellt wurde, geht $\mathcal{B}$ wie bei der Unfälschbarkeit im Fall (1.2) vor.

Münzverfolgung-Beschuldigung: Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Münzverfolgung-Beschuldigung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das DLog-Problem in der Gruppe $\mathbb{G}_2$ zu lösen.

Der Angreifer $\mathcal{B}$ erhält die Parameter $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \hat{e}, u_0, h)$ sowie das Paar $(u_0^a, h^a) \in \mathbb{G}_1 \times \mathbb{G}_2$ und soll den diskreten Logarithmus $a \in \mathbb{Z}_p^*$ berechnen.

Simulation der Initialisierung: Der Angreifer $\mathcal{B}$ generiert eine zyklische Gruppe $\mathbb{G}_p$ der Ordnung p. Dann wählt $\mathcal{B}$ zufällig drei Zahlen $\alpha, y, z \in_{\mathcal{R}} \mathbb{Z}_p^*$, vier Generatoren $g, g_0, g_1, g_2 \in_{\mathcal{R}} \mathbb{G}_1$ und einen Generator $\mathfrak{g} \in_{\mathcal{R}} \mathbb{G}_p$. Danach berechnet $\mathcal{B}$ die Generatoren $\mathfrak{g}_0 = \mathfrak{g}^y, Z = h^z$ und $u_i = u_0^{\alpha^i}$ sowie $h_i = h^{\alpha^i}$ für $1 \leq i \leq K$. Anschließend sendet $\mathcal{B}$ die perfekt simulierten Systemparameter $\mathsf{sp} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \mathbb{G}_p', \hat{e}, g, g_0, g_1, g_2, u_0, \ldots, u_K, h, h_1, \ldots, h_K, \mathfrak{g})$ sowie den perfekt simulierten öffentlichen Schlüssel des Deanonymisierers $\mathsf{pk}_{\mathcal{D}} = (\mathfrak{g}_0, Z)$ an den Angreifer $\mathcal{A}$.

Der Angreifer $\mathcal{A}$ generiert den öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}} = X$ der Bank und sendet diesen an $\mathcal{B}$.

Simulation von $\mathcal{O}^{\mathsf{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{A}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}_{\mathcal{K}}^{\mathsf{A}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}_{\mathcal{K}}^{\mathsf{W}}$: Der Angreifer $\mathcal{B}$ wählt zufällig die Zahlen $\kappa_{0,0} \in_{\mathcal{R}} \mathbb{Z}_p^*$ und $\delta, a_0 \in_{\mathcal{R}} \mathbb{Z}_p$ sowie das Element $C \in_{\mathcal{R}} \mathbb{G}_1$ und berechnet das Element A_0 gemäß Protokoll. Dann berechnet $\mathcal{B}$ das Element $E = (h^a)^{\delta} = h^{a\delta} = h^{t'}$ für $t' := a \cdot \delta \mod p$. Somit sind die Elemente C, E, A_0 perfekt simuliert. Anschließend simuliert $\mathcal{B}$ die Signature-of-Knowledge SoK_{W} perfekt. Nach erfolgreicher Durchführung wird ein Eintrag $(i, \mathbb{W}_i = (\delta, \Sigma, e, s'', t'', \mathsf{info}), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$ sowie ein Eintrag $(i, \mathbb{T} = (I, C, E, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K))$ in einer Liste $\mathbb{T}_H$ gespeichert und der Zähler i um 1 erhöht.

Falls die Anfrage nicht erfolgreich durchgeführt wurde, wird ein Eintrag $(i, \mathbb{W}_i = (\delta, \perp, \perp, \perp, \perp, \mathsf{info}), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$ sowie ein Eintrag $(i, \mathbb{T} = (I, C, E, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K))$ in einer Liste $\mathbb{T}_H$ gespeichert und der Zähler i um 1 erhöht.

Simulation von $\mathcal{O}_{\mathcal{H}}^{\mathsf{S}^\star}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird u. a. ein Eintrag (j, coin) in einer Liste $\mathbb{C}_{\mathcal{A}}$ gespeichert und der Zähler j um 1 erhöht.

Simulation von $\mathcal{O}_{\mathcal{K}}^{\mathsf{S}^\star}$: Der Angreifer $\mathcal{B}$ verwendet den Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i) \in \mathbb{W}_H$, berechnet die Elemente S, T, A_1, B_1, B_2 gemäß Protokoll und berechnet den korrekten Münz-Identifikator $T_C = \hat{e}\left(B_1, (h^a)^{\delta z}\, Z^{t''}\right) = \hat{e}\left(B_1, Z^{a\delta + t''}\right) = \hat{e}\left(B_1^t, Z\right)$ ohne Kenntnis der Zahl t. Dann simuliert $\mathcal{B}$ die Signature-of-Knowledge SoK perfekt. Anschließend wird der Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i)$ zu $(i, \mathbb{W}_i', \mathsf{pk}_{\mathcal{K}}, \$_i - k)$ aktualisiert und ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_{H,\mathcal{A}}$ gespeichert.

Simulation von $\mathcal{O}_{\mathcal{K},\mathcal{H}}^{\mathsf{S}^\star}$: Der Angreifer $\mathcal{B}$ verwendet den Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i) \in \mathbb{W}_H$, berechnet die Elemente S, T, A_1, B_1, B_2 gemäß Protokoll und berechnet den korrekten Münz-Identifikator $T_C = \hat{e}\left(B_1, (h^a)^{\delta z}\, Z^{t''}\right) = \hat{e}\left(B_1, Z^{a\delta + t''}\right) = \hat{e}\left(B_1^t, Z\right)$ ohne Kenntnis der Zahl t. Dann simuliert $\mathcal{B}$ die Signature-of-Knowledge SoK perfekt.

Anschließend wird der Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i)$ zu $(i, \mathbb{W}'_i, \mathsf{pk}_{\mathcal{K}}, \$_i - k)$ aktualisiert, ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_H$ sowie ein Eintrag (j, coin) in einer Liste $\mathbb{C}$ gespeichert und der Zähler j um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{D}}_{\mathcal{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{UT}}_{\mathcal{D}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den geheimen Schlüssel $y \in \mathbb{Z}^*_p$ kennt.

Simulation von $\mathcal{O}^{\mathsf{CT}}_{\mathcal{D}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den geheimen Schlüssel $z \in \mathbb{Z}^*_p$ kennt.

Spielende: Der Angreifer $\mathcal{A}$ sendet mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ Informationen $(\mathbb{T} = (I, C, E, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = t'')$ an $\mathcal{B}$. Sei $\mathsf{coin} = (k, S, T, T_C, R, \Phi = (A_1, B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$ eine Münze, die u. a. durch das Protokoll CTrace ausgegeben wird. Folglich gilt $T_C = \hat{e}\left(B_1, E^z Z^{t''}\right)$, obwohl

- $(i, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(j, \mathbb{T}) \in \mathbb{T}_H$ mit $j \neq i$ ist, oder
- $(i, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(\cdot, \mathbb{T}) \notin \mathbb{T}_H$ ist, oder
- $(\cdot, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(j, \mathbb{T}) \in \mathbb{T}_H$ ist.

- $(i, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(j, \mathbb{T}) \in \mathbb{T}_H$ mit $j \neq i$:

  In diesem Fall gibt es einen Eintrag $(j, \mathbb{W}_j = (\delta, \cdots), \cdot, \cdot) \in \mathbb{W}_H$. Zudem gibt es die Einträge $(i, \mathbb{W}_i = (\delta_i, \cdots, t''_i, \cdots), \cdot, \cdot) \in \mathbb{W}_H$ und $(i, \cdot, \mathsf{coin} = (\cdots, T_C, \cdots, B_1, \cdots)) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ mit $T_C = \hat{e}\left(B_1^{a\delta_i + t''_i}, Z\right)$. Weiter muss nach Voraussetzung aber auch $T_C = \hat{e}\left(B_1, E^z Z^{t''}\right) = \hat{e}\left(B_1^{a\delta + t''}, Z\right)$ gelten, woraus $a\delta_i + t''_i = a\delta + t''$ folgt. Somit berechnet $\mathcal{B}$ den diskreten Logarithmus $a = (t'' - t''_i)/(\delta_i - \delta) \mod p$, da der Fall $\delta_i = \delta$ vernachlässigbar ist.

- $(i, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(\cdot, \mathbb{T}) \notin \mathbb{T}_H$:

  In diesem Fall wurde SoK_{W} von $\mathcal{A}$ erstellt und $\mathcal{B}$ kann die Zahl $t' \in \mathbb{Z}_p$ mit $E = h^{t'}$ extrahieren. Zudem gibt es die Einträge $(i, \mathbb{W}_i = (\delta_i, \cdots, t''_i, \cdots), \cdot, \cdot) \in \mathbb{W}_H$ und $(i, \cdot, \mathsf{coin} = (\cdots, T_C, \cdots, B_1, \cdots)) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ mit $T_C = \hat{e}\left(B_1^{a\delta_i + t''_i}, Z\right)$. Weiter muss nach Voraussetzung aber auch $T_C = \hat{e}\left(B_1, E^z Z^{t''}\right) = \hat{e}\left(B_1^{t' + t''}, Z\right)$ gelten, woraus $a\delta_i + t''_i = t' + t''$ folgt. Somit berechnet $\mathcal{B}$ den diskreten Logarithmus $a = (t' + t'' - t''_i)/\delta_i \mod p$, da der Fall $\delta_i = 0$ vernachlässigbar ist.

- $(\cdot, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(j, \mathbb{T}) \in \mathbb{T}_H$:

  In diesem Fall wurde SoK von $\mathcal{A}$ erstellt und $\mathcal{B}$ kann die Zahl $t \in \mathbb{Z}_p$ mit $T_C = \hat{e}(B_1^t, Z)$ extrahieren. Zudem gibt es einen Eintrag $(j, \mathbb{W}_j = (\delta, \cdots), \cdot, \cdot) \in \mathbb{W}_H$. Nach Voraussetzung muss aber auch $T_C = \hat{e}\left(B_1, E^z Z^{t''}\right) = \hat{e}\left(B_1^{a\delta + t''}, Z\right)$ gelten,

woraus $a\delta + t'' = t$ folgt. Somit berechnet $\mathcal{B}$ den diskreten Logarithmus $a = (t - t'')/\delta \mod p$, da der Fall $\delta = 0$ vernachlässigbar ist.

Somit löst $\mathcal{B}$ das DLog-Problem in $\mathbb{G}_2$ mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist. □

Bemerkung 7.3.1. *Analog zu Bemerkung 6.5.6 sind die Sicherheitsziele Anonymität, Double-Spending-Beschuldigung, Abhebe-Beschuldigung, Kundenverfolgung-Beschuldigung und Münzverfolgung-Beschuldigung auch dann noch gewährleistet, wenn der Angreifer $\mathcal{A}$ die Zahl $\alpha \in \mathbb{Z}_p^*$ wählt. Des Weiteren erfüllt das Geldsystem auch die in [Tro06] definierte Anonymität, bei der der Angreifer $\mathcal{A}$ in der Challenge-Phase zusätzlich alle privaten Schlüssel der ehrlichen Kunden erhält.*

7.4 Das faire teilbare elektronische Geldsystem FTG-II

In dem vorherigen Abschnitt 7.3 wurde das teilbare Geldsystem TG-I aus Abschnitt 6.5 zu dem fairen teilbaren Geldsystem FTG-I erweitert. Analog dazu wird in diesem Abschnitt das teilbare Geldsystem TG-II aus Abschnitt 6.6 zu einem fairen teilbaren Geldsystem FTG-II erweitert.

Die Grundidee dieser Konstruktion ist, bei jeder Münze einen Teil eines XDH Tupels $\left(g, g^a, g^b, g^{ab}, h\right) =: (u_0, Q, u_0^t, T_C, h) \in \mathbb{G}_1^4 \times \mathbb{G}_2$ einzubauen. Die Elemente u_0 und h sind öffentliche Generatoren mit $\mathbb{G}_1 = \langle u_0 \rangle$ und $\mathbb{G}_2 = \langle h \rangle$.

Der Münz-Identifikationsschlüssel ist erneut die Zufallszahl $t \in_{\mathcal{R}} \mathbb{Z}_p$, die gemeinsam vom Kunden und der Bank generiert wird, aber nur dem Kunden bekannt ist. Der Kunde erhält nun im Abhebeprotokoll eine BBS+A-Signatur $\sigma = (\Sigma, e, s)$ auf die Nachricht $m = (t, \phi(\mathsf{x}))$ und die Bank erhält eine Verschlüsselung des Elements $h^t \in \mathbb{G}_2$. Zusätzlich erhält die Bank bei einer Münze das zufällige Element $Q \in_{\mathcal{R}} \mathbb{G}_1$ sowie den zum Paar (Q, u_0^t) gehörenden Münz-Identifikator $T_C = Q^t \in \mathbb{G}_1$.

Aus dem XDH-Problem und der semantischen Sicherheit des Verschlüsselungsverfahrens folgt, dass der Münz-Identifikator T_C nicht effizient von einem zufälligen Gruppenelement unterschieden werden kann. Somit kann die Bank den Münz-Identifikator T_C nicht der entsprechenden Verschlüsselung des Elements h^t zuordnen.

Möchte die Bank nun im Verdachtsfall alle zum Element h^t gehörenden Münzen verfolgen, erhält sie vom Deanonymisierer das entschlüsselte Element $E^\star = h^t$. Somit kann die Bank bei jeder Münze die Gleichung $\hat{e}(T_C, h) \stackrel{?}{=} \hat{e}(Q, E^\star)$ überprüfen, unabhängig davon, ob die Münze bereits eingelöst wurde oder nicht.

Wird das Element $E^\star$ auf einer schwarzen Liste veröffentlicht um diese Münzen zu sperren, überprüft der Händler bereits während des Bezahlprotokolls obige Gleichung und akzeptiert die Münze bei Gleichheit nicht.

Die wesentliche Veränderung zum vorherigen fairen Geldsystem FTG-I besteht darin, dass der Münz-Identifikator nun in der Gruppe $\mathbb{G}_1$ statt in $\mathbb{G}_T$ berechnet wird und somit eine kürzere Bitdarstellung besitzt. So wird dieser nun durch $T_C = Q^t$ berechnet. Daraus folgt, dass während des Abhebeprotokolls eine Verschlüsselung von $h^{t'}$ generiert werden muss und das Element nicht mehr im Klartext gesendet werden darf. Die Idee der Berechnung des Münz-Identifikators $T_C = Q^t$ wurde auch bereits in [Au09, Abschnitt 5.6] vorgestellt, wobei Q dem dortigen *One-Time-Tag* entspricht. Allerdings kann durch die Verwendung eines Pairings eine Münze durch Überprüfung der Gleichung $\hat{e}(T_C, h) \stackrel{?}{=} \hat{e}(Q, h^t)$ verfolgt werden. Dadurch muss nicht wie in [Au09] die Zahl t bzw. t' verifizierbar verschlüsselt werden, sondern das Element h^t bzw. $h^{t'}$. Aus diesem Grund kann ein effizienteres Verschlüsselungsverfahren zusammen mit Zero-Knowledge-Beweisen verwendet werden. Des Weiteren besitzt die in [Au09] angewandte Methode nicht die Sicherheitseigenschaft Münzverfolgung-Beschuldigung gemäß Definition 4.4.8. Denn nachdem die Bank die Zahl t kennt, kann sie beliebige Münzen bzgl. eines Münz-Identifikators $T_C^\star = (Q^\star)^t$ erstellen (siehe Unterabschnitt 7.6.1).

Da das Geldsystem wegen der Berechnung des Münz-Identifikators nun nur noch unter der XDH-Annahme für $\mathsf{Setup}_{\mathsf{Bil}}$ anonym ist, kann dieses nicht im Typ-1-Pairing eingesetzt werden. Denn bei zwei Münzen bzgl. der Paare (Q, T_C) und (Q', T_C') könnte dann die Gleichung $\hat{e}(T_C, Q') \stackrel{?}{=} \hat{e}(Q, T_C')$ überprüft werden. Wenn beide Münzen von derselben Geldbörse stammen, gilt $\hat{e}(T_C, Q') = \hat{e}(Q, Q')^t = \hat{e}(Q, T_C')$.

Da für $\mathsf{Setup}_{\mathsf{Bil}}$ die XDH-Annahme gelten muss, können auch die Seriennummern und Sicherheitstags (wie in den Geldsystemen TG-II und FTG-0) in der Gruppe $\mathbb{G}_1$ berechnet werden. Analog könnten diese aber auch weiterhin in der Gruppe $\mathbb{G}_p$ berechnet werden (falls beispielsweise gewünscht wird, dass die Bank nach einem Double-Spending oder bei einer Kundenverfolgung nur Informationen über den Kunden, aber nicht noch über das entsprechende Abhebeprotokoll erhält). Allerdings behandeln wir lediglich den ersten Fall und nur für das Typ-3-Pairing, da dies effizienter ist. Wenn das System im Typ-2-Pairing eingesetzt werden soll, muss während des Abhebeprotokolls lediglich für die Verschlüsselung des Elements $h^{t'}$ das lineare (oder ein anderes sicheres Verfahren) anstelle des ElGamal-Verschlüsselungsverfahrens verwendet werden. Da die Seriennummern in $\mathbb{G}_1$ berechnet werden, gilt für jede Seriennummer S im Random-Oracle-Modell $S \in_{\mathcal{R}} \mathbb{G}_1$, da das Random-Oracle die Zahl $\kappa \in_{\mathcal{R}} \mathbb{Z}_p^*$ mit $S = g^\kappa$ zufällig wählt. Somit kann ähnlich zum fairen teilbaren Geldsystem FTG-I das Element $Q := S$ gesetzt werden.

Im Folgenden werden hauptsächlich die Veränderungen im Vergleich zu TG-II detailliert dargestellt. Der Algorithmus BGen und das Protokoll Account werden wie beim teilbaren Geldsystem TG-II durchgeführt. Da sich die Abhebe- und Bezahlprotokolle Withdraw und Spend bzw. Spend-K verändern, werden die Algorithmen Identify, VerifyGuilt und VerifyWithdraw dann analog zum teilbaren Geldsystem TG-II durchgeführt und aus diesem Grund nicht beschrieben.

Setup wählt zusätzlich einen weiteren, zufälligen Generator $g_1 \in_{\mathcal{R}} \mathbb{G}_1$. Der Generator

$g_0 \in \mathbb{G}_1$ wird nicht zufällig gewählt. Die Ausgabe sind die Systemparameter

$$\mathsf{sp} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_1, u_0, \ldots, u_K, h, h_1, \ldots, h_K, \mathsf{H}) \leftarrow \mathsf{Setup}\left(1^\lambda, K\right).$$

DGen wählt bei Eingabe der Systemparameter sp zufällig zwei Zahlen $y, z \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Elemente $g_0 = g^y \in \mathbb{G}_1$ sowie $Z = h^z \in \mathbb{G}_2$. Die Zahl y wird erneut für eine Kundenverfolgung und die Zahl z für eine Münzverfolgung verwendet. Die Ausgabe ist das Schlüsselpaar

$$(\mathsf{pk}_{\mathcal{D}}, \mathsf{sk}_{\mathcal{D}}) = ((g_0, Z), (y, z)) \leftarrow \mathsf{DGen}(\mathsf{sp}).$$

In der folgenden Tabelle 7.3 werden die benötigten Gruppen, deren öffentliche Generatoren, deren Verwendung und die zugrunde liegenden, kryptografischen Annahmen zusammengefasst.

Gruppen	Generatoren	Verwendung	Annahmen
$\mathbb{G}_1$	$g, g_0, g_1, u_0, \ldots, u_K$	Seriennummern, Sicherheitstags und Münz-Identifikatoren	
$\mathbb{G}_2$	$h, h_1, \ldots, h_K, X, Z$	ElGamal-Verschlüsselung	
$(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$		Nguyen-Akkumulator und BBS+A-Signatur	SXDH, q-SDH und q-polyDH

Tabelle 7.3: Öffentliche Parameter von FTG-II

Withdraw: Damit ein Kunde $\mathcal{K}$ eine Geldbörse mit dem Wert $K = 2^L$ bei der Bank $\mathcal{B}$ abheben kann, führen $\mathcal{K}$ und $\mathcal{B}$ das Abhebeprotokoll Withdraw durch, welches in Abbildung 7.9 dargestellt ist. Zuvor authentifizieren sich jedoch beide Parteien gegenseitig.

Schritt 1: Der Kunde $\mathcal{K}$ wählt zufällig zwei Zahlen $s', t' \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das perfekt verbergende Commitment $C = g_0^{s'} g_1^{t'} u_0^{\phi(\alpha)}$, wobei $\phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ wie bisher das Polynom $\phi(\mathsf{x}) = \prod_{j=0}^{K-1}(\mathsf{x} + \kappa_{L+1,j})$ und $\kappa_{L+1,0}, \ldots, \kappa_{L+1,K-1}$ die K Serienschlüssel sind. Zusätzlich generiert der Kunde eine ElGamal-Verschlüsselung des Elements $h^{t'}$. Dazu wählt $\mathcal{K}$ zufällig eine Zahl $m \in_{\mathcal{R}} \mathbb{Z}_p$, berechnet die Verschlüsselung $(E_1, E_2) = \left(h^{t'} Z^m, h^m\right)$ und sendet diese ebenfalls an $\mathcal{B}$. Dann erstellt der Kunde $\mathcal{K}$ die folgende Signature-of-Knowledge:

$$SoK_{\mathsf{W}}\Big[(a_0, m, s', t', u, \phi(l)) : I = g^u \ \wedge \ E_1 = h^{t'} Z^m \ \wedge \ E_2 = h^m \ \wedge \\ \hat{e}(C, h)\, \hat{e}\left(A_0, h_1 h^{-l}\right)^{-1} = \hat{e}\left(g_0^{s'} g_1^{t'} u_0^{\phi(l)+a_0 l} u_1^{-a_0}, h\right)\Big](\mathsf{ts}||K).$$

Schritt 2: Die Bank $\mathcal{B}$ berechnet den Hashwert $l = \mathsf{H}(C)$, verifiziert SoK_{W}, wählt zufällig drei Zahlen $e, s'', t'' \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Element $\Sigma = \left(gg_0^{s''} g_1^{t''} C\right)^{\frac{1}{x+e}}$. Dann sendet $\mathcal{B}$ das Tupel (Σ, e, s'', t'') an $\mathcal{K}$ und bucht dem Kunden $\mathcal{K}$ den Betrag K vom Konto mit der Kontonummer I ab. Anschließend speichert $\mathcal{B}$ die Protokollansicht $\mathsf{view} = (\mathbb{T} = (I, C, E_1, E_2, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s'', t''))$ in ihrer Datenbank $\mathbb{D}_W$ ab.

Schritt 3: Der Kunde $\mathcal{K}$ berechnet $s = s' + s'' \mod p$ sowie $t = t' + t'' \mod p$, verifiziert die Signatur durch Überprüfung der Gleichung $\hat{e}(\Sigma, Xh^e) \stackrel{?}{=} \hat{e}\left(gg_0^{s''} g_1^{t''} C, h\right)$ und speichert $\mathbb{W} = (\Sigma, e, s, t, \mathsf{info})$ ab.

$\boxed{\mathcal{K}unde}$ $(\mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{K}})$	$(\mathsf{sp}, \mathsf{pk}_{\mathcal{D}}, K, \mathsf{ts})$	$\boxed{\mathcal{B}ank}$ $(\mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{K}})$
$\kappa_{0,0} \in_{\mathcal{R}} \mathbb{Z}_p^*, s', t', a_0, m \in_{\mathcal{R}} \mathbb{Z}_p$		
Generiere $\mathbb{K}_0, \mathbb{K}_1$		
$\phi(\mathsf{x}) = \prod_{j=0}^{K-1} (\mathsf{x} + \kappa_{L+1,j})$		
$C = g_0^{s'} g_1^{t'} u_0^{\phi(\alpha)}$		
$E_1 = h^{t'} Z^m$		
$E_2 = h^m$		
$l = \mathsf{H}(C)$		
$A_0 = W_l u_0^{a_0}$		
	$\xrightarrow{C, E_1, E_2, A_0}$	
$SoK_{\mathsf{W}}\Big[(a_0, m, s', t', u, \phi(l)) : I = g^u \ \wedge\ E_1 = h^{t'} Z^m \ \wedge\ E_2 = h^m \ \wedge$ $\hat{e}(C, h)\, \hat{e}\left(A_0, h_1 h^{-l}\right)^{-1} = \hat{e}\left(g_0^{s'} g_1^{t'} u_0^{\phi(l)+a_0 l} u_1^{-a_0}, h\right)\Big] (\mathsf{ts} \| K)$		
		$l = \mathsf{H}(C)$
		Verifiziere SoK_{W}
		$e, s'', t'' \in_{\mathcal{R}} \mathbb{Z}_p$
		$\Sigma = \left(gg_0^{s''} g_1^{t''} C\right)^{\frac{1}{x+e}}$
	$\xleftarrow{\Sigma, e, s'', t''}$	
$s = s' + s'' \mod p$		
$t = t' + t'' \mod p$		
$\hat{e}(\Sigma, Xh^e) \stackrel{?}{=} \hat{e}\left(gg_0^{s''} g_1^{t''} C, h\right)$		
$\mathbb{W} = (\Sigma, e, s, t, \mathsf{info})$ oder $\perp$		view oder $\perp$

Abbildung 7.9: Abhebeprotokoll Withdraw von FTG-II

Spend: Damit ein Kunde $\mathcal{K}$, der im Besitz der Geldbörse $\mathbb{W} = (\Sigma, e, s, t, \mathsf{info})$ ist, bei

einem Händler $\mathcal{H}$ mit einer elektronischen Münze coin mit dem Wert $k = 2^\ell < K$ bezahlen kann, führen $\mathcal{K}$ und $\mathcal{H}$ das Bezahlprotokoll Spend durch, welches in Abbildung 7.10 dargestellt ist. Zuvor authentifiziert sich der Händler $\mathcal{H}$ gegenüber dem Kunden $\mathcal{K}$ und beide Parteien gleichen den Zeitpunkt ts ab.

$\boxed{\mathcal{K}unde}$ $(\mathsf{pk}_\mathcal{H}, \mathsf{sk}_\mathcal{K}, \mathbb{W})$	$(\mathsf{sp}, \mathsf{pk}_\mathcal{B}, \mathsf{pk}_\mathcal{D}, k, \mathsf{ts})$	$\boxed{\mathcal{H}ändler}$ $(\mathsf{sk}_\mathcal{H})$
$R = \mathsf{H}(\mathsf{pk}_\mathcal{H}\|\|\mathsf{ts}\|\|k)$		$R = \mathsf{H}(\mathsf{pk}_\mathcal{H}\|\|\mathsf{ts}\|\|k)$
$a_1 \in_\mathcal{R} \mathbb{Z}_p$		
$\kappa := \kappa_{L-\ell,j_0}$		
$S = g^\kappa$		
$T = \Sigma g_0^{R\kappa}$		
$T_C = S^t$		
$l = \mathsf{H}(S\|\|T\|\|T_C)$		
$A_1 = W_{I,l} u_0^{a_1}$		
	$\xrightarrow{S, T, T_C, A_1}$	
$SoK\Big[(a_1, \beta, e, s, t, \kappa, \phi_I(l)) : S = g^\kappa \ \wedge\ 1 = S^e g^{-\beta} \ \wedge\ T_C = S^t \ \wedge$ $\frac{\hat{e}(T, X)}{\hat{e}(g, h)\,\hat{e}(A_1, h_{l,I})} = \hat{e}\left(g_0^{R\kappa}, X\right) \hat{e}\left(g_0^{s+R\beta} g_1^t u_{l,I}^{-a_1} u_I^{\phi_I(l)} T_1^{-e}, h\right)\Big](R)$		
		Berechne $\kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}$
		$l = \mathsf{H}(S\|\|T\|\|T_C)$
		Verifiziere SoK
$\mathbb{W}'$ oder $\perp$		coin oder $\perp$

Abbildung 7.10: Bezahlprotokoll Spend von FTG-II

Schritt 1: Der Händler $\mathcal{H}$ und der Kunde $\mathcal{K}$ berechnen beide den Hashwert $R = \mathsf{H}(\mathsf{pk}_\mathcal{H}\|\|\mathsf{ts}\|\|k)$.

Schritt 2: Der Kunde $\mathcal{K}$ geht wie im teilbaren Geldsystem TG-II vor und berechnet zusätzlich den Münz-Identifikator $T_C = S^t$. Folglich berechnet $\mathcal{K}$ den Hashwert $l = \mathsf{H}(S\|\|T\|\|T_C)$, sendet die Elemente S, T, T_C, A_1 an $\mathcal{H}$ und erstellt die folgende Signature-of-Knowledge SoK, wobei $\beta = e\kappa, u_I = u_0^{\prod_{j\in I}(\alpha+\kappa_{L+1,j})}, u_{l,I} = u_0^{(\alpha-l)\prod_{j\in I}(\alpha+\kappa_{L+1,j})}$ und $h_{l,I} = h^{(\alpha-l)\prod_{j\in I}(\alpha+\kappa_{L+1,j})}$ ist:

$$SoK\Bigg[(a_1, \beta, e, s, t, \kappa, \phi_I(l)) : S = g^\kappa \ \wedge\ 1 = S^e g^{-\beta} \ \wedge\ T_C = S^t \ \wedge \\ \frac{\hat{e}(T, X)}{\hat{e}(g, h)\,\hat{e}(A_1, h_{l,I})} = \hat{e}\left(g_0^{R\kappa}, X\right) \hat{e}\left(g_0^{s+R\beta} g_1^t u_{l,I}^{-a_1} u_I^{\phi_I(l)} T_1^{-e}, h\right)\Bigg](R)\,.$$

Anschließend aktualisiert $\mathcal{K}$ seine Geldbörse zu $\mathbb{W}' = (\Sigma, e, s, t, \mathsf{info}')$.

Schritt 3: Der Händler $\mathcal{H}$ berechnet aus der Seriennummer S alle k Seriennschlüssel sowie den Hashwert $l = \mathsf{H}(S||T||T_C)$, verifiziert SoK und speichert die Münze $\mathsf{coin} = (k, S, T, T_C, R, \Phi = (A_1, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$ ab.

Spend-K: Damit ein Kunde $\mathcal{K}$, der im Besitz der Geldbörse $\mathbb{W} = (\Sigma, e, s, t, \mathsf{info})$ ist, bei einem Händler $\mathcal{H}$ mit einer elektronischen Münze coin mit dem Wert $K = 2^L$ bezahlen kann, führen $\mathcal{K}$ und $\mathcal{H}$ das Bezahlprotokoll Spend-K durch, welches in Abbildung 7.11 dargestellt ist. Zuvor authentifiziert sich der Händler $\mathcal{H}$ gegenüber dem Kunden $\mathcal{K}$ und beide Parteien gleichen den Zeitpunkt ts ab.

$\boxed{\mathcal{K}unde}$ $(\mathsf{pk}_{\mathcal{H}}, \mathsf{sk}_{\mathcal{K}}, \mathbb{W})$	$(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{D}}, K, \mathsf{ts})$	$\boxed{\mathcal{H}ändler}$ $(\mathsf{sk}_{\mathcal{H}})$
$R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}}\|\|\mathsf{ts}\|\|K)$		$R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}}\|\|\mathsf{ts}\|\|K)$
$\kappa := \kappa_{0,0}$ $S = g^{\kappa}$ $T = \Sigma g_0^{R\kappa}$ $T_C = S^t$		
	$\xrightarrow{S, T, T_C}$	
$SoK\Big[(\beta, e, s, t, \kappa) : S = g^{\kappa} \wedge 1 = S^e g^{-\beta} \wedge$ $T_C = S^t \wedge \frac{\hat{e}(T, X)}{\hat{e}(gV, h)} = \hat{e}\left(g_0^{R\kappa}, X\right) \hat{e}\left(g_0^{s+R\beta} g_1^t T^{-e}, h\right)\Big](R)$		
		Berechne $\kappa_{L+1,0}, \ldots, \kappa_{L+1,K-1}$ Verifiziere SoK
$\mathbb{W}'$ oder $\perp$		coin oder $\perp$

Abbildung 7.11: Bezahlprotokoll Spend-K von FTG-II

Schritt 1: Der Händler $\mathcal{H}$ und der Kunde $\mathcal{K}$ berechnen beide den Hashwert $R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}}||\mathsf{ts}||K)$.

Schritt 2: Der Kunde $\mathcal{K}$ geht wie im teilbaren Geldsystem TG-II vor und berechnet zusätzlich den Münz-Identifikator $T_C = S^t$. Folglich sendet $\mathcal{K}$ die Elemente S, T, T_C an $\mathcal{H}$ und erstellt die folgende Signature-of-Knowledge SoK, wobei $\beta = e\kappa$ und $V = u_0^{\prod_{j=0}^{K-1}(\alpha+\kappa_{L+1,j})}$ ist:

$$SoK\Big[(\beta, e, s, t, \kappa) : S = g^{\kappa} \wedge 1 = S^e g^{-\beta} \wedge T_C = S^t \wedge \frac{\hat{e}(T, X)}{\hat{e}(gV, h)} = \hat{e}\left(g_0^{R\kappa}, X\right) \hat{e}\left(g_0^{s+R\beta} g_1^t T^{-e}, h\right)\Big](R)\,.$$

Anschließend aktualisiert $\mathcal{K}$ seine Geldbörse zu $\mathbb{W}' = (\Sigma, e, s, t, \mathsf{info}')$.

Schritt 3: Der Händler $\mathcal{H}$ berechnet aus der Seriennummer S alle K Serienschlüssel sowie den Akkumulator V, verifiziert SoK und speichert die Münze $\mathsf{coin} = (K, S, T, T_C, R, \Phi = (SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$ ab.

Deposit: Dieses Protokoll wird analog zum teilbaren Geldsystem TG-II ausgeführt. Die Bank $\mathcal{B}$ kann zusätzlich sofort das Element $\hat{e}(T_C, h)$ berechnen und abspeichern, damit dieses nicht bei einer Münzverfolgung immer wieder neu berechnet werden muss.

UTrace: Möchte die Bank $\mathcal{B}$ gemeinsam mit dem Deanonymisierer $\mathcal{D}$ den Eigentümer einer eingelösten Münze bestimmen, führen $\mathcal{B}$ und $\mathcal{D}$ das Kundenverfolgungsprotokoll UTrace durch, welches in Abbildung 7.12 dargestellt ist. Zuvor authentifizieren sich jedoch beide Parteien gegenseitig.

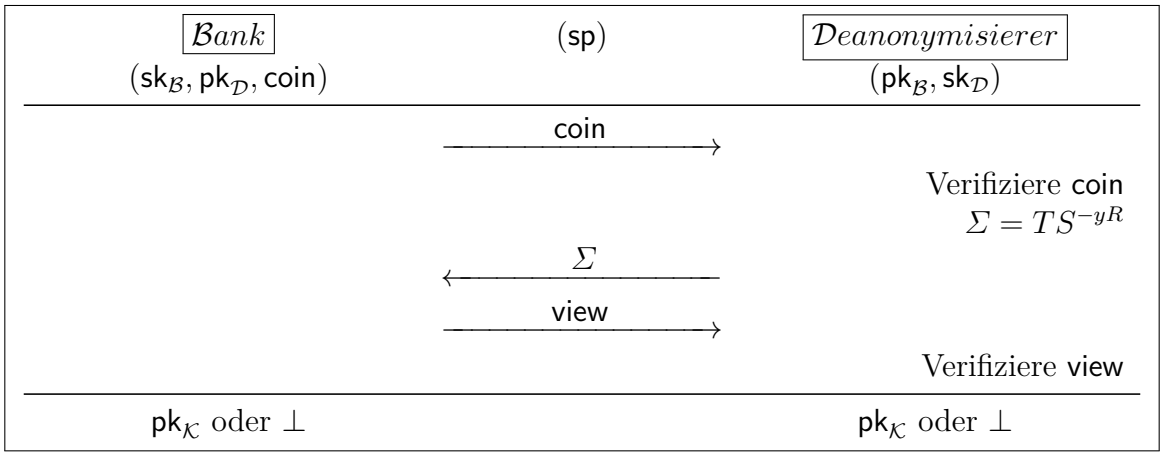

Abbildung 7.12: Kundenverfolgungsprotokoll UTrace von FTG-II

Schritt 1: Die Bank $\mathcal{B}$ sendet eine Münze $\mathsf{coin} = (k, S, T, T_C, R, \Phi)$ an $\mathcal{D}$.

Schritt 2: Der Deanonymisierer $\mathcal{D}$ verifiziert die Münze coin und berechnet das Element

$$TS^{-yR} = \Sigma g_0^{R\kappa} g^{-yR\kappa} = \Sigma g_0^{R\kappa} g_0^{-R\kappa} = \Sigma.$$

Anschließend sendet $\mathcal{D}$ das Element Σ an $\mathcal{B}$.

Schritt 3: Die Bank durchsucht ihre Datenbank $\mathbb{D}_W$ und kann den Kunden $\mathcal{K}$ anhand von Σ eindeutig identifizieren. Dann sendet $\mathcal{B}$ die Protokollansicht des entsprechenden Abhebeprotokolls $\mathsf{view} = (\mathbb{T} = (I, C, E_1, E_2, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s'', t''))$ an $\mathcal{D}$.

Schritt 4: Der Deanonymisierer $\mathcal{D}$ verifiziert SoK_{W} sowie die Signatur durch Überprüfung der Gleichung $\hat{e}\left(\Sigma, Xh^{e}\right) \stackrel{?}{=} \hat{e}\left(gg_0^{s''}g_1^{t''}C, h\right)$, was der Durchführung des Algorithmus VerifyWithdraw entspricht, und gibt den öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{K}} = I$ des Kunden aus.

CTrace: Möchte die Bank $\mathcal{B}$ gemeinsam mit dem Deanonymisierer $\mathcal{D}$ alle Münzen einer zu einem bestimmten Abhebeprotokoll Withdraw gehörenden Geldbörse verfolgen, führen $\mathcal{B}$ und $\mathcal{D}$ das Münzverfolgungsprotokoll CTrace durch, welches in Abbildung 7.13 dargestellt ist. Zuvor authentifizieren sich jedoch beide Parteien gegenseitig.

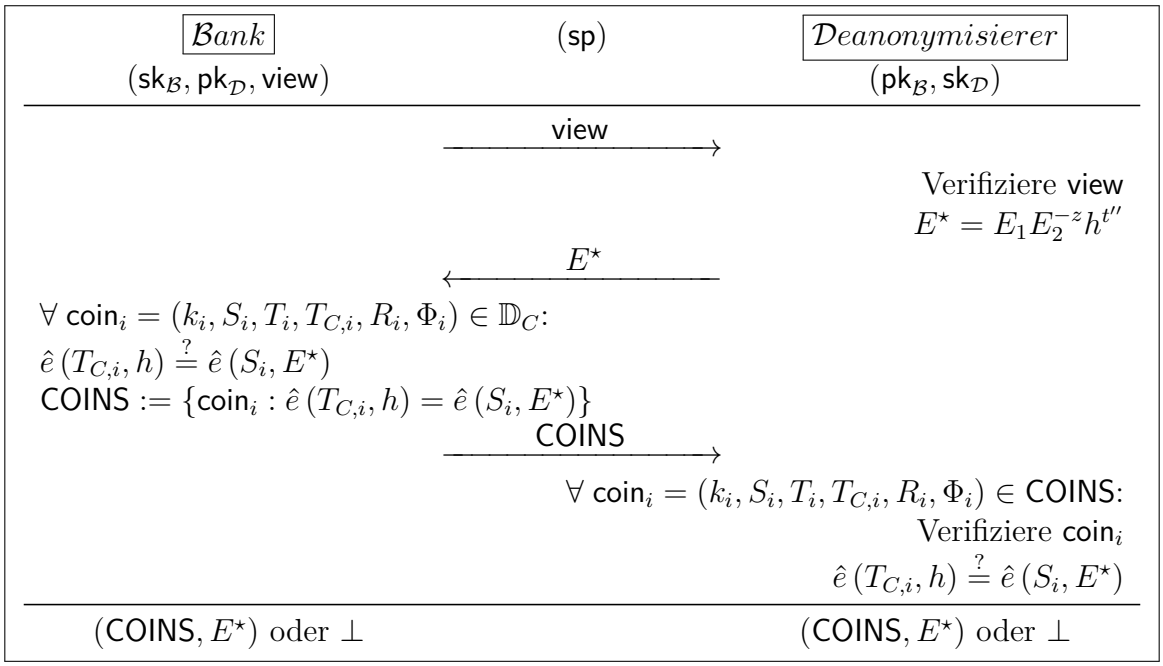

Abbildung 7.13: Münzverfolgungsprotokoll CTrace von FTG-II

Schritt 1: Die Bank $\mathcal{B}$ sendet eine Protokollansicht $\mathsf{view} = (\mathbb{T} = (I, C, E_1, E_2, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s'', t''))$ eines bestimmten Abhebeprotokolls an $\mathcal{D}$.

Schritt 2: Der Deanonymisierer $\mathcal{D}$ verifiziert SoK_{W} und die Signatur durch Überprüfung der Gleichung $\hat{e}\left(\Sigma, Xh^{e}\right) \stackrel{?}{=} \hat{e}\left(gg_0^{s''}g_1^{t''}C, h\right)$, was der Durchführung des Algorithmus VerifyWithdraw entspricht. Dann berechnet $\mathcal{D}$ das Element $E^{\star} = E_1 E_2^{-z} h^{t''} = h^{t'+t''} = h^{t}$ für $t = t' + t'' \mod p$ und sendet $E^{\star}$ an $\mathcal{B}$.

Schritt 3: Die Bank $\mathcal{B}$ kann dann bei jeder Münze $\mathsf{coin}_i = (k_i, S_i, T_i, T_{C,i}, R_i, \Phi_i)$ die Gleichung

$$\hat{e}\left(T_{C,i}, h\right) \stackrel{?}{=} \hat{e}\left(S_i, E^{\star}\right)$$

überprüfen, egal, ob diese bereits eingelöst wurde oder erst zukünftig verwendet wird. (Beachte, dass die Bank das Element $\hat{e}(T_{C,i}, h)$ bereits zusätzlich mit der Münze coin_i während des Deposit-Protokolls berechnet und gespeichert haben kann.) Die Bank kann das Element $E^\star$ auch auf einer schwarzen Liste veröffentlichen und so alle Münzen dieser Geldbörse sperren lassen.

Damit der Deanonymisierer eine mögliche Bestrafung überprüfen kann, sendet $\mathcal{B}$ alle Münzen coin_i mit $\hat{e}(T_{C,i}, h) = \hat{e}(S_i, E^\star)$ an $\mathcal{D}$. Es sei $\mathsf{COINS} := \{\mathsf{coin}_i : \hat{e}(T_{C,i}, h) = \hat{e}(S_i, E^\star)\}$ die Menge dieser Münzen.

Schritt 4: Der Deanonymisierer $\mathcal{D}$ verifiziert bei jeder Münze $\mathsf{coin}_i \in \mathsf{COINS}$ die Münze coin_i und die Gleichung $\hat{e}(T_{C,i}, h) \stackrel{?}{=} \hat{e}(S_i, E^\star)$.

7.4.1 Sicherheitsanalyse von FTG-II

Satz 7.4.1. *Das faire teilbare elektronische Geldsystem FTG-II ist im Random-Oracle-Modell ein sicheres faires elektronisches Geldsystem, falls die SXDH-Annahme, die q-SDH-Annahme und die q-polyDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gelten.*

Beweis. Wir beweisen Unfälschbarkeit, Anonymität, Double-Spending-Beschuldigung, Abhebe-Beschuldigung, korrekte Kundenverfolgung, Kundenverfolgung-Beschuldigung, korrekte Münzverfolgung und Münzverfolgung-Beschuldigung.

Unfälschbarkeit: Die Unfälschbarkeit des Geldsystems FTG-II folgt direkt aus der Unfälschbarkeit des teilbaren Geldsystems TG-II. Denn die Abhebe- bzw. Bezahlprotokolle bestehen im Wesentlichen aus denen des teilbaren Geldsystems TG-II und den zusätzlichen Elementen (E_1, E_2) bzw. T_C. Somit werden nun bei den Orakel-Anfragen entsprechend Nachrichten-Signatur-Paare $((t, \phi(\mathsf{x})), (\Sigma, e, s))$ erstellt bzw. extrahiert. Bei der Simulation der Initialisierung wählt $\mathcal{B}$ den privaten Schlüssel $z \in_{\mathcal{R}} \mathbb{Z}_p^*$ des Deanonymisierers. Das Element g_0 ist nun nicht Teil der Systemparameter sp sondern ein Teil des öffentlichen Schlüssels $\mathsf{pk}_{\mathcal{D}}$. Ansonsten geht $\mathcal{B}$ analog zu TG-II vor. Da der Angreifer $\mathcal{A}$ nun allerdings die beiden Orakel $\mathcal{O}^{\mathsf{UT}}_{\mathcal{B},\mathcal{D}}$ und $\mathcal{O}^{\mathsf{CT}}_{\mathcal{B},\mathcal{D}}$ befragen darf, wird im Folgenden detailliert beschrieben, wie $\mathcal{B}$ diese (im Fall (1.1)) ohne Kenntnis des geheimen Schlüssels $y \in \mathbb{Z}_p^*$ mit $g_0 = g^y$ simuliert.

Simulation von $\mathcal{O}^{\mathsf{UT}}_{\mathcal{B},\mathcal{D}}$: Der Angreifer $\mathcal{B}$ erhält eine Münze $\mathsf{coin} = (k, S, T, T_C, R, \Phi = (A_1, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$.

- Falls der Angreifer $\mathcal{B}$ die Münze coin zum Bezahlen verwendet hat, gibt es die Einträge $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i) \in \mathbb{W}_H$ mit $\mathbb{W}_i = (\Sigma, \cdots)$. Somit kennt $\mathcal{B}$ das Element Σ und gibt dieses aus, wodurch die Anfrage perfekt simuliert ist.
- Falls der Angreifer $\mathcal{B}$ die Münze coin nicht zum Bezahlen verwendet hat, wurde SoK vom Angreifer $\mathcal{A}$ erstellt. Daher kann $\mathcal{B}$ die Zahl $\kappa \in \mathbb{Z}_p$ mit

$S = g^{\kappa}$ sowie $T = \Sigma g_0^{R\kappa}$ aus SoK extrahieren und das Element $\Sigma = T g_0^{-R\kappa}$ korrekt berechnen. Dann gibt $\mathcal{B}$ das Element Σ aus, wodurch die Anfrage perfekt simuliert ist. Falls $\mathcal{B}$ die Zahl κ nicht extrahieren kann, bricht $\mathcal{B}$ ab. Dies ist aber nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Simulation von $\mathcal{O}^{\mathsf{CT}}_{\mathcal{B},\mathcal{D}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den geheimen Schlüssel $z \in \mathbb{Z}_p^*$ kennt.

Vorbemerkungen zur Anonymität: Wir benötigen zunächst das folgende Lemma 7.4.1 sowie das Korollar 7.4.1, das direkt aus Lemma 7.4.1 folgt.

Lemma 7.4.1. *Angenommen, es gilt die XDH-Annahme für* $\mathsf{Setup_{Bil}}$ *und* $(\mathsf{GenEnc}, \mathsf{Enc}, \mathsf{Dec})$ *ist ein semantisch sicheres Verschlüsselungsverfahren, wobei* GenEnc *die Ausgabe von* $\mathsf{Setup_{Bil}}$ *zur Generierung der Schlüssel verwendet und* $\mathbb{M}_{\mathsf{Enc}} = \mathbb{G}_2 = \langle h \rangle$ *ist. Seien* $q_0, q_1 \in \mathbb{N}$ *polynomiell beschränkt. Dann gibt es für jeden polynomiellen Angreifer* $\mathcal{A}$ *eine vernachlässigbare Funktion* ν*, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\Big|\Pr\Big[\mathsf{sp_{Bil}} \leftarrow \mathsf{Setup_{Bil}}\left(1^{\lambda}\right); (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenEnc}\left(1^{\lambda}\right); g_0, g_1, g_{0,1}, \ldots, g_{0,q_0}, g_{1,1}, \ldots, g_{1,q_1} \in_{\mathcal{R}} \mathbb{G}_1;$$
$$a_0, a_1 \in_{\mathcal{R}} \mathbb{Z}_p\colon\ 1 \leftarrow \mathcal{A}(\mathsf{sp_{Bil}}, \mathsf{pk}, g_0, g_1, \{g_{0,i}\}_{i=1}^{q_0}, \{g_{1,j}\}_{j=1}^{q_1},$$
$$\{g_{0,i}^{a_0}\}_{i=1}^{q_0}, \{g_{1,j}^{a_1}\}_{j=1}^{q_1}, \mathsf{Enc}\left(\mathsf{pk}, h^{a_0}\right), \mathsf{Enc}\left(\mathsf{pk}, h^{a_1}\right), g_0^{a_0}, g_1^{a_1})\Big]$$
$$- \Pr\Big[\mathsf{sp_{Bil}} \leftarrow \mathsf{Setup_{Bil}}\left(1^{\lambda}\right); (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenEnc}\left(1^{\lambda}\right); g_0, g_1, g_{0,1}, \ldots, g_{0,q_0}, g_{1,1}, \ldots, g_{1,q_1} \in_{\mathcal{R}} \mathbb{G}_1;$$
$$A, B \in_{\mathcal{R}} \mathbb{G}_1; C, D \in_{\mathcal{R}} \mathbb{G}_2; a_0, a_1 \in_{\mathcal{R}} \mathbb{Z}_p\colon\ 1 \leftarrow \mathcal{A}(\mathsf{sp_{Bil}}, \mathsf{pk}, g_0, g_1, \{g_{0,i}\}_{i=1}^{q_0}, \{g_{1,j}\}_{j=1}^{q_1},$$
$$\{g_{0,i}^{a_0}\}_{i=1}^{q_0}, \{g_{1,j}^{a_1}\}_{j=1}^{q_1}, \mathsf{Enc}\left(\mathsf{pk}, C\right), \mathsf{Enc}\left(\mathsf{pk}, D\right), A, B)\Big]\Big| \leq \nu(\lambda).$$

Beweis. Angenommen, es existiert ein polynomieller Angreifer $\mathcal{A}$, der mit Wahrscheinlichkeit $1/2 + \epsilon(\lambda)$ für eine nicht vernachlässigbare Funktion ϵ bei Eingabe des Tupels $\left(\mathsf{sp_{Bil}}, \mathsf{pk}, g_0, g_1, \{g_{0,i}\}_{i=1}^{q_0}, \{g_{1,j}\}_{j=1}^{q_1}, \{g_{0,i}^{a_0}\}_{i=1}^{q_0}, \{g_{1,j}^{a_1}\}_{j=1}^{q_1}, \mathsf{Enc}\left(\mathsf{pk}, C\right), \mathsf{Enc}\left(\mathsf{pk}, D\right), A, B\right)$ entscheiden kann, ob $A, B \in_{\mathcal{R}} \mathbb{G}_1 \wedge C, D \in_{\mathcal{R}} \mathbb{G}_2$ ist oder, ob $A = g_0^{a_0} \wedge B = g_1^{a_1} \wedge C = h^{a_0} \wedge D = h^{a_1}$ gilt.

Wir konstruieren einen polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das XDH-Problem zu lösen, unter der Annahme, dass $(\mathsf{GenEnc}, \mathsf{Enc}, \mathsf{Dec})$ ein semantisch sicheres Verschlüsselungsverfahren ist.

Schlüsselgenerierung: Der Angreifer $\mathcal{B}$ erhält eine zufällige XDH-Probleminstanz $(p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, h, g^a, g^b, \Gamma)$, wobei entweder $\Gamma = g^{ab}$ oder $\Gamma \in_{\mathcal{R}} \mathbb{G}_1$ ist. Zunächst generiert $\mathcal{B}$ einen öffentlichen Schlüssel pk mit $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenEnc}\left(1^{\lambda}\right)$. Dann wählt $\mathcal{B}$ zufällig $q_0 + q_1 + 3$ Zahlen $\gamma, \gamma', \delta, \beta_{0,1}, \ldots, \beta_{0,q_0}, \beta_{1,1}, \ldots, \beta_{1,q_1} \in_{\mathcal{R}} \mathbb{Z}_p^*$ und definiert die Generatoren $g_0 := g^a, g_1 := (g^a)^{\gamma} g^{\gamma'}, g_{0,i} := g^{\beta_{0,i}}$ und $g_{1,j} := g^{\beta_{1,j}}$ für $1 \leq i \leq q_0$ und $1 \leq j \leq q_1$. Weiter definiert $\mathcal{B}$ die Generatoren $g_{0,i}^{a_0} := \left(g^b\right)^{\beta_{0,i}}$ und $g_{1,j}^{a_1} := \left(g^b g^{\delta}\right)^{\beta_{1,j}}$ für $1 \leq i \leq q_0$ und $1 \leq j \leq q_1$. Nach diesen Definitionen gilt $a_0 = b$

und $a_1 = b + \delta \mod p$. Anschließend definiert $\mathcal{B}$ die Elemente $A := \Gamma$ sowie $B := \Gamma^{\gamma} (g^a)^{\gamma\delta} \left(g^b\right)^{\gamma'} g^{\gamma'\delta}$, wählt zufällig zwei Elemente $C, D \in_{\mathcal{R}} \mathbb{G}_2$ und erstellt die beiden Verschlüsselungen $\mathsf{Enc}(\mathsf{pk}, C)$ sowie $\mathsf{Enc}(\mathsf{pk}, D)$.

Falls $\Gamma = g^{ab}$ ist, folgt $A = g_0^{a_0}$ sowie $B = g^{ab\gamma + a\gamma\delta + b\gamma' + \gamma'\delta} = g^{(a\gamma+\gamma')(b+\delta)} = g_1^{a_1}$ wie gewünscht. Die beiden Verschlüsselungen sind aufgrund der semantischen Sicherheit des Verschlüsselungsverfahrens rechnerisch ununterscheidbar simuliert.

Falls $\Gamma \in_{\mathcal{R}} \mathbb{G}_1$ ist, sei $\Gamma = g^{ab+d}$ mit $d \in_{\mathcal{R}} \mathbb{Z}_p^*$. Dann folgt $A = g_0^{a_0} g^d$ und $B = g_1^{a_1} g^{d\gamma}$. Daher gilt $A, B \in_{\mathcal{R}} \mathbb{G}_1$ und somit sind die beiden Elemente perfekt simuliert. Weiter sind in diesem Fall auch die beiden Verschlüsselungen perfekt simuliert.

Schließlich sendet $\mathcal{B}$ das rechnerisch ununterscheidbar simulierte Tupel $\Big(\mathsf{sp}_{\mathsf{Bil}}, \mathsf{pk}, g_0, g_1, \{g_{0,i}\}_{i=1}^{q_0}, \{g_{1,j}\}_{j=1}^{q_1}, \{g_{0,i}^{a_0}\}_{i=1}^{q_0}, \{g_{1,j}^{a_1}\}_{j=1}^{q_1}, \mathsf{Enc}(\mathsf{pk}, C), \mathsf{Enc}(\mathsf{pk}, D), A, B\Big)$ an $\mathcal{A}$.

Ausgabe: Der Angreifer $\mathcal{A}$ gibt ein Bit $b \in \{0, 1\}$ aus, welches $\mathcal{B}$ ebenfalls ausgibt.

Somit löst der Angreifer $\mathcal{B}$ das XDH-Problem mit Wahrscheinlichkeit $1/2 + \epsilon(\lambda) - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist. □

Korollar 7.4.1. *Angenommen, es gilt die XDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *und* $(\mathsf{GenEnc}, \mathsf{Enc}, \mathsf{Dec})$ *ist ein semantisch sicheres Verschlüsselungsverfahren, wobei* GenEnc *die Ausgabe von* $\mathsf{Setup}_{\mathsf{Bil}}$ *zur Generierung der Schlüssel verwendet und* $\mathbb{M}_{\mathsf{Enc}} = \mathbb{G}_2 = \langle h \rangle$ *ist. Seien* $q_0, q_1 \in \mathbb{N}$ *polynomiell beschränkt. Dann gibt es für jeden polynomiellen Angreifer* $\mathcal{A}$ *eine vernachlässigbare Funktion* ν*, so dass für alle hinreichend großen* $\lambda \in \mathbb{N}$ *gilt:*

$$\begin{aligned}
&\Big|\Pr\Big[\mathsf{sp}_{\mathsf{Bil}} \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right); (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenEnc}\left(1^\lambda\right); g_0, g_1, g_{0,1}, \ldots, g_{0,q_0}, g_{1,1}, \ldots, g_{1,q_1} \in_{\mathcal{R}} \mathbb{G}_1; \\
&\qquad a_0, a_1 \in_{\mathcal{R}} \mathbb{Z}_p\colon\ 1 \leftarrow \mathcal{A}(\mathsf{sp}_{\mathsf{Bil}}, \mathsf{pk}, g_0, g_1, \{g_{0,i}\}_{i=1}^{q_0}, \{g_{1,j}\}_{j=1}^{q_1}, \\
&\qquad \{g_{0,i}^{a_0}\}_{i=1}^{q_0}, \{g_{1,j}^{a_1}\}_{j=1}^{q_1}, \mathsf{Enc}(\mathsf{pk}, h^{a_0}), \mathsf{Enc}(\mathsf{pk}, h^{a_1}), g_0^{a_0}, g_1^{a_1})\Big] \\
&- \Pr\Big[\mathsf{sp}_{\mathsf{Bil}} \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right); (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenEnc}\left(1^\lambda\right); g_0, g_1, g_{0,1}, \ldots, g_{0,q_0}, g_{1,1}, \ldots, g_{1,q_1} \in_{\mathcal{R}} \mathbb{G}_1; \\
&\qquad A, B \in_{\mathcal{R}} \mathbb{G}_1; a_0, a_1 \in_{\mathcal{R}} \mathbb{Z}_p\colon\ 1 \leftarrow \mathcal{A}(\mathsf{sp}_{\mathsf{Bil}}, \mathsf{pk}, g_0, g_1, \{g_{0,i}\}_{i=1}^{q_0}, \{g_{1,j}\}_{j=1}^{q_1}, \\
&\qquad \{g_{0,i}^{a_0}\}_{i=1}^{q_0}, \{g_{1,j}^{a_1}\}_{j=1}^{q_1}, \mathsf{Enc}(\mathsf{pk}, h^{a_0}), \mathsf{Enc}(\mathsf{pk}, h^{a_1}), A, B)\Big]\Big| \leq \nu(\lambda).
\end{aligned}$$

Beweis. Zur Vereinfachung werden die Systemparameter $\mathsf{sp}_{\mathsf{Bil}}$, der öffentliche Schlüssel pk, die Generatoren $g_0, g_1, \{g_{0,i}\}_{i=1}^{q_0}, \{g_{1,j}\}_{j=1}^{q_1} \in_{\mathcal{R}} \mathbb{G}_1$ und die Elemente $\{g_{0,i}^{a_0}\}_{i=1}^{q_0}, \{g_{1,j}^{a_1}\}_{j=1}^{q_1} \in \mathbb{G}_1$ bei der Eingabe von $\mathcal{A}$ weggelassen. Seien wie oben $a_0, a_1 \in_{\mathcal{R}} \mathbb{Z}_p$, $A, B \in_{\mathcal{R}} \mathbb{G}_1$ und $C, D \in_{\mathcal{R}} \mathbb{G}_2$. Dann gilt

$$\begin{aligned}
&\Big|\Pr\Big[1 \leftarrow \mathcal{A}\Big(\mathsf{Enc}(h^{a_0}), \mathsf{Enc}(h^{a_1}), g_0^{a_0}, g_1^{a_1}\Big)\Big] - \Pr\Big[1 \leftarrow \mathcal{A}\Big(\mathsf{Enc}(h^{a_0}), \mathsf{Enc}(h^{a_1}), A, B\Big)\Big]\Big| \\
=&\Big|\Pr\Big[1 \leftarrow \mathcal{A}\Big(\mathsf{Enc}(h^{a_0}), \mathsf{Enc}(h^{a_1}), g_0^{a_0}, g_1^{a_1}\Big)\Big] - \Pr\Big[1 \leftarrow \mathcal{A}\Big(\mathsf{Enc}(C), \mathsf{Enc}(D), A, B\Big)\Big] \\
&+ \Pr\Big[1 \leftarrow \mathcal{A}\Big(\mathsf{Enc}(C), \mathsf{Enc}(D), A, B\Big)\Big] - \Pr\Big[1 \leftarrow \mathcal{A}\Big(\mathsf{Enc}(h^{a_0}), \mathsf{Enc}(h^{a_1}), A, B\Big)\Big]\Big|
\end{aligned}$$

$$
\begin{aligned}
&\leq \Big|\Pr\Big[1 \leftarrow \mathcal{A}\Big(\mathsf{Enc}\left(h^{a_0}\right), \mathsf{Enc}\left(h^{a_1}\right), g_0^{a_0}, g_1^{a_1}\Big)\Big] - \Pr\Big[1 \leftarrow \mathcal{A}\Big(\mathsf{Enc}\left(C\right), \mathsf{Enc}\left(D\right), A, B\Big)\Big]\Big| \\
&\quad + \Big|\Pr\Big[1 \leftarrow \mathcal{A}\Big(\mathsf{Enc}\left(C\right), \mathsf{Enc}\left(D\right), A, B\Big)\Big] - \Pr\Big[1 \leftarrow \mathcal{A}\Big(\mathsf{Enc}\left(h^{a_0}\right), \mathsf{Enc}\left(h^{a_1}\right), A, B\Big)\Big]\Big| \\
&\leq \nu_1(\lambda) + \nu_2(\lambda) =: \nu(\lambda),
\end{aligned}
$$

da nach Lemma 7.4.1

$$
\left|\Pr\left[1 \leftarrow \mathcal{A}(\mathsf{Enc}\left(h^{a_0}\right), \mathsf{Enc}\left(h^{a_1}\right), g_0^{a_0}, g_1^{a_1})\right] - \Pr\left[1 \leftarrow \mathcal{A}(\mathsf{Enc}\left(C\right), \mathsf{Enc}\left(D\right), A, B)\right]\right| \leq \nu_1(\lambda).
$$

und aufgrund der semantischen Sicherheit des Verschlüsselungsverfahrens

$$
\left|\Pr\left[1 \leftarrow \mathcal{A}(\mathsf{Enc}\left(C\right), \mathsf{Enc}\left(D\right), A, B)\right] - \Pr\left[1 \leftarrow \mathcal{A}(\mathsf{Enc}\left(h^{a_0}\right), \mathsf{Enc}\left(h^{a_1}\right), A, B)\right]\right| \leq \nu_2(\lambda)
$$

gilt, wobei ν_1 und ν_2 vernachlässigbare Funktionen sind. □

Anonymität: Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Anonymität des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, die semantische Sicherheit des ElGamal-Verschlüsselungsverfahrens in der Gruppe $\mathbb{G}_1$ mit $(p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, h) \leftarrow \mathsf{Setup}_{\mathsf{Bil}}\left(1^\lambda\right)$ zu brechen, was ein Widerspruch zur XDH-Annahme für $\mathsf{Setup}_{\mathsf{Bil}}$ ist.

Der Angreifer $\mathcal{B}$ erhält die Parameter $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, h)$ sowie das Paar $(g, g_0) \in \mathbb{G}_1^2$, wobei $(p, \mathbb{G}_1', g, g_0)$ dem öffentlichen Schlüssel des ElGamal-Verschlüsselungsverfahrens entspricht.

Simulation der Initialisierung: Der Angreifer $\mathcal{B}$ wählt zufällig zwei Zahlen $\alpha, z \in_{\mathcal{R}} \mathbb{Z}_p^*$ und einen Generator $g_1 \in_{\mathcal{R}} \mathbb{G}_1$. Danach berechnet $\mathcal{B}$ die Elemente $Z = h^z, u_i = u_0^{\alpha^i}$ sowie $h_i = h^{\alpha^i}$ für $1 \leq i \leq K$. Anschließend sendet $\mathcal{B}$ die perfekt simulierten Systemparameter $\mathsf{sp} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_1, u_0, \ldots, u_K, h, h_1, \ldots, h_K)$ sowie den perfekt simulierten öffentlichen Schlüssel des Deanonymisierers $\mathsf{pk}_{\mathcal{D}} = (g_0, Z)$ an den Angreifer $\mathcal{A}$.

Der Angreifer $\mathcal{A}$ generiert den öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}} = (X, SoK_{\mathcal{B}})$ der Bank und sendet diesen an $\mathcal{B}$.

Simulation der Probe-Phase: Der Angreifer $\mathcal{B}$ verhält sich bei allen Orakel-Anfragen, außer bei $\mathcal{O}_{\mathcal{D}}^{\mathsf{UT}}$, wie das jeweilige Orakel und kann somit diese Anfragen perfekt simulieren.

Simulation von $\mathcal{O}_{\mathcal{D}}^{\mathsf{UT}}$: Der Angreifer $\mathcal{B}$ erhält eine Münze $\mathsf{coin} = (k, S, T, T_C, R, \Phi = (A_1, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$.

- Falls der Angreifer $\mathcal{B}$ die Münze coin zum Bezahlen verwendet hat, gibt es die Einträge $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i) \in \mathbb{W}_H$ mit $\mathbb{W}_i = (\Sigma, \cdots)$. Somit kennt $\mathcal{B}$ das Element Σ und gibt dieses aus, wodurch die Anfrage perfekt simuliert ist.

- Falls der Angreifer $\mathcal{B}$ die Münze coin nicht zum Bezahlen verwendet hat, wurde SoK vom Angreifer $\mathcal{A}$ erstellt. Daher kann $\mathcal{B}$ die Zahl $\kappa \in \mathbb{Z}_p$ mit $S = g^{\kappa}$ sowie $T = \Sigma g_0^{R\kappa}$ aus SoK extrahieren und das Element $\Sigma = T g_0^{-R\kappa}$ korrekt berechnen. Dann gibt $\mathcal{B}$ das Element Σ aus, wodurch die Anfrage perfekt simuliert ist. Falls $\mathcal{B}$ die Zahl κ nicht extrahieren kann, bricht $\mathcal{B}$ ab. Dies ist aber nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Simulation der Challenge-Phase: Der Angreifer $\mathcal{B}$ geht wie beim teilbaren Geldsystem TG-II in Abschnitt 6.6 beschrieben vor. Somit sind $S_b = g^{\kappa_b}$ und $S_{\bar{b}} = g^{\kappa_{\bar{b}}}$ zufällige Generatoren in $\mathbb{G}_1$. Zusätzlich wählt $\mathcal{B}$ zufällig zwei Münz-Identifikatoren $T_{C,b}, T_{C,\bar{b}} \in_{\mathcal{R}} \mathbb{G}_1$. Schließlich sendet $\mathcal{B}$ die beiden Münzen $\mathsf{coin}_b = (k_b, S_b, T_b, T_{C,b}, R_b, A_{1,b}, SoK_b, \mathsf{ts}_b, \mathsf{pk}_b)$ und $\mathsf{coin}_{\bar{b}} = \left(k_{\bar{b}}, S_{\bar{b}}, T_{\bar{b}}, T_{C,\bar{b}}, R_{\bar{b}}, A_{1,\bar{b}}, SoK_{\bar{b}}, \mathsf{ts}_{\bar{b}}, \mathsf{pk}_{\bar{b}}\right)$ an den Angreifer $\mathcal{A}$.

Nun müssen wir zeigen, dass beide Münzen rechnerisch ununterscheidbar simuliert sind. Im Gegensatz zum teilbaren Geldsystem TG-II kennt der Angreifer $\mathcal{A}$ zusätzlich pro Abhebeanfrage ein Tripel $\left(E_1 = h^{t'} Z^m, E_2 = h^m, t''\right) \in \mathbb{G}_2^2 \times \mathbb{Z}_p$ und pro Bezahlanfrage ein Paar $(S, T_C) \in \mathbb{G}_1^2$ mit der Eigenschaft $T_C = S^t$ für $t = t' + t''$ mod p. Im Folgenden wird gezeigt, warum diese Werte dem Angreifer $\mathcal{A}$ unter der SXDH-Annahme nur mit vernachlässigbarer Wahrscheinlichkeit helfen, das Spiel zu gewinnen.

Seien $\mathsf{view}_0 = (I_0, \cdots, E_{1,0}, E_{2,0}, \cdots, t_0'')$ und $\mathsf{view}_1 = (I_1, \cdots, E_{1,1}, E_{2,1}, \cdots, t_1'')$ mit $\mathsf{Enc}\,(\mathsf{pk}, h^{t_0}) := \left(h^{t_0''} E_{1,0}, E_{2,0}\right)$ sowie $\mathsf{Enc}\,(\mathsf{pk}, h^{t_1}) := \left(h^{t_1''} E_{1,1}, E_{2,1}\right)$ die beiden Protokollansichten und $\mathbb{W}_0$ und $\mathbb{W}_1$ die beiden Geldbörsen bzgl. der Indizes i_0 und i_1. Seien weiter $(S_{0,i}, T_{C,0,i})$ bzw. $(S_{1,j}, T_{C,1,j})$ für $1 \leq i \leq q_0$ bzw. $1 \leq j \leq q_1$ die Paare der entsprechenden Münzen, die mit $\mathbb{W}_0$ bzw. $\mathbb{W}_1$ bezahlt wurden.

Wir müssen zeigen, dass die Münz-Identifikatoren $T_{C,b}, T_{C,\bar{b}}$ rechnerisch ununterscheidbar simuliert sind. Da $\mathcal{B}$ die beiden Elemente $T_{C,b}, T_{C,\bar{b}} \in_{\mathcal{R}} \mathbb{G}_1$ zufällig wählt, sind diese falsch berechnet. Doch dies bemerkt $\mathcal{A}$ nach Korollar 7.4.1 nur mit vernachlässigbarer Wahrscheinlichkeit unter der XDH-Annahme für $\mathsf{Setup}_{\mathsf{Bil}}$ und der semantischen Sicherheit des Verschlüsselungsverfahrens $(\mathsf{GenEnc}, \mathsf{Enc}, \mathsf{Dec})$ mit $\mathbb{M}_{\mathsf{Enc}} = \mathbb{G}_2 = \langle h \rangle$ und $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenEnc}\left(1^{\lambda}\right)$. (Im Geldsystem FTG-II ist dies das ElGamal-Verschlüsselungsverfahren mit $(\mathsf{pk}, \mathsf{sk}) = ((p, \mathbb{G}_2', h, Z), z)$, das semantisch sicher ist, falls die SXDH-Annahme gilt.) Denn das dem Angreifer $\mathcal{A}$ bekannte Tupel

$$(\mathsf{sp}_{\mathsf{Bil}}, \mathsf{pk}, S_b, S_{\bar{b}}, \{S_{0,i}\}_{i=1}^{q_0}, \{S_{1,j}\}_{j=1}^{q_1}, \{T_{C,0,i}\}_{i=1}^{q_0}, \{T_{C,1,j}\}_{j=1}^{q_1}, \mathsf{Enc}(\mathsf{pk}, h^{t_0}), \mathsf{Enc}\left(\mathsf{pk}, h^{t_1}\right), T_{C,b}, T_{C,\bar{b}})$$

entspricht dem Tupel

$$(\mathsf{sp}_{\mathsf{Bil}}, \mathsf{pk}, g_0, g_1, \{g_{0,i}\}_{i=1}^{q_0}, \{g_{1,j}\}_{j=1}^{q_1}, \{g_{0,i}^{a_0}\}_{i=1}^{q_0}, \{g_{1,j}^{a_1}\}_{j=1}^{q_1},$$

$\mathsf{Enc}(\mathsf{pk}, h^{a_0}), \mathsf{Enc}(\mathsf{pk}, h^{a_1}), A, B)$

aus Korollar 7.4.1. Daher sind die beiden Münz-Identifikatoren rechnerisch ununterscheidbar simuliert.

Somit sind beide Münzen coin_b und $\mathsf{coin}_{\bar{b}}$ rechnerisch ununterscheidbar bzgl. der Elemente Σ_b und $\Sigma_{\bar{b}}$ simuliert.

Anschließend aktualisiert $\mathcal{B}$ die beiden Geldbörsen $\mathbb{W}_0$ und $\mathbb{W}_1$ jeweils so, als würde bei beiden mit dem Geldbetrag $\max\{k_b, k_{\bar{b}}\}$ bezahlt. Daher gilt für die Einträge nun $(i_0, \mathbb{W}'_0, I_0, \$'_0 = \$_0 - \max\{k_b, k_{\bar{b}}\})$ bzw. $(i_1, \mathbb{W}'_1, I_1, \$'_1 = \$_1 - \max\{k_b, k_{\bar{b}}\})$ und somit wurde ein Eintrag korrekt und der andere falsch aktualisiert.

Simulation der Post-Challenge-Phase: Der Angreifer $\mathcal{B}$ verhält sich bei allen Orakel-Anfragen, außer bei $\mathcal{O}_{\mathcal{K}}^{\mathsf{S}^\star}$ und $\mathcal{O}_{\mathcal{K},\mathcal{H}}^{\mathsf{S}^\star}$, wie bei der Simulation der Probe-Phase.

Simulation von $\mathcal{O}_{\mathcal{K}}^{\mathsf{S}^\star}$ und $\mathcal{O}_{\mathcal{K},\mathcal{H}}^{\mathsf{S}^\star}$: Für jeden Index $i \notin \{i_0, i_1\}$ verhält sich der Angreifer $\mathcal{B}$ wie das Orakel.

Falls der Index $i \in \{i_0, i_1\}$ ist, kann sich der Angreifer $\mathcal{B}$ nicht genau wie das Orakel verhalten, da $\mathcal{B}$ die beiden Geldbörsen $\mathbb{W}_0$ und $\mathbb{W}_1$ wie oben beschrieben anders als die Orakel aktualisiert. Der Angreifer $\mathcal{B}$ verhält sich aber wie der entsprechende ehrliche Kunde mit aktualisierter Geldbörse, wodurch die Anfrage im Random-Oracle-Modell perfekt simuliert ist, da der Angreifer $\mathcal{A}$ in dieser Phase bzgl. $\hat{b} \in \{0, 1\}$ nur Münzen im Gesamtwert von $\$_{\hat{b}} - \max\{k_b, k_{\bar{b}}\}$ anfragen darf.

Spielende: Der Angreifer $\mathcal{A}$ gibt mit Wahrscheinlichkeit $1/2 + \epsilon(\lambda)$ das korrekte Bit b aus, wobei ϵ eine nicht vernachlässigbare Funktion ist. Der Angreifer $\mathcal{B}$ leitet das Bit b an das ElGamal-Verschlüsselungs-Orakel weiter und bricht somit die semantische Sicherheit des ElGamal-Verschlüsselungsverfahrens mit Wahrscheinlichkeit $1/2 + \epsilon(\lambda) - \nu(\lambda)$ bricht, wobei ν eine vernachlässigbare Funktion ist.

Double-Spending-Beschuldigung: Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Double-Spending-Beschuldigung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das DLog-Problem in der Gruppe $\mathbb{G}_1$ zu lösen.

Der Angreifer $\mathcal{B}$ erhält die Parameter $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \hat{e}, u_0, h)$ sowie das Tripel $(g, g_0, g_1) \in \mathbb{G}_1^3$ und soll das Darstellungsproblem bzgl. des Generatortupels (g, g_0, g_1, u_0) lösen. Seien $a, b \in \mathbb{Z}_p^*$ die diskreten Logarithmen mit $g_0 = g^a$ und $u_0 = g_0^b$.

Vereinfacht formuliert, kennt $\mathcal{B}$ nach der Ausgabe von $\mathcal{A}$ acht Zahlen $a_1, a_2, a_3, a_4, a'_1, a'_2, a'_3, a'_4 \in \mathbb{Z}_p$ mit $g^{a_1} g_0^{a_2} g_1^{a_3} u_0^{a_4} = g^{a'_1} g_0^{a'_2} g_1^{a'_3} u_0^{a'_4}$. Wie beim teilbaren Geldsystem TG-II müssen wir die beiden Fälle (1) $a_1 \neq a'_1 \ \vee \ a_2 \neq a'_2$ und (2) $a_1 = a'_1 \ \wedge \ a_2 = a'_2$ unterscheiden. Im Folgenden wird die Konstruktion des Angreifers $\mathcal{B}$ detailliert beschrieben und an den nötigen Stellen die beiden Fälle einzeln behandelt.

Simulation der Initialisierung: Der Angreifer $\mathcal{B}$ wählt zufällig zwei Zahlen $\alpha, z \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die Elemente $Z = h^z, u_i = u_0^{\alpha^i}$ sowie $h_i = h^{\alpha^i}$ für $1 \leq i \leq K$. Anschließend sendet $\mathcal{B}$ die perfekt simulierten Systemparameter $\mathsf{sp} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_1, u_0, \ldots, u_K, h, h_1, \ldots, h_K)$ sowie den perfekt simulierten öffentlichen Schlüssel des Deanonymisierers $\mathsf{pk}_{\mathcal{D}} = (g_0, Z)$ an den Angreifer $\mathcal{A}$.

Der Angreifer $\mathcal{A}$ generiert den öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}} = (X, SoK_{\mathcal{B}})$ der Bank und sendet diesen an $\mathcal{B}$, wobei $\mathcal{B}$ die Zahl $x \in \mathbb{Z}_p^*$ aus $SoK_{\mathcal{B}}$ extrahiert.

Simulation von $\mathcal{O}^{\mathsf{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{A}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}_{\mathcal{K}}^{\mathsf{A}}$: Der Angreifer $\mathcal{B}$ wählt zufällig eine Zahl $\gamma \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet die perfekt simulierte Kontonummer $I = g_0 g^{\gamma} = g^{a+\gamma} = g^u$ für $u := a + \gamma \mod p$. Dann speichert $\mathcal{B}$ den Eintrag (I, γ) in einer Liste $\mathcal{U}_H$.

Simulation von $\mathcal{O}_{\mathcal{K}}^{\mathsf{W}}$: Wir unterscheiden die beiden Fälle:

Fall 1: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, außer dass $\mathcal{B}$ die Signature-of-Knowledge SoK_{W} perfekt simuliert (da $\mathcal{B}$ die Zahl $u = a + \gamma \mod p$ nicht kennt). Nach erfolgreicher Durchführung wird ein Eintrag $(i, \mathbb{W}_i = (s', t', \Sigma, e, s'', t'', \mathsf{info}, \phi(\mathsf{x})), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$ sowie ein Eintrag $(i, \mathbb{T} = (I, C, E_1, E_2, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K))$ in einer Liste $\mathbb{T}_H$ gespeichert und der Zähler i um 1 erhöht.

Falls die Anfrage nicht erfolgreich durchgeführt wurde, wird ein Eintrag $(i, \mathbb{W}_i = (s', t', \bot, \bot, \bot, \bot, \mathsf{info}, \phi(\mathsf{x})), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$ sowie ein Eintrag $(i, \mathbb{T} = (I, C, E_1, E_2, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K))$ in einer Liste $\mathbb{T}_H$ gespeichert und der Zähler i um 1 erhöht.

Fall 2: Der Angreifer $\mathcal{B}$ wählt zufällig die Zahl $\kappa_{0,0} \in_{\mathcal{R}} \mathbb{Z}_p^*$ sowie die vier Zahlen $t', a_0, m, \delta \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Commitment $C = u_0 g_0^{\delta} g_1^{t'} u_0^{\phi(\alpha)} = g_0^{b+\delta} g_1^{t'} u_0^{\phi(\alpha)} = g_0^{s'} g_1^{t'} u_0^{\phi(\alpha)}$ für $s' := b + \delta \mod p$ und die Elemente E_1, E_2, A_0 gemäß Protokoll. Anschließend simuliert $\mathcal{B}$ die Signature-of-Knowledge SoK_{W} perfekt, wodurch das gesamte Protokoll perfekt simuliert ist. Nach erfolgreicher Durchführung wird ein Eintrag $(i, \mathbb{W}_i = (\delta, t', \Sigma, e, s'', t'', \mathsf{info}), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$ sowie ein Eintrag $(i, \mathbb{T} = (I, C, E_1, E_2, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K))$ in einer Liste $\mathbb{T}_H$ gespeichert und der Zähler i um 1 erhöht.

Falls die Anfrage nicht erfolgreich durchgeführt wurde, wird ein Eintrag $(i, \mathbb{W}_i = (\delta, t', \bot, \bot, \bot, \bot, \mathsf{info}), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$ sowie ein Eintrag $(i, \mathbb{T} = (I, C, E_1, E_2, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K))$ in einer Liste $\mathbb{T}_H$ gespeichert und der Zähler i um 1 erhöht.

Simulation von $\mathcal{O}_{\mathcal{H}}^{\mathsf{S}^\star}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird u. a. ein Eintrag (j, coin) in einer Liste $\mathbb{C}_{\mathcal{A}}$ gespeichert und der Zähler j um 1 erhöht.

Simulation von $\mathcal{O}_{\mathcal{K}}^{\mathsf{S}^\star}$: Wir unterscheiden die beiden Fälle:

Fall 1: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird u. a. ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_{H,\mathcal{A}}$ gespeichert.

Fall 2: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, außer bei der Signature-of-Knowledge SoK, die $\mathcal{B}$ perfekt simuliert (da $\mathcal{B}$ die Zahl $s = b + \delta + s'' \mod p$ nicht kennt). Somit wird u. a. ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_{H,\mathcal{A}}$ gespeichert.

Simulation von $\mathcal{O}_{\mathcal{K},\mathcal{H}}^{\mathsf{S}^\star}$: Wir unterscheiden die beiden Fälle:

Fall 1: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird u. a. ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_H$ sowie ein Eintrag (j, coin) in einer Liste $\mathbb{C}$ gespeichert und der Zähler j um 1 erhöht.

Fall 2: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, außer bei der Signature-of-Knowledge SoK, die $\mathcal{B}$ perfekt simuliert (da $\mathcal{B}$ die Zahl $s = b + \delta + s'' \mod p$ nicht kennt). Somit wird u. a. ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_H$ sowie ein Eintrag (j, coin) in einer Liste $\mathbb{C}$ gespeichert und der Zähler j um 1 erhöht.

Simulation von $\mathcal{O}_{\mathcal{H}}^{\mathsf{D}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird ein Eintrag (j, coin) in einer Liste $\mathbb{D}_H$ gespeichert.

Simulation von $\mathcal{O}_{\mathcal{D}}^{\mathsf{UT}}$: Der Angreifer $\mathcal{B}$ erhält eine Münze $\mathsf{coin} = (k, S, T, T_C, R, \Phi = (A_1, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$.

- Falls der Angreifer $\mathcal{B}$ die Münze coin zum Bezahlen verwendet hat, gibt es die Einträge $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i) \in \mathbb{W}_H$ mit $\mathbb{W}_i = (\Sigma, \cdots)$. Somit kennt $\mathcal{B}$ das Element Σ und gibt dieses aus, wodurch die Anfrage perfekt simuliert ist.
- Falls der Angreifer $\mathcal{B}$ die Münze coin nicht zum Bezahlen verwendet hat, wurde SoK vom Angreifer $\mathcal{A}$ erstellt. Daher kann $\mathcal{B}$ die Zahl $\kappa \in \mathbb{Z}_p$ mit $S = g^\kappa$ sowie $T = \Sigma g_0^{R\kappa}$ aus SoK extrahieren und das Element $\Sigma = T g_0^{-R\kappa}$ korrekt berechnen. Dann gibt $\mathcal{B}$ das Element Σ aus, wodurch die Anfrage perfekt simuliert ist. Falls $\mathcal{B}$ die Zahl κ nicht extrahieren kann, bricht $\mathcal{B}$ ab. Dies ist aber nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Simulation von $\mathcal{O}_{\mathcal{D}}^{\mathsf{CT}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den geheimen Schlüssel $z \in \mathbb{Z}_p^*$ kennt.

Spielende: Der Angreifer $\mathcal{A}$ gibt mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ eine Kontonummer $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H$ sowie einen Beweis Π_G, der eine Protokollansicht $\mathsf{view} = (\mathbb{T} = (\mathsf{pk}_{\mathcal{K}}, C, E_1, E_2, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s'', t''))$ und zwei Münzen $\mathsf{coin} = (k, S, T, T_C, R, A_1, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}})$ sowie $\mathsf{coin}' = (k', S', T', T_C', R', A_1', SoK', \mathsf{ts}', \mathsf{pk}_{\mathcal{H}}')$ enthält, aus, so dass $\mathsf{VerifyGuilt}(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{D}}, \mathsf{pk}_{\mathcal{K}}, \Pi_G) = 1$ gilt, obwohl nicht $(\cdot, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin}), (\cdot, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin}') \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ ist.

Die Fälle A und B sind wie beim teilbaren Geldsystem TG-II vernachlässigbar. Im Fall E geht $\mathcal{B}$ wie im teilbaren Geldsystem TG-II vor. Somit müssen nur die Fälle C und D betrachtet werden, in denen $\mathcal{B}$ analog zum teilbaren Geldsystem TG-II vorgeht. Wir nehmen an, dass im Folgenden die Signature-of-Knowledge SoK_{W} stets von $\mathcal{B}$ durchgeführt wurde und somit $(\cdot, \mathbb{T}) \in \mathbb{T}_H$ ist, da $\mathcal{B}$ im Fall E: $(\cdot, \mathbb{T}) \notin \mathbb{T}_H$ das DLog-Problem löst. Dann gilt analog zu TG-II je nach Fall:

Fall 1: Es gibt einen Eintrag $(i, \mathbb{T}) \in \mathbb{T}_H$ und einen Eintrag $(i, \mathbb{W}_i = (s', t', \cdot, \cdot, \cdot, \cdot, \mathsf{info}, \phi(\mathsf{x})), \mathsf{pk}_{\mathcal{K}}, \$_i) \in \mathbb{W}_H$. Dieser Eintrag wird zu $(i, \mathbb{W}_i = (s', t', \Sigma, e, s'', t'', \mathsf{info}, \phi(\mathsf{x})), \mathsf{pk}_{\mathcal{K}}, \$_i)$ überschrieben, wobei $\sigma' = (\Sigma, e, s'', t'')$ in der Ausgabe des Angreifers $\mathcal{A}$ enthalten ist. Seien $s := s' + s'' \mod p$ und $t := t' + t'' \mod p$, dann kennt $\mathcal{B}$ das Nachrichten-Signatur-Paar $((t, \phi(\mathsf{x})), (\Sigma, e, s))$.

Fall 2: Es gibt einen Eintrag $(i, \mathbb{T}) \in \mathbb{T}_H$ und einen Eintrag $(i, \mathbb{W}_i = (\delta, t', \cdot, \cdot, \cdot, \cdot, \mathsf{info}, \phi(\mathsf{x})), \mathsf{pk}_{\mathcal{K}}, \$_i) \in \mathbb{W}_H$. Dieser Eintrag wird zu $(i, \mathbb{W}_i = (\delta, t', \Sigma, e, s'', t'', \mathsf{info}, \phi(\mathsf{x})), \mathsf{pk}_{\mathcal{K}}, \$_i)$ überschrieben, wobei $\sigma' = (\Sigma, e, s'', t'')$ in der Ausgabe des Angreifers $\mathcal{A}$ enthalten ist. Seien $s := b + \delta + s'' \mod p$ sowie $t := t' + t'' \mod p$ und $((t, \phi(\mathsf{x})), (\Sigma, e, s))$ das entsprechende Nachrichten-Signatur-Paar, welches $\mathcal{B}$ nicht kennt.

Fall C: $(\cdot, \cdot, \mathsf{coin}), (\cdot, \cdot, \mathsf{coin}') \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$:

Analog zu TG-II extrahiert $\mathcal{B}$ nun die Nachrichten-Signature-Paare $((t_1, \phi_1(\mathsf{x})), (\Sigma_1, e_1, s_1))$ und $((t_2, \phi_2(\mathsf{x})), (\Sigma_2, e_2, s_2))$. Das weitere Vorgehen von $\mathcal{B}$ verläuft nun analog, wobei entsprechend $\Sigma = \left(g g_0^s g_1^t u_0^{\phi(\alpha)}\right)^{\frac{1}{x+e}}, \Sigma_1 = \left(g g_0^{s_1} g_1^{t_1} u_0^{\phi_1(\alpha)}\right)^{\frac{1}{x+e_1}}$ und $\Sigma_2 = \left(g g_0^{s_2} g_1^{t_2} u_0^{\phi_2(\alpha)}\right)^{\frac{1}{x+e_2}}$ gelten.

Fall D: Sei o.B.d.A. $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(\cdot, \cdot, \mathsf{coin}') \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$:

Analog zu TG-II extrahiert $\mathcal{B}$ nun das Nachrichten-Signature-Paar $((t_2, \phi_2(\mathsf{x})), (\Sigma_2, e_2, s_2))$. Entsprechend muss nun je nach Fall unterschieden werden:

Fall 1: Es sei $\mathbb{W}_j = (s_1', t_1', \Sigma_1, e_1, s_1'', t_1'', \cdots)$ und $s_1 := s_1' + s_1'' \mod p$ sowie $t_1 := t_1' + t_1'' \mod p$.

Fall 2: Es sei $\mathbb{W}_j = (\delta_1, t_1', \Sigma_1, e_1, s_1'', t_1'', \cdots)$ und $s_1 := b + \delta_1 + s_1'' \mod p$ sowie $t_1 := t_1' + t_1'' \mod p$.

Das weitere Vorgehen von $\mathcal{B}$ verläuft nun analog, wobei wie im Fall C entsprechend $\Sigma = \left(g g_0^s g_1^t u_0^{\phi(\alpha)}\right)^{\frac{1}{x+e}}, \Sigma_1 = \left(g g_0^{s_1} g_1^{t_1} u_0^{\phi_1(\alpha)}\right)^{\frac{1}{x+e_1}}$ und $\Sigma_2 = \left(g g_0^{s_2} g_1^{t_2} u_0^{\phi_2(\alpha)}\right)^{\frac{1}{x+e_2}}$ gelten.

Abhebe-Beschuldigung: Dies folgt direkt aus dem teilbaren Geldsystem TG-II.

Korrekte Kundenverfolgung: Dies folgt wie bei FTG-0 direkt aus der Unfälschbarkeit dieses Geldsystems. Sei $\mathcal{A}$ ein polynomieller Angreifer, der die korrekte Kundenverfolgung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das q-SDH-Problem zu lösen. Der Angreifer $\mathcal{A}$ gibt am Ende eine Münze $\mathsf{coin} = (k, S, T, T_C, R, \Phi = (A_1, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}})) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ aus.

Somit wurde SoK vom Angreifer $\mathcal{A}$ erstellt und daher kann $\mathcal{B}$ die Zahl $\kappa \in \mathbb{Z}_p$ und das Nachrichten-Signatur-Paar $((t, \phi(\mathsf{x})), (\Sigma, e, s))$ mit $S = g^{\kappa}$ und $T = \Sigma g_0^{R\kappa}$ aus SoK extrahieren. Da kein Kunde $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_{\mathcal{A}}$ eindeutig durch das Protokoll UTrace ausgegeben wird, wird entweder (1) kein Kunde $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_{\mathcal{A}}$ ausgegeben oder (2) mindestens zwei verschiedene Kunden $\mathsf{pk}_{\mathcal{K}}, \mathsf{pk}'_{\mathcal{K}} \in \mathcal{U}_{\mathcal{A}} \cup \mathcal{U}_H$ ausgegeben. Da nach Protokollablauf allerdings jedes Element Σ einmalig berechnet wird, ist Fall (2) ausgeschlossen. Somit wird kein Kunde $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_{\mathcal{A}}$ ausgegeben und das Nachrichten-Signatur-Paar $((t, \phi(\mathsf{x})), (\Sigma, e, s))$ wurde folglich nie bei einer $\mathcal{O}_{\mathcal{B}}^{\mathsf{W}}$-Anfrage erstellt.

- Falls das obige Nachrichten-Signatur-Paar niemals bei einer Orakel-Anfrage erstellt wurde, geht $\mathcal{B}$ wie bei der Unfälschbarkeit im Fall (1.1) vor.
- Falls das obige Nachrichten-Signatur-Paar bei einer $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{W}}$-Anfrage erstellt wurde, geht $\mathcal{B}$ wie bei der Unfälschbarkeit im Fall (1.2) vor.

Kundenverfolgung-Beschuldigung: Dies folgt wie bei FTG-0 im Wesentlichen analog zur Double-Spending-Beschuldigung dieses Geldsystems. Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Kundenverfolgung-Beschuldigung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das DLog-Problem in $\mathbb{G}_1$ zu lösen.

Simulation: Der Angreifer $\mathcal{B}$ erhält dieselben Parameter wie bei der Double-Spending-Beschuldigung und simuliert die Initialisierung sowie die Orakel-Anfragen genauso.

Spielende: Der Angreifer $\mathcal{A}$ sendet erst eine Münze $\mathsf{coin} = (k, S, T, T_C, R, \Phi)$ und nach Erhalt eines Elements Σ eine Protokollansicht $\mathsf{view} = (\mathbb{T} = (\mathsf{pk}_{\mathcal{K}}, C, E_1, E_2, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s'', t''))$ an $\mathcal{B}$, wobei $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H$ oder $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ gilt, obwohl $(\cdot, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ ist.

In den Fällen A bis D geht $\mathcal{B}$ analog zum Geldsystem FTG-0 aus Abschnitt 7.2 vor, wobei, wie bei der Double-Spending-Beschuldigung dieses Geldsystems, entsprechend $\Sigma = \left(g g_0^s g_1^t u_0^{\phi(\alpha)}\right)^{\frac{1}{x+e}}$ und $\Sigma_1 = \left(g g_0^{s_1} g_1^{t_1} u_0^{\phi_1(\alpha)}\right)^{\frac{1}{x+e_1}}$ gelten.

Korrekte Münzverfolgung: Dies folgt wie bei FTG-0 (ebenfalls wie die korrekte Kundenverfolgung) direkt aus der Unfälschbarkeit dieses Geldsystems. Sei $\mathcal{A}$ ein polynomieller Angreifer, der die korrekte Münzverfolgung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer

$\mathcal{A}$ dazu verwendet, das q-SDH-Problem zu lösen. Der Angreifer $\mathcal{A}$ gibt eine Münze $\mathsf{coin} = (k, S, T, T_C, R, \Phi = (A_1, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}})) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ aus.

Somit wurde SoK vom Angreifer $\mathcal{A}$ erstellt und daher kann $\mathcal{B}$ das Nachrichten-Signatur-Paar $((t, \phi(\mathsf{x})), (\Sigma, e, s))$ mit $T_C = S^t$ aus SoK extrahieren. Da die Münze coin nicht eindeutig durch das Protokoll CTrace verfolgt werden kann, folgt entweder (1) $\hat{e}(T_C, h) \neq \hat{e}\left(S, E_{1,i}E_{2,i}^{-z}h^{t_i''}\right) = \hat{e}(S^{t_i}, h)$ für jede Protokollansicht $\mathsf{view}_i = (\cdots, E_{1,i}, E_{2,i}, \cdots, t_i'') \in \mathsf{view}_{\mathcal{A}}$ oder (2) es gibt zwei verschiedene Protokollansichten $\mathsf{view}_0 = (\cdots, E_{1,0}, E_{2,0}, \cdots, t_0''), \mathsf{view}_1 = (\cdots, E_{1,1}, E_{2,1}, \cdots, t_1'') \in \mathsf{view}_{\mathcal{A}} \cup \mathsf{view}_H$ mit $\hat{e}(T_C, h) = \hat{e}\left(S, E_{1,0}E_{2,0}^{-z}h^{t_0''}\right) = \hat{e}\left(S, E_{1,1}E_{2,1}^{-z}h^{t_1''}\right)$.

Wir zeigen zunächst, dass Fall (2) vernachlässigbar ist. Seien $t_0', t_1', m_0, m_1 \in \mathbb{Z}_p$ die bei den Abhebeanfragen (von $\mathcal{A}$ oder $\mathcal{B}$) gewählten Zahlen mit $E_{1,0} = h^{t_0'}Z^{m_0}, E_{2,0} = h^{m_0}, E_{1,1} = h^{t_1'}Z^{m_1}, E_{2,1} = h^{m_1}$ und $t_0'', t_1'' \in_{\mathcal{R}} \mathbb{Z}_p$ die zugehörenden von $\mathcal{B}$ zufällig gewählten Zahlen. Dann folgt $E_{1,0}E_{2,0}^{-z}h^{t_0''} = h^{t_0'+t_0''} = h^{t_1'+t_1''} = E_{1,1}E_{2,1}^{-z}h^{t_1''}$ und somit $t_1'' = t_0' + t_0'' - t_1' \mod p$. Da allerdings $t_1'' \in_{\mathcal{R}} \mathbb{Z}_p$ zufällig von $\mathcal{B}$ gewählt wird, ist dies vernachlässigbar.

Im Fall (1) wurde das Nachrichten-Signatur-Paar nie bei einer $\mathcal{O}_{\mathcal{B}}^{\mathsf{W}}$-Anfrage erstellt.

- Falls das obige Nachrichten-Signatur-Paar niemals bei einer Orakel-Anfrage erstellt wurde, geht $\mathcal{B}$ wie bei der Unfälschbarkeit im Fall (1.1) vor.
- Falls das obige Nachrichten-Signatur-Paar bei einer $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{W}}$-Anfrage erstellt wurde, geht $\mathcal{B}$ wie bei der Unfälschbarkeit im Fall (1.2) vor.

Münzverfolgung-Beschuldigung: Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Münzverfolgung-Beschuldigung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das DLog-Problem in der Gruppe $\mathbb{G}_2$ zu lösen.

Der Angreifer $\mathcal{B}$ erhält die Parameter $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, h)$ sowie das Paar $(u_0^a, h^a) \in \mathbb{G}_1 \times \mathbb{G}_2$ und soll den diskreten Logarithmus $a \in \mathbb{Z}_p^*$ berechnen.

Simulation der Initialisierung: Der Angreifer $\mathcal{B}$ wählt zufällig vier Zahlen $\alpha, \beta, y, z \in_{\mathcal{R}} \mathbb{Z}_p^*$ und einen Generator $g_1 \in_{\mathcal{R}} \mathbb{G}_1$. Danach berechnet $\mathcal{B}$ die Generatoren $g = u_0^{\beta}, g_0 = g^y, Z = h^z$ und $u_i = u_0^{\alpha^i}$ sowie $h_i = h^{\alpha^i}$ für $1 \leq i \leq K$. Anschließend sendet $\mathcal{B}$ die perfekt simulierten Systemparameter $\mathsf{sp} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_1, u_0, \ldots, u_K, h, h_1, \ldots, h_K)$ sowie den perfekt simulierten öffentlichen Schlüssel des Deanonymisierers $\mathsf{pk}_{\mathcal{D}} = (g_0, Z)$ an den Angreifer $\mathcal{A}$.

Der Angreifer $\mathcal{A}$ generiert den öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}} = (X, SoK_{\mathcal{B}})$ der Bank und sendet diesen an $\mathcal{B}$.

Simulation von $\mathcal{O}^{\mathsf{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{A}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}_{\mathcal{K}}^{\mathsf{A}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}_{\mathcal{K}}^{\mathsf{W}}$: Der Angreifer $\mathcal{B}$ wählt zufällig die Zahlen $\kappa_{0,0} \in_{\mathcal{R}} \mathbb{Z}_p^*$ und δ, a_0, $m \in_{\mathcal{R}} \mathbb{Z}_p$ sowie das Element $C \in_{\mathcal{R}} \mathbb{G}_1$ und berechnet das Element A_0 gemäß Protokoll. Dann berechnet $\mathcal{B}$ die Verschlüsselung (E_1, E_2) durch $E_1 = (h^a)^\delta Z^m = h^{a\delta} Z^m = h^{t'} Z^m$ und $E_2 = h^m$ für $t' := a \cdot \delta \mod p$. Somit sind die Elemente C, E_1, E_2, A_0 perfekt simuliert. Anschließend simuliert $\mathcal{B}$ die Signature-of-Knowledge SoK_{W} perfekt. Nach erfolgreicher Durchführung wird ein Eintrag $(i, \mathbb{W}_i = (\delta, \Sigma, e, s'', t'', \mathsf{info}), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$ sowie ein Eintrag $(i, \mathbb{T} = (I, C, E_1, E_2, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K))$ in einer Liste $\mathbb{T}_H$ gespeichert und der Zähler i um 1 erhöht.

Falls die Anfrage nicht erfolgreich durchgeführt wurde, wird ein Eintrag $(i, \mathbb{W}_i = (\delta, \perp, \perp, \perp, \perp, \mathsf{info}), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$ sowie ein Eintrag $(i, \mathbb{T} = (I, C, E_1, E_2, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K))$ in einer Liste $\mathbb{T}_H$ gespeichert und der Zähler i um 1 erhöht.

Simulation von $\mathcal{O}_{\mathcal{H}}^{\mathsf{S}^\star}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird u. a. ein Eintrag (j, coin) in einer Liste $\mathbb{C}_{\mathcal{A}}$ gespeichert und der Zähler j um 1 erhöht.

Simulation von $\mathcal{O}_{\mathcal{K}}^{\mathsf{S}^\star}$: Der Angreifer $\mathcal{B}$ verwendet den Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i) \in \mathbb{W}_H$, berechnet die Elemente S, T, A_1 gemäß Protokoll und berechnet den korrekten Münz-Identifikator $T_C = (u_0^a)^{\beta\kappa\delta} g^{\kappa t''} = S^{a\delta + t''} = S^t$ ohne Kenntnis der Zahl t, wobei $\kappa \in \mathbb{Z}_p^*$ die Zahl mit $S = g^\kappa$ ist. Dann simuliert $\mathcal{B}$ die Signature-of-Knowledge SoK perfekt. Anschließend wird der Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i)$ zu $(i, \mathbb{W}_i', \mathsf{pk}_{\mathcal{K}}, \$_i - k)$ aktualisiert und ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_{H,\mathcal{A}}$ gespeichert.

Simulation von $\mathcal{O}_{\mathcal{K},\mathcal{H}}^{\mathsf{S}^\star}$: Der Angreifer $\mathcal{B}$ verwendet den Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i) \in \mathbb{W}_H$, berechnet die Elemente S, T, A_1 gemäß Protokoll und berechnet den korrekten Münz-Identifikator $T_C = (u_0^a)^{\beta\kappa\delta} g^{\kappa t''} = S^{a\delta + t''} = S^t$ ohne Kenntnis der Zahl t, wobei $\kappa \in \mathbb{Z}_p^*$ die Zahl mit $S = g^\kappa$ ist. Dann simuliert $\mathcal{B}$ die Signature-of-Knowledge SoK perfekt. Anschließend wird der Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i)$ zu $(i, \mathbb{W}_i', \mathsf{pk}_{\mathcal{K}}, \$_i - k)$ aktualisiert, ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_H$ sowie ein Eintrag (j, coin) in einer Liste $\mathbb{C}$ gespeichert und der Zähler j um 1 erhöht.

Simulation von $\mathcal{O}_{\mathcal{H}}^{\mathsf{D}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}_{\mathcal{D}}^{\mathsf{UT}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den geheimen Schlüssel $y \in \mathbb{Z}_p^*$ kennt.

Simulation von $\mathcal{O}_{\mathcal{D}}^{\mathsf{CT}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den geheimen Schlüssel $z \in \mathbb{Z}_p^*$ kennt.

Spielende: Der Angreifer $\mathcal{A}$ sendet mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ eine Protokollansicht $\mathsf{view} = (\mathbb{T} = (I, C, E_1, E_2, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e,$

$s'', t''))$ an $\mathcal{B}$. Sei $\mathsf{coin} = (k, S, T, T_C, R, \Phi = (A_1, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$ eine Münze, die u. a. durch das Protokoll CTrace ausgegeben wird. Somit gilt die Gleichung $\hat{e}\,(T_C, h) = \hat{e}\left(S, E_1 E_2^{-z} h^{t''}\right)$, obwohl

- $(i, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(j, \mathbb{T}) \in \mathbb{T}_H$ mit $j \neq i$ ist, oder
- $(i, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(\cdot, \mathbb{T}) \notin \mathbb{T}_H$ ist, oder
- $(\cdot, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(j, \mathbb{T}) \in \mathbb{T}_H$ ist.

- $(i, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(j, \mathbb{T}) \in \mathbb{T}_H$ mit $j \neq i$:

  In diesem Fall gibt es einen Eintrag $(j, \mathbb{W}_j = (\delta, \cdots), \cdot, \cdot) \in \mathbb{W}_H$. Zudem gibt es die Einträge $(i, \mathbb{W}_i = (\delta_i, \cdots, t_i'', \cdots), \cdot, \cdot) \in \mathbb{W}_H$ und $(i, \cdot, \mathsf{coin} = (\cdot, S, \cdot, T_C, \cdots)) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ mit $T_C = S^{a\delta_i + t_i''}$. Weiter muss nach Voraussetzung aber auch $\hat{e}\,(T_C, h) = \hat{e}\left(S, E_1 E_2^{-z} h^{t''}\right) = \hat{e}\left(S^{a\delta + t''}, h\right)$ gelten, woraus $a\delta_i + t_i'' = a\delta + t''$ folgt. Somit berechnet $\mathcal{B}$ den diskreten Logarithmus $a = (t'' - t_i'')/(\delta_i - \delta) \mod p$, da der Fall $\delta_i = \delta$ vernachlässigbar ist.

- $(i, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(\cdot, \mathbb{T}) \notin \mathbb{T}_H$:

  In diesem Fall wurde SoK_{W} von $\mathcal{A}$ erstellt und $\mathcal{B}$ kann die Zahlen $t', m \in \mathbb{Z}_p$ mit $E_1 = h^{t'} Z^m, E_2 = h^m$ aus SoK_{W} extrahieren. Zudem gibt es die Einträge $(i, \mathbb{W}_i = (\delta_i, \cdots, t_i'', \cdots), \cdot, \cdot) \in \mathbb{W}_H$ und $(i, \cdot, \mathsf{coin} = (\cdot, S, \cdot, T_C, \cdots)) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ mit $T_C = S^{a\delta_i + t_i''}$. Weiter muss nach Voraussetzung aber auch $\hat{e}\,(T_C, h) = \hat{e}\left(S, E_1 E_2^{-z} h^{t''}\right) = \hat{e}\left(S^{t' + t''}, h\right)$ gelten, woraus $a\delta_i + t_i'' = t' + t''$ folgt. Somit berechnet $\mathcal{B}$ den diskreten Logarithmus $a = (t' + t'' - t_i'')/\delta_i \mod p$, da der Fall $\delta_i = 0$ vernachlässigbar ist.

- $(\cdot, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(j, \mathbb{T}) \in \mathbb{T}_H$:

  In diesem Fall wurde SoK von $\mathcal{A}$ erstellt und $\mathcal{B}$ kann die Zahl $t \in \mathbb{Z}_p$ mit $T_C = S^t$, extrahieren. Zudem gibt es einen Eintrag $(j, \mathbb{W}_j = (\delta, \cdots), \cdot, \cdot) \in \mathbb{W}_H$. Nach Voraussetzung muss aber auch $\hat{e}\,(T_C, h) = \hat{e}\left(S, E_1 E_2^{-z} h^{t''}\right) = \hat{e}\left(S^{a\delta + t''}, h\right)$ gelten, woraus $a\delta + t'' = t$ folgt. Somit berechnet $\mathcal{B}$ den diskreten Logarithmus $a = (t - t'')/\delta \mod p$, da der Fall $\delta = 0$ vernachlässigbar ist.

Somit löst $\mathcal{B}$ das DLog-Problem in $\mathbb{G}_2$ mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon\,(\lambda) - \nu\,(\lambda)$, wobei ν eine vernachlässigbare Funktion ist. □

Bemerkung 7.4.1. *Analog zu Bemerkung 6.6.2 sind die Sicherheitsziele Anonymität, Double-Spending-Beschuldigung, Abhebe-Beschuldigung, Kundenverfolgung-Beschuldigung und Münzverfolgung-Beschuldigung auch dann noch gewährleistet, wenn der Angreifer $\mathcal{A}$ die Zahl $\alpha \in_{\mathcal{R}} \mathbb{Z}_p^*$ wählt. Des Weiteren erfüllt das Geldsystem auch die in [Tro06] definierte Anonymität, bei der der Angreifer $\mathcal{A}$ in der Challenge-Phase zusätzlich alle privaten Schlüssel der ehrlichen Kunden erhält.*

7.5 Effizienzvergleich der fairen und teilbaren elektronischen Geldsysteme

Analog zu der Tabelle 6.3 in Abschnitt 6.7 werden in der folgenden Tabelle 7.4 die Unterschiede zwischen dem fairen teilbaren Geldsystem FTG-II aus Abschnitt 7.4 bzgl. des fairen teilbaren Geldsystems FTG-I aus Abschnitt 7.3 aufgelistet, wobei p erneut eine 256-Bit Primzahl ist und zur Simplifizierung $\mathbb{G}_p := \mathbb{G}_1$ gesetzt wird (was im Typ-2- und Typ-3-Pairing möglich ist).

	Unterschied	
sp	$-\mathbb{G}_1^3$	$\Rightarrow -99$ Bytes
$\mathsf{pk}_{\mathcal{B}}$	$+\mathbb{Z}_p^2$	$\Rightarrow +64$ Bytes
Withdraw	$+\mathbb{G}_2$	$\Rightarrow +65$ Bytes
Kunde	$+2$ ME	
Bank	$+1$ ME	
$(\mathbb{T}, \sigma')$	$+\left(\mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{Z}_p^3\right)$	$\Rightarrow +194$ Bytes
Spend bzw. Spend-K	$-\left(\mathbb{G}_1 \times \mathbb{G}_T \times \mathbb{Z}_p^4\right)$	$\Rightarrow -289$ Bytes
Kunde	$-(4$ ME $+ 2$ P$)$	
Händler	$-(2$ ME $+ 1$ P$)$	
Deposit	$-\left(\mathbb{G}_1 \times \mathbb{G}_T \times \mathbb{Z}_p^4\right)$	$\Rightarrow -289$ Bytes
Bank	$-(2$ ME $+ 1$ P$)$	
$\mathsf{coin} \in \mathbb{D}_C$	$-\left(\mathbb{G}_1 \times \mathbb{G}_T \times \mathbb{Z}_p^4\right)$	$\Rightarrow -289$ Bytes

Tabelle 7.4: Unterschied FTG-II im Vergleich zu FTG-I

In der folgenden Tabelle 7.5 werden die fairen teilbaren Geldsysteme FTG-I und FTG-II mit den ihnen zugrunde liegenden teilbaren Geldsystemen TG-I und TG-II für eine 256-Bit Primzahl p verglichen. Dabei wird lediglich der Mehraufwand der fairen teilbaren Geldsysteme angegeben.

Abschließend werden in der Tabelle 7.6 die vier teilbaren elektronischen Geldsysteme TG-I, TG-II, FTG-I und FTG-II für eine 256-Bit Primzahl p miteinander verglichen.

	FTG-I		FTG-II	
	Kunden-verfolgung	Münz-verfolgung	Kunden-verfolgung	Münz-verfolgung
$\mathsf{sp} \cup \mathsf{pk}_{\mathcal{D}}$		$\mathbb{G}_1 \times \mathbb{G}_2$		$\mathbb{G}_1 \times \mathbb{G}_2$
Bytes		98		98
$\mathsf{sk}_{\mathcal{D}}$	$\mathbb{Z}_p$	$\mathbb{Z}_p$	$\mathbb{Z}_p$	$\mathbb{Z}_p$
Bytes	32	32	32	32
Withdraw		$\mathbb{G}_2 \times \mathbb{Z}_p^2$		$\mathbb{G}_2^2 \times \mathbb{Z}_p^3$
Bytes		129		226
$\mathcal{K}unde$		2 ME		4 ME
$\mathcal{B}ank$		1 ME		2 ME
$\mathbb{W}$		$\mathbb{Z}_p$		$\mathbb{Z}_p$
Bytes		32		32
$(\mathbb{T}, \sigma')$		$\mathbb{G}_2 \times \mathbb{Z}_p^2$		$\mathbb{G}_2^2 \times \mathbb{Z}_p^3$
Bytes		129		226
Spend		$\mathbb{G}_T \times \mathbb{Z}_p$		$\mathbb{G}_1 \times \mathbb{Z}_p$
Bytes		160		65
$\mathcal{K}unde$		2 ME + 2 P		2 ME
$\mathcal{H}ändler$		1 ME + 1 P		1 ME
Deposit		$\mathbb{G}_T \times \mathbb{Z}_p$		$\mathbb{G}_1 \times \mathbb{Z}_p$
Bytes		160		65
$\mathcal{B}ank$		1 ME + 1 P		1 ME
$\mathsf{coin} \in \mathbb{D}_C$		$\mathbb{G}_T \times \mathbb{Z}_p$		$\mathbb{G}_1 \times \mathbb{Z}_p$
Bytes		160		65

Tabelle 7.5: Mehraufwand für faire teilbare Geldsysteme

	TG-I	TG-II	FTG-I	FTG-II
sp Bytes	$\mathbb{G}_p^2 \times \mathbb{G}_1^{K+4} \times \mathbb{G}_2^{K+1}$ 100.615	$\mathbb{G}_1^{K+3} \times \mathbb{G}_2^{K+1}$ 100.516	$\mathbb{G}_p \times \mathbb{G}_1^{K+5} \times \mathbb{G}_2^{K+1}$ 100.615	$\mathbb{G}_1^{K+3} \times \mathbb{G}_2^{K+1}$ 100.516
$\mathsf{pk}_{\mathcal{B}}$ Bytes	$\mathbb{G}_2$ 65	$\mathbb{G}_2 \times \mathbb{Z}_p^2$ 129	$\mathbb{G}_2$ 65	$\mathbb{G}_2 \times \mathbb{Z}_p^2$ 129
$\mathsf{sk}_{\mathcal{B}}$ Bytes	$\mathbb{Z}_p$ 32	$\mathbb{Z}_p$ 32	$\mathbb{Z}_p$ 32	$\mathbb{Z}_p$ 32
$\mathsf{pk}_{\mathcal{D}}$ Bytes			$\mathbb{G}_p \times \mathbb{G}_2$ 98	$\mathbb{G}_1 \times \mathbb{G}_2$ 98
$\mathsf{sk}_{\mathcal{D}}$ Bytes			$\mathbb{Z}_p^2$ 64	$\mathbb{Z}_p^2$ 64
Withdraw Bytes *Kunde* *Bank*	$\mathbb{G}_1^3 \times \mathbb{Z}_p^7$ 323 $(2K+5)$ ME + 3 P 4 ME + 2 P	$\mathbb{G}_1^3 \times \mathbb{Z}_p^7$ 323 $(2K+5)$ ME + 3 P 4 ME + 2 P	$\mathbb{G}_1^3 \times \mathbb{G}_2 \times \mathbb{Z}_p^9$ 452 $(2K+7)$ ME + 3 P 5 ME + 2 P	$\mathbb{G}_1^3 \times \mathbb{G}_2^2 \times \mathbb{Z}_p^{10}$ 549 $(2K+9)$ ME + 3 P 6 ME + 2 P
tree Bytes	$\mathbb{Z}_p^{3K-2}$ 98.240	$\mathbb{Z}_p^{3K-2}$ 98.240	$\mathbb{Z}_p^{3K-2}$ 98.240	$\mathbb{Z}_p^{3K-2}$ 98.240
$\mathbb{W}$ Bytes	$\mathbb{G}_1 \times \mathbb{Z}_p^3$ 129	$\mathbb{G}_1 \times \mathbb{Z}_p^3$ 129	$\mathbb{G}_1 \times \mathbb{Z}_p^4$ 161	$\mathbb{G}_1 \times \mathbb{Z}_p^4$ 161
$(\mathbb{T}, \sigma')$ Bytes	$\mathbb{G}_p \times \mathbb{G}_1^2 \times \mathbb{Z}_p^5$ 259	$\mathbb{G}_1^4 \times \mathbb{Z}_p^7$ 356	$\mathbb{G}_p \times \mathbb{G}_1^2 \times \mathbb{G}_2 \times \mathbb{Z}_p^7$ 388	$\mathbb{G}_1^4 \times \mathbb{G}_2^2 \times \mathbb{Z}_p^{10}$ 582
Spend Bytes *Kunde* *Händler*	$\mathbb{G}_p^2 \times \mathbb{G}_1^3 \times \mathbb{Z}_p^{11}$ 517 11 ME + 2 P $(2k+5)$ ME + 3 P	$\mathbb{G}_1^3 \times \mathbb{Z}_p^7$ 323 7 ME + 2 P $(2k+3)$ ME + 3 P	$\mathbb{G}_p^2 \times \mathbb{G}_1^3 \times \mathbb{G}_T \times \mathbb{Z}_p^{12}$ 677 13 ME + 4 P $(2k+6)$ ME + 4 P	$\mathbb{G}_1^4 \times \mathbb{Z}_p^8$ 388 9 ME + 2 P $(2k+4)$ ME + 3 P
Deposit Bytes *Bank*	$\mathbb{G}_p^2 \times \mathbb{G}_1^3 \times \mathbb{Z}_p^{12}$ 549 $(2k+5)$ ME + 3 P	$\mathbb{G}_1^3 \times \mathbb{Z}_p^8$ 355 $(2k+3)$ ME + 3 P	$\mathbb{G}_p^2 \times \mathbb{G}_1^3 \times \mathbb{G}_T \times \mathbb{Z}_p^{13}$ 709 $(2k+6)$ ME + 4 P	$\mathbb{G}_1^4 \times \mathbb{Z}_p^9$ 420 $(2k+4)$ ME + 3 P
$\mathsf{coin} \in \mathbb{D}_C$ Bytes	$\mathbb{G}_p^3 \times \mathbb{G}_1^3 \times \mathbb{Z}_p^{k+12}$ 4.678	$\mathbb{G}_1^4 \times \mathbb{Z}_p^{k+8}$ 4.484	$\mathbb{G}_p^3 \times \mathbb{G}_1^3 \times \mathbb{G}_T \times \mathbb{Z}_p^{k+13}$ 4.838	$\mathbb{G}_1^5 \times \mathbb{Z}_p^{k+9}$ 4.549

Tabelle 7.6: Effizienzvergleich zwischen TG-I, TG-II, FTG-I und FTG-II

7.6 Alternative Varianten der Münzverfolgung

In diesem Abschnitt werden zwei alternative Varianten der Münzverfolgung angegeben. In Unterabschnitt 7.6.1 wird die bereits in Abschnitt 7.1 und 7.4 erwähnte Methode vorgestellt, die von M. Au in [Au09, Abschnitt 5.6] vorgeschlagen wurde. Zudem wird bewiesen, dass dieses Verfahren nicht die Sicherheitseigenschaft Münzverfolgung-Beschuldigung gemäß Definition 4.4.8 gewährleistet und eine Lösung des Problems vorgeschlagen. Da sich die Art der Münzverfolgung dieser Methode im Wesentlichen aber kaum von der Münzverfolgung der Geldsysteme FTG-I bzw. FTG-II unterscheidet, wird in Unterabschnitt 7.6.2 eine andere Variante beschrieben, die zu einer effizienteren Münzverfolgung führt.

7.6.1 Münzverfolgung mit der Methode von Au

Wie bereits in Abschnitt 7.4 erwähnt, erfüllt die in [Au09, Abschnitt 5.6] vorgeschlagene Münzverfolgung nicht die Sicherheitseigenschaft Münzverfolgung-Beschuldigung. In [Au09] verschlüsselt der Kunde im Abhebeprotokoll den Münz-Identifikationsschlüssel $t \in \mathbb{Z}_p$ verifizierbar unter dem öffentlichen Schlüssel des Deanonymisierers und berechnet bei jeder Münze den Münz-Identifikator $T_C = Q^t$ für ein zufälliges Gruppenelement $Q \in \mathbb{G}_1$. Möchte die Bank eine Münzverfolgung durchführen, sendet sie die entsprechende Protokollansicht an den Deanonymisierer und erhält den Münz-Identifikationsschlüssel t. Im Anschluss kann die Bank (oder der Händler) bei jeder Münze coin_i mit dem Paar $(T_{C,i}, Q_i)$ die Gleichung $T_{C,i} \stackrel{?}{=} Q_i^t$ überprüfen um festzustellen, ob die Münze coin_i zum entsprechenden Abhebeprotokoll gehört oder nicht.

Satz 7.6.1. *Die in [Au09, Abschnitt 5.6] vorgeschlagene Münzverfolgung erfüllt nicht die Sicherheitseigenschaft Münzverfolgung-Beschuldigung gemäß Definition 4.4.8.*

Beweis. Wir konstruieren einen polynomiellen Angreifer $\mathcal{A}$, der das Spiel 4.4.8 mit Wahrscheinlichkeit 1 gewinnt und somit die Münzverfolgung-Beschuldigung gemäß Definition 4.4.8 bricht.

Schritt 1: Der Angreifer $\mathcal{A}$ wählt zufällig den privaten Schlüssel $x \in_{\mathcal{R}} \mathbb{Z}_p^*$ und sendet den öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}}$ an den Challenger $\mathcal{C}$.

Schritt 2: Der Angreifer $\mathcal{A}$ stellt zwei Anfragen an das Orakel $\mathcal{O}_{\mathcal{K}}^{\mathsf{A}}$, damit ein ehrlicher Kunde $\mathsf{pk}_{\mathcal{K}}$ sowie ein ehrlicher Händler $\mathsf{pk}_{\mathcal{H}}$ erstellt werden.

Schritt 3: Der Angreifer $\mathcal{A}$ stellt eine Anfrage an das Orakel $\mathcal{O}_{\mathcal{K}}^{\mathsf{W}}$ bzgl. $\mathsf{pk}_{\mathcal{K}}$ und speichert die Protokollansicht $\mathsf{view} = (\mathbb{T}, \sigma')$. Das Orakel speichert einen Eintrag $(1, \mathbb{T})$ in der Liste $\mathbb{T}_H$.

Schritt 4: Der Angreifer $\mathcal{A}$ stellt eine Anfrage an das Orakel $\mathcal{O}_{\mathcal{D}}^{\mathsf{CT}}$ bzgl. view und erhält den Münz-Identifikationsschlüssel $t \in \mathbb{Z}_p$.

Schritt 5: Da der Angreifer $\mathcal{A}$ den geheimen Schlüssel x kennt, erstellt $\mathcal{A}$ eine gültige Münze $\mathsf{coin} = (\cdots, T_C, \cdots, Q, \cdots)$ mit $T_C = Q^t$. Diese Münze kann mit beliebigen Zahlen $u, s, \cdots \in \mathbb{Z}_p$ gemäß Protokoll erstellt werden. Daher kann $\mathcal{A}$ auch die Signature-of-Knowledge SoK korrekt erstellen.

Schritt 6: Der Angreifer $\mathcal{A}$ führt nun mit dem Challenger das Protokoll CTrace bzgl. view aus und sendet die Münze coin an den Challenger. Somit enthält die Ausgabe die Münze coin, obwohl $(1, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(1, \mathbb{T}) \in \mathbb{T}_H$ ist.

Damit bricht der Angreifer $\mathcal{A}$ die Münzverfolgung-Beschuldigung. □

Wenn also die Bank den Münz-Identifikationsschlüssel $t \in \mathbb{Z}_p$ kennt, kann sie beliebige Münzen generieren, die mit CTrace fälschlicherweise verfolgt werden. Eine Möglichkeit dies zu verhindern, ist bei jeder durch CTrace ausgegebenen Münze im Anschluss das Kundenverfolgungsprotokoll UTrace auszuführen. In diesem Fall kann der Angreifer $\mathcal{A}$ keine beliebigen Münzen ausgeben, da $\mathcal{A}$ dazu eine Signature-of-Knowledge über die Kenntnis eines geheimen Schlüssels $\mathsf{sk}_{\mathcal{K}}$ mit $(\mathsf{pk}_{\mathcal{K}}, \mathsf{sk}_{\mathcal{K}}) \in \mathcal{U}_H$ erstellen muss (bei FTG-I in coin und bei FTG-II in $\mathbb{T}$). Da nur der Kunde und der Angreifer $\mathcal{A}$ den Münz-Identifikationsschlüssel t kennen, sind nur diese beiden in der Lage, Münzen zu erstellen, die mit CTrace bzgl. t verfolgt werden. Somit hat die anschließende Ausführung von UTrace keine Auswirkung auf die Anonymität der anderen Kunden oder der anderen Geldbörsen dieses Kunden (vgl. auch Unterabschnitt 7.6.2).

7.6.2 Münzverfolgung mit der Methode von Camenisch *et al.*

Während eine Kundenverfolgung in den fairen teilbaren Geldsystemen FTG-0, FTG-I und FTG-II im Wesentlichen identisch durchgeführt wird und sehr effizient ist (insbesondere, da sich dazu die Abhebe- und Bezahlprotokolle im Vergleich zu den teilbaren Geldsystemen TG-I bzw. TG-II nicht ändern), werden für eine Münzverfolgung bei jeder Münze im fairen teilbaren Geldsystem FTG-I zusätzlich ein Münz-Identifikator $T_C = \hat{e}\,(B_1^t, Z) \in \mathbb{G}_T$ und im fairen teilbaren Geldsystem FTG-II ein zusätzlicher Münz-Identifikator $T_C = S^t \in \mathbb{G}_1$ berechnet. Damit die Bank $\mathcal{B}$ zusammen mit dem Deanonymisierer $\mathcal{D}$ eine Münzverfolgung durchführen kann, müssen im Abhebeprotokoll in FTG-I zusätzlich das Element $E = h^{t'} \in \mathbb{G}_2$ und in FTG-II die ElGamal-Verschlüsselung $(E_1, E_2) = \left(h^{t'} Z^m, h^m\right) \in \mathbb{G}_2^2$ an die Bank gesendet werden. Die Bank erhält in Verdachtsfällen vom Deanonymisierer dann das Element $E^\star = Z^t$ in FTG-I bzw. $E^\star = h^t$ in FTG-II und kann dann bei jeder eingelösten Münze die Gleichung $T_C \stackrel{?}{=} \hat{e}\,(B_1, E^\star)$ bzw. $\hat{e}\,(T_C, h) \stackrel{?}{=} \hat{e}\,(S, E^\star)$ überprüfen. Der Nachteil beider Varianten (und auch der Variante von Au in Unterabschnitt 7.6.1) ist allerdings, dass die Bank $\mathcal{B}$ bei jeder eingelösten Münze in der Datenbank $\mathbb{D}_C$ die Gleichung überprüfen muss und nicht direkt ermitteln kann, welche Münzen aus der Datenbank zu einem bestimmten Abhebeprotokoll gehören.

Dies ist allerdings mit der Technik aus [CHL05, CHL06b] möglich, indem für die Berechnung des Münz-Identifikators eine sichere Pseudozufallsfunktion verwendet wird. Im Folgenden werden kurz die Änderungen des Algorithmus DGen sowie der Protokolle $\mathsf{Withdraw}, \mathsf{Spend}$ und CTrace beschrieben, die sich aus der in [CHL05, CHL06b] eingesetzten Methode zur Münzverfolgung ergeben.

DGen wählt ein sicheres verifizierbares Verschlüsselungsverfahren $(\mathsf{GenEnc}, \mathsf{Enc}, \mathsf{Dec})$ mit $\mathbb{M}_{\mathsf{Enc}} = \mathbb{Z}_p$, generiert ein Schlüsselpaar $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{GenEnc}\left(1^\lambda\right)$ und veröffentlicht $\mathsf{pk}_{\mathcal{D}} = (\mathfrak{g}_0, \mathsf{pk})$, wobei in FTG-II $\mathfrak{g}_0 = g_0$ ist.

Withdraw: Im Abhebeprotokoll muss der Kunde, anstelle des Elements E in FTG-I bzw. der Verschlüsselung (E_1, E_2) in FTG-II, eine verifizierbare Verschlüsselung $\mathsf{Enc}(\mathsf{pk}, t')$ der zufällig gewählten Zahl $t' \in_{\mathcal{R}} \mathbb{Z}_p$ unter dem öffentlichen Schlüssel pk des Deanonymisierers $\mathcal{D}$ erstellen und die korrekte Berechnung der Verschlüsselung $\mathsf{Enc}(\mathsf{pk}, t')$ bzgl. des Commitments C beweisen (vgl. [CHL05, CHL06b]). Die Verschlüsselung $\mathsf{Enc}(\mathsf{pk}, t')$ fließt zur Erstellung der Signature-of-Knowledge SoK_{W} mit als Eingabe in die Hashfunktion H ein. Die Bank speichert in FTG-I die Informationen $(\mathbb{T} = (I, C, \mathsf{Enc}(\mathsf{pk}, t'), A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = t'')$ und in FTG-II die Protokollansicht $\mathsf{view} = (\mathbb{T} = (I, C, \mathsf{Enc}(\mathsf{pk}, t'), A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s'', t''))$ in ihrer Datenbank $\mathbb{D}_W$ ab.

Spend: Im Bezahlprotokoll berechnet der Kunde den Münz-Identifikator $T_C = \mathfrak{g}^{\frac{1}{t+J+1}}$, wobei in FTG-II $\mathfrak{g} = g$ ist (vgl. Kapitel 5) und beweist die korrekte Berechnung von T_C sowie die Relation $0 \leq J \leq K-1$. Dies kann der Kunde direkt wie in [CHL05] beweisen, oder durch öffentliche Signaturen der Zahlen $0, \ldots, K-1$ wie in [ASM07] oder indem die Zahlen $0, \ldots, K-1$ akkumuliert werden und der Akkumulator sowie die Zeugen veröffentlicht werden (wie in Abschnitt 5.1).

CTrace: Möchte die Bank in Verdachtsfällen alle Münzen einer bestimmten Geldbörse verfolgen, sendet $\mathcal{B}$ die Informationen $(\mathbb{T}, \sigma')$ an den Deanonymisierer $\mathcal{D}$. Dieser verifiziert SoK_{W}, entschlüsselt $t' = \mathsf{Dec}(\mathsf{pk}, \mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, t'))$ und sendet t' an die Bank $\mathcal{B}$. Die Bank $\mathcal{B}$ berechnet den Münz-Identifikationsschlüssel $t = t' + t'' \bmod p$ und kann anschließend alle zu t gehörenden Münz-Identifikatoren $T_C^{(0)} = \mathfrak{g}^{1/(t+1)}, \ldots, T_C^{(K-1)} = \mathfrak{g}^{1/(t+K)}$ berechnen, wobei in FTG-II $\mathfrak{g} = g$ ist. Es ist zu beachten, dass der Kunde üblicherweise weniger als K Münz-Identifikatoren generiert, da es nur dann genau K Münz-Identifikatoren sind, wenn der Kunde im Bezahlprotokoll immer mit einer Münze mit dem Wert 1 bezahlt.

Durch diese Methode muss $\mathcal{B}$ nicht jede eingelöste Münze in der Datenbank $\mathbb{D}_C$ überprüfen. Um alle Münzen einer Geldbörse zu sperren, veröffentlicht $\mathcal{B}$ die Münz-Identifikatoren $T_C^{(0)}, \ldots, T_C^{(K-1)}$ auf einer schwarzen Liste, so dass der Händler bereits während des Bezahlprotokolls überprüfen kann, ob der gesendete Münz-Identifikator T_C gültig ist oder gesperrt wurde.

Um den in Unterabschnitt 7.6.1 beschriebenen Angriff gegen die Münzverfolgung-Beschuldigung zu verhindern, muss auch hier bei jeder durch CTrace ausgegebenen Münze im Anschluss das Kundenverfolgungsprotokoll UTrace durchgeführt werden.

Für detaillierte Informationen sei auf [CHL05, CHL06b] verwiesen.

Satz 7.6.2. *Die fairen teilbaren elektronischen Geldsysteme FTG-I und FTG-II sind auch mit oben beschriebener Methode der Münzverfolgung unter der Sicherheit der Pseudozufallsfunktion sowie der Sicherheit des verifizierbaren Verschlüsselungsverfahrens im Random-Oracle-Modell sichere faire elektronische Geldsysteme.*

Beweis. Die Sicherheitseigenschaften **Unfälschbarkeit**, **Double-Spending-Beschuldigung**, **Abhebe-Beschuldigung**, **korrekte Kundenverfolgung** sowie **Kundenverfolgung-Beschuldigung** bleiben unverändert. Somit müssen nur die Sicherheitsziele Anonymität, korrekte Münzverfolgung und Münzverfolgung-Beschuldigung bewiesen werden.

Anonymität: Der Angreifer $\mathcal{B}$ geht wie bei FTG-I bzw. FTG-II beschrieben vor. Dass die Münz-Identifikatoren $T_{C,b}$ und $T_{C,\bar{b}}$ falsch berechnet sind, bemerkt $\mathcal{A}$ unter der Sicherheit der Pseudozufallsfunktion sowie der semantischen Sicherheit des verifizierbaren Verschlüsselungsverfahrens nur mit vernachlässigbarer Wahrscheinlichkeit (vgl. [CHL06b, Theorem 4.1 und Theorem 4.4]).

Korrekte Münzverfolgung: Dies folgt direkt aus der Unfälschbarkeit. Sei $\mathcal{A}$ ein polynomieller Angreifer, der die korrekte Münzverfolgung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das q-SDH-Problem zu lösen. Der Angreifer $\mathcal{A}$ gibt eine Münze $\mathsf{coin} = (k, S, T, T_C, R, \Phi = (A_1, B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}})) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ aus.

Somit wurde SoK vom Angreifer $\mathcal{A}$ erstellt und daher kann $\mathcal{B}$ die Zahl J mit $0 \leq J \leq K-1$ sowie das Nachrichten-Signatur-Paar $((u, t, \phi(\mathsf{x})), (\Sigma, e, s))$ mit $T_C = \mathfrak{g}^{\frac{1}{t+J+1}}$ aus SoK extrahieren. Da die Münze coin nicht eindeutig durch das Protokoll CTrace verfolgt werden kann, folgt entweder (1) $T_C \neq \mathfrak{g}^{\frac{1}{t_i+J+1}}$ für jede Protokollansicht $\mathsf{view}_i = (\cdots, \mathsf{Enc}(\mathsf{pk}, t_i'), \cdots, t_i'') \in \mathsf{view}_{\mathcal{A}}$ mit $t_i := \mathsf{Dec}(\mathsf{pk}, \mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, t_i')) + t_i'' \mod p$ für alle $0 \leq J \leq K-1$ oder (2) es gibt zwei verschiedene Protokollansichten $\mathsf{view}_0 = (\cdots, \mathsf{Enc}(\mathsf{pk}, t_0'), \cdots, t_0''), \mathsf{view}_1 = (\cdots, \mathsf{Enc}(\mathsf{pk}, t_1'), \cdots, t_1'') \in \mathsf{view}_{\mathcal{A}} \cup \mathsf{view}_H$ mit $T_C = \mathfrak{g}^{\frac{1}{t_0+J_0+1}} = \mathfrak{g}^{\frac{1}{t_1+J_1+1}}$ für $0 \leq J_0, J_1 \leq K-1$ und $t_0 = \mathsf{Dec}(\mathsf{pk}, \mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, t_0')) + t_0'' \mod p$ sowie $t_1 = \mathsf{Dec}(\mathsf{pk}, \mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, t_1')) + t_1'' \mod p$.

Wir zeigen zunächst, dass Fall (2) vernachlässigbar ist. Seien $t_0', t_1' \in \mathbb{Z}_p$ die während der Abhebeanfragen (von $\mathcal{A}$ oder $\mathcal{B}$) gewählten Zahlen und $t_0'', t_1'' \in_{\mathcal{R}} \mathbb{Z}_p$ die zugehörenden von $\mathcal{B}$ zufällig gewählten Zahlen. Dann folgt $\mathfrak{g}^{\frac{1}{t_0'+t_0''+J_0+1}} = \mathfrak{g}^{\frac{1}{t_1'+t_1''+J_1+1}}$ bzw. $t_0' + t_0'' + J_0 = t_1' + t_1'' + J_1$ und damit $|t_0' + t_0'' - (t_1' + t_1'')| < K$ (vgl. [CHL06b, Theorem 4.1]). Da allerdings $t_0'', t_1'' \in_{\mathcal{R}} \mathbb{Z}_p$ zufällig von $\mathcal{B}$ gewählt werden, ist dies vernachlässigbar.

In Fall (1) wurde das Nachrichten-Signatur-Paar nie bei einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{B}}$-Anfrage erstellt.

- Falls das obige Nachrichten-Signatur-Paar niemals bei einer Orakel-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.1) vor.
- Falls das obige Nachrichten-Signatur-Paar bei einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.2) vor.

Der Beweis für das faire teilbare Geldsystem FTG-II erfolgt analog, indem $\mathfrak{g} = g$ ist und die beiden Elemente B_1, B_2 nicht in der Münze coin enthalten sind.

Münzverfolgung-Beschuldigung: Durch die zusätzliche Kundenverfolgung im Anschluss an das Protokoll CTrace folgt die Münzverfolgung-Beschuldigung direkt aus der Kundenverfolgung-Beschuldigung. Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Münzverfolgung-Beschuldigung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das DLog-Problem in der Gruppe $\mathbb{G}_p$ (bzw. $\mathbb{G}_1$ bei FTG-II) zu lösen.

Simulation: Der Angreifer $\mathcal{B}$ erhält dieselben Parameter wie bei der Kundenverfolgung-Beschuldigung und simuliert die Initialisierung sowie die Orakel-Anfragen außer $\mathcal{O}^{\mathsf{W}}_{\mathcal{K}}$ genauso.

Simulation von $\mathcal{O}^{\mathsf{W}}_{\mathcal{K}}$: Der Angreifer $\mathcal{B}$ wählt zufällig die drei Zahlen $\kappa_{0,0} \in_{\mathcal{R}} \mathbb{Z}_p^*$ und $t', a_0 \in_{\mathcal{R}} \mathbb{Z}_p$ sowie das Element $C \in_{\mathcal{R}} \mathbb{G}_1$. Dann berechnet $\mathcal{B}$ das Element A_0 sowie die Verschlüsselung $\mathsf{Enc}(\mathsf{pk}, t')$ gemäß Protokoll und simuliert den Beweis über die korrekte Verschlüsselung sowie SoK_{W} perfekt. Nach erfolgreicher Durchführung wird ein Eintrag $(i, \mathbb{W}_i = (t', \Sigma, e, s'', t'', \mathsf{info}), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$ sowie ein Eintrag $(i, \mathbb{T} = (I, C, \mathsf{Enc}(\mathsf{pk}, t'), A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K))$ in einer Liste $\mathbb{T}_H$ gespeichert und der Zähler i um 1 erhöht.

Falls die Anfrage nicht erfolgreich durchgeführt wurde, wird ein Eintrag $(i, \mathbb{W}_i = (t', \bot, \bot, \bot, \bot, \mathsf{info}), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$ sowie ein Eintrag $(i, \mathbb{T} = (I, C, \mathsf{Enc}(\mathsf{pk}, t'), A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K))$ in einer Liste $\mathbb{T}_H$ gespeichert und der Zähler i um 1 erhöht.

Spielende: Der Angreifer $\mathcal{A}$ sendet mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$

bei FTG-I Informationen $(\mathbb{T} = (I, C, \mathsf{Enc}(\mathsf{pk}, t'), A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = t'')$ an $\mathcal{B}$. Sei $\mathsf{coin} = (k, S, T, T_C, R, \Phi = (A_1, B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$ eine Münze mit $I = TS^{-yR}$,

bei FTG-II eine Protokollansicht $\mathsf{view} = (\mathbb{T} = (I, C, \mathsf{Enc}(\mathsf{pk}, t'), A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s'', t'')$ an $\mathcal{B}$. Sei $\mathsf{coin} = (k, S, T, T_C, R, \Phi = (A_1, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$ eine Münze mit $\Sigma = TS^{-yR}$,

die u. a. durch das Protokoll CTrace ausgegeben wird. Somit gilt $T_C = \mathfrak{g}^{\frac{1}{t+J+1}}$ für $t = \mathsf{Dec}(\mathsf{pk}, \mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, t')) + t'' \mod p$ und eine Zahl $J \in [0, K-1]$, obwohl

- $(i, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(j, \mathbb{T}) \in \mathbb{T}_H$ mit $j \neq i$ ist, oder
- $(i, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(\cdot, \mathbb{T}) \notin \mathbb{T}_H$ ist, oder
- $(\cdot, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(j, \mathbb{T}) \in \mathbb{T}_H$ ist.

- $(i, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(j, \mathbb{T}) \in \mathbb{T}_H$ mit $j \neq i$:

  In diesem Fall gibt es einen Eintrag $(j, \mathbb{W}_j = (t', \cdots), \cdot, \cdot) \in \mathbb{W}_H$. Zudem gibt es die Einträge $(i, \mathbb{W}_i = (t'_i, \cdots, t''_i, \cdots), \cdot, \cdot) \in \mathbb{W}_H$ und $(i, \cdot, \mathsf{coin} = (\cdot, S, T, T_C, R, \Phi)) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ mit $T_C = \mathfrak{g}^{\frac{1}{t'_i + t''_i + J_i + 1}}$ für eine Zahl $J_i \in [0, K-1]$. Weiter muss nach Voraussetzung aber auch $T_C = \mathfrak{g}^{\frac{1}{t+J+1}}$ für ein $J \in [0, K-1]$ und $t = \mathsf{Dec}(\mathsf{pk}, \mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, t')) + t'' \mod p$ gelten. Damit folgt $t'_i + t''_i + J_i = t' + t'' + J$ und somit $|t' + t'' - (t'_i + t''_i)| < K$. Da allerdings $t', t'_i \in_{\mathcal{R}} \mathbb{Z}_p$ zufällig von $\mathcal{B}$ gewählt werden, ist dies vernachlässigbar (vgl. [CHL06b], Theorem 4.1).

- $(i, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(\cdot, \mathbb{T}) \notin \mathbb{T}_H$:

  In diesem Fall wurde SoK_{W} von $\mathcal{A}$ erstellt und $\mathcal{B}$ kann die Zahl $u \in \mathbb{Z}_p$ mit $I = \mathfrak{g}^u$ aus SoK_{W} extrahieren. Der Angreifer $\mathcal{B}$ sucht das Paar $(I = \mathfrak{g}^u, \gamma) \in \mathcal{U}_H$ und berechnet den diskreten Logarithmus $a = u - \gamma \mod p$.

- $(\cdot, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $(j, \mathbb{T}) \in \mathbb{T}_H$:

  In diesem Fall gilt $(\cdot, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ und $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H$ mit $\mathbb{T} = (\mathsf{pk}_{\mathcal{K}}, \cdots)$. Dies ist ein direkter Widerspruch zur Kundenverfolgung-Beschuldigung. Daher geht $\mathcal{B}$ wie bei der Kundenverfolgung-Beschuldigung vor (bei FTG-II ist dies Fall C).

Somit löst $\mathcal{B}$ das DLog-Problem in $\mathbb{G}_p$ (bzw. in $\mathbb{G}_1$ bei FTG-II) mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$ (bzw. $\epsilon(\lambda)/2 - \nu(\lambda)$ bei FTG-II), wobei ν eine vernachlässigbare Funktion ist. □

7.7 Elektronische Geldsysteme mit erstattungsfähigen Geldbörsen

Die beiden fairen teilbaren elektronischen Geldsysteme FTG-I (Abschnitt 7.3) und FTG-II (Abschnitt 7.4) garantieren der Bank eine Münzverfolgung in Zusammenarbeit mit dem Deanonymisierer. Somit kann die Bank mit Hilfe des Deanonymisierers alle Münzen bzgl. eines bestimmten Abhebeprotokolls verfolgen und gegebenenfalls sperren. Als *Erstattungsfähigkeit* bezeichnen wir die Funktion, die es den Kunden ermöglicht in Kooperation mit einer Trusted-Third-Party $\mathcal{TTP}$ (dies kann auch der Deanonymisierer

sein) die eigenen Geldbörsen im Fall von Verlust oder Diebstahl sperren zu lassen, damit der Geldbetrag der noch nicht ausgegebenen Münzen von der Bank erstattet wird. Dazu wird ein Recovery-Protokoll benötigt, das auf Wunsch eines Kunden zwischen der Bank und der Trusted-Third-Party ausgeführt wird. Das erste elektronische Offline-Geldsystem mit Erstattungsfähigkeit wurde von J. Liu, P. Tsang und D. Wong [LTW05] entwickelt.

Das Hauptziel eines elektronischen Geldsystems mit Erstattungsfähigkeit sollte sein, dass alle bereits ausgegebenen Münzen weiterhin anonym bleiben. Dies wird in [LTW05] jedoch offensichtlich nicht erfüllt (siehe [LTW05, Abschnitt 3.4] und den folgenden Punkt 4.).

Da elektronische Geldsysteme zwar ein digitales Äquivalent zum herkömmlichen Bargeld darstellen und eine Erstattungsfähigkeit nach Verlust oder Diebstahl des Bargelds nicht möglich ist, ist dies dennoch eine wünschenswerte Eigenschaft für elektronische Geldsysteme. Im Gegensatz zum Bargeld verliert ein Kunde nämlich nicht nur sein aktuelles Guthaben der Geldbörse. Denn wenn ein Dieb an die geheimen Daten der Geldbörse gelangt, kann er gültige Münzen erstellen und diese auch mehrfach ausgeben. In diesem Fall wird ein unschuldiger Kunde für Double-Spendings bestraft, die mit seiner gestohlenen Geldbörse begangen wurden.

Zunächst werden jedoch die Probleme aufgelistet, die durch eine Integration der Erstattungsfähigkeit in ein bestehendes elektronisches Geldsystem entstehen, selbst wenn dieses weder teilbar, kompakt noch fair ist.

1. Damit die bereits bezahlten Münzen eines Kunden nicht ihre Anonymität verlieren, darf es nach einem Diebstahl der Geldbörse nicht möglich sein, dass der Dieb ein Double-Spending begehen kann. Somit müssen alle Informationen gelöscht werden, die ein Double-Spending ermöglichen würden. Daher dürfte bei obigen elektronischen Geldsystemen TG-I, TG-II, FTG-0, FTG-I und FTG-II die Zahl $\kappa_{0,0} \in \mathbb{Z}_p^*$ nicht in der Geldbörse gespeichert bleiben, sondern der Kunde muss die Listen $\mathbb{K}_0$ und $\mathbb{K}_1$ wie beschrieben verwalten. Somit ist dies für die in dieser Arbeit entwickelten elektronischen Geldsysteme ein leicht zu lösendes Problem, nicht jedoch bei [CHL05, ASM07, CG07]. (Beachte, dass ein korrupter Händler die Münze an die korrupte Bank weiterleiten könnte, selbst wenn die Münze gesperrt und damit ungültig ist.)

2. Da ein korrupter Dieb allerdings die Geldbörse an die Bank weiterleiten kann, dürfen keine Informationen über bereits bezahlte Münzen gespeichert werden. Dieses Problem kann in den in dieser Arbeit entwickelten elektronischen Geldsysteme jedoch nicht behoben werden, da der Kunde die Liste $\mathbb{K}_1$ (oder andere Informationen über die Serienschlüssel) speichern muss, um im Bezahlprotokoll den entsprechenden Zeugen zu berechnen.

3. Damit wirklich alle Geldbörsen eines Kunden gesperrt werden, muss die Trusted-Third-Party die Datenbank $\mathbb{D}_W$ der Bank einsehen können und selbst nach den

entsprechenden Protokollansichten des Kunden suchen. Würde die Bank die entsprechenden Protokollansichten an die Trusted-Third-Party senden, könnte sie einige weglassen. Die Münzen dieser Geldbörsen würden dann nicht gesperrt werden und die Bank müsste dem Kunden möglicherweise einen geringeren Geldbetrag zurückerstatten.

Eine Alternative wäre, dass der Kunde unmittelbar nach Durchführung des Abhebeprotokolls die vollständige Protokollansicht (der Bank) $\mathsf{view} = (\mathbb{T}, \sigma')$ selbst an die Trusted-Third-Party sendet.

4. Damit die bereits bezahlten Münzen eines Kunden nicht ihre Anonymität verlieren und dem Kunden nach Sperrung der Geldbetrag seiner noch nicht bezahlten Münzen zurückerstattet wird, muss die Trusted-Third-Party die Datenbank $\mathbb{D}_C$ der Bank einsehen können und selbst nach den entsprechenden Münzen bzgl. des Kunden suchen. Denn andernfalls müsste die Bank feststellen, welche eingelösten Münzen von diesem Kunden ausgegeben wurden, wodurch diese direkt ihre Anonymität verlieren würden. (Genau dies ist in [LTW05] der Fall.)

   Analog wäre eine Alternative, dass der Kunde unmittelbar nach Durchführung des Bezahlprotokolls die Münze coin selbst an die Trusted-Third-Party sendet.

5. Da die Trusted-Third-Party jedoch nur die bereits eingelösten Münzen überprüfen kann, aber der Kunde auch bereits einige Münzen bezahlt haben kann, die noch nicht eingelöst wurden, entspricht der Geldbetrag der eingelösten Münzen in der Datenbank $\mathbb{D}_C$ nicht unbedingt dem Geldbetrag der ausgegebenen Münzen. (Dieses Problem wurde in [LTW05] völlig außer Acht gelassen.) Damit einem Kunden also nicht zu viel Geld zurückerstattet wird, müssen erst die Händler alle erhaltenen Münzen vor dem Zeitpunkt der Sperrung bei der Bank einlösen. Erst dann kann die Trusted-Third-Party die eingelösten Münzen überprüfen und den Geldbetrag ermitteln, der noch nicht ausgegeben wurde.

   Auch hier wäre die Alternative, dass der Kunde unmittelbar nach Durchführung des Bezahlprotokolls die Münze coin selbst an die Trusted-Third-Party sendet. Da dies aber ein Nachteil für den Kunden ist, würde ein korrupter Kunde dies nicht machen.

6. Damit allerdings dem Kunden nach Sperrung der Geldbetrag seiner noch nicht ausgegebenen Münzen von der Bank zurückerstattet werden kann, muss die Trusted-Third-Party bestimmte Informationen an die Bank senden. Aus diesen Informationen muss mindestens hervorgehen, welcher Geldbetrag noch nicht bezahlt wurde und dem Kunden zurückerstattet werden muss. Jedoch ist dann die Anonymität nicht mehr zu gewährleisten, da der Angreifer $\mathcal{B}$ diese Informationen im Sicherheitsbeweis nicht simulieren kann. Im folgenden Unterabschnitt 7.7.1 wird gezeigt, wie ein polynomieller Angreifer $\mathcal{A}$ allein durch die Information des noch nicht ausgegebenen Geldbetrages die Anonymität bricht.

7.7.1 Unvereinbarkeit von Anonymität und Erstattungsfähigkeit

Da bei elektronischen Geldsystemen mit Erstattungsfähigkeit ein weiteres Protokoll Recovery benötigt wird, ist für die Sicherheitsanalyse entsprechend ein weiteres Orakel $\mathcal{O}^{\mathsf{Rec}}$ erforderlich, welches der Angreifer $\mathcal{A}$ befragen darf. Wie oben beschrieben, muss das Orakel, um eine Erstattungsfähigkeit realisieren zu können, Informationen an die Bank senden, aus denen mindestens hervorgeht, welcher Geldbetrag eines Kunden noch nicht bezahlt wurde. Aus diesem Grund definieren wir das Orakel $\mathcal{O}^{\mathsf{Rec}}$, ohne genauer auf das eigentliche Protokoll Recovery einzugehen.

- $\mathcal{O}^{\mathsf{Rec}}$: Mit diesem Orakel kann der Angreifer $\mathcal{A}$ das Protokoll Recovery durchführen. Der Angreifer $\mathcal{A}$ sendet einen öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H$ an das Orakel. Seien $(\cdot, \cdot, \mathsf{pk}_{\mathcal{K}}, \$_1), \ldots, (\cdot, \cdot, \mathsf{pk}_{\mathcal{K}}, \$_q) \in \mathbb{W}_H$ alle Tupel bzgl. des öffentlichen Schlüssels $\mathsf{pk}_{\mathcal{K}}$. Dann sendet das Orakel den Geldbetrag $\$ = \sum_{j=1}^{q} \$_j$ der noch nicht ausgegebenen Münzen an den Angreifer $\mathcal{A}$.

Durch das neue Orakel muss die Sicherheitsdefinition der Anonymität angepasst werden.

Spiel 7.7.1 (**Anonymität bei Erstattungsfähigkeit**). *Dieses Spiel verläuft wie das Spiel 4.4.2 (Anonymität), allerdings darf der Angreifer $\mathcal{A}$ in der Probe-Phase und in der Post-Challenge-Phase zusätzlich adaptiv eine polynomiell beschränkte Anzahl an Anfragen an das Orakel $\mathcal{O}^{\mathsf{Rec}}$ stellen.*

Definition 7.7.1 (**Anonymität bei Erstattungsfähigkeit**). *Ein elektronisches Geldsystem mit Erstattungsfähigkeit erfüllt die Sicherheitseigenschaft* Anonymität*, wenn es für jeden polynomiellen Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion ν gibt, so dass für alle hinreichend großen $\lambda \in \mathbb{N}$ gilt:*

$$\Pr\left[\mathcal{A} \textit{ gewinnt Spiel 7.7.1}\right] \leq \frac{1}{2} + \nu(\lambda).$$

Nun können wir zeigen, dass kein elektronisches Geldsystem mit Erstattungsfähigkeit die in Definition 7.7.1 definierte Anonymität garantieren kann. Auch dann nicht, wenn das Geldsystem weder teilbar, kompakt noch fair ist.

Satz 7.7.1. *Kein elektronisches Geldsystem mit Erstattungsfähigkeit besitzt die Anonymität gemäß Definition 7.7.1.*

Beweis. Wir konstruieren einen polynomiellen Angreifer $\mathcal{A}$, der das Spiel 7.7.1 mit Wahrscheinlichkeit 1 gewinnt und somit die Anonymität gemäß Definition 7.7.1 bricht.

Schritt 1: Der Angreifer $\mathcal{A}$ generiert das Schlüsselpaar der Bank gemäß Vorschrift und sendet den öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}}$ an den Challenger $\mathcal{C}$.

Schritt 2: Der Angreifer $\mathcal{A}$ stellt drei Anfragen an das Orakel $\mathcal{O}_{\mathcal{K}}^{\mathsf{A}}$, damit zwei ehrliche Kunden $\mathsf{pk}_0, \mathsf{pk}_1$ und ein ehrlicher Händler $\mathsf{pk}_{\mathcal{H}}$ erstellt werden.

Schritt 3: Der Angreifer $\mathcal{A}$ stellt bzgl. jedes Kunden eine Anfrage an das Orakel $\mathcal{O}_{\mathcal{K}}^{\mathsf{W}}$ bzgl. desselben Geldbetrages K (wobei $K = 1$ ist, wenn das Geldsystem weder teilbar noch kompakt ist). Seien $(0, \mathbb{W}_0, \mathsf{pk}_0, K), (1, \mathbb{W}_1, \mathsf{pk}_1, K) \in \mathbb{W}_H$ die vom Orakel $\mathcal{O}_{\mathcal{K}}^{\mathsf{W}}$ gespeicherten Einträge.

Schritt 4: Der Angreifer $\mathcal{A}$ sendet als Challenge die Indizes $0, 1$, zwei öffentliche Schlüssel $\mathsf{pk}_{\mathcal{H}_b}, \mathsf{pk}_{\mathcal{H}_{\bar{b}}}$ mit $\mathsf{pk}_{\mathcal{H}_b} = \mathsf{pk}_{\mathcal{H}_{\bar{b}}} = \mathsf{pk}_{\mathcal{H}}$, zwei beliebige Zeitpunkte $\mathsf{ts}_b, \mathsf{ts}_{\bar{b}}$ und zwei Geldbeträge $k_b = 0, k_{\bar{b}} = 1$ an den Challenger $\mathcal{C}$. Anschließend werden die beiden Einträge zu $(0, \mathbb{W}_0', \mathsf{pk}_0, \$_0 = K - k_0)$ und $(1, \mathbb{W}_1', \mathsf{pk}_1, \$_1 = K - k_1)$ aktualisiert.

Schritt 5: In der Post-Challenge-Phase stellt der Angreifer $\mathcal{A}$ bzgl. des Kunden pk_0 eine Anfrage an das Orakel $\mathcal{O}^{\mathsf{Rec}}$ und erhält den Geldbetrag $\$_0$ der noch nicht ausgegebenen Münzen des Kunden pk_0.

Schritt 6: Der Angreifer $\mathcal{A}$ überprüft, ob $\$_0 \stackrel{?}{=} K$ oder $\$_0 \stackrel{?}{=} K - 1$ gilt. Wenn $\$_0 = K$ ist, gibt der Angreifer $\mathcal{A}$ das Bit $b' = 0$ aus und wenn $\$_0 = K - 1$ ist, gibt der Angreifer $\mathcal{A}$ das Bit $b' = 1$ aus.

Damit gewinnt der Angreifer $\mathcal{A}$ das Spiel 7.7.1. □

Der Angreifer $\mathcal{A}$ würde das Spiel 7.7.1 selbst dann noch gewinnen, wenn der Challenger in der Challenge-Phase die beiden Indizes bzw. die beiden Kunden wählt (wie es bei anderen Autoren definiert ist, vgl. Unterabschnitt 4.5.1), da der Challenger ja diese beiden Indizes wählen muss. Des Weiteren wird deutlich, dass der Angreifer $\mathcal{A}$ die beiden Münzen coin_b und $\mathsf{coin}_{\bar{b}}$ überhaupt nicht benötigt um das Spiel zu gewinnen.

Durch die bisherige Münzverfolgung in den fairen teilbaren Geldsystemen FTG-I und FTG-II kann die Erstattungsfähigkeit realisiert werden, wenn die obigen unter 3. bis 5. genannten Probleme entsprechend behoben werden. Im Fall von Verlust oder Diebstahl der Geldbörsen könnten sich die Kunden dann also aussuchen, ob sie ihre noch nicht ausgegebenen Münzen erstattet bekommen und gleichzeitig aber auf die Anonymität aller ihrer bereits ausgegebenen Münzen verzichten.

Alternativ können nach Verlust oder Diebstahl die noch nicht verwendeten Seriennschlüssel vom Kunden selbst an die Bank gesendet werden. Dies setzt allerdings voraus, dass die Kunden in regelmäßigen Abständen Sicherungskopien ihrer Daten anlegen. Diese Variante wäre daher auch bei elektronischen Geldsystemen möglich, bei denen keine Trusted-Third-Party involviert ist. Allerdings muss sichergestellt werden, dass eine korrupte Bank dem Kunden auch wirklich den entsprechenden Geldbetrag zurückerstattet. Analog zu dem obigen Beweis gewinnt der Angreifer $\mathcal{A}$ das Spiel 7.7.1 aber ebenfalls mit Wahrscheinlichkeit 1.

KAPITEL 8

ELEKTRONISCHE GELDSYSTEME MIT MODIFIZIERTEN PROTOKOLLEN

In diesem Kapitel werden die in den vorherigen Kapiteln 6 und 7 entwickelten teilbaren elektronischen Geldsysteme modifiziert, so dass Kunden im Abhebeprotokoll Geldbörsen mit einem beliebigen Geldbetrag $K' \leq K$ abheben und im Bezahlprotokoll mit einer Münze mit einem beliebigen Wert $k \leq K'$ bezahlen können. Nach bestem Wissen besitzt nur das Geldsystem von [CDG+09] diese Eigenschaften, welches allerdings nur eine sehr schwache Anonymität bietet und die Identifikation eines Double-Spenders nur mit Hilfe einer Trusted-Third-Party garantiert (vgl. Abschnitt 4.5 und siehe [CDG+09] für Details).

Die Modifizierungen werden dabei bei den fairen teilbaren Geldsystemen FTG-I und FTG-II vorgenommen um zu zeigen, dass die Änderungen keine Auswirkungen auf die Sicherheitsziele der fairen teilbaren Geldsysteme besitzen. Dies impliziert, dass die Modifizierungen bei den teilbaren Geldsystemen TG-I und TG-II die Sicherheitsmerkmale ebenfalls nicht beeinflussen. Somit können die folgenden Veränderungen analog auf die teilbaren Geldsysteme TG-I und TG-II übertragen werden.

8.1 Teilbare Münzen mit beliebigem Geldwert

In diesem Abschnitt wird das Bezahlprotokoll Spend der beiden fairen teilbaren elektronischen Geldsysteme FTG-I und FTG-II aus Abschnitt 7.3 und 7.4 so konstruiert, dass mit einer teilbaren Münze mit beliebigem Wert bezahlt werden kann. Somit muss der Wert k keine Zweierpotenz 2^ℓ sein, wodurch der Bezahlvorgang und damit auch das Einlösen erheblich effizienter werden. Die weiteren Protokolle und Algorithmen werden

dementsprechend angepasst.

Möchte ein Kunde $\mathcal{K}$ eine teilbare Münze mit einem Wert $k < K$ zum Bezahlen verwenden, teilt $\mathcal{K}$ den Wert k eindeutig in $k = 2^{\ell_1} + \cdots + 2^{\ell_n}$ mit $\ell_1 > \cdots > \ell_n$ auf, was der Bitdarstellung von k entspricht. Somit müssen statt $n = O(\log(k))$ Bezahlprotokollen nur noch ein einziges Bezahlprotokoll durchgeführt werden. Der Kunde wählt dann jeweils den gültigen Schlüssel $\kappa_{L-\ell_i,j_i}$ für $1 \leq i \leq n$ der $(L-\ell_i)$-ten Zeile des Binär-Baums mit dem kleinsten Wert j_i und berechnet mit diesen n Schlüsseln die n Seriennummern $S_i = \mathfrak{g}^{\kappa_{L-\ell_i,j_i}}$ in FTG-I (bzw. $S_i = g^{\kappa_{L-\ell_i,j_i}}$ in FTG-II) sowie die zugehörigen n Sicherheitstags $T_i = \mathfrak{g}^u \mathfrak{g}_0^{R\kappa_{L-\ell_i,j_i}}$ (bzw. $T_i = \Sigma g_0^{R\kappa_{L-\ell_i,j_i}}$).

Da der Händler ebenfalls die n Zahlen $\ell_1, \ldots, \ell_n$ mit $k = 2^{\ell_1} + \cdots + 2^{\ell_n}$ und $\ell_1 > \cdots > \ell_n$ kennt, kann er aus den n Seriennummern alle k Serienschlüssel berechnen. Der Vorteil dieser Bezahlprotokolle ist nun der, dass der Kunde nur einmal die Elemente A_1, B_1, B_2 (bzw. A_1 bei FTG-II) berechnen und die Signature-of-Knowledge SoK nur einmal erstellen muss. Weiter müssen Händler und Bank in den entsprechenden Protokollen nur eine Signature-of-Knowledge verifizieren. Folglich kann auch eine teilbare Münze mit einem beliebigen Wert $k < K$ bei der Bank eingelöst werden.

Obwohl die wesentliche Veränderung nur das Protokoll Spend betrifft, muss zudem der Algorithmus Identify angepasst werden. Die Algorithmen Setup, DGen und BGen sowie die Protokolle Account und Spend-K werden wie beim fairen teilbaren Geldsystem FTG-I bzw. FTG-II durchgeführt. Die Algorithmen VerifyGuilt sowie VerifyWithdraw und die Protokolle Deposit, UTrace sowie CTrace werden dann analog zu FTG-I bzw. FTG-II durchgeführt und aus diesem Grund nicht beschrieben.

Spend FTG-I: Damit ein Kunde $\mathcal{K}$, der im Besitz der Geldbörse $\mathbb{W} = (\Sigma, e, s, t, \mathsf{info})$ ist, bei einem Händler $\mathcal{H}$ mit einer elektronischen Münze coin mit dem Wert $k = 2^{\ell_1} + \cdots + 2^{\ell_n} < K$ für $\ell_1 > \cdots > \ell_n$ bezahlen kann, führen $\mathcal{K}$ und $\mathcal{H}$ das Bezahlprotokoll Spend durch, welches in Abbildung 8.1 dargestellt ist. Zuvor authentifiziert sich $\mathcal{H}$ gegenüber $\mathcal{K}$ und beide gleichen den Zeitpunkt ts ab.

Schritt 1: Der Händler $\mathcal{H}$ und der Kunde $\mathcal{K}$ berechnen beide den Hashwert $R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}} || \mathsf{ts} || k)$.

Schritt 2: Der Kunde $\mathcal{K}$ teil den Wert k in $k = 2^{\ell_1} + \cdots + 2^{\ell_n}$ mit $\ell_1 > \cdots > \ell_n$ auf und wählt jeweils den gültigen Schlüssel $\kappa_{L-\ell_i,j_i} := \kappa_i \in \mathbb{K}_0$ mit dem kleinsten Wert j_i. Dementsprechend berechnet $\mathcal{K}$ die n Seriennummern $S_i = \mathfrak{g}^{\kappa_i}$ sowie die n Sicherheitstags $T_i = \mathfrak{g}^u \mathfrak{g}_0^{R\kappa_i}$ für $1 \leq i \leq n$. Weiter wählt der Kunde $\mathcal{K}$ die beiden Zahlen $b_1, b_2 \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Elemente $B_1 = g_0^{b_1} g_1^{b_2}$ sowie $B_2 = \Sigma g_1^{b_2}$. Dann berechnet $\mathcal{K}$ den Münz-Identifikator $T_C = \hat{e}(B_1^t, Z)$ sowie den Hashwert $l = \mathsf{H}(S_1 || \ldots || S_n || T_1 || \ldots || T_n || T_C || B_1 || B_2)$. Weiter berechnet $\mathcal{K}$ das Element $A_1 = W_{I,l} u_0^{a_1}$ wie gewohnt. Anschließend sendet $\mathcal{K}$ die Elemente $S_1, \ldots, S_n, T_1, \ldots, T_n, T_C, A_1, B_1, B_2$ an $\mathcal{H}$ und erstellt die folgende Signature-of-Knowledge SoK, wobei $\beta_1 = b_1 e, \beta_2 = b_2 e, u_I =$

$u_0^{\prod_{j\in I}(\alpha+\kappa_{L+1,j})}, u_{l,I} = u_0^{(\alpha-l)\prod_{j\in I}(\alpha+\kappa_{L+1,j})}$ und $h_{l,I} = h^{(\alpha-l)\prod_{j\in I}(\alpha+\kappa_{L+1,j})}$ ist:

$$SoK\Bigg[(a_1,b_1,b_2,\beta_1,\beta_2,e,s,t,u,\kappa_1,\ldots,\kappa_n,\phi_I(l)) : B_1 = g_0^{b_1}g_1^{b_2} \wedge$$
$$1 = B_1^e g_0^{-\beta_1} g_1^{-\beta_2} \ \wedge\ S_1 = \mathfrak{g}^{\kappa_1},\ldots,S_n = \mathfrak{g}^{\kappa_n},\ \wedge T_1 = \mathfrak{g}^u\mathfrak{g}_0^{R\kappa_1},\ldots,$$
$$T_n = \mathfrak{g}^u\mathfrak{g}_0^{R\kappa_n} \ \wedge\ T_C = \hat{e}\left(B_1^t, Z\right) \wedge$$
$$\frac{\hat{e}(B_2,X)}{\hat{e}(g,h)\,\hat{e}(A_1,h_{l,I})} = \hat{e}\left(g_1^{b_1},X\right)\hat{e}\left(g_0^s g_1^{u+\beta_1} g_2^t u_{l,I}^{-a_1} u_I^{\phi_I(l)} B_2^{-e}, h\right)\Bigg](R).$$

Schritt 3: Der Händler $\mathcal{H}$ berechnet aus den Seriennummern $S_1,\ldots,S_n$ alle k Serienschlüssel sowie den Hashwert $l = \mathsf{H}(S_1||\ldots||S_n||T_1||\ldots||T_n||T_C||B_1||B_2)$, verifiziert SoK und speichert die Münze $\mathsf{coin} = (k, S_1,\ldots,S_n,T_1,\ldots,T_n, T_C, R, \Phi = (A_1,B_1,B_2,SoK,\mathsf{ts},\mathsf{pk}_{\mathcal{H}}))$ ab.

$\boxed{\mathcal{K}unde}$ $(\mathsf{pk}_{\mathcal{H}},\mathsf{sk}_{\mathcal{K}},\mathbb{W})$	$(\mathsf{sp},\mathsf{pk}_{\mathcal{B}},\mathsf{pk}_{\mathcal{D}},k,\mathsf{ts})$	$\boxed{\mathcal{H}ändler}$ $(\mathsf{sk}_{\mathcal{H}})$
$R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}}\|\|\mathsf{ts}\|\|k)$		$R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}}\|\|\mathsf{ts}\|\|k)$
$a_1,b_1,b_2 \in_{\mathcal{R}} \mathbb{Z}_p$		
$\kappa_i := \kappa_{L-\ell_i,j_i}$		
$S_1 = \mathfrak{g}^{\kappa_1},\ldots,S_n = \mathfrak{g}^{\kappa_n}$		
$T_1 = \mathfrak{g}^u\mathfrak{g}_0^{R\kappa_1},\ldots,T_n = \mathfrak{g}^u\mathfrak{g}_0^{R\kappa_n}$		
$B_1 = g_0^{b_1}g_1^{b_2}$		
$B_2 = \Sigma g_1^{b_1}$		
$T_C = \hat{e}(B_1^t, Z)$		
$l = \mathsf{H}(S_1\|\|\ldots\|\|S_n\|\|T_1\|\|\ldots\|\|T_n\|\|T_C\|\|B_1\|\|B_2)$		
$A_1 = W_{I,l}u_0^{a_1}$		
	$\xrightarrow{S_i,T_i,T_C,A_1,B_1,B_2}$	
$SoK\Big[(a_1,b_1,b_2,\beta_1,\beta_2,e,s,t,u,\kappa_1,\ldots,\kappa_n,\phi_I(l)) : B_1 = g_0^{b_1}g_1^{b_2} \ \wedge\ 1 = B_1^e g_0^{-\beta_1}g_1^{-\beta_2}$		
$\wedge\ S_1 = \mathfrak{g}^{\kappa_1},\ldots,S_n = \mathfrak{g}^{\kappa_n} \ \wedge\ T_1 = \mathfrak{g}^u\mathfrak{g}_0^{R\kappa_1},\ldots,T_n = \mathfrak{g}^u\mathfrak{g}_0^{R\kappa_n} \ \wedge\ T_C = \hat{e}(B_1^t,Z) \ \wedge$		
$\frac{\hat{e}(B_2,X)}{\hat{e}(g,h)\,\hat{e}(A_1,h_{l,I})} = \hat{e}\left(g_1^{b_1},X\right)\hat{e}\left(g_0^s g_1^{u+\beta_1} g_2^t u_{l,I}^{-a_1} u_I^{\phi_I(l)} B_2^{-e}, h\right)\Big](R)$		
		Berechne $\kappa_{L+1,0},\ldots,\kappa_{L+1,k-1}$
		$l = \mathsf{H}(S_1\|\|\ldots\|\|S_n\|\|T_1\|\|\ldots\|\|T_n\|\|B_1\|\|B_2)$
		Verifiziere SoK
$\mathbb{W}'$ oder $\perp$		coin oder $\perp$

Abbildung 8.1: Modifiziertes Bezahlprotokoll Spend von FTG-I

Spend FTG-II: Damit ein Kunde $\mathcal{K}$, der im Besitz der Geldbörse $\mathbb{W} = (\Sigma, e, s, t, \mathsf{info})$ ist, bei einem Händler $\mathcal{H}$ mit einer elektronischen Münze coin mit dem Wert $k = 2^{\ell_1} + \cdots + 2^{\ell_n} < K$ für $\ell_1 > \cdots > \ell_n$ bezahlen kann, führen $\mathcal{K}$ und $\mathcal{H}$ das Bezahlprotokoll Spend durch, welches in Abbildung 8.2 dargestellt ist. Zuvor authentifiziert sich $\mathcal{H}$ gegenüber $\mathcal{K}$ und beide gleichen den Zeitpunkt ts ab.

Schritt 1: Der Händler $\mathcal{H}$ und der Kunde $\mathcal{K}$ berechnen beide den Hashwert $R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}}||\mathsf{ts}||k)$.

Schritt 2: Der Kunde $\mathcal{K}$ teil den Wert k in $k = 2^{\ell_1} + \cdots + 2^{\ell_n}$ mit $\ell_1 > \cdots > \ell_n$ auf und wählt jeweils den gültigen Schlüssel $\kappa_{L-\ell_i,j_i} := \kappa_i \in \mathbb{K}_0$ mit dem kleinsten Wert j_i. Dementsprechend berechnet $\mathcal{K}$ die n Seriennummern $S_i = g^{\kappa_i}$ sowie die n Sicherheitstags $T_i = \Sigma g_0^{R\kappa_i}$ für $1 \leq i \leq n$. Dann berechnet $\mathcal{K}$ den Münz-Identifikator $T_C = S_1^t$ sowie den Hashwert $l = \mathsf{H}(S_1||\ldots||S_n||T_1||\ldots||T_n||T_C)$. Weiter berechnet $\mathcal{K}$ das Element $A_1 = W_{I,l}u_0^{a_1}$ wie gewohnt. Anschließend sendet $\mathcal{K}$ die Elemente $S_1, \ldots, S_n, T_1, \ldots, T_n, T_C, A_1$ an $\mathcal{H}$ und erstellt die Signature-of-Knowledge SoK. Bei einem naiven Ansatz würden in SoK die folgenden Gleichungen für $\beta_i = e\kappa_i$ für alle $1 \leq i \leq n$ sowie $u_I = u_0^{\prod_{j\in I}(\alpha+\kappa_{L+1,j})}, u_{l,I} = u_0^{(\alpha-l)\prod_{j\in I}(\alpha+\kappa_{L+1,j})}$ und $h_{l,I} = h^{(\alpha-l)\prod_{j\in I}(\alpha+\kappa_{L+1,j})}$ bewiesen werden:

$$S_i = g^{\kappa_i} \ \wedge \ 1 = S_i^e g^{-\beta_i} \ \wedge$$
$$\frac{\hat{e}(T_i, X)}{\hat{e}(g,h)\,\hat{e}(A_1, h_{l,I})} = \hat{e}\left(g_0^{R\kappa_i}, X\right)\hat{e}\left(g_0^{s+R\beta_i} g_1^t u_{l,I}^{-a_1} u_I^{\phi_I(l)} T_i^{-e}, h\right).$$

Da allerdings die Gleichungen $T_1 = \Sigma g_0^{R\kappa_1}, \ldots, T_n = \Sigma g_0^{R\kappa_n}$ gelten, ist es ausreichend, die Relationen

$$S_i = g^{\kappa_i} \ \wedge \ \frac{T_i}{T_1} = \frac{g_0^{R\kappa_i}}{g_0^{R\kappa_1}}$$

für alle $2 \leq i \leq n$ zu beweisen.

Daher wird SoK folgendermaßen für $\beta = e\kappa_1$ erstellt:

$$SoK\Bigg[(a_1, \beta, e, s, t, \kappa_1, \ldots, \kappa_n, \phi_I(l)) : S_1 = g^{\kappa_1}, \ldots, S_n = g^{\kappa_n} \ \wedge$$
$$1 = S_1^e g^{-\beta} \ \wedge \ T_2/T_1 = \left(g_0^R\right)^{\kappa_2-\kappa_1}, \ldots, T_n/T_1 = \left(g_0^R\right)^{\kappa_n-\kappa_1} \ \wedge \ T_C = S_1^t$$
$$\wedge \ \frac{\hat{e}(T_1, X)}{\hat{e}(g,h)\,\hat{e}(A_1, h_{l,I})} = \hat{e}\left(g_0^{R\kappa_1}, X\right)\hat{e}\left(g_0^{s+R\beta} g_1^t u_{l,I}^{-a_1} u_I^{\phi_I(l)} T_1^{-e}, h\right)\Bigg](R).$$

Schritt 3: Der Händler $\mathcal{H}$ berechnet aus den Seriennummern $S_1, \ldots, S_n$ alle k Serienschlüssel sowie den Hashwert $l = \mathsf{H}(S_1||\ldots||S_n||T_1||\ldots||T_n||T_C)$, verifiziert SoK und speichert die Münze $\mathsf{coin} = (k, S_1, \ldots, S_n, T_1, \ldots, T_n, T_C, R, \Phi = (A_1, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$ ab.

$\boxed{\mathcal{K}unde}$ $(\mathsf{pk}_{\mathcal{H}}, \mathsf{sk}_{\mathcal{K}}, \mathbb{W})$	$(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{D}}, k, \mathsf{ts})$	$\boxed{\mathcal{H}ändler}$ $(\mathsf{sk}_{\mathcal{H}})$
$R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}}\|\|\mathsf{ts}\|\|k)$		$R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}}\|\|\mathsf{ts}\|\|k)$
$a_1 \in_{\mathcal{R}} \mathbb{Z}_p$		
$\kappa_i := \kappa_{L-\ell_i,j_i}$		
$S_1 = g^{\kappa_1}, \ldots, S_n = g^{\kappa_n}$		
$T_1 = \Sigma g_0^{R\kappa_1}, \ldots, T_n = \Sigma g_0^{R\kappa_n}$		
$T_C = S_1^t$		
$l = \mathsf{H}(S_1\|\|\ldots\|\|S_n\|\|T_1\|\|\ldots\|\|T_n\|\|T_C)$		
$A_1 = W_{I,l} u_0^{a_1}$		
	$\xrightarrow{S_i, T_i, T_C, A_1}$	
$SoK\Big[(a_1, \beta, e, s, t, \kappa_1, \ldots, \kappa_n, \phi_I(l)) : S_1 = g^{\kappa_1}, \ldots, S_n = g^{\kappa_n} \ \wedge\ 1 = S_1^e g^{-\beta} \ \wedge$		
$T_2/T_1 = \left(g_0^R\right)^{\kappa_2-\kappa_1}, \ldots, T_n/T_1 = \left(g_0^R\right)^{\kappa_n-\kappa_1} \ \wedge\ T_C = S_1^t \ \wedge$		
$\frac{\hat{e}(T_1, X)}{\hat{e}(g,h)\,\hat{e}(A_1, h_{l,I})} = \hat{e}\left(g_0^{R\kappa_1}, X\right)\hat{e}\left(g_0^{s+R\beta} g_1^t u_{l,I}^{-a_1} u_I^{\phi_I(l)} T_1^{-e}, h\right)\Big](R)$		
		Berechne $\kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}$
		$l = \mathsf{H}(S_1\|\|\ldots\|\|S_n\|\|T_1\|\|\ldots\|\|T_n\|\|T_C)$
		Verifiziere SoK
$\mathbb{W}'$ oder $\perp$		coin oder $\perp$

Abbildung 8.2: Modifiziertes Bezahlprotokoll Spend von FTG-II

Identify: Seien $\mathsf{coin} = \left(k = 2^{\ell_1} + \cdots + 2^{\ell_n}, S_1, \ldots, S_n, T_1, \ldots, T_n, T_C, R, \Phi\right)$ und $\mathsf{coin}' = \left(k' = 2^{\ell_1} + \cdots + 2^{\ell_{n'}}, S_1', \ldots, S_{n'}', T_1', \ldots, T_{n'}', T_C', R', \Phi'\right)$ die zwei eingelösten Münzen. Seien o.B.d.A. S_i und $S_{i'}'$ mit $1 \leq i \leq n$ und $1 \leq i' \leq n'$ die beiden Seriennummern, bei denen ein Double-Spending auftrat, d.h. aus denen mindestens ein gleicher Serienschlüssel berechnet wird. Dann werden entsprechend zur Berechnung der Kontonummer $I \in \mathbb{G}_p$ bei FTG-I bzw. des Elements $\Sigma \in \mathbb{G}_1$ bei FTG-II die Tupel $\left(k_i := 2^{\ell_i}, S_i, T_i, R\right)$ sowie $\left(k_{i'}' := 2^{\ell_{i'}}, S_{i'}', T_{i'}', R'\right)$ verwendet. Analog zum fairen teilbaren Geldsystem FTG-I bzw. FTG-II müssen dann die beiden Fälle $k_i = k_{i'}'$ und $k_i \neq k_{i'}'$ unterschieden und die entsprechenden Berechnungen ausgeführt werden.

In der folgenden Tabelle 8.1 werden die modifizierten Bezahlprotokolle mit den ursprünglichen Bezahlprotokollen hinsichtlich der Effizienz verglichen, wobei p erneut eine 256-Bit Primzahl ist und exemplarisch $k = 124 = 2^6 + 2^5 + 2^4 + 2^3 + 2^2$ mit $n = 5$ gewählt wird.

FTG-I	ursprünglich	modifiziert
Spend	$\mathbb{G}_p^{2n} \times \mathbb{G}_1^{3n} \times \mathbb{G}_T^n \times \mathbb{Z}_p^{12n}$	$\mathbb{G}_p^{2n} \times \mathbb{G}_1^3 \times \mathbb{G}_T \times \mathbb{Z}_p^{n+11}$
Bytes	$677n$	$579 + 98n$
$n = 5$	3.385	1.069
Kunde	$13n$ ME $+ 4n$ P	$4n + 9$ ME $+ 4$ P
Händler	$2k + 6n$ ME $+ 4n$ P	$2k + 6$ ME $+ 4$ P
Deposit	$\mathbb{G}_p^{2n} \times \mathbb{G}_1^{3n} \times \mathbb{G}_T^n \times \mathbb{Z}_p^{13n}$	$\mathbb{G}_p^{2n} \times \mathbb{G}_1^3 \times \mathbb{G}_T \times \mathbb{Z}_p^{n+12}$
Bytes	$709n$	$611 + 98n$
$n = 5$	3.545	1.101
Bank	$2k + 6n$ ME $+ 4n$ P	$2k + 6$ ME $+ 4$ P
$\mathsf{coin} \in \mathbb{D}_C$	$\mathbb{G}_p^{3n} \times \mathbb{G}_1^{3n} \times \mathbb{G}_T^n \times \mathbb{Z}_p^{k+13n}$	$\mathbb{G}_p^{2n+1} \times \mathbb{G}_1^3 \times \mathbb{G}_T \times \mathbb{Z}_p^{k+n+12}$
Bytes	$742n + 32k$	$644 + 98n + 32k$
$n = 5, k = 124$	7.678	5.102

FTG-II	ursprünglich	modifiziert
Spend	$\mathbb{G}_1^{4n} \times \mathbb{Z}_p^{8n}$	$\mathbb{G}_1^{2n+2} \times \mathbb{Z}_p^{n+7}$
Bytes	$388n$	$290 + 98n$
$n = 5$	1.940	780
Kunde	$9n$ ME $+ 2n$ P	$4n + 5$ ME $+ 2$ P
Händler	$2k + 4n$ ME $+ 3n$ P	$2k + 4$ ME $+ 3$ P
Deposit	$\mathbb{G}_1^{4n} \times \mathbb{Z}_p^{9n}$	$\mathbb{G}_1^{2n+2} \times \mathbb{Z}_p^{n+8}$
Bytes	$420n$	$322 + 98n$
$n = 5$	2.100	812
Bank	$2k + 4n$ ME $+ 3n$ P	$2k + 4$ ME $+ 3$ P
$\mathsf{coin} \in \mathbb{D}_C$	$\mathbb{G}_1^{5n} \times \mathbb{Z}_p^{k+9n}$	$\mathbb{G}_1^{2n+3} \times \mathbb{Z}_p^{k+n+8}$
Bytes	$453n + 32k$	$355 + 98n + 32k$
$n = 5, k = 124$	6.233	4.813

Tabelle 8.1: Effizienzvergleich der ursprünglichen und modifizierten Bezahlprotokolle

8.1.1 Sicherheitsanalyse

Satz 8.1.1. *Die oben beschriebenen Modifizierungen haben keine Auswirkungen auf die Sicherheit der fairen teilbaren elektronischen Geldsysteme FTG-I und FTG-II.*

Bemerkung 8.1.1. *Analog haben die Modifizierungen keine Auswirkungen auf die Sicherheit der teilbaren elektronischen Geldsysteme TG-I und TG-II.*

Beweis. Die Sicherheitseigenschaften **Unfälschbarkeit**, **Double-Spending-Beschuldigung**, **Abhebe-Beschuldigung**, **korrekte Kundenverfolgung**, **Kundenverfol-**

gung-Beschuldigung, **korrekte Münzverfolgung** und **Münzverfolgung-Beschuldigung** bleiben analog zum fairen teilbaren Geldsystem FTG-I bzw. FTG-II erhalten. Denn jede Münze $\mathsf{coin} = \left(k = 2^{\ell_1} + \cdots + 2^{\ell_n}, S_1, \ldots, S_n, T_1, \ldots, T_n, T_C, R, \Phi\right)$ kann in die Tupel $(2^{\ell_1}, S_1, T_1, T_C, R, \Phi), \ldots, (2^{\ell_n}, S_n, T_n, T_C, R, \Phi)$ unterteilt werden. Somit folgen jeweils dieselben Eigenschaften wie im fairen teilbaren Geldsystem FTG-I bzw. FTG-II bzgl. der Münze

Daher muss nur die Anonymität bewiesen werden.

Anonymität: Die Anonymität kann mit einer Veränderung in der Challenge-Phase analog zu der Anonymität des fairen teilbaren Geldsystems FTG-I bzw. FTG-II bewiesen werden. Ansonsten geht $\mathcal{B}$ wie beim Geldsystem FTG-I bzw. FTG-II beschrieben vor.

Simulation der Challenge-Phase: Der Angreifer $\mathcal{A}$ sendet zwei verschiedene Indizes i_0, i_1, zwei öffentliche Schlüssel $\mathsf{pk}_b, \mathsf{pk}_{\bar{b}}$, zwei Zeitpunkte $\mathsf{ts}_b, \mathsf{ts}_{\bar{b}}$ (wobei sogar $(\mathsf{pk}_b, \mathsf{ts}_b) = (\mathsf{pk}_{\bar{b}}, \mathsf{ts}_{\bar{b}})$ erlaubt ist) und zwei gültige Werte $k_b, k_{\bar{b}}$ mit $\$_{i_0}, \$_{i_1} \geq \max\{k_b, k_{\bar{b}}\}$ (wobei auch $k_b = k_{\bar{b}}$ erlaubt ist) für $k_b = 2^{\ell_{1,b}} + \cdots + 2^{\ell_{n_b,b}}$ mit $\ell_{1,b} > \cdots > \ell_{n_b,b}$ sowie $k_{\bar{b}} = 2^{\ell_{1,\bar{b}}} + \cdots + 2^{\ell_{n_{\bar{b}},\bar{b}}}$ mit $\ell_{1,\bar{b}} > \cdots > \ell_{n_{\bar{b}},\bar{b}}$.

Der Angreifer $\mathcal{B}$ geht wie beim Geldsystem FTG-I beschrieben vor und erhält die beiden Geheimtexte $c_b = (c_{1,b}, c_{2,b}) = (I_b \mathfrak{g}_0^{r_b}, \mathfrak{g}^{r_b})$ sowie $c_{\bar{b}} = \left(c_{1,\bar{b}}, c_{2,\bar{b}}\right) = \left(I_{\bar{b}} \mathfrak{g}_0^{r_{\bar{b}}}, \mathfrak{g}^{r_{\bar{b}}}\right)$ mit $r_0, r_1 \in_{\mathcal{R}} \mathbb{Z}_p$ und $b, \bar{b} \in \{0,1\}$ mit $b \neq \bar{b}$.

Der Angreifer $\mathcal{B}$ wählt zufällig $n_b + n_{\bar{b}}$ Zahlen $\delta_{1,b}, \ldots, \delta_{n_b,b}, \delta_{1,\bar{b}}, \ldots, \delta_{n_{\bar{b}},\bar{b}} \in_{\mathcal{R}} \mathbb{Z}_p^*$. Dann werden die Seriennummern $S_{i,b} := \left(c_{2,b} \cdot \mathfrak{g}^{\delta_{i,b}}\right)^{R_b^{-1}}$ sowie $S_{j,\bar{b}} := \left(c_{2,\bar{b}} \cdot \mathfrak{g}^{\delta_{j,\bar{b}}}\right)^{R_{\bar{b}}^{-1}}$ und die Sicherheitstags $T_{i,b} := c_{1,b} \cdot \mathfrak{g}_0^{\delta_{i,b}}$ sowie $T_{j,\bar{b}} := c_{1,\bar{b}} \cdot \mathfrak{g}_0^{\delta_{j,\bar{b}}}$ für $1 \leq i \leq n_b$ und $1 \leq j \leq n_{\bar{b}}$ definiert.

Seien $\kappa_{i,b} := (r_b + \delta_{i,b}) \cdot R_b^{-1} \mod p$ und $\kappa_{j,\bar{b}} := \left(r_{\bar{b}} + \delta_{j,\bar{b}}\right) \cdot R_{\bar{b}}^{-1} \mod p$, dann sind alle Seriennummern $S_{i,b} = \mathfrak{g}^{\kappa_{i,b}}$ sowie $S_{j,\bar{b}} = \mathfrak{g}^{\kappa_{j,\bar{b}}}$ mit $\kappa_{i,b}, \kappa_{j,\bar{b}} \in_{\mathcal{R}} \mathbb{Z}_p^*$ für $1 \leq i \leq n_b$ und $1 \leq j \leq n_{\bar{b}}$ im Random-Oracle-Modell analog zu FTG-I perfekt simuliert. Somit sind auch alle Sicherheitstags $T_{i,b} := I_b \mathfrak{g}_0^{\kappa_{i,b} R_b}$ sowie $T_{j,\bar{b}} := I_{\bar{b}} \mathfrak{g}_0^{\kappa_{j,\bar{b}} R_{\bar{b}}}$ für $1 \leq i \leq n_b$ und $1 \leq j \leq n_{\bar{b}}$ perfekt simuliert.

Ansonsten geht $\mathcal{B}$ wie beim Geldsystem FTG-I vor und sendet schließlich die beiden rechnerisch ununterscheidbar simulierten Münzen $\mathsf{coin}_b = (k_b, S_{1,b}, \ldots, S_{n_b,b}, T_{1,b}, \ldots, T_{n_b,b}, T_{C,b}, R_b, A_{1,b}, B_{1,b}, B_{2,b}, SoK_b, \mathsf{ts}_b, \mathsf{pk}_b)$ und $\mathsf{coin}_{\bar{b}} = \left(k_{\bar{b}}, S_{1,\bar{b}}, \ldots, S_{n_{\bar{b}},\bar{b}}, T_{1,\bar{b}}, \ldots, T_{n_{\bar{b}},\bar{b}}, T_{C,\bar{b}}, R_{\bar{b}}, A_{1,\bar{b}}, B_{1,\bar{b}}, B_{2,\bar{b}}, SoK_{\bar{b}}, \mathsf{ts}_{\bar{b}}, \mathsf{pk}_{\bar{b}}\right)$ an den Angreifer $\mathcal{A}$.

Der Beweis für das Geldsystem FTG-II verläuft analog.

□

8.2 Geldbörsen mit beliebigem Geldbetrag

In diesem Abschnitt wird das Abhebeprotokoll Withdraw modifiziert, so dass jeder Kunde einen beliebigen Geldbetrag $K' \leq K$ abheben kann. Die Grundidee ist, dass während des Abhebeprotokolls bereits k Serienschlüssel wieder bei der Bank „eingelöst“ werden, um eine Geldbörse mit dem Wert $K' = K - k$ abzuheben. Der Kunde $\mathcal{K}$ verhält sich zunächst so, als würde er eine Geldbörse vom Wert K abheben. Dann teilt $\mathcal{K}$ den Wert k wie in Abschnitt 8.1 in $k = 2^{\ell_1} + \cdots + 2^{\ell_n}$ mit $\ell_1 > \cdots > \ell_n$ auf und sendet die entsprechenden n Schlüssel $\kappa_{L-\ell_1,j_1}, \ldots, \kappa_{L-\ell_n,j_n} \in \mathbb{K}_0$ an die Bank. Diese n Schlüssel bzw. die entsprechenden k Serienschlüssel gelten dann als bezahlt und eingelöst.

Somit bleiben die Größe der öffentlichen Parameter und der Berechnungsaufwand des Kunden unverändert, während die Bank zusätzlich die k Serienschlüssel aus den n Schlüsseln berechnen muss. Will ein Kunde bspw. eine Geldbörse vom Wert $K' = K/2$ abheben, müssen zusätzlich lediglich der Schlüssel $\kappa_{1,0}$ gesendet und die daraus resultierenden Serienschlüssel von der Bank berechnet werden. Insgesamt betragen für das Abhebeprotokoll die Komplexität des Datentransfers und die Berechnungskomplexität der Bank nun allerdings $O(\lambda \cdot \log(K))$.

Obwohl die wesentliche Veränderung nur das Protokoll Withdraw betrifft, müssen zudem die Algorithmen Identify und VerifyGuilt angepasst werden. Die Algorithmen Setup, DGen und BGen sowie die Protokolle Account, Spend, Spend-K und Deposit werden wie beim (modifizierten) fairen teilbaren Geldsystem FTG-I bzw. FTG-II durchgeführt. Der Algorithmus VerifyWithdraw und die Protokolle UTrace sowie CTrace werden dann analog zu FTG-I bzw. FTG-II durchgeführt und aus diesem Grund nicht beschrieben.

Withdraw: Damit ein Kunde $\mathcal{K}$ eine Geldbörse mit dem Wert $K' = K - k$ bei der Bank $\mathcal{B}$ abheben kann, führen $\mathcal{K}$ und $\mathcal{B}$ das Abhebeprotokoll Withdraw durch, welches (für FTG-I) in Abbildung 8.3 dargestellt ist. Zuvor authentifizieren sich jedoch beide Parteien gegenseitig und gleichen den aktuellen Zeitpunkt ts ab.

Schritt 1: Der Kunde $\mathcal{K}$ geht zunächst wie im fairen teilbaren Geldsystem FTG-I bzw. FTG-II vor und erzeugt die Listen $\mathbb{K}_0, \mathbb{K}_1$ sowie das Commitment C und das Element E bzw. die Verschlüsselung (E_1, E_2). Dann teilt der Kunde den Wert k in $k = 2^{\ell_1} + \cdots + 2^{\ell_n}$ mit $\ell_1 > \cdots > \ell_n$ auf, wählt jeweils den Schlüssel $\kappa_{L-\ell_i,j_i} := \kappa_i \in \mathbb{K}_0$ mit dem kleinsten Wert j_i und sendet $\kappa_1, \ldots, \kappa_n$ ebenfalls an die Bank. Der Kunde $\mathcal{K}$ berechnet den Hashwert $l = \mathsf{H}(C||\kappa_1|| \ldots ||\kappa_n)$ und das Element $A_0 = W_{I,l} u_0^{a_0}$ wie gewohnt. Anschließend erstellt $\mathcal{K}$ für FTG-I die folgende Signature-of-Knowledge SoK_{W}, wobei $u_I = u_0^{\prod_{j \in I}(\alpha + \kappa_{L+1,j})}$, $u_{l,I} = u_0^{(\alpha-l)\prod_{j \in I}(\alpha+\kappa_{L+1,j})}$ und $h_{l,I} = h^{(\alpha-l)\prod_{j \in I}(\alpha+\kappa_{L+1,j})}$ ist:

$$SoK_{\mathsf{W}}\Big[(a_0, s', t', u, \phi_I(l)) : I = \mathfrak{g}^u \ \wedge \ E = h^{t'} \ \wedge \ \hat{e}(C,h)\,\hat{e}(A_0, h_{l,I})^{-1} = \hat{e}\left(g_0^{s'} g_1^u g_2^{t'} u_I^{\phi_I(l)} u_{l,I}^{-a_0}, h\right)\Big](\mathsf{ts}||K').$$

Die Signature-of-Knowledge für das Geldsystem FTG-II wird dementsprechend analog erstellt.

Schritt 2: Die Bank $\mathcal{B}$ berechnet aus den n Schlüsseln $\kappa_1, \ldots, \kappa_n$ die entsprechenden k Serienschlüssel. Falls mindestens einer dieser Serienschlüssel bereits in der Datenbank $\mathbb{D}_W$ gespeichert ist, bricht die Bank ab und sendet $\perp$ an $\mathcal{K}$. Andernfalls geht $\mathcal{B}$ analog zu FTG-I bzw. FTG-II vor, wobei die n Schlüssel $\kappa_1, \ldots, \kappa_n$ sowie die entsprechenden k Serienschlüssel (diese seien o.B.d.A.) $\kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}$ ebenfalls in $\mathbb{T}$ gespeichert werden.

Schritt 3: Der Kunde $\mathcal{K}$ geht wie in FTG-I bzw. FTG-II vor.

$\boxed{\mathcal{K}unde}$ $(\mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{K})$	$(\mathsf{sp}, K', \mathsf{ts})$	$\boxed{\mathcal{B}ank}$ $(\mathsf{sk}_\mathcal{B}, \mathsf{pk}_\mathcal{K})$
$\kappa_{0,0} \in_\mathcal{R} \mathbb{Z}_p^*, s', t', a_0 \in_\mathcal{R} \mathbb{Z}_p$ Generiere $\mathbb{K}_0, \mathbb{K}_1$ $\phi(\mathsf{x}) = \prod_{j=0}^{K-1} (\mathsf{x} + \kappa_{L+1,j})$ $C = g_0^{s'} g_1^u g_2^{t'} u_0^{\phi(\alpha)}$ $E = h^{t'}$ $\kappa_i := \kappa_{L-\ell_i, j_i}$ $l = \mathsf{H}(C \Vert \kappa_1 \Vert \ldots \Vert \kappa_n)$ $A_0 = W_{I,l} u_0^{a_0}$		
	$\xrightarrow{C, E, \kappa_1, \ldots, \kappa_n, A_0}$	
$SoK_\mathsf{W}\Big[(a_0, s', u, t', \phi_I(l)) : I = \mathfrak{g}^u \ \wedge \ E = h^{t'} \ \wedge$ $\hat{e}(C, h)\, \hat{e}(A_0, h_{l,I})^{-1} = \hat{e}\left(g_0^{s'} g_1^u g_2^{t'} u_I^{\phi_I(l)} u_{l,I}^{-a_0}, h\right)\Big] (\mathsf{ts} \Vert K')$		
		Berechne $\kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}$ $\kappa_{L+1,j} \notin \mathbb{D}_W$? $l = \mathsf{H}(C \Vert \kappa_1 \Vert \ldots \Vert \kappa_n)$ Verifiziere SoK_W $e, s'', t'' \in_\mathcal{R} \mathbb{Z}_p$ $\Sigma = \left(g g_0^{s''} g_2^{t''} C\right)^{\frac{1}{x+e}}$
	$\xleftarrow{\Sigma, e, s'', t''}$	
$s = s' + s'' \mod p$ $t = t' + t'' \mod p$ $\hat{e}(\Sigma, X h^e) \stackrel{?}{=} \hat{e}\left(g g_0^{s''} g_2^{t''} C, h\right)$		
$\mathbb{W} = (\Sigma, e, s, t, \mathsf{info})$ oder $\perp$		view oder $\perp$

Abbildung 8.3: Modifiziertes Abhebeprotokoll Withdraw von FTG-I

Bemerkung 8.2.1. *Falls mindestens einer der k Serienschlüssel bereits in der Datenbank $\mathbb{D}_W$ gespeichert ist, muss die Bank das Protokoll abbrechen, da andernfalls die Sicherheitseigenschaft Unfälschbarkeit nicht erfüllt wäre. Denn würden zwei Kunden bspw. den gleichen Serienschlüssel $\kappa_{L+1,0}$ während des Abhebeprotokolls an die Bank senden und einer der beiden Kunden diesen Serienschlüssel dennoch zum Bezahlen verwendet (was einem Double-Spending entspricht), kann im folgenden Fall (2.2) nicht festgestellt werden, welcher der beiden Kunden der Double-Spender ist.*

Identify: Wir müssen nun den zusätzlichen Fall (2) betrachten, dass ein Double-Spending durch ein Abhebe- und ein Einlöseprotokoll auftritt.

FTG-I (2): Seien $\mathbb{T} = (I, C, E, A_0, SoK_{\mathsf{W}}, \kappa_1, \ldots, \kappa_n, \kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}, \mathsf{ts}, K')$ eine Abhebeinformation und $\mathsf{coin} = (k', S_1, \ldots, S_{n'}, T_1, \ldots, T_{n'}, T_C, R, \Phi)$ eine eingelöste Münze, die (mindestens) einen Serienschlüssel aus $\{\kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}\}$ enthält. Seien κ_i und $S_{i'}$ mit $1 \leq i \leq n$ und $1 \leq i' \leq n'$ die Werte, aus denen mindestens ein identischer Serienschlüssel aus $\{\kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}\}$ berechnet wird und seien k_i bzw. $k'_{i'}$ die entsprechenden Geldbeträge (d.h. aus κ_i werden k_i und aus $S_{i'}$ werden $k'_{i'}$ Serienschlüssel berechnet).

Im Fall (2.1) $k_i \geq k'_{i'}$ kann aus dem Schlüssel κ_i der zu $S_{i'}$ gehörige Schlüssel $\kappa'_{i'}$ mit $S_{i'} = \mathfrak{g}^{\kappa'_{i'}}$ und $T_{i'} = I\mathfrak{g}_0^{R\kappa'_{i'}}$ berechnet werden (für $k_i = k'_{i'}$ ist $\kappa'_{i'} = \kappa_i$). In diesem Fall wird die Gleichung $T_{i'}/\mathfrak{g}_0^{R\kappa'_{i'}} \stackrel{?}{=} I$ verifiziert und entweder der gespeicherte öffentliche Schlüssel I mit einem Beweis $\Pi_G = (\mathbb{T}, \mathsf{coin})$ oder das Symbol $\perp$ ausgegeben.

Im Fall (2.2) $k_i < k'_{i'}$ ist die Ausgabe der gespeicherte öffentliche Schlüssel I mit einem Beweis $\Pi_G = (\mathbb{T}, \mathsf{coin})$.

FTG-II (2): Seien analog $\mathsf{view} = (\mathbb{T} = (I, C, E_1, E_2, A_0, SoK_{\mathsf{W}}, \kappa_1, \ldots, \kappa_n, \kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}, \mathsf{ts}, K', \sigma' = (\Sigma, e, s'', t''))$ eine Protokollansicht und $\mathsf{coin} = (k', S_1, \ldots, S_{n'}, T_1, \ldots, T_{n'}, T_C, R, \Phi)$ eine eingelöste Münze, die (mindestens) einen Serienschlüssel aus $\{\kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}\}$ enthält. Seien κ_i und $S_{i'}$ mit $1 \leq i \leq n$ und $1 \leq i' \leq n'$ die Werte, aus denen mindestens ein identischer Serienschlüssel aus $\{\kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}\}$ berechnet wird und seien k_i bzw. $k'_{i'}$ die entsprechenden Geldbeträge.

Im Fall (2.1) $k_i \geq k'_{i'}$ kann aus dem Schlüssel κ_i der zu $S_{i'}$ gehörige Schlüssel $\kappa'_{i'}$ mit $S_{i'} = g^{\kappa'_{i'}}$ und $T_{i'} = \Sigma g_0^{R\kappa'_{i'}}$ berechnet werden (für $k_i = k'_{i'}$ ist $\kappa'_{i'} = \kappa_i$). In diesem Fall wird die Gleichung $T_{i'}/g_0^{R\kappa'_{i'}} \stackrel{?}{=} \Sigma$ verifiziert und entweder der gespeicherte öffentliche Schlüssel I mit einem Beweis $\Pi_G = (\mathsf{view}, \mathsf{coin})$ oder das Symbol $\perp$ ausgegeben.

Im Fall (2.2) $k_i < k'_{i'}$ ist die Ausgabe der gespeicherte öffentliche Schlüssel I mit einem Beweis $\Pi_G = (\mathsf{view}, \mathsf{coin})$.

VerifyGuilt: Wir müssen nun analog zum Identify-Algorithmus den zusätzlichen Fall (2) betrachten.

FTG-I (2): Der Algorithmus verifiziert bei Eingabe der Systemparameter sp, eines öffentlichen Schlüssels $\mathsf{pk}_{\mathcal{K}}$, einer Abhebeinformation $\mathbb{T} = (I, C, E, A_0, SoK_{\mathsf{W}}, \kappa_1, \ldots, \kappa_n, \kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}, \mathsf{ts}, K')$ und einer Münze $\mathsf{coin} = (k', S_1, \ldots, S_{n'}, T_1, \ldots, T_{n'}, T_C, R, \Phi = (A_1, B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$ die zwei Signatures-of-Knowledge SoK_{W} und SoK und führt den Identify-Algorithmus aus.

FTG-II (2): Der Algorithmus verifiziert bei Eingabe der Systemparameter sp, eines öffentlichen Schlüssels $\mathsf{pk}_{\mathcal{K}}$, einer Protokollansicht $\mathsf{view} = (\mathbb{T} = (I, C, E_1, E_2, A_0, SoK_{\mathsf{W}}, \kappa_1, \ldots, \kappa_n, \kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}, \mathsf{ts}, K'), \sigma' = (\Sigma, e, s'', t'')$ und einer Münze $\mathsf{coin} = (k', S_1, \ldots, S_{n'}, T_1, \ldots, T_{n'}, T_C, R, \Phi = (A_1, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$ die beiden Signatures-of-Knowledge SoK_{W} und SoK sowie die Signatur und führt den Identify-Algorithmus aus.

Somit kann jeder verifizieren, dass der Kunde $\mathcal{K}$ mit Kontonummer $\mathsf{pk}_{\mathcal{K}} = I$ einen Serienschlüssel mehrfach ausgegeben hat.

Bemerkung 8.2.2. *Es ist zu beachten, dass bei dieser Modifizierung eine Münze mit dem Wert $k \leq K$ nur von den Geldbörsen stammen kann, bei denen ein Betrag $K' \geq k$ abgehoben wurde. Somit kann die Bank immer nachvollziehen, dass eine Münze mit dem Wert k nur zu einer Geldbörse vom Wert $K' \geq k$ gehören kann. Diese Eigenschaft ist allerdings unvermeidbar und beeinflusst die Anonymität gemäß Definition 4.4.2 nicht, da gefordert wird, dass beide Münzwerte k_b und $k_{\bar{b}}$ von beiden Geldbörsen $\mathbb{W}_0$ und $\mathbb{W}_1$ erzeugt werden können.*

8.2.1 Sicherheitsanalyse

Satz 8.2.1. *Die oben beschriebenen Modifizierungen haben keine Auswirkungen auf die Sicherheit der fairen teilbaren elektronischen Geldsysteme FTG-I und FTG-II.*

Bemerkung 8.2.3. *Analog haben die Modifizierungen keine Auswirkungen auf die Sicherheit der teilbaren elektronischen Geldsysteme TG-I und TG-II.*

Beweis. Die Sicherheitsziele **Anonymität**, **Abhebe-Beschuldigung**, **korrekte Kundenverfolgung**, **Kundenverfolgung-Beschuldigung**, **korrekte Münzverfolgung** und **Münzverfolgung-Beschuldigung** bleiben analog zum fairen teilbaren Geldsystem FTG-I bzw. FTG-II erhalten.

Somit müssen nur für die Unfälschbarkeit und Double-Spending-Beschuldigung weitere Fälle betrachtet werden.

Unfälschbarkeit: Die Fälle A bis F sowie (1.1), (1.2), (2.1) und (2.2) sind analog zum fairen teilbaren Geldsystem FTG-I bzw. FTG-II vernachlässigbar. Da ein Double-Spending nun allerdings auch durch eine Abhebeinformation $\mathbb{T} = (I, C, E, A_0, SoK_{\mathsf{W}}, \kappa_1,$

$\ldots, \kappa_n, \kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}, \mathsf{ts}, K')$ beim Geldsystem FTG-I bzw. eine Protokollansicht $\mathsf{view} = (\mathbb{T} = (I, C, E_1, E_2, A_0, SoK_{\mathsf{W}}, \kappa_1, \ldots, \kappa_n, \kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}, \mathsf{ts}, K'), \sigma' = (\Sigma', e', s'', t''))$ beim Geldsystem FTG-II und eine Münze $\mathsf{coin} = (k', S_1, \ldots, S_{n'}, T_1, \ldots, T_{n'}, T_C, R, \Phi)$ entstehen kann, müssen weitere Fälle betrachtet werden. Da durch den Identify-Algorithmus kein Kunde $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_{\mathcal{A}}$ identifiziert wird, ist die Ausgabe entweder $I \in \mathcal{U}_H$ oder $\perp$. Durch die Fälle A bis F wurde bereits gezeigt, dass nur mit vernachlässigbarer Wahrscheinlichkeit $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ gilt. Für $I \in \mathcal{U}_H$ folgt $(\cdot, \mathsf{view} = (\mathbb{T}, \cdot)) \in \mathsf{view}_H$ und für $I \in \mathcal{U}_{\mathcal{A}}$ folgt $(\cdot, \mathsf{view} = (\mathbb{T}, \cdot)) \in \mathsf{view}_{\mathcal{A}}$. Somit gibt es die beiden Fälle, dass I die Kontonummer eines (2.3) ehrlichen oder eines (2.4) korrupten Kunden ist.

Seien κ_i und $S_{i'}$ mit $1 \leq i \leq n$ und $1 \leq i' \leq n'$ die Werte, aus denen mindestens ein identischer Serienschlüssel aus $\{\kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}\}$ berechnet wird und seien k_i bzw. $k'_{i'}$ die entsprechenden Geldbeträge.

Fall (2.3): $I \in \mathcal{U}_H$, $(\cdot, \mathsf{view} = (\mathbb{T}, \cdot)) \in \mathsf{view}_H$ und $(\cdot, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$:

In diesem Fall wurde die Signature-of-Knowledge SoK der Münze coin vom Angreifer $\mathcal{A}$ erstellt. Daher kann $\mathcal{B}$ die Zahl $\kappa'_{i'} \in \mathbb{Z}_p$ sowie

FTG-I: das Nachrichten-Signatur-Paar $((u, t, \phi(\mathsf{x})), (\Sigma, e, s))$ mit $S_{i'} = \mathfrak{g}^{\kappa'_{i'}}$ und $T_{i'} = \mathfrak{g}^u \mathfrak{g}_0^{R\kappa'_{i'}}$

FTG-II: das Nachrichten-Signatur-Paar $((t, \phi(\mathsf{x})), (\Sigma, e, s))$ mit $S_{i'} = g^{\kappa'_{i'}}$ und $T_{i'} = \Sigma g_0^{R\kappa'_{i'}}$

aus SoK extrahieren. Da $(\cdot, \mathsf{view} = (\mathbb{T}, \cdot)) \in \mathsf{view}_H$ ist, gilt insbesondere $(\cdot, \mathsf{view} = (\mathbb{T}, \sigma')) \notin \mathsf{view}_{\mathcal{A}}$ und somit wurde das Nachrichten-Signatur-Paar niemals bei einer $\mathcal{O}_{\mathcal{B}}^{\mathsf{W}}$-Anfrage erstellt.

- Falls das Nachrichten-Signatur-Paar niemals bei einer Orakel-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.1) vor.
- Falls das Nachrichten-Signatur-Paar bei einer $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{W}}$-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.2) vor.

Fall (2.4): $I \in \mathcal{U}_{\mathcal{A}}$, $(\cdot, \mathsf{view} = (\mathbb{T}, \cdot)) \in \mathsf{view}_{\mathcal{A}}$ und $(\cdot, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$:

Auch in diesem Fall wurde die Signature-of-Knowledge SoK der Münze coin vom Angreifer $\mathcal{A}$ erstellt. Daher kann $\mathcal{B}$ die Zahl $\kappa'_{i'} \in \mathbb{Z}_p$

FTG-I: das Nachrichten-Signatur-Paar $((u, t, \phi(\mathsf{x})), (\Sigma, e, s))$ mit $S_{i'} = \mathfrak{g}^{\kappa'_{i'}}$ und $T_{i'} = \mathfrak{g}^u \mathfrak{g}_0^{R\kappa'_{i'}}$

FTG-II: das Nachrichten-Signatur-Paar $((t, \phi(\mathsf{x})), (\Sigma, e, s))$ mit $S_{i'} = g^{\kappa'_{i'}}$ und $T_{i'} = \Sigma g_0^{R\kappa'_{i'}}$

aus SoK extrahieren. Weiter muss in diesem Fall die Ausgabe von Identify nach Voraussetzung das Symbol $\perp$ sein, da kein korrupter Kunde identifiziert wird.

Somit muss $k_i \geq k'_{i'}$ gelten, da für $k_i < k'_{i'}$ der korrekte öffentliche Schlüssel I ausgegeben wird.

- Falls $k_i \geq k'_{i'}$ ist und der zu $S_{i'}$ gehörige Schlüssel $\kappa'_{i'}$ aus dem Schlüssel κ_i berechnet werden kann, folgt mit dem $\mathsf{Identify}$-Algorithmus für FTG-I:

$$\mathsf{pk}^\star_{\mathcal{K}} = \frac{T_{i'}}{\mathfrak{g}_0^{R\kappa'_{i'}}} = \frac{\mathfrak{g}^u \mathfrak{g}_0^{R\kappa'_{i'}}}{\mathfrak{g}_0^{R\kappa'_{i'}}} = \mathfrak{g}^u \neq I.$$

  Entsprechend folgt für FTG-II:

$$\Sigma^\star = \frac{T_{i'}}{g_0^{R\kappa'_{i'}}} = \frac{\Sigma g_0^{R\kappa'_{i'}}}{g_0^{R\kappa'_{i'}}} = \Sigma \neq \Sigma'.$$

  - Falls das Nachrichten-Signatur-Paar niemals bei einer Orakel-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.1) vor.
  - Falls das Nachrichten-Signatur-Paar bei einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$-Anfrage erstellt wurde, geht $\mathcal{B}$ wie im Fall (1.2) vor.
  - Falls obige Fälle nicht eintreten, wurde das Nachrichten-Signatur-Paar bei einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{B}}$-Anfrage erstellt. In diesem Fall handelt es sich nicht um ein Double-Spending, da

    FTG-I: zwei verschiedene korrupte Kunden $\mathsf{pk}^\star_{\mathcal{K}} = \mathfrak{g}^u \in \mathcal{U}_{\mathcal{A}}$ und $\mathsf{pk}_{\mathcal{K}} = I \in \mathcal{U}_{\mathcal{A}}$ zwei Geldbörsen abgehoben haben.

    FTG-II: entweder von einem oder von zwei korrupten Kunden zwei verschiedene Geldbörsen $\mathbb{W} = (\Sigma, \cdots)$ und $\mathbb{W}' = (\Sigma', \dots)$ abgehoben wurden.

    Insbesondere kann $\mathcal{A}$ in diesem Fall aber nicht Münzen im Wert von mehr als \$ ausgeben oder einlösen und somit das Spiel nicht gewinnen.

- Falls $k_i \geq k'_{i'}$ ist, aber der zu $S_{i'}$ gehörige Schlüssel $\kappa'_{i'}$ nicht aus dem Schlüssel κ_i berechnet werden kann, müssen aus dem Schlüssel κ_i und der Seriennummer $S_{i'}$ mindestens einmal derselbe Serienschlüssel $\kappa_j \in \mathbb{Z}_p^*$ berechnet werden (denn es ist ja ein Double-Spending aufgetreten). Somit gibt es einen Nachfolger S^* von κ_i sowie einen Nachfolger S'^* von $S_{i'}$ (wobei auch $S'^* = S_{i'}$ erlaubt ist) und eine Zahl $\kappa^* \in \mathbb{Z}_p^*$ mit $\kappa^* = \mathsf{H}(S^*||b) = \mathsf{H}(S'^*||b')$ für $b, b' \in \{0, 1\}$. Dies ist allerdings nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Double-Spending-Beschuldigung: Für den Fall (1), dass der Angreifer $\mathcal{A}$ zwei Münzen $\mathsf{coin} = (k, S_1, \dots, S_n, T_1, \dots, T_n, T_C, R, \Phi)$ und $\mathsf{coin}' = (k', S'_1, \dots, S'_{n'}, T'_1, \dots, T'_{n'}, T'_C, R', \Phi')$ ausgibt, kann analog zum fairen teilbaren Geldsystem FTG-I bzw. FTG-II verfahren werden, um das DLog-Problem zu lösen. Insbesondere sind die Fälle A bis E somit vernachlässigbar.

Es bleibt zu zeigen, dass der Angreifer $\mathcal{A}$ auch nicht eine Abhebeinformation $\mathbb{T} = (I, C, E, A_0, SoK_{\mathsf{W}}, \kappa_1, \ldots, \kappa_n, \kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}, \mathsf{ts}, K')$ bei FTG-I bzw. eine Protokollansicht $\mathsf{view} = (\mathbb{T} = (I, C, E_1, E_2, A_0, SoK_{\mathsf{W}}, \kappa_1, \ldots, \kappa_n, \kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}, \mathsf{ts}, K'), \sigma' = (\Sigma, e, s'', t''))$ bei FTG-II sowie eine Münze $\mathsf{coin} = (k', S_1, \ldots, S_{n'}, T_1, \ldots, T_{n'}, T_C, R, \Phi)$ ausgeben kann, um das Spiel zu gewinnen.

Simulation: Der Angreifer $\mathcal{B}$ erhält dieselben Parameter wie bei der Double-Spending Beschuldigung des fairen teilbaren Geldsystems FTG-I bzw. FTG-II und simuliert die Initialisierung sowie die Orakel-Anfragen entsprechend.

Spielende: Der Angreifer $\mathcal{A}$ gibt einen öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{K}} \in \mathcal{U}_H$ sowie einen Beweis Π_G, der eine Abhebeinformation $\mathbb{T} = (I, C, E, A_0, SoK_{\mathsf{W}}, \kappa_1, \ldots, \kappa_n, \kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}, \mathsf{ts}, K')$ bei FTG-I bzw. eine Protokollansicht $\mathsf{view} = (\mathbb{T} = (I, C, E_1, E_2, A_0, SoK_{\mathsf{W}}, \kappa_1, \ldots, \kappa_n, \kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}, \mathsf{ts}, K'), \sigma' = (\Sigma, e, s'', t''))$ bei FTG-II sowie eine Münze $\mathsf{coin} = (k', S_1, \ldots, S_{n'}, T_1, \ldots, T_{n'}, T_C, R, \Phi)$ enthält, aus, so dass $\mathsf{VerifyGuilt}(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{D}}, \mathsf{pk}_{\mathcal{K}}, \Pi_G) = 1$ gilt, obwohl nicht $(\cdot, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ ist.

Analog zu den Fällen B, C und D gibt es nun die Möglichkeiten, dass die Abhebeinformation und die Münze von $\mathcal{B}$ bzgl. verschiedener Kunden, oder jeweils eine der Daten von $\mathcal{B}$ und die andere von $\mathcal{A}$, oder beide Daten von $\mathcal{A}$ erstellt wurden. Zunächst zeigen wir, dass $\mathcal{A}$ die Abhebeinformationen nur mit vernachlässigbarer Wahrscheinlichkeit erzeugt hat.

Fall F: $(\cdot, \mathbb{T}) \notin \mathbb{T}_H$:

In diesem Fall wurde SoK_{W} vom Angreifer $\mathcal{A}$ erstellt. Daher kann $\mathcal{B}$ den privaten Schlüssel $u \in \mathbb{Z}_p$ mit $I = \mathfrak{g}^u$ extrahieren und den diskreten Logarithmus $a = u - \gamma \mod p$ berechnen.

Fall G: $(\cdot, \mathbb{T} = (\mathsf{pk}_{\mathcal{K}}, \cdots)) \in \mathbb{T}_H$ und $(\cdot, \mathsf{pk}'_{\mathcal{K}}, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ mit $\mathsf{pk}_{\mathcal{K}} \neq \mathsf{pk}'_{\mathcal{K}}$:

Die Wahrscheinlichkeit, dass $\mathcal{B}$ die Abhebeinformation $\mathbb{T}$ bzgl. $\mathsf{pk}_{\mathcal{K}}$ erstellt und die Münze coin zum Bezahlen verwendet hat, ist vernachlässigbar, da $\mathcal{B}$ dazu entweder denselben Master-Schlüssel gewählt haben oder eine Kollision bei der Simulation des Random-Oracles auftreten müsste.

Fall H: $(\cdot, \mathbb{T}) \in \mathbb{T}_H$ und $(\cdot, \cdot, \mathsf{coin}) \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$:

Seien κ_i und $S_{i'}$ die Werte, aus denen mindestens ein identischer Serienschlüssel aus $\{\kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}\}$ berechnet wird und seien k_i bzw. $k'_{i'}$ die entsprechenden Geldbeträge. Nun müssen wir die beiden Fälle $k_i \geq k'_{i'}$ sowie $k_i < k'_{i'}$ unterscheiden.

- Sei $k_i \geq k'_{i'}$. Der Angreifer geht nun analog zum Fall D, Unterpunkt $k > k'$, vor. Wir unterscheiden je nach Geldsystem:

FTG-I: In diesem Fall wurde SoK vom Angreifer $\mathcal{A}$ erstellt und daher kann $\mathcal{B}$ die Zahlen $u', \kappa'_{i'} \in \mathbb{Z}_p$ mit $S_{i'} = \mathfrak{g}^{\kappa'_{i'}}$ und $T_{i'} = \mathfrak{g}^{u'}\mathfrak{g}_0^{R\kappa'_{i'}}$ aus SoK extrahieren. Sei $(\mathsf{pk}_\mathcal{K}, \gamma_i) \in \mathcal{U}_H$ der entsprechende Eintrag mit $\mathsf{pk}_\mathcal{K} = I = \mathfrak{g}^{u_i} = \mathfrak{g}^{a+\gamma_i}$. Dann folgt mit dem Identify-Algorithmus:

$$\mathsf{pk}_\mathcal{K} = \frac{T_{i'}}{\mathfrak{g}_0^{R\kappa'_{i'}}} = \frac{\mathfrak{g}^{u'}\mathfrak{g}_0^{R\kappa'_{i'}}}{\mathfrak{g}_0^{R\kappa'_{i'}}} = \mathfrak{g}^{u'}.$$

Somit ist $u_i = u'$ und $\mathcal{B}$ berechnet den diskreten Logarithmus $a = u' - \gamma_i \mod p$.

FTG-II: In diesem Fall wurde SoK vom Angreifer $\mathcal{A}$ erstellt und daher kann $\mathcal{B}$ die Zahl $\kappa'_{i'} \in \mathbb{Z}_p$ mit $S_{i'} = g^{\kappa'_{i'}}$ und $T_{i'} = \Sigma_2 g_0^{R\kappa'_{i'}}$ sowie das Nachrichten-Signatur-Paar $((t_2, \phi_2(\mathsf{x})), (\Sigma_2, e_2, s_2))$ aus SoK extrahieren. Dann folgt mit dem Identify-Algorithmus analog zu Gleichung 6.2 wie bei FTG-II:

$$\Sigma = \frac{T_{i'}}{g_0^{R\kappa'_{i'}}} = \frac{\Sigma_2 g_0^{R\kappa'_{i'}}}{g_0^{R\kappa'_{i'}}} = \Sigma_2$$
$$\Leftrightarrow g^{(x+e_2)} g_0^{s(x+e_2)} g_1^{t(x+e_2)} u_0^{\phi(\alpha)(x+e_2)} = g^{(x+e)} g_0^{s_2(x+e)} g_1^{t_2(x+e)} u_0^{\phi_2(\alpha)(x+e)}.$$

Somit kann $\mathcal{B}$ analog zu TG-II und FTG-II im Fall 1 zwei verschiedene Darstellungen ausgeben und im Fall 2 den diskreten Logarithmus b berechnen.

- Sei $k_i < k'_{i'}$. Dann kann aus der Seriennummer $S_{i'}$ der Schlüssel κ_i berechnet werden, obwohl der Angreifer $\mathcal{A}$ das Element $S_{i'}$ nicht auf eine Orakel-Anfrage erhalten hat (sonst hätte der entsprechende Kunde ja die Seriennummer eingelöst und es würde $(\cdot, \cdot, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$ gelten). Dies ist aber im Random-Oracle-Modell nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Somit löst $\mathcal{B}$ das DLog-Problem in $\mathbb{G}_p$ (bzw. in $\mathbb{G}_1$ bei FTG-II) mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$ (bzw. $\epsilon(\lambda)/2 - \nu(\lambda)$ bei FTG-II), wobei ν eine vernachlässigbare Funktion ist. □

8.3 Geldbörsen mit unterschiedlichen Geldbeträgen

In diesem Abschnitt wird eine alternative Methode zu der in Abschnitt 8.2 vorgestellt, mit der Kunden unterschiedliche Geldbeträge abheben können. Der Vorteil des in Abschnitt 8.2 beschriebenen Abhebeprotokolls ist zwar, dass ohne Veränderung der öffentlichen Parameter jeder beliebige Betrag $K' \leq K$ abgehoben werden kann, doch die

Nachteile sind, dass der Kunde dennoch den vollständigen Binär-Baum für K Serienschlüssel berechnen und speichern muss. Zudem muss die Bank ebenfalls die „unnötigen“ $k = K - K'$ Serienschlüssel berechnen und in ihrer Datenbank abspeichern, um die Double-Spending-Detection zu gewährleisten. Dadurch betragen für das Abhebeprotokoll sowohl die Komplexität des Datentransfers als auch die Berechnungskomplexität der Bank jeweils $O(\lambda \cdot \log(K))$.

Die alternative Variante besteht nun darin, dass ein Kunde während des Abhebevorgangs beweist, dass das im Commitment C gebundene Polynom $\phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ einen bestimmten Grad $K' \leq K$ besitzt und somit im Akkumulator $V = u_0^{\phi(\alpha)}$ maximal $K' \leq K$ Serienschlüssel akkumuliert sind. Dies kann folgendermaßen realisiert werden:

Es werden nicht nur für einen Wert K sondern für mehrere Werte $K_1 = 2^{L_1}, \ldots, K_n = 2^{L_n}$ mit $L_1 < \cdots < L_n$ und $K_n = K$ die Parameter für das Akkumulator-Schema generiert. Somit werden zusätzlich $n-1$ zufällige Zahlen $\alpha_1, \ldots \alpha_{n-1} \in_{\mathcal{R}} \mathbb{Z}_p^*$ gewählt und die Generatoren $u_{i,j} = u_0^{(\alpha_i)^j}$ für $1 \leq i \leq n-1$ und $1 \leq j \leq K_i$ veröffentlicht (analog zu bspw. [ASM08]). Die entsprechenden Generatoren $h_{i,j} = h^{(\alpha_i)^j}$ müssen nicht veröffentlicht werden. Es werden lediglich die Generatoren $h_{i,1} = h^{\alpha_i}$ für $1 \leq i \leq n-1$ von der Bank benötigt. Somit wählt die Bank die Zahlen $\alpha_1, \ldots \alpha_{n-1} \in_{\mathcal{R}} \mathbb{Z}_p^*$ und speichert entweder diese oder die berechneten Generatoren $h_{i,1} = h^{\alpha_i}$ für $1 \leq i \leq n-1$ in $\mathsf{sk}_{\mathcal{B}}$ ab.

Im Abhebeprotokoll berechnet der Kunde nun neben dem bisherigen Commitment $C = g_0^{s'} g_1^u g_2^t u_0^{\phi(\alpha_n)}$ (in FTG-I), wobei $\alpha_n = \alpha$ ist, ein weiteres Commitment $C' = g_0^r u_0^{\phi(\alpha_i)}$ für eine Zufallszahl $r \in_{\mathcal{R}} \mathbb{Z}_p$ und beweist in zero-knowledge, dass es sich jeweils um dasselbe Polynom $\phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ handelt. Das Commitment C' wird also mit den Generatoren $u_0, u_{i,1}, \ldots, u_{i,K_i} \in \mathbb{G}_1$ berechnet, wodurch garantiert ist, dass das Polynom $\phi(\mathsf{x})$ maximal den Grad $K_i \leq K_n = K$ besitzt. Dabei ist es notwendig, dass beide Commitments C und C' berechnet werden, da wie bisher der Akkumulator $V = u_0^{\phi(\alpha)}$ signiert werden muss. Denn würde stattdessen der Akkumulator $V' = u_0^{\phi(\alpha_i)}$ signiert werden, müsste der Händler im Bezahlprotokoll die entsprechenden Generatoren $u_0, u_{i,0}, \ldots, u_{i,K_i}$ zur Verifikation der Signature-of-Knowledge SoK verwenden, da V' mit diesen berechnet wird. Somit müsste der Kunde dem Händler den Index i mitteilen, wodurch die Anonymität nicht mehr gewährleistet ist.

Die Vorteile dieser Variante sind, dass ein Kunde nicht einen vollständigen Binär-Baum für K Serienschlüssel generieren und die Bank nicht die $k = K - K'$ Serienschlüssel im Abhebeprotokoll berechnen und in ihrer Datenbank speichern muss. Somit betragen für das Abhebeprotokoll sowohl die Komplexität des Datentransfers als auch die Berechnungskomplexität der Bank jeweils $O(\lambda)$ und sind damit unabhängig von der Größe der Geldbörse.

Die Nachteile sind allerdings, dass die öffentlichen Parameter mehr Generatoren beinhalten und nicht jeder beliebige Geldbetrag $K' = K - k$, sondern lediglich einer der n vorgegebenen Geldbeträge $K_1, \ldots, K_n$ abgehoben werden kann. Es ist allerdings auch möglich $n = K$ zu wählen, so dass das Abheben jedes beliebigen Betrages $1, \ldots, K$ ermöglicht wird. In diesem Fall werden allerdings neben $u_0, \ldots, u_K \in \mathbb{G}_1$ weitere $K(K-1)/2$ Gene-

ratoren $u_{i,1}, \ldots, u_{i,i} \in \mathbb{G}_1$ für $1 \leq i \leq K-1$ benötigt, wodurch die Speicherkomplexität der öffentlichen Parameter auf $O(\lambda \cdot K^2)$ erhöht wird.

Bemerkung 8.3.1. *Da bei dem ESS+-Signaturverfahren keinerlei Relation über den verwendeten Akkumulator bewiesen wird, kann die hier vorgestellte Methode nicht allein durch das ESS+-Signaturverfahren realisiert werden.*

Die Veränderung betreffen nur den Algorithmus BGen und das Protokoll Withdraw. Die Algorithmen Setup, DGen, Identify und VerifyGuilt sowie die Protokolle Account, Spend, Spend-K und Deposit werden wie beim (modifizierten) fairen teilbaren Geldsystem FTG-I bzw. FTG-II durchgeführt. Der Algorithmus VerifyWithdraw und die Protokolle UTrace sowie CTrace werden dann analog zu FTG-I bzw. FTG-II durchgeführt und aus diesem Grund nicht beschrieben. Allerdings wird ein weiteres Bezahlprotokoll Spend-K_i benötigt, wenn im Abhebeprotokoll eine Geldbörse vom Wert $K_i < K$ abgehoben wurde und mit einer Münze mit diesem Wert bezahlt werden will.

BGen geht wie beim fairen teilbaren Geldsystem FTG-I bzw. FTG-II vor. Zusätzlich werden zufällig $n-1$ Zahlen $\alpha_1, \ldots \alpha_{n-1} \in_{\mathcal{R}} \mathbb{Z}_p^*$ gewählt, die Generatoren $u_{i,j} = u_0^{(\alpha_i)^j}$ für $1 \leq i \leq n-1$ und $1 \leq j \leq K_i$ berechnet und unter $\mathsf{pk}_{\mathcal{B}}$ veröffentlicht. Somit gibt es die weiteren Generatoren $\{u_{i,1}, \ldots, u_{i,K_i}\}_{i=1}^{n-1}$. Die Bank speichert in $\mathsf{sk}_{\mathcal{B}}$ entweder zusätzlich die Zahlen $\alpha_1, \ldots \alpha_{n-1}$ oder die Generatoren $h_{i,1} = h^{\alpha_i}$ für $1 \leq i \leq n-1$ ab.

In der folgenden Tabelle 8.2 werden die benötigten Gruppen, deren öffentliche Generatoren, deren Verwendung und die zugrunde liegenden, kryptografischen Annahmen für das faire teilbare Geldsystem FTG-I zusammengefasst.

Gruppen	**Generatoren**	**Verwendung**	**Annahmen**
$\mathbb{G}_1$	$g, g_0, g_1, g_2,$ $u_0, \ldots, u_K$ $\{u_{i,1}, \ldots, u_{i,K_i}\}_{i=1}^{n-1}$		
$\mathbb{G}_2$	$h, h_1, \ldots, h_K, X, Z$		
$(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$		Nguyen-Akkumulator, BBS+A-Signatur und Münz-Identifikatoren	q-SDH, q-polyDH und DBDH
$\mathbb{G}_p$	$\mathfrak{g}, \mathfrak{g}_0$	Seriennummern und Sicherheitstags	DDH

Tabelle 8.2: Öffentliche Parameter von FTG-I bei dieser Modifikation

Withdraw: Damit ein Kunde $\mathcal{K}$ eine Geldbörse mit dem Wert $K_i = 2^{L_i} \leq K_n = K$ bei der Bank $\mathcal{B}$ abheben kann, führen $\mathcal{K}$ und $\mathcal{B}$ das Abhebeprotokoll Withdraw durch, welches für FTG-I in Abbildung 8.4 dargestellt ist. Zuvor authentifizieren sich jedoch beide Parteien gegenseitig und gleichen den aktuellen Zeitpunkt ts ab.

$\boxed{\mathcal{K}unde}$ $(\mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{K})$	$(\mathsf{sp}, K_i, \mathsf{ts})$	$\boxed{\mathcal{B}ank}$ $(\mathsf{sk}_\mathcal{B}, \mathsf{pk}_\mathcal{K})$
$\kappa_{0,0} \in_\mathcal{R} \mathbb{Z}_p^*, r, s', t', a_0, a_0' \in_\mathcal{R} \mathbb{Z}_p$		
Generiere $\mathbb{K}_0, \mathbb{K}_1$		
$\phi(\mathsf{x}) = \prod_{j=0}^{K_i-1} (\mathsf{x} + \kappa_{L_i+1,j})$		
$C = g_0^{s'} g_1^u g_2^{t'} u_0^{\phi(\alpha)}$		
$C' = g_0^r u_0^{\phi(\alpha_i)}$		
$E = h^{t'}$		
$l = \mathsf{H}(C \| C')$		
$A_0 = W_l u_0^{a_0}$		
$A_0' = W_l' u_0^{a_0'}$		
	$\xrightarrow{C, C', E, A_0, A_0'}$	
$SoK_\mathsf{W}\Big[(a_0, a_0', r, s', t', u, \phi(l)) : I = \mathfrak{g}^u \ \wedge \ E = h^{t'} \ \wedge$		
$\hat{e}(C, h)\, \hat{e}\left(A_0, h_1 h^{-l}\right)^{-1} = \hat{e}\left(g_0^{s'} g_1^u g_2^{t'} u_0^{\phi(l)+a_0 l} u_1^{-a_0}, h\right)$		
$\hat{e}(C', h)\, \hat{e}\left(A_0', h_{i,1} h^{-l}\right)^{-1} = \hat{e}\left(g_0^r u_0^{\phi(l)+a_0' l} u_{i,1}^{-a_0'}, h\right)\Big]\ (\mathsf{ts} \| K_i)$		
		Verifiziere SoK_W
		$e, s'', t'' \in_\mathcal{R} \mathbb{Z}_p$
		$\Sigma = \left(g g_0^{s''} g_2^{t''} C\right)^{\frac{1}{x+e}}$
	$\xleftarrow{\Sigma, e, s'', t''}$	
$s = s' + s'' \mod p$		
$t = t' + t'' \mod p$		
$\hat{e}(\Sigma, X h^e) \stackrel{?}{=} \hat{e}\left(g g_0^{s''} g_2^{t''} C, h\right)$		
$\mathbb{W} = (\Sigma, e, s, t, \mathsf{info})$ oder $\perp$		view oder $\perp$

Abbildung 8.4: Modifiziertes Abhebeprotokoll Withdraw von FTG-I

Schritt 1: Der Kunde $\mathcal{K}$ geht wie beim fairen teilbaren Geldsystem FTG-I bzw. FTG-II vor, außer dass der Binär-Baum nur für $K_i = 2^{L_i} \leq K$ Serienschlüssel $\kappa_{L_i+1,0}, \ldots, \kappa_{L_i+1,K_i-1}$ generiert und das Polynom $\phi(\mathsf{x}) \in \mathbb{Z}_p$ folglich durch $\phi(\mathsf{x}) = \prod_{j=0}^{K_i-1} (\mathsf{x} + \kappa_{L_i+1,j})$ mit $\deg(\phi) = K_i$ berechnet wird. Dann wählt $\mathcal{K}$ zusätzlich eine Zufallszahl $r \in_\mathcal{R} \mathbb{Z}_p$ und berechnet das Commitment $C' = g_0^r u_0^{\phi(\alpha_i)}$. Dieses berechnet $\mathcal{K}$ mit den öffentlichen Generato-

ren $u_0, u_{i,1}, \ldots, u_{i,K_i} \in \mathbb{G}_1$. Dann berechnet der Kunde den Hashwert $l = \mathsf{H}(C||C')$, die Zahl $\phi(l) \in \mathbb{Z}_p$, das Polynom $\phi_l(\mathsf{x}) = (\phi(\mathsf{x}) - \phi(l))\,(\mathsf{x} - l)^{-1}$ und die Zeugen $W_l = u_0^{\phi(\alpha)}$ mit den Generatoren $u_0, \ldots, u_K$ sowie $W_l' = u_0^{\phi(\alpha_i)}$ mit den Generatoren $u_0, u_{i,1}, \ldots, u_{i,K_i}$. Der Kunde $\mathcal{K}$ wählt zufällig zwei Zahlen $a_0, a_0' \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Elemente $A_0 = W_l u_0^{a_0}$ sowie $A_0' = W_l' u_0^{a_0'}$, um beide Zeugen perfekt zu verbergen. Dann erstellt $\mathcal{K}$ die folgenden Signature-of-Knowledge (für FTG-I):

$$SoK_{\mathsf{W}}\Big[(a_0, a_0', r, s', t', u, \phi\,(l)) : I = \mathfrak{g}^u \ \wedge\ E = h^{t'} \ \wedge \\ \hat{e}\,(C, h)\,\hat{e}\left(A_0, h_1 h^{-l}\right)^{-1} = \hat{e}\left(g_0^{s'} g_1^{u} g_2^{t'} u_0^{\phi(l)+a_0 l} u_1^{-a_0}, h\right) \ \wedge \\ \hat{e}\,(C', h)\,\hat{e}\left(A_0', h_{i,1} h^{-l}\right)^{-1} = \hat{e}\left(g_0^{r} u_0^{\phi(l)+a_0' l} u_{i,1}^{-a_0'}, h\right)\Big]\,(\mathsf{ts}||K_i)\,.$$

Für das Geldsystem FTG-II wird die Signature-of-Knowledge SoK_{W} analog erzeugt.

Schritt 2: Die Bank $\mathcal{B}$ verifiziert SoK_{W}, wählt zufällig drei Zahlen $e, s'', t'' \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Element $\Sigma = \left(g g_0^{s''} g_2^{t''} C\right)^{\frac{1}{x+e}}$. Dann sendet $\mathcal{B}$ das Tupel (Σ, e, s'', t'') an $\mathcal{K}$ und bucht dem Kunden $\mathcal{K}$ den Betrag K_i vom Konto mit der Kontonummer I ab. Anschließend speichert $\mathcal{B}$ die Informationen $(\mathbb{T} = (I, C, C', E, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K_i), \sigma' = t'')$ bei FTG-I bzw. die Protokollansicht $\mathsf{view} = (\mathbb{T} = (I, C, C', E_1, E_2, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K_i), \sigma' = (\Sigma, e, s'', t''))$ bei FTG-II in ihrer Datenbank $\mathbb{D}_W$ ab.

Schritt 3: Der Kunde $\mathcal{K}$ geht wie bei FTG-I bzw. FTG-II vor.

Spend-K_i: Damit ein Kunde $\mathcal{K}$ mit Geldbörse $\mathbb{W} = (\Sigma, e, s, t, \mathsf{info})$ bei einem Händler $\mathcal{H}$ mit einer elektronischen Münze coin mit dem Wert $k = K_i < K$ bezahlen kann, wobei die Geldbörse $\mathbb{W}$ mit dem Wert K_i abgehoben wurde, führen $\mathcal{K}$ und $\mathcal{H}$ das Bezahlprotokoll Spend-K_i durch.

Dieses läuft wie das Bezahlprotokoll Spend ab, außer dass der Kunde $\phi_I(l) := 1\ (\in \mathbb{Z}_p^*)$ und $W_{I,l} := 1\ (\in \mathbb{G}_1)$ definiert. Denn für $k = K_i$ gilt $I = \{0, \ldots, K_i - 1\}$ und somit ist das entsprechende Polynom $\phi_I(\mathsf{x}) = \prod_{j=0, j\notin I}^{K_i-1}(\mathsf{x} + \kappa_{L_i+1,j})$ nicht definiert. Daher können weder der Wert $\phi_I(l)$ noch der Zeuge $W_{I,l}$ gemäß dem Bezahlprotokoll Spend berechnet werden.

Allerdings erfüllen die obigen Definitionen die folgende, benötigte (allgemeine) Gleichung 3.17 aus Unterabschnitt 3.2.1

$$\hat{e}\,(\Sigma, X h^e) = \hat{e}\,(g g_0^s g_1^{m_1} \ldots g_L^{m_L}, h)\,\hat{e}\left(W_{I,l}, h^{(\alpha-l)\prod_{j=0}^{k-1}(\alpha+n_j)}\right)\hat{e}\left(u_0^{\phi_I(l)}, h^{\prod_{j=0}^{k-1}(\alpha+n_j)}\right) \\ = \hat{e}\,(g g_0^s g_1^{m_1} \ldots g_L^{m_L}, h)\,\hat{e}\left(u_0^{\prod_{j=0}^{k-1}(\alpha+n_j)}, h\right) = \hat{e}\left(g g_0^s g_1^{m_1} \ldots g_L^{m_L} u_0^{\phi(\alpha)}, h\right),$$

wobei die k Werte $n_0, \ldots, n_{k-1}$ in diesem Fall den k Serienschlüsseln $\kappa_{L_i+1,0}, \ldots, \kappa_{L_i+1,k-1}$ entsprechen und somit für das im Abhebeprotokoll generierte Polynom $\phi(\mathsf{x}) = \prod_{j=0}^{k-1} (\mathsf{x} + n_j) \in \mathbb{Z}_p[\mathsf{x}]$ gilt, da $k = K_i$ ist.

Bemerkung 8.3.2. *Analog zu Bemerkung 8.2.2 kann auch bei dieser Modifizierung eine Münze mit dem Wert $k \leq K$ nur von den Geldbörsen stammen, bei denen ein Betrag $K_i \geq k$ abgehoben wurde. Somit kann die Bank immer nachvollziehen, dass eine Münze mit dem Wert k nur zu einer Geldbörse vom Wert $K_i \geq k$ gehören kann.*

8.3.1 Sicherheitsanalyse

Satz 8.3.1. *Die oben beschriebenen Modifizierungen haben keine Auswirkungen auf die Sicherheit der fairen teilbaren elektronischen Geldsysteme FTG-I und FTG-II.*

Bemerkung 8.3.3. *Analog haben die Modifizierungen keine Auswirkungen auf die Sicherheit der teilbaren elektronischen Geldsysteme TG-I und TG-II.*

Beweis. Die Sicherheitsziele **Double-Spending-Beschuldigung**, **Abhebe-Beschuldigung**, **korrekte Kundenverfolgung**, **Kundenverfolgung-Beschuldigung**, **korrekte Münzverfolgung** und **Münzverfolgung-Beschuldigung** bleiben analog zum fairen teilbaren Geldsystem FTG-I bzw. FTG-II erhalten.

Somit müssen nur Veränderungen für die Unfälschbarkeit und Anonymität betrachtet werden.

Unfälschbarkeit: Der Angreifer $\mathcal{B}$ geht analog zum Geldsystem FTG-I bzw. FTG-II vor. Zusätzlich extrahiert $\mathcal{B}$ bei jeder $\mathcal{O}_{\mathcal{B}}^{\mathsf{W}}$-Anfrage die Zahl $r \in \mathbb{Z}_p$ und interpoliert wie bei FTG-I bzw. FTG-II das Polynom $\phi(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ mit den $K+1$ extrahierten Paaren $\left(l^{(1)}, \phi\left(l^{(1)}\right)\right), \ldots, \left(l^{(K+1)}, \phi\left(l^{(K+1)}\right)\right)$. Weiter interpoliert $\mathcal{B}$ das Polynom $\phi'(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ mit den ersten K_i+1 extrahierten Paaren $\left(l^{(1)}, \phi\left(l^{(1)}\right)\right), \ldots, \left(l^{(K_i+1)}, \phi\left(l^{(K_i+1)}\right)\right)$. Falls $C' \neq g_0^r u_0^{\phi'(\alpha_i)}$ ist, bricht $\mathcal{B}$ ab. Dies tritt aber nur mit vernachlässigbarer Wahrscheinlichkeit ein, da, außer mit vernachlässigbarer Wahrscheinlichkeit, jeder Wert $\phi\left(l^{(j)}\right)$ die korrekte Berechnung des Polynoms $\phi'(\mathsf{x})$ an der Stelle $l^{(j)}$ für $1 \leq j \leq K_i + 1$ (Berechnungsgebundenheit) und $\deg(\phi') \leq K_i$ (Beschränktheit) ist. Falls $\phi'(\mathsf{x}) \neq \phi(\mathsf{x})$ ist, bricht $\mathcal{B}$ ebenfalls ab. Da für unterschiedliche Polynome $\phi(\mathsf{x}), \phi'(\mathsf{x}) \in \mathbb{Z}_p[\mathsf{x}]$ und eine zufällige Zahl $l \in_{\mathcal{R}} \mathbb{Z}_p$ aber nur mit vernachlässigbarer Wahrscheinlichkeit $\phi(l) = \phi'(l)$ gilt, ist dies ebenfalls nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Anonymität: Da das zusätzliche Commitment $C' \in \mathbb{G}_1$ perfekt verbergend ist und der zusätzliche Zeuge $W_l' \in \mathbb{G}_1$ perfekt verborgen ist, ändert sich nichts an der Anonymität der Geldsysteme FTG-I bzw. FTG-II. Somit geht der Angreifer $\mathcal{B}$ analog zu FTG-I bzw. FTG-II vor. □

KAPITEL 9

FAIRE TEILBARE ELEKTRONISCHE GELDSYSTEME MIT OBSERVERN

Alle bisher beschriebenen teilbaren elektronischen Geldsysteme TG-I, TG-II, FTG-0, FTG-I und FTG-II aus den vorherigen Kapiteln 6 bis 8 erfüllen zwar die geforderten Sicherheitseigenschaften Unfälschbarkeit, Anonymität, Double-Spending-Beschuldigung und Abhebe-Beschuldigung aus Abschnitt 4.4, allerdings ist es jedem Kunden möglich, einige Serienschlüssel mehrmals zum Bezahlen zu verwenden. Dieses Double-Spending wird von der Bank aber erst bemerkt, nachdem die entsprechenden Münzen eingelöst wurden. Durch die Double-Spending-Detection kann der Double-Spender dann anschließend identifiziert werden. Damit ein Double-Spending jedoch bereits unmittelbar während des Bezahlvorgangs verhindert werden kann, setzen elektronische Geldsysteme eine zusätzliche manipulationssichere Hardware ein, die als *Observer* bezeichnet wird (bspw. [CP93, CP94, Bra94, Sch09]). Durch die Integration der Observer sollen die bisherigen Sicherheitseigenschaften und die zusätzliche Sicherheitseigenschaft *Double-Spending-Prevention* gewährleistet werden. Dazu darf es dem Kunden nicht möglich sein, ohne seinen Observer ein Bezahlprotokoll durchzuführen.

In Abschnitt 9.2 wird gezeigt, wie die Observer effizient in die (modifizierten) teilbaren elektronischen Geldsysteme TG-I, TG-II, FTG-0, FTG-I und FTG-II integriert werden können, wobei dies exemplarisch nur für das Geldsystem FTG-I explizit beschrieben wird. Die Integration der Observer lässt sich aber völlig analog auf die übrigen Geldsysteme, auch auf die modifizierten Varianten aus Kapitel 8, übertragen.

Da es bei einem fairen elektronischen Geldsystem auch denkbar ist, dass die Observer anstatt von der Bank vom Deanonymisierer an die Kunden verteilt werden, wird in Abschnitt 9.3 beschrieben, wie sich die vom Deanonymisierer ausgegebenen Observer in

die fairen teilbaren elektronischen Geldsysteme FTG-0, FTG-I und FTG-II integrieren lassen. Da man in diesem Fall davon ausgehenden kann, dass die vom Observer generierten Daten keine versteckten Informationen an die Bank erhalten, kann eine effizientere Integration realisiert werden.

Beide Varianten erfüllen die bisherigen Sicherheitseigenschaften selbst dann, wenn die auf dem Observer gespeicherten Daten manipuliert und ausgelesen werden. Die Double-Spending-Prevention ist nach einer Manipulation eines Observers jedoch nicht mehr gewährleistet. Falls viel Vertrauen in die Manipulationssicherheit der Observer gelegt wird, kann analog zu [Sch09, Kapitel 7] ein noch effizienteres teilbares elektronisches Geldsystem konstruiert werden. Dieses wird in Abschnitt 9.4 vorgestellt.

Zunächst wird jedoch im nächsten Abschnitt 9.1 gezeigt, dass kein elektronisches Geldsystem die Anonymität gemäß Definition 4.4.2 erfüllen kann, wenn die von der Bank ausgehändigten Observer wieder an diese zurückgegeben werden (bspw. im Falle einer Kontoauflösung oder eines Defekts).

9.1 Rückgabe der Observer an die Bank

Bisher gibt es in der Literatur einige elektronische Geldsysteme, die Observer einsetzen um eine Double-Spending-Prevention zu garantieren (bspw. [CP93, Bra93, CP94, Bra94, NS03, NS06, Sch09]). Wie in diesen Publikationen beschrieben, muss die Anonymität der Kunden auch dann noch gewährleistet sein, wenn die Bank die von ihr ausgehändigten Observer wiederbekommt. Allerdings werden wir im Folgenden zeigen, dass kein elektronisches Geldsystem anonym sein kann, sobald die Bank ihre ausgegebenen Observer zurückerhält.

Satz 9.1.1. *Kein elektronisches Geldsystem, bei dem der Kunde zur Erstellung einer Münze mit einem von der Bank ausgehändigtem Observer kommunizieren muss, besitzt die Anonymität gemäß Definition 4.4.2, falls die Bank in den Besitz der Observer gelangt.*

Beweis. Wir konstruieren einen polynomiellen Angreifer $\mathcal{A}$, der das Spiel 4.4.2 mit Wahrscheinlichkeit 1 gewinnt und somit die Anonymität gemäß Definition 4.4.2 bricht, wenn $\mathcal{A}$ die Observer an die Kunden verteilt und diese später zurückfordern kann.

Schritt 1: Der Angreifer $\mathcal{A}$ generiert das Schlüsselpaar der Bank gemäß Vorschrift und sendet den öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}}$ an den Challenger $\mathcal{C}$.

Schritt 2: Der Angreifer $\mathcal{A}$ stellt drei Anfragen an das Orakel $\mathcal{O}_{\mathcal{K}}^{\mathsf{A}}$, damit zwei ehrliche Kunden $\mathsf{pk}_0, \mathsf{pk}_1$ und ein ehrlicher Händler $\mathsf{pk}_{\mathcal{H}}$ erstellt werden. Des Weiteren wird jeder Kunde pk_i für $i = 0, 1$ mit einem manipulierten Observer ausgestattet, der zusätzlich einen Zähler $J_i = 0$ besitzt und diesen nach jedem beteiligten Bezahlvorgang um 1 erhöht.

Schritt 3: Der Angreifer $\mathcal{A}$ stellt bzgl. jedes Kunden eine Anfrage an das Orakel $\mathcal{O}_{\mathcal{K}}^{\mathsf{W}}$ bzgl. desselben Geldbetrages K (wobei $K = 1$ ist, wenn das Geldsystem weder teilbar noch kompakt ist). Seien $(0, \mathbb{W}_0, \mathsf{pk}_0, K), (1, \mathbb{W}_1, \mathsf{pk}_1, K) \in \mathbb{W}_H$ die vom Orakel $\mathcal{O}_{\mathcal{K}}^{\mathsf{W}}$ gespeicherten Einträge.

Schritt 4: Der Angreifer $\mathcal{A}$ sendet als Challenge die Indizes $0, 1$, zwei öffentliche Schlüssel $\mathsf{pk}_{\mathcal{H}_b}, \mathsf{pk}_{\mathcal{H}_{\bar{b}}}$ mit $\mathsf{pk}_{\mathcal{H}_b} = \mathsf{pk}_{\mathcal{H}_{\bar{b}}} = \mathsf{pk}_{\mathcal{H}}$, zwei beliebige Zeitpunkte $\mathsf{ts}_b, \mathsf{ts}_{\bar{b}}$ und zwei Geldbeträge $k_b = 0, k_{\bar{b}} = 1$ an den Challenger $\mathcal{C}$. Anschließend wird der Zähler $J_{\bar{b}}$ zu $J_{\bar{b}} = 1$ aktualisiert, während weiterhin $J_b = 0$ gilt.

Schritt 5: In der Post-Challenge-Phase verlangt der Angreifer $\mathcal{A}$ den Observer des Kunden pk_0 zurück.

Schritt 6: Der Angreifer $\mathcal{A}$ liest den Zähler des Observers ab und überprüft, ob $J_0 \stackrel{?}{=} 0$ oder $J_0 \stackrel{?}{=} 1$ gilt. Für $J_0 = 0$ gibt der Angreifer $\mathcal{A}$ das Bit $b' = 0$ und für $J_0 = 1$ das Bit $b' = 1$ aus.

Damit gewinnt der Angreifer $\mathcal{A}$ das Spiel 4.4.2. □

Die Problematik der bisher vorhandenen elektronischen Geldsysteme mit Observern ist, dass es keine formale Definition der Anonymität gibt (z.B. in [CP93, Bra93, CP94, Bra94]), oder eine wesentlich schwächere Definition als Definition 4.4.2 verwendet wird, bei der insbesondere in der Challenge-Phase $k_b = k_{\bar{b}}$ gefordert wird (z.B. in [NS03, NS06, Sch09]). Folglich bieten alle bisherigen elektronischen Geldsysteme tatsächlich nur dann die gewünschte Anonymität, sofern die Bank niemals in den Besitz der von ihr ausgehändigten Observer kommt.

9.2 Das faire teilbare elektronische Geldsystem mit Observern FTGO-I

In diesem Abschnitt sollen Observer in die vorgestellten teilbaren elektronischen Geldsysteme integriert werden, um eine Mehrfachausgabe der Münzen unmittelbar zum Bezahlzeitpunkt zu verhindern, wobei die Observer von der Bank an die Kunden verteilt werden. Wie im vorherigen Abschnitt 9.1 beschrieben, dürfen die Observer allerdings niemals an die Bank zurückgegeben werden, da diese ansonsten die auf dem Observer gespeicherten Daten auslesen kann, um anschließend die Anonymität der Kunden zu brechen. Dennoch ist es uns leider nur möglich, anstelle eines Double-Spendings lediglich ein sogenanntes *Over-Spending* zu verhindern (vgl. [Wei05]). Somit verhindert der Observer unmittelbar zum Bezahlzeitpunkt, dass ein Kunde einen höheren Geldbetrag ausgeben kann, als er abgehoben hat. Dadurch ist eine Over-Spending-Prevention zwar schwächer als eine Double-Spending-Prevention, erfüllt aber im Grunde dennoch genau die von der Bank gewünschte Eigenschaft. Wenn ein Kunde beispielsweise Geldbörsen

im Wert von insgesamt 1.024 Euro bei der Bank abhebt, kann dieser Kunde maximal Münzen im Wert von 1.024 Euro zum Bezahlen verwenden. Somit ist es zwar möglich, dass dieser Kunde eine einzige Münze (genauer: einen einzigen Serienschlüssel) insgesamt 1.024-mal zum Bezahlen verwendet und somit ein Double-Spending begeht, aber der Observer wird bei jedem weiteren Bezahlvorgang abbrechen, so dass der Kunde dann keine weiteren Münzen ausgeben kann. Dieses Beispiel zeigt deutlich, dass die Kunden keinen Vorteil dadurch besitzen, dass lediglich eine Over-Spending-Prevention anstelle einer Double-Spending-Prevention gewährleistet wird. Denn der Kunde kann maximal Münzen im Wert von 1.024 Euro zum Bezahlen verwenden und im Falle eines Double-Spendings verliert er durch die Double-Spending-Detection die Anonymität aller mehrfach ausgegebenen Münzen.

Die explizite Integration der Observer wird dabei exemplarisch für das faire teilbare elektronische Geldsystem FTG-I gezeigt, kann aber wie bereits erwähnt, völlig analog auf die übrigen Geldsysteme übertragen werden. Bei jedem teilbaren Geldsystem wird ein zufälliger Generator $g_1 \in_{\mathcal{R}} \mathbb{G}_1$ sowie ein weiterer, zufälliger Generator $g_{\mathcal{O}} \in_{\mathcal{R}} \mathbb{G}_1$ benötigt.

Bevor das Geldsystem beschrieben wird, werden zunächst die wesentlichen Charakteristika der Integration aufgelistet.

Jeder Observer besitzt ein Schlüsselpaar $(\mathsf{pk}_{\mathcal{O}}, \mathsf{sk}_{\mathcal{O}})$, wobei der geheime Schlüssel $o \in_{\mathcal{R}} \mathbb{Z}_p^*$ zufällig von der Bank gewählt wird. Der öffentliche Schlüssel $\mathsf{pk}_{\mathcal{O}} = (I_{\mathcal{O}}, O) = (g_{\mathcal{O}}^o, h^o) \in \mathbb{G}_1 \times \mathbb{G}_2$ setzt sich aus der Identität $I_{\mathcal{O}}$ sowie einem Signaturschlüssel O zusammen. Zusätzlich werden auf dem Observer die benötigten Systemparameter gespeichert. Alle Daten, die auf dem Observer gespeichert sind, können zwar möglicherweise ausgelesen aber nicht manipuliert werden. Die Daten, die auf dem *nichtauslesbaren* Teil gespeichert werden, können weder ausgelesen noch manipuliert werden. Somit ist es für die Over-Spending-Prevention wichtig, dass der geheime Schlüssel $\mathsf{sk}_{\mathcal{O}} = o$ auf dem nichtauslesbaren Teil abgespeichert wird.

Die Grundidee der Integration der Observer ist nun, dass der Observer einen Zähler J verwaltet, der nach jedem Abhebeprotokoll um den Wert K erhöht und nach jedem Bezahlprotokoll entsprechend um den Wert k verringert wird. Unmittelbar während des Bezahlvorgangs überprüft der Observer, ob die Bedingung $k \leq J$ erfüllt ist und bricht für $k > J$ ab. In diesem Fall kann der Kunde dann keine gültige Münze erzeugen.

Um dies zu garantieren, wird die Identität des Observers $I_{\mathcal{O}}$ während des Abhebeprotokolls ebenfalls von der Bank mit dem BBS+A-Signaturverfahren signiert. Dadurch ist sichergestellt, dass der Kunde keine Münze ohne den Observer generieren kann, da er ohne Kenntnis des geheimen Schlüssels $\mathsf{sk}_{\mathcal{O}}$ nicht die benötigte Signature-of-Knowledge SoK erstellen kann. Zusätzlich erhält der Kunde von der Bank bei jedem Abhebevorgang eine Signature-of-Knowledge $SoK_X\left[(x) : X = h^x\right](I_{\mathcal{O}}||K)$ (bzw. eine Schnorr-Signatur auf die Nachricht $m = (I_{\mathcal{O}}||K)$, vgl. Unterabschnitt 2.8.1 und 2.10.2). Dabei wird das Commitment nicht alleine durch die Bank erzeugt, sondern vom Kunden beeinflusst, damit einerseits eingehende Informationen verhindert werden und andererseits ein Kunde stets eine neue Signature-of-Knowledge erhält. Diese Signature-of-Knowledge

wird anschließend an den Observer weitergeleitet, der verifiziert, dass sie neu und valide ist. Im Anschluss erhöht der Observer den aktuellen Zähler J um den Wert K. Da die Werte $I_{\mathcal{O}}$ und K zur Erstellung von SoK_X in die Hashfunktion einfließen, ist sichergestellt, dass nur der korrekte Observer bzgl. $I_{\mathcal{O}}$ seinen Zähler J um den korrekten Betrag K erhöht.

Möchte ein Kunde nun eine Münze vom Wert k zum Bezahlen verwenden, benötigt dieser vom Observer eine Signatur auf den Hashwert R sowie den Geldbetrag k. Somit erstellt der Observer bei jedem Bezahlprotokoll eine BBS+-Signatur $\sigma = (\Sigma', e', r)$ auf die Nachricht $m = (R, k) \in \mathbb{Z}_p^2$ bzgl. des Signaturschlüssels O (vgl. Unterabschnitt 2.12.1). Der Kunde erstellt nun gemeinsam mit seinem Observer die Signature-of-Knowledge SoK sowie eine weitere Signature-of-Knowledge SoK_σ und beweist damit, dass er eine gültige BBS+-Signatur auf die dem Händler bekannte Nachricht $m = (R, k)$ kennt, wobei der zur Erstellung dieser Signatur benötigte geheime Schlüssel o von der Bank signiert wurde. Durch die BBS+-Signatur auf den Geldbetrag k wird sichergestellt, dass sich der Händler während des Bezahlprotokolls bzw. die Bank während des Einlöseprotokolls davon überzeugen kann, dass der Observer den Zähler um den korrekten Wert k verringert hat. Der Hashwert R fließt mit in die zu signierende Nachricht ein, damit jede vom Observer erstelle BBS+-Signatur $\sigma = (\Sigma', e', r)$ genau einmal verwendet werden kann, nämlich nur beim Bezahlprotokoll zum Hashwert R.

Im Folgenden werden hauptsächlich die Veränderungen im Vergleich zu FTG-I detailliert beschrieben. Die Algorithmen BGen und DGen werden wie beim fairen teilbaren Geldsystem FTG-I durchgeführt. Da sich die Kontoeröffnungs-, Abhebe- und Bezahlprotokolle $\mathsf{Account}, \mathsf{Withdraw}$ und Spend sowie der Algorithmus $\mathsf{VerifyWithdraw}$ verändern, werden die Algorithmen $\mathsf{Identify}$ und $\mathsf{VerifyGuilt}$ sowie die Protokolle Spend-$K, \mathsf{Deposit}$, UTrace und CTrace dann analog zum fairen teilbaren Geldsystem FTG-I durchgeführt und aus diesem Grund nicht beschrieben.

Setup wählt zusätzlich einen weiteren, zufälligen Generator $g_{\mathcal{O}} \in_{\mathcal{R}} \mathbb{G}_1$. Die Ausgabe sind die Systemparameter

$$\mathsf{sp} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \mathbb{G}_p', \hat{e}, g, g_0, g_1, g_2, g_{\mathcal{O}}, u_0, \ldots, u_K, h, h_1, \ldots, h_K, \mathfrak{g}, \mathsf{H}) \leftarrow \mathsf{Setup}\left(1^\lambda, K\right).$$

Account: Damit ein Kunde $\mathcal{K}$ dem System beitreten und ein Konto bei der Bank $\mathcal{B}$ eröffnen kann, führen $\mathcal{K}$ und $\mathcal{B}$ das Kontoeröffnungsprotokoll $\mathsf{Account}$ durch. Zuvor muss sich der Kunde $\mathcal{K}$ gegenüber $\mathcal{B}$ ausweisen, beispielsweise mit einem Personalausweis.

Schritt 1: Der Kunde $\mathcal{K}$ erzeugt sein Schlüsselpaar $(\mathsf{pk}_{\mathcal{K}}, \mathsf{sk}_{\mathcal{K}}) = (I, u) \in \mathbb{G}_p \times \mathbb{Z}_p^*$. Dazu wählt $\mathcal{K}$ zufällig seinen geheimen Schlüssel $u \in_{\mathcal{R}} \mathbb{Z}_p^*$, berechnet seinen öffentlichen Schlüssel $I = \mathfrak{g}^u$ und sendet den öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{K}}$ an die Bank.

Schritt 2: Die Bank $\mathcal{B}$ initialisiert einen Observer $\mathcal{O}$ für den Kunden $\mathcal{K}$. Dazu werden auf $\mathcal{O}$ der benötigte Teil der Systemparameter $\mathsf{sp}' = (p, \mathbb{G}_1', \mathbb{G}_2', g, g_0, g_1, g_{\mathcal{O}}, u_0, h, \mathsf{H})$, der öffentliche Signaturschlüssel X, der öffentliche Schlüssel I und ein Zähler J, der mit $J = 0$ initialisiert wird, gespeichert. Weiter wählt die Bank $\mathcal{B}$ zufällig einen geheimen Schlüssel $o \in_{\mathcal{R}} \mathbb{Z}_p^*$ und berechnet den öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{O}} = (I_{\mathcal{O}}, O) = (g_{\mathcal{O}}^o, h^o) \in \mathbb{G}_1 \times \mathbb{G}_2$, wobei $I_{\mathcal{O}}$ die Identität und O der öffentliche Signaturschlüssel des Observers ist. Der geheime Schlüssel $\mathsf{sk}_{\mathcal{O}} = o$ wird auf dem nichtauslesbaren Teil des Observers gespeichert. Anschließend wird der Observer $\mathcal{O}$ zusammen mit dem öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{O}}$ an den Kunden $\mathcal{K}$ übergeben. Die Bank $\mathcal{B}$ speichert (I, o) zusammen mit persönlichen Daten von $\mathcal{K}$ in ihrer Datenbank $\mathbb{D}_{\mathcal{K}}$ und legt ein Konto für $\mathcal{K}$ an. Anhand von I kann $\mathcal{K}$ eindeutig identifiziert werden, weshalb I auch als Kontonummer oder Identität von $\mathcal{K}$ bezeichnet wird.

Schritt 3: Der Kunde $\mathcal{K}$ speichert sein Schlüsselpaar $(\mathsf{pk}_{\mathcal{K}}, \mathsf{sk}_{\mathcal{K}})$ sowie den öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{O}}$ ab.

Falls entweder das Element I oder die Zahl o bereits in der Datenbank $\mathbb{D}_{\mathcal{K}}$ gespeichert ist, muss dieses Protokoll erneut durchgeführt werden. Zudem muss der Kunde mit Kontonummer I ein neues Konto eröffnen, da die Bank nun die Kontonummer I sperren muss. Dies ist allerdings nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Withdraw: Damit ein Kunde $\mathcal{K}$ gemeinsam mit seinem Observer $\mathcal{O}$ eine Geldbörse mit dem Wert $K = 2^L$ bei der Bank $\mathcal{B}$ abheben kann, führen $\mathcal{O}, \mathcal{K}$ und $\mathcal{B}$ das Abhebeprotokoll Withdraw durch, welches in Abbildung 9.1 dargestellt ist. Zuvor authentifizieren sich jedoch $\mathcal{O}$ und $\mathcal{K}$ sowie $\mathcal{K}$ und $\mathcal{B}$ gegenseitig. Weiter gleichen $\mathcal{K}$ und $\mathcal{B}$ den aktuellen Zeitpunkt ts ab.

Schritt 1: Die Bank $\mathcal{B}$ wählt zufällig eine Zahl $\rho_x' \in_{\mathcal{R}} \mathbb{Z}_p$, berechnet das Commitment $T_X' = h^{\rho_x'}$ und sendet T_X' an $\mathcal{K}$.

Schritt 2: Der Kunde $\mathcal{K}$ geht wie beim Geldsystem FTG-I vor, wählt zusätzlich eine zufällige Zahl $\rho_x'' \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Commitment $T_X = T_X' h^{\rho_x''}$. Bei der Erstellung der Signature-of-Knowledge SoK_{W} fließen das Commitment T_X sowie das Element $I_{\mathcal{O}}$ zur Berechnung der Challenge c mit in die Hashfunktion H ein. Die Zahl ρ_x'' wird ebenfalls an die Bank gesendet.

Schritt 3: Die Bank $\mathcal{B}$ berechnet das Commitment $T_X = T_X' h^{\rho_x''}$, verifiziert SoK_{W}, wählt zufällig drei Zahlen $e, s'', t'' \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Element $\Sigma = \left(g g_0^{s''} g_2^{t''} I_{\mathcal{O}} C\right)^{\frac{1}{x+e}}$. Zudem erstellt $\mathcal{B}$ eine Signature-of-Knowledge $SoK_X\left[(x) : X = h^x\right](I_{\mathcal{O}} || K)$. Dazu berechnet $\mathcal{B}$ die Challenge $d = \mathsf{H}(T_X || I_{\mathcal{O}} || K)$ und die Response $z_x = \rho_x' + \rho_x'' + dx \mod p$. Dann sendet $\mathcal{B}$ das Tupel $(\Sigma, e, s'', t'', z_x)$ an $\mathcal{K}$ und bucht dem Kunden $\mathcal{K}$ den Betrag K vom Konto mit der Kontonummer I ab. Anschließend speichert $\mathcal{B}$ die Informationen

$(\mathbb{T} = (I, I_{\mathcal{O}}, C, E, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K, d), \sigma' = (t'', z_x))$ in ihrer Datenbank $\mathbb{D}_W$.

Schritt 4: Der Kunde $\mathcal{K}$ berechnet $s = s' + s'' \mod p$ sowie $t = t' + t'' \mod p$, verifiziert die Signatur durch Überprüfung der Gleichung $\hat{e}(\Sigma, Xh^e) \stackrel{?}{=} \hat{e}\left(gg_0^{s''}g_2^{t''}I_{\mathcal{O}}C, h\right)$ und speichert $\mathbb{W} = (\Sigma, e, s, t, \mathsf{info})$ ab. Weiter verifiziert der Kunde $\mathcal{K}$ die Signature-of-Knowledge SoK_X, indem $\mathcal{K}$ den Hashwert $d = \mathsf{H}(T_X||I_{\mathcal{O}}||K)$ berechnet und die Gleichung $T_X \stackrel{?}{=} h^{z_x}X^{-d}$ überprüft. Dann sendet $\mathcal{K}$ die Signature-of-Knowledge SoK_X, also das Paar $(d, z_x) \in \mathbb{Z}_p^2$ mit $T_X = h^{z_x}X^{-d}$, an $\mathcal{O}$.

Schritt 5: Der Observer $\mathcal{O}$ überprüft zunächst, ob sich die Challenge d bereits in der Liste $\mathbb{D}$ befindet. Falls ja bricht $\mathcal{O}$ ab. Andernfalls verifiziert der Observer $\mathcal{O}$ die Signature-of-Knowledge SoK_X. Schließlich erhöht $\mathcal{O}$ den Zähler J um K und speichert die Challenge d in der Liste $\mathbb{D}$ ab.

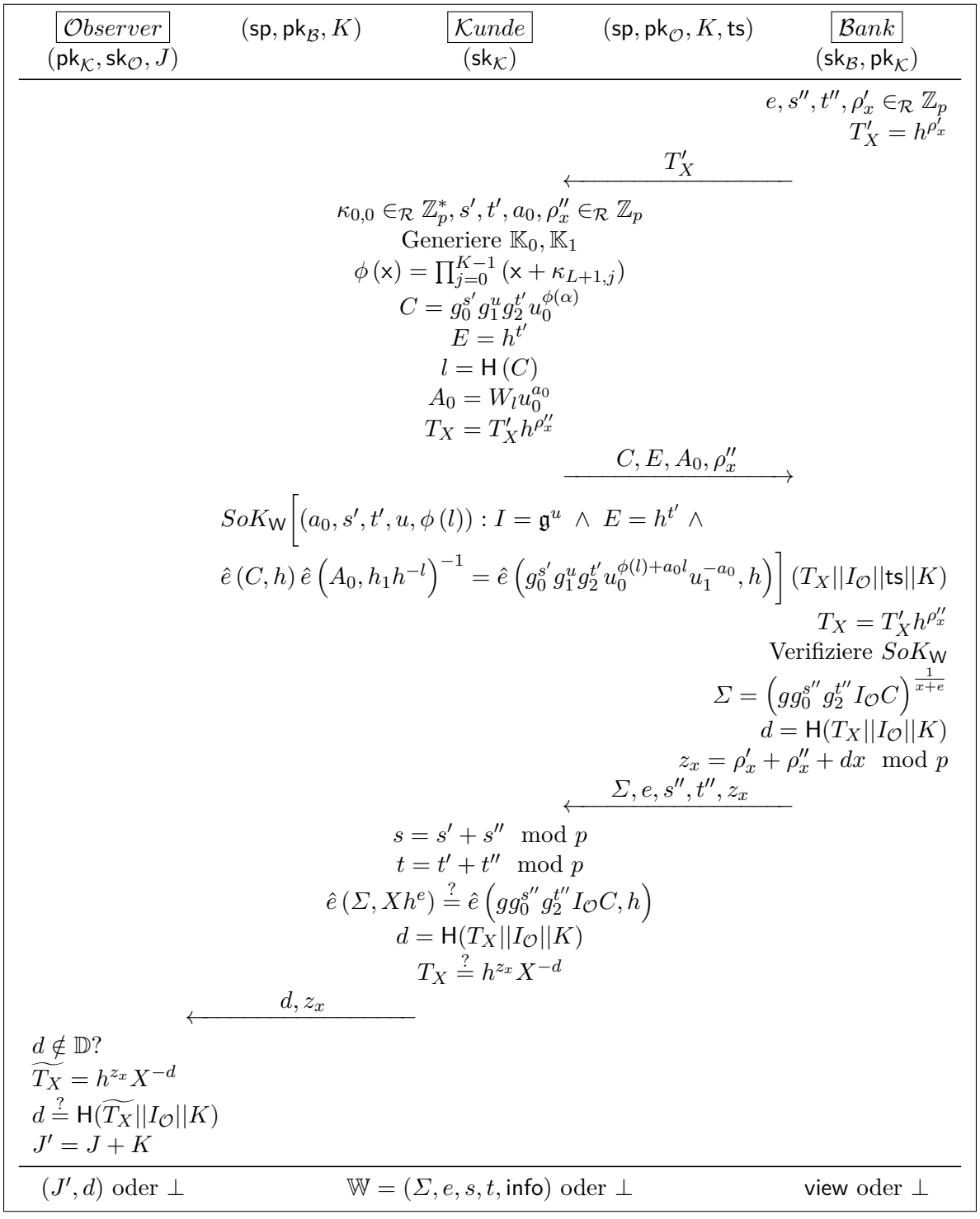

Abbildung 9.1: Abhebeprotokoll Withdraw von FTGO-I

Spend: Damit ein Kunde $\mathcal{K}$, der im Besitz der Geldbörse $\mathbb{W} = (\Sigma, e, s, t, \mathsf{info})$ ist, gemeinsam mit seinem Observer $\mathcal{O}$, der im Besitz eines Zählers J ist, bei einem Händler $\mathcal{H}$ mit einer elektronischen Münze coin mit dem Wert $k = 2^\ell < K$ bezahlen kann, führen $\mathcal{O}, \mathcal{K}$ und $\mathcal{H}$ das Bezahlprotokoll Spend durch, welches in Abbildung 9.3 dargestellt ist. Zuvor authentifizieren sich jedoch $\mathcal{O}$ und $\mathcal{K}$ gegenseitig. Weiter authentifiziert sich der Händler $\mathcal{H}$ gegenüber dem Kunden $\mathcal{K}$ und beide Parteien gleichen den Zeitpunkt ts ab. Weiter überprüft der Observer zunächst, ob die Bedingung $k \leq J$ erfüllt ist. Andernfalls bricht $\mathcal{O}$ direkt ab und sendet das Symbol $\perp$ an $\mathcal{K}$.

Schritt 1: Der Händler $\mathcal{H}$ und der Kunde $\mathcal{K}$ berechnen beide den Hashwert $R = \mathsf{H}(\mathsf{pk}_\mathcal{H} || \mathsf{ts} || k)$.

Schritt 2: Der Kunde $\mathcal{K}$ geht zunächst wie im Geldsystem FTG-I vor und berechnet die Elemente S, T, T_C, A_1, B_1, B_2. Anschließend sendet $\mathcal{K}$ die Zahl R an $\mathcal{O}$.

Schritt 3: Der Observer $\mathcal{O}$ verifiziert SoK_D und erstellt mit seinem privaten Schlüssels o eine BBS+-Signatur (Σ', e', r) auf die Nachricht (R, k). Dazu wählt $\mathcal{O}$ zufällig zwei Zahlen $r'', e' \in_\mathcal{R} \mathbb{Z}_p$ und berechnet das Element $\Sigma' = \left(g g_0^{r''} D u_0^k\right)^{\frac{1}{o+e'}}$. Dann sendet $\mathcal{O}$ das Tripel (Σ', e', r'') an $\mathcal{K}$ und verringert den Zähler J um k.

Schritt 4: Der Kunde $\mathcal{K}$ berechnet die Zahl $r = r' + r'' \mod p$ und verifiziert die Signatur durch Überprüfung der Gleichung $\hat{e}\left(\Sigma', O h^{e'}\right) \stackrel{?}{=} \hat{e}\left(g g_0^r g_1^R u_0^k, h\right)$. Anschließend wählt $\mathcal{K}$ zufällig drei Zahlen $c_1, c_2, c_3 \in_\mathcal{R} \mathbb{Z}_p$ und berechnet die Elemente $C_1 = g_0^{c_1} g_1^{c_2} u_0^{c_3}, C_2 = \Sigma' g_1^{c_1}$ sowie $C_3 = O h^{c_3}$, damit die Elemente Σ' sowie O perfekt verborgen sind und $\mathcal{K}$ rechnerisch an die Zahlen c_1 und c_3 gebunden ist. (Das Commitment C_1 und das Element Σ' werden mit dem Generator u_0 statt g_2 berechnet, da g_2 bei der nicht fairen Version nicht benötigt wird.) Dann sendet $\mathcal{K}$ die Elemente $S, T, T_C, A_1, B_1, B_2, C_1, C_2, C_3$ an $\mathcal{H}$.

Schritt 5: Schließlich erstellen $\mathcal{O}$ und $\mathcal{K}$ gemeinsam die folgende Signature-of-Knowledge SoK, wobei $\beta_1 = b_1 e, \beta_2 = b_2 e, u_I = u_0^{\prod_{j \in I}(\alpha + \kappa_{L+1,j})}, u_{l,I} = u_0^{(\alpha-l)\prod_{j \in I}(\alpha+\kappa_{L+1,j})}$ und $h_{l,I} = h^{(\alpha-l)\prod_{j \in I}(\alpha+\kappa_{L+1,j})}$ ist (siehe auch Abbildung 9.2 und Abschnitt A.5):

$$
\begin{aligned}
SoK\Big[(a_1, b_1, b_2, \beta_1, \beta_2, c_1, c_2, c_3, e, o, s, t, u, \kappa, \phi_I(l)) : \\
B_1 = g_0^{b_1} g_1^{b_2} \ \wedge\ 1 = B_1^e g_0^{-\beta_1} g_1^{-\beta_2} \ \wedge\ S = \mathfrak{g}^\kappa \ \wedge\ T = \mathfrak{g}^u \mathfrak{g}_0^{R\kappa} \ \wedge \\
T_C = \hat{e}\left(B_1^t, Z\right) \ \wedge\ C_1 = g_0^{c_1} g_1^{c_2} u_0^{c_3} \ \wedge\ C_3 = h^{o+c_3} \ \wedge
\end{aligned}
$$

$$\frac{\hat{e}\left(B_2, X\right)}{\hat{e}\left(g, h\right)\hat{e}\left(A_1, h_{l,I}\right)} = \hat{e}\left(g_1^{b_1}, X\right)\hat{e}\left(g_0^{s} g_1^{u+\beta_1} g_2^{t} g_{\mathcal{O}}^{o} u_{l,I}^{-a_1} u_I^{\phi_I(l)} B_2^{-e}, h\right)\Bigg](R)\,.$$

Dabei ist die Zahl o nur dem Observer $\mathcal{O}$ bekannt und alle anderen Zahlen sind nur dem Kunden $\mathcal{K}$ bekannt. Ähnliche Proofs-of-Knowledge bzw. Signatures-of-Knowledge mit zwei Provern sind bspw. in [CP93, Bra93, Sch09] zu finden.

Die Signature-of-Knowledge SoK wird dabei folgendermaßen vom Observer $\mathcal{O}$ und vom Kunden $\mathcal{K}$ erstellt (siehe auch Abschnitt A.5).

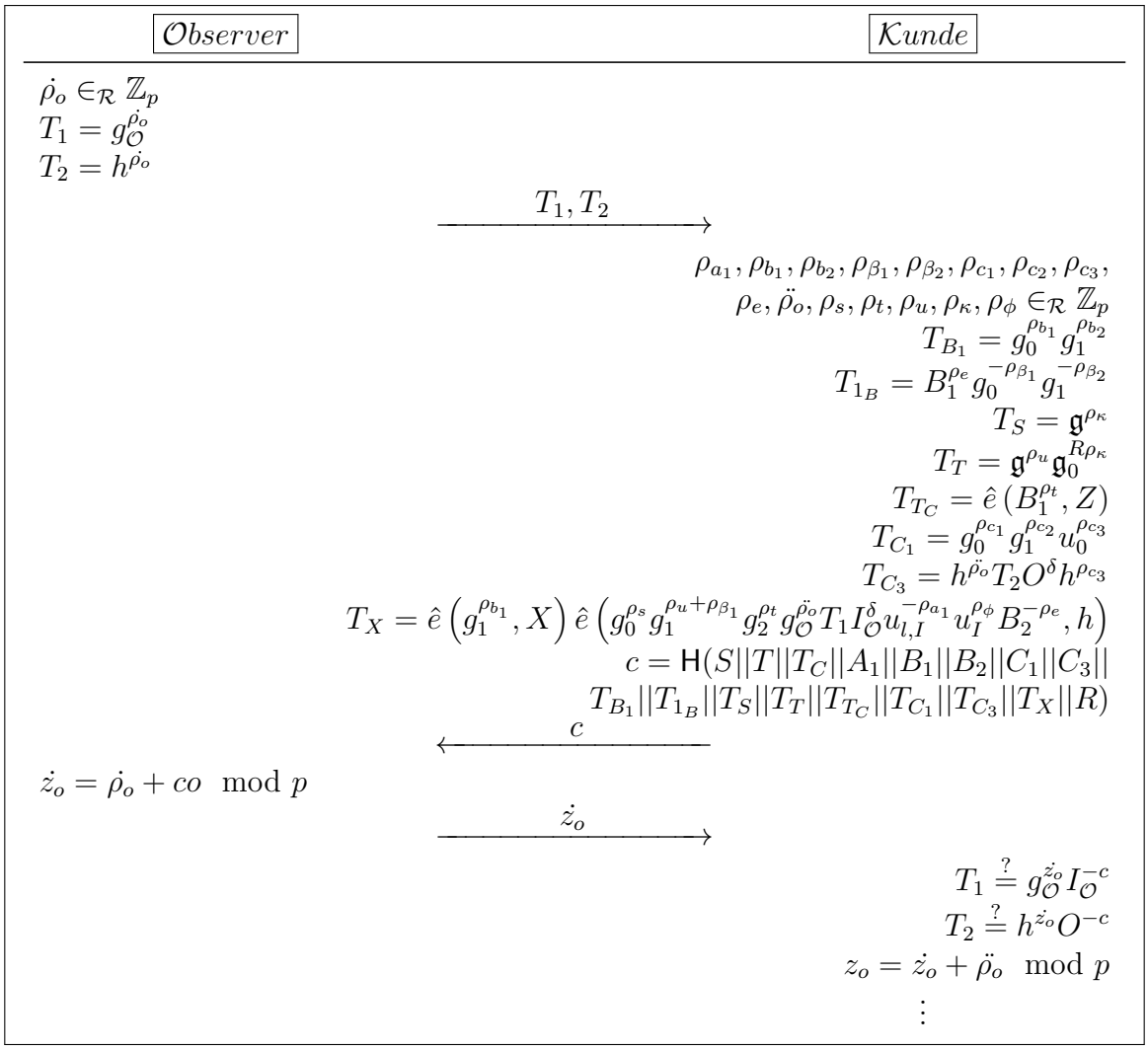

Abbildung 9.2: SoK des Bezahlprotokolls Spend von FTGO-I

Alle weiteren Responses berechnet der Kunde $\mathcal{K}$ wie gewohnt. Die detaillierte Ausführung von SoK ist in Abschnitt A.5 beschrieben.

Des Weiteren muss $\mathcal{K}$ beweisen, im Besitz einer gültigen BBS+-Signatur auf die Nachricht (R, k) bzgl. des öffentlichen Schlüssels $O = h^o$ zu sein. Aus diesem Grund erstellt $\mathcal{K}$ (ohne den Observer) die folgende Signature-of-Knowledge SoK_σ, wobei $c_4 = c_1, c_5 = c_2, c_6 = c_3, \gamma_1 = c_4(e' - c_6), \gamma_2 = c_5(e' - c_6)$ und $\gamma_3 = c_6(e' - c_6)$ ist (siehe auch Abschnitt A.6):

$$SoK_\sigma\Bigg[(c_4, c_5, c_6, \gamma_1, \gamma_2, \gamma_3, e', r) : C_1 = g_0^{c_4} g_1^{c_5} u_0^{c_6} \ \wedge \ 1 = C_1^{e'-c_6} g_0^{-\gamma_1} g_1^{-\gamma_2} u_0^{-\gamma_3} \\ \wedge \ \frac{\hat{e}\,(C_2, C_3)}{\hat{e}\left(g g_1^R u_0^k, h\right)} = \hat{e}\,(g_1^{c_4}, C_3)\,\hat{e}\left(g_0^r g_1^{\gamma_1} C_2^{c_6 - e'}, h\right)\Bigg]\,(R||SoK)\,,$$

wobei das Challenge-Response-Tupel der Signature-of-Knowledge SoK zur Berechnung der Challenge c_σ mit in die Hashfunktion einfließt.

Schritt 6: Der Händler $\mathcal{H}$ berechnet aus der Seriennummern S alle k Seriennschlüssel $\kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}$. Dann berechnet $\mathcal{H}$ den Hashwert $l = \mathsf{H}(S||T||T_C||B_1||B_2)$, verifiziert SoK sowie SoK_σ und speichert die Münze $\mathsf{coin} = (k, S, T, T_C, R, \Phi = (A_1, B_1, B_2, C_1, C_2, C_3, SoK, SoK_\sigma, \mathsf{ts}, \mathsf{pk}_\mathcal{H}))$ ab.

VerifyWithdraw verifiziert bei Eingabe der Systemparameter sp, der öffentlichen Schlüssel $\mathsf{pk}_\mathcal{B}$ und $\mathsf{pk}_\mathcal{D}$, eines öffentlichen Schlüssels $\mathsf{pk}_\mathcal{K}$ und einer Protokollansicht $\mathsf{view} = (\mathbb{T} = (\mathsf{pk}_\mathcal{K}, I_\mathcal{O}, C, E, A_0, SoK_\mathsf{W}, \mathsf{ts}, K, d), \sigma' = (\Sigma, e, s'', t'', z_x))$ die BBS+A-Signatur durch Überprüfung der Gleichung $\hat{e}\,(\Sigma, X h^e) \stackrel{?}{=} \hat{e}\left(g g_0^{s''} g_2^{t''} I_\mathcal{O} C, h\right)$. Anschließend wird die Signature-of-Knowledge SoK_X verifiziert, indem das Commitment $\widetilde{T_X} = h^{z_x} X^{-d}$ berechnet und die Gleichung $d \stackrel{?}{=} \mathsf{H}(\widetilde{T_X}||I_\mathcal{O}||K)$ überprüft wird. Danach wird die Signature-of-Knowledge SoK_W verifiziert, da dazu das Element $\widetilde{T_X}$ benötigt wird. Somit kann eine dritte Partei verifizieren, dass der Kunde $\mathsf{pk}_\mathcal{K}$ dieses Abhebeprotokoll durchgeführt hat.

$\boxed{\mathcal{O}bserver}$	(sp, k)	$\boxed{\mathcal{K}unde}$	$(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{D}}, \mathsf{ts}, k)$	$\boxed{\mathcal{H}ändler}$
$(\mathsf{pk}_{\mathcal{K}}, \mathsf{sk}_{\mathcal{O}}, J)$		$(\mathsf{pk}_{\mathcal{H}}, \mathsf{sk}_{\mathcal{K}}, \mathsf{pk}_{\mathcal{O}}, \mathbb{W})$		$(\mathsf{sk}_{\mathcal{H}})$

$$R = \mathsf{H}\left(\mathsf{pk}_{\mathcal{H}} || \mathsf{ts} || k\right) \qquad R = \mathsf{H}\left(\mathsf{pk}_{\mathcal{H}} || \mathsf{ts} || k\right)$$

$$a_1, b_1, b_2, c_1, c_2, c_3 \in_{\mathcal{R}} \mathbb{Z}_p$$

$$\kappa := \kappa_{L-\ell, j_0}$$

$$S = \mathfrak{g}^{\kappa}$$

$$T = \mathfrak{g}^{u} \mathfrak{g}_0^{R\kappa}$$

$$B_1 = g_0^{b_1} g_1^{b_2}, B_2 = \Sigma g_1^{b_1}$$

$$T_C = \hat{e}\left(B_1^t, Z\right)$$

$$l = \mathsf{H}\left(S || T || T_C || B_1 || B_2\right)$$

$$A_1 = W_{I,l} u_0^{a_1}$$

$$\xrightarrow{S, T, T_C, A_1, B_1, B_2}$$

$$\xleftarrow{R}$$

$k \leq J$?

$$e', r \in_{\mathcal{R}} \mathbb{Z}_p$$

$$\Sigma' = \left(g g_0^r g_1^R u_0^k\right)^{\frac{1}{o+e'}}$$

$$J' = J - k$$

$$\xrightarrow{\Sigma', e', r}$$

$$\hat{e}\left(\Sigma', O h^{e'}\right) \stackrel{?}{=} \hat{e}\left(g g_0^r g_1^R u_0^k, h\right)$$

$$C_1 = g_0^{c_1} g_1^{c_2} u_0^{c_3}, C_2 = \Sigma' g_1^{c_1}$$

$$C_3 = O h^{c_3}$$

$$\xrightarrow{C_1, C_2, C_3}$$

$$SoK\Big[(a_1, b_1, b_2, \beta_1, \beta_2, c_1, c_2, c_3, e, o, s, t, u, \kappa, \phi_I(l)) : B_1 = g_0^{b_1} g_1^{b_2} \ \wedge \ 1 = B_1^e g_0^{-\beta_1} g_1^{-\beta_2} \ \wedge$$

$$S = \mathfrak{g}^{\kappa} \ \wedge \ T = \mathfrak{g}^{u} \mathfrak{g}_0^{R\kappa} \ \wedge \ T_C = \hat{e}\left(B_1^t, Z\right) \ \wedge \ C_1 = g_0^{c_1} g_1^{c_2} u_0^{c_3} \ \wedge \ C_3 = h^{o+c_3} \ \wedge$$

$$\frac{\hat{e}\left(B_2, X\right)}{\hat{e}\left(g, h\right) \hat{e}\left(A_1, h_{l,I}\right)} = \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(g_0^s g_1^{u+\beta_1} g_2^t g_{\mathcal{O}}^o u_{l,I}^{-a_1} u_I^{\phi_I(l)} B_2^{-e}, h\right)\Big] (R)$$

$$SoK_{\sigma}\Big[(c_4, c_5, c_6, \gamma_1, \gamma_2, \gamma_3, e', r) : C_1 = g_0^{c_4} g_1^{c_5} u_0^{c_6} \ \wedge \ 1 = C_1^{e'-c_6} g_0^{-\gamma_1} g_1^{-\gamma_2} u_0^{-\gamma_3} \ \wedge$$

$$\frac{\hat{e}\left(C_2, C_3\right)}{\hat{e}\left(g g_1^R u_0^k, h\right)} = \hat{e}\left(g_1^{c_4}, C_3\right) \hat{e}\left(g_0^r g_1^{\gamma_1} C_2^{c_6-e'}, h\right)\Big] (R || SoK)$$

Berechne $\kappa_{L+1,0}, \ldots, \kappa_{L+1,k-1}$

$$l = \mathsf{H}\left(S || T || T_C || B_1 || B_2\right)$$

Verifiziere SoK, SoK_{σ}

J' oder $\perp$	$\mathbb{W}'$ oder $\perp$	coin oder $\perp$

Abbildung 9.3: Bezahlprotokoll Spend von FTGO-I

9.2.1 Sicherheitsanalyse von FTGO-I

Zunächst benötigen wir die folgende Annahme:

Annahme 9.1. *Das Schnorr-Identifikationsverfahren ist unter der DLog-Annahme auch nach polynomiell vielen Durchführungen sicher.*

Bemerkung 9.2.1. *Diese Annahme wird ebenfalls beim elektronischen Geldsystem mit Observern in [Bra94] verwendet. Um die Sicherheit des Schnorr-Identifikationsverfahrens zu modellieren, darf der Angreifer $\mathcal{B}$ im Folgenden neben den Signatur-Anfragen auch zusätzlich das Schnorr-Identifikationsverfahren in den Gruppen $\mathbb{G}_1$ und $\mathbb{G}_2$ polynomiell häufig mit dem BBS+A-Signatur-Orakel durchführen.*

Satz 9.2.1. *Das faire teilbare elektronische Geldsystem mit Observern FTGO-I ist im Random-Oracle-Modell unter der Annahme 9.1 ein sicheres faires elektronisches Geldsystem, falls die q-SDH-Annahme, die q-polyDH-Annahme sowie die DBDH-Annahme für $\mathsf{Setup}_{\mathsf{Bil}}$ gelten und die DDH-Annahme für $\mathsf{Setup}_{\mathsf{G}}$ gilt. Zudem erfüllt dieses Geldsystem unter den genannten Annahmen im Random-Oracle-Modell die Sicherheitseigenschaft Over-Spending-Prevention. Dabei sind alle Sicherheitseigenschaften außer der Over-Spending-Prevention sogar dann erfüllt, wenn die Observer vom Angreifer $\mathcal{A}$ manipuliert werden. Weiter werden sowohl eingehende als auch ausgehende Informationen verhindert.*

Beweis. Zunächst zeigen wir, dass dieses Geldsystem **eingehende** und **ausgehende Informationen** verhindert.

Die einzigen Informationen, die der Observer während des Abhebeprotokolls erhält, ist die Signature-of-Knowledge SoK_X, also das Paar (d, z_x) und damit das Element $T_X = h^{z_x} X^{-d}$. Da das Commitment allerdings durch $T_X = T'_X h^{\rho''_x}$ berechnet wird und die Zahl $\rho''_x \in_{\mathcal{R}} \mathbb{Z}_p$ zufällig vom Kunden gewählt wird, ist das Element T_X zufällig und gleichverteilt in $\mathbb{G}_2$ und somit kann die Bank keine Informationen in T_X integrieren. Da die Challenge $d \in_{\mathcal{R}} \mathbb{Z}_p^*$ die Ausgabe des Random-Oracles und abhängig vom Commitment T_X ist, kann die Bank auch hier keine Informationen integrieren. Schließlich ist die Response $z_x = \rho'_x + \rho''_x + dx \mod p$ durch die Zahlen ρ''_x und d zufällig und gleichverteilt in $\mathbb{Z}_p$. Die einzigen Informationen, die der Observer während des Bezahlprotokolls erhält, sind die Zahl R und die Challenge c. Da beide Werte die Ausgabe einer Hashfunktion sind und vom Kunden an den Observer gesendet werden, können auch hier keine Informationen der Bank bzw. des Händlers integriert werden. Somit werden eingehende Informationen verhindert.

Im Abhebeprotokoll sendet der Observer überhaupt keine Informationen. Die einzigen Informationen, die der Observer während des Bezahlprotokolls sendet, ist das Tripel (Σ', e', r'') sowie die Commitments T_1, T_2 und die Response $\dot{z}_o$. Doch da der Kunde die Werte für die Signatures-of-Knowledge SoK und SoK_σ entsprechend perfekt blendet, kann der Observer keine Informationen integrieren. Somit werden ausgehende Informationen verhindert.

Unfälschbarkeit: Die Unfälschbarkeit des Geldsystems FTGO-I folgt direkt aus der Unfälschbarkeit des fairen teilbaren Geldsystems FTG-I, selbst dann wenn der Angreifer $\mathcal{A}$ die erhaltenen Observer manipuliert. Um dies zu modellieren, erhält der Angreifer $\mathcal{A}$ bei jeder $\mathcal{O}_{\mathcal{B}}^{\mathsf{A}}$-Anfrage zusätzlich den geheimen Schlüssel $\mathsf{sk}_{\mathcal{O}}$, der von $\mathcal{B}$ gemäß Protokoll gewählt wird. Da der Angreifer $\mathcal{A}$ nun selbst die Rolle der Observer übernehmen kann bzw. darf, werden die Orakel $\mathcal{O}_{\mathcal{B}}^{\mathsf{A}}, \mathcal{O}_{\mathcal{B}}^{\mathsf{W}}, \mathcal{O}_{\mathcal{H}}^{\mathsf{S}}, \mathcal{O}_{\mathcal{H}}^{\mathsf{S}^*}$ nicht durch die entsprechenden Orakel $\mathcal{O}_{\mathcal{O},\mathcal{B}}^{\mathsf{A}}, \mathcal{O}_{\mathcal{O},\mathcal{B}}^{\mathsf{W}}, \mathcal{O}_{\mathcal{O},\mathcal{H}}^{\mathsf{S}}, \mathcal{O}_{\mathcal{O},\mathcal{H}}^{\mathsf{S}^*}$ ersetzt. Dies gilt auch für die folgenden Sicherheitseigenschaften, außer für die Over-Spending-Prevention.

Bei den Orakel-Anfragen der Abhebe-, Bezahl- und Einlöseprotokolle werden nun entsprechend die jeweilige BBS+A-Signatur auf die Nachricht $(u, t, o, \phi(\mathsf{x}))$ erstellt bzw. extrahiert.

Es bleibt zu zeigen, dass $\mathcal{B}$ im Fall (1.1) bei jeder $\mathcal{O}_{\mathcal{B}}^{\mathsf{W}}$- und $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{W}}$-Anfrage die Signature-of-Knowledge SoK_X ohne Kenntnis des geheimen Schlüssels $\mathsf{sk}_{\mathcal{B}} = x$ perfekt simulieren kann. Dazu verwenden wir obige Annahme 9.1 und Bemerkung 9.2.1. Der Angreifer $\mathcal{B}$ führt das Schnorr-Identifikationsverfahren mit dem Signatur-Orakel durch. Das Signatur-Orakel sendet ein Commitment $T \in \mathbb{G}_2$ an $\mathcal{B}$, welcher $T'_X := T$ definiert. Bei einer $\mathcal{O}_{\mathcal{B}}^{\mathsf{W}}$-Anfrage erhält $\mathcal{B}$ vom Angreifer $\mathcal{A}$ eine Zahl $\rho''_x \in \mathbb{Z}_p$ und bei einer $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{W}}$-Anfrage wählt $\mathcal{B}$ die Zahl $\rho''_x \in_{\mathcal{R}} \mathbb{Z}_p$ zufällig gemäß Protokoll. Dann berechnet $\mathcal{B}$ das Commitment $T_X = T'_X h^{\rho''_x}$ gemäß Protokoll und erzeugt den Hashwert $d = \mathsf{H}(T_X || I_{\mathcal{O}} || K)$ gemäß dem Random-Oracle. Der Angreifer $\mathcal{B}$ sendet nun den Hashwert d als Challenge an das Signatur-Orakel. Dieses antwortet mit einer Response $z \in \mathbb{Z}_p$ mit $T = h^z X^{-d}$. Der Angreifer $\mathcal{B}$ berechnet die Response $z_x = z + \rho''_x \mod p$, so dass wie gewünscht $T_X = T'_X h^{\rho''_x} = T h^{\rho''_x} = h^z X^{-d} h^{\rho''_x} = h^{z_x} X^{-d}$ gilt und (d, z_x) die korrekte Verteilung besitzt.

Anonymität: Die Anonymität des Geldsystems FTGO-I folgt direkt aus der Anonymität des fairen teilbaren Geldsystems FTG-I. Um zu modellieren, dass sich der Observer möglicherweise nicht gemäß Protokoll verhält, übernimmt ein Angreifer $\mathcal{A}_{\mathcal{O}}$ die Rolle der Observer. Allerdings können die Angreifer $\mathcal{A}$ und $\mathcal{A}_{\mathcal{O}}$ nach der Initialisierung nicht miteinander kommunizieren. Dadurch wird modelliert, dass die Observer nach Ausgabe an die Kunden nicht wieder in den Besitz des Angreifers $\mathcal{A}$ gelangen.

Die einzige Änderung im Vergleich zu FTG-I betrifft nur die Simulation der Münzen coin_b und $\mathsf{coin}_{\bar{b}}$ in der Challenge-Phase. Der Angreifer $\mathcal{B}$ kann beide Münzen dabei ohne $\mathcal{A}_{\mathcal{O}}$ simulieren. Zur Simulation der Münze coin_b wählt $\mathcal{B}$ lediglich zusätzlich drei zufällige Elemente $C_{1,b}, C_{2,b} \in_{\mathcal{R}} \mathbb{G}_1$ sowie $C_{3,b} \in_{\mathcal{R}} \mathbb{G}_2$ und sendet diese ebenfalls an $\mathcal{A}$. Anschließend simuliert $\mathcal{B}$ die Signature-of-Knowledge SoK_b sowie $SoK_{\sigma,b}$ perfekt (vgl. auch Abschnitt A.5 und Abschnitt A.6). Da der Angreifer $\mathcal{A}$ keinen Kontakt zu $\mathcal{A}_{\mathcal{O}}$ hat, bemerkt $\mathcal{A}$ nicht, dass die Münze ohne Kommunikation mit $\mathcal{A}_{\mathcal{O}}$ erstellt wurde. Dann sendet $\mathcal{B}$ die Münze $\mathsf{coin}_b = (k_b, S_b, T_b, T_{C,b}, R_b, A_{1,b}, B_{1,b}, B_{2,b}, C_{1,b}, C_{2,b}, C_{3,b}, SoK_b, SoK_{\sigma,b}, \mathsf{ts}_b, \mathsf{pk}_b)$ an den Angreifer $\mathcal{A}$. Analog verhält sich $\mathcal{B}$ zur Simulation der Münze $\mathsf{coin}_{\bar{b}}$.

Double-Spending-Beschuldigung: Die Double-Spending-Beschuldigung des Geldsystems FTGO-I folgt direkt aus der Double-Spending-Beschuldigung des fairen teilbaren Geldsystems FTG-I. Auch hier darf der Angreifer $\mathcal{A}$ die Rolle der Observer übernehmen.

Abhebe-Beschuldigung: Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Abhebe-Beschuldigung des Systems bricht. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das DLog-Problem in der Gruppe $\mathbb{G}_p$ zu lösen.

Der Angreifer $\mathcal{B}$ erhält die Parameter $\left(p, \mathbb{G}'_p, \mathfrak{g}, \mathfrak{g}^a\right)$ und soll den diskreten Logarithmus $a \in \mathbb{Z}_p^*$ berechnen.

Simulation: Der Angreifer $\mathcal{B}$ simuliert die Systemparameter sowie die Orakel-Anfragen wie bei der Double-Spending-Beschuldigung.

Spielende: Der Angreifer $\mathcal{A}$ gibt mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ eine Protokollansicht $\mathsf{view} = (\mathbb{T} = (\mathsf{pk}_\mathcal{K}, I_\mathcal{O}, C, E, A_0, SoK_\mathsf{W}, \mathsf{ts}, K, d), \sigma' = (\Sigma, e, s'', t'', z_x))$ mit $\mathsf{pk}_\mathcal{K} \in \mathcal{U}_H$ aus, so dass $\mathsf{VerifyWithdraw}(\mathsf{sp}, \mathsf{pk}_\mathcal{B}, \mathsf{pk}_\mathcal{D}, \mathsf{pk}_\mathcal{K}, \mathsf{view}) = 1$ gilt, obwohl $(\cdot, \mathbb{T}) \notin \mathbb{T}_H$ ist.

Wir müssen zeigen, dass SoK_W, außer mit vernachlässigbarer Wahrscheinlichkeit, von $\mathcal{A}$ erstellt wurde.

Angenommen, SoK_W wurde von $\mathcal{B}$ erstellt. Da $(\cdot, \mathbb{T}) \notin \mathbb{T}_H$ ist, gibt es einen Eintrag $\mathbb{T}' = (\mathsf{pk}'_\mathcal{K}, I'_\mathcal{O}, C', E', A'_0, SoK'_\mathsf{W}, \mathsf{ts}', K', d') \neq \mathbb{T}$ mit $SoK'_\mathsf{W} = SoK_\mathsf{W}$ und $(\cdot, \mathbb{T}') \in \mathbb{T}_H$.

Falls $(\mathsf{pk}_\mathcal{K}, I_\mathcal{O}, C, E, A_0, \mathsf{ts}, K) \neq (\mathsf{pk}'_\mathcal{K}, I'_\mathcal{O}, C', E', A'_0, \mathsf{ts}', K')$ gilt, muss für die Challenge $c = \mathsf{H}(\mathsf{pk}_\mathcal{K}||C||E||A_0|| \cdots ||I_\mathcal{O}||\mathsf{ts}||K) = \mathsf{H}(\mathsf{pk}'_\mathcal{K}||C'||E'||A'_0|| \cdots ||I'_\mathcal{O}||\mathsf{ts}'||K') = c'$ gelten, was nur mit vernachlässigbarer Wahrscheinlichkeit möglich ist.

Falls $(\mathsf{pk}_\mathcal{K}, I_\mathcal{O}, C, E, A_0, \mathsf{ts}, K) = (\mathsf{pk}'_\mathcal{K}, I'_\mathcal{O}, C', E', A'_0, \mathsf{ts}', K')$ gilt, muss $d \neq d'$ gelten, da nach Voraussetzung $\mathbb{T}' \neq \mathbb{T}$ ist. Sei $T^\star_X$ das vom Angreifer $\mathcal{B}$ bzgl. $\mathbb{T}'$ erzeugte Commitment, das zur Berechnung der Challenge c' mit in die Hashfunktion einfließt. Damit $\mathsf{VerifyWithdraw}(\mathsf{sp}, \mathsf{pk}_\mathcal{B}, \mathsf{pk}_\mathcal{D}, \mathsf{pk}_\mathcal{K}, \mathsf{view}) = 1$ gilt, müssen $d = \mathsf{H}\left(\widetilde{T_X}||I_\mathcal{O}||K\right)$ und $c = \mathsf{H}\left(\mathsf{pk}_\mathcal{K}||C||E||A_0|| \cdots ||\widetilde{T_X}||I_\mathcal{O}||\mathsf{ts}||K\right)$ für $\widetilde{T_X} = h^{z_x} X^{-d}$ gelten. Da $\mathsf{H}\left(\widetilde{T_X}||I_\mathcal{O}||K\right) = d \neq d' = \mathsf{H}\left(T^\star_X||I'_\mathcal{O}||K'\right)$ mit $(I_\mathcal{O}, K) = (I'_\mathcal{O}, K')$ ist, folgt $\widetilde{T_X} \neq T^\star_X$. Allerdings gilt für die Challenge $c = \mathsf{H}\left(\cdots||\widetilde{T_X}||I_\mathcal{O}||\mathsf{ts}||K\right) = \mathsf{H}\left(\cdots||T^\star_X||I'_\mathcal{O}||\mathsf{ts}'||K'\right) = c'$, was nur mit vernachlässigbarer Wahrscheinlichkeit möglich ist.

Somit wurde SoK_W, außer mit vernachlässigbarer Wahrscheinlichkeit, von $\mathcal{A}$ erstellt und $\mathcal{B}$ kann den privaten Schlüssel $u \in \mathbb{Z}_p$ mit $\mathsf{pk}_\mathcal{K} = \mathfrak{g}^u$ extrahieren und somit den diskreten Logarithmus $a = u - \gamma \mod p$ berechnen.

Somit löst $\mathcal{B}$ das DLog-Problem in $\mathbb{G}_p$ mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$, wobei ν

Da die beiden Sicherheitseigenschaften **korrekte Kundenverfolgung** sowie **korrekte Münzverfolgung** erneut direkt aus der Unfälschbarkeit dieses Geldsystems folgen, folgen diese Sicherheitseigenschaften direkt aus denen des fairen teilbaren Geldsystems FTG-I. Analog folgen die Sicherheitseigenschaften **Kundenverfolgung-Beschuldigung** und **Münzverfolgung-Beschuldigung** ebenfalls direkt aus denen des fairen teilbaren Geldsystems FTG-I.

Over-Spending-Prevention: Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Over-Spending-Prevention des Systems bricht. Seien $q_{W,1}$ die Anzahl der Anfragen an das Orakel $\mathcal{O}^{\mathsf{W}}_{\mathcal{B}}$, $q_{W,2}$ die Anzahl der Anfragen an das Orakel $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$, q_S die Anzahl der Anfragen an das Orakel $\mathcal{O}^{\mathsf{S}}_{\mathcal{H}}$ und q_V die Anzahl der Anfragen an das Orakel $\mathcal{O}^{\mathsf{VW}}_{\mathcal{B}}$. Sei k_1 der Wert der Münzen, die $\mathcal{A}$ in den $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$-Anfragen erhält. Wir konstruieren einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$, der den Angreifer $\mathcal{A}$ dazu verwendet, das q-SDH-Problem mit $q \leq q_{W,1} + q_{W,2} + q_S + q_V$ zu lösen (bzw. $q \leq q_{W,1} + q_{W,2} + q_S + q_V + 1$ für ein Typ-1- und Typ-2-Pairing, vgl Unterabschnitt 3.2.3).

Da die Over-Spending-Prevention nur dann gewährleistet ist, falls der Angreifer $\mathcal{A}$ die Observer nicht manipulieren kann, werden die Orakel $\mathcal{O}^{\mathsf{A}}_{\mathcal{B}}, \mathcal{O}^{\mathsf{W}}_{\mathcal{B}}$ und $\mathcal{O}^{\mathsf{S}}_{\mathcal{H}}$ entsprechend durch die Orakel $\mathcal{O}^{\mathsf{A}}_{\mathcal{O},\mathcal{B}}, \mathcal{O}^{\mathsf{W}}_{\mathcal{O},\mathcal{B}}$ und $\mathcal{O}^{\mathsf{S}}_{\mathcal{O},\mathcal{H}}$ ersetzt.

Am Ende hat $\mathcal{A}$ mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ durch die Orakel $\mathcal{O}^{\mathsf{S}}_{\mathcal{O},\mathcal{H}}$ und $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$ Münzen im Wert von $\$^* = \$ + k^*$ für $k^* \geq 1$ ausgegeben oder eingelöst. Da der Angreifer $\mathcal{A}$ durch die $\mathcal{O}^{\mathsf{W}}_{\mathcal{O},\mathcal{B}}$-Anfragen $q_{W,1} \cdot K$ und durch die $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$-Anfragen k_1 Serienschlüssel erhält, ist $\$ = q_{W,1} \cdot K + k_1$.

Unabhängig davon, wie $\mathcal{B}$ die Orakel-Anfragen simuliert, werden zunächst die verschiedenen Fälle aufgelistet, in denen der Angreifer $\mathcal{A}$ das Spiel gewinnt.

Die Fälle A bis F sind genau wie bei der Unfälschbarkeit vernachlässigbar.

Insgesamt wurden durch den Angreifer $\mathcal{A}$ Münzen im Wert von $\$^* > q_{W,1} \cdot K + k_1$ ausgegeben oder eingelöst. Dann gewinnt $\mathcal{A}$ das Spiel, wenn alle $\$^* = q_{W,1} \cdot K + k_1 + k^*$ Serienschlüssel für $k^* \geq 1$ verschieden sind, oder einige Serienschlüssel mehrmals ausgegeben oder eingelöst werden. Durch die Unfälschbarkeit wurde bereits gezeigt, dass nur mit vernachlässigbarer Wahrscheinlichkeit alle $\$^*$ Serienschlüssel verschieden sind (sogar wenn der Angreifer $\mathcal{A}$ die geheimen Schlüssel der Observer kennt). Somit müssen wir nur den Fall betrachten, dass einige Serienschlüssel mehrmals ausgegeben oder eingelöst werden.

Dann gibt es die beiden Möglichkeiten, dass der Angreifer $\mathcal{A}$ (1) jede Münze gemäß dem Bezahlprotokoll generiert oder (2) mindestens eine Münze nicht gemäß dem Bezahlprotokoll generiert. Es wird gezeigt, dass $\mathcal{A}$ im ersten Fall eine Signature-of-Knowledge SoK_X fälschen muss, damit der Observer den Zähler J erhöht und im zweiten Fall eine BBS+-Signatur bzgl. eines Observers fälschen muss (da $\mathcal{A}$ nur mit vernachlässigbarer Wahrscheinlichkeit eine BBS+A-Signatur der Bank fälschen kann). Beide Fälle können

auf die Sicherheit des BBS+A-Signaturverfahrens reduziert werden. Des Weiteren wird im Fall (1) bewiesen, dass bei einer $\mathcal{O}^{\mathsf{S}}_{\mathcal{O},\mathcal{H}}$- bzw. einer $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$-Anfrage nur mit vernachlässigbarer Wahrscheinlichkeit $c_3 \neq c_6$ gilt, da dies für den Fall (2) benötigt wird. Im Folgenden wird die Konstruktion von $\mathcal{B}$ detailliert beschrieben.

Fall (1): Der Angreifer $\mathcal{B}$ erhält den öffentlichen Schlüssel $(p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \hat{e}, g, g_0, g_1, g_2, g_3, u_0, \ldots, u_K, h, h_1, \ldots, h_K, X)$ des BBS+A-Signaturverfahrens und darf q Signatur-Anfragen an das Signatur-Orakel stellen. Zusätzlich darf $\mathcal{B}$ das Schnorr-Identifikationsverfahren polynomiell häufig mit dem Signatur-Orakel durchführen (vgl. Annahme 9.1 und Bemerkung 9.2.1).

Simulation der Initialisierung: Der Angreifer $\mathcal{B}$ generiert eine zyklische Gruppe $\mathbb{G}_p$ der Ordnung p, wählt zufällig zwei Zahlen $y, z \in_{\mathcal{R}} \mathbb{Z}^*_p$ sowie einen Generator $\mathfrak{g} \in_{\mathcal{R}} \mathbb{G}_p$, berechnet die Generatoren $\mathfrak{g}_0 = \mathfrak{g}^y$ sowie $Z = h^z$ und definiert $g_{\mathcal{O}} := g_3$. Anschließend sendet $\mathcal{B}$ die perfekt simulierten Systemparameter $\mathsf{sp} = (p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \mathbb{G}'_p, \hat{e}, g, g_0, g_1, g_2, g_{\mathcal{O}}, u_0, \ldots, u_K, h, h_1, \ldots, h_K, \mathfrak{g})$ sowie die perfekt simulierten öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}} = X$ und $\mathsf{pk}_{\mathcal{D}} = (\mathfrak{g}_0, Z)$ an den Angreifer $\mathcal{A}$.

Simulation von $\mathcal{O}^{\mathsf{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{A}}_{\mathcal{O},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wählt $\mathcal{B}$ insbesondere den privaten Schlüssel $o \in_{\mathcal{R}} \mathbb{Z}^*_p$ und speichert einen Eintrag $(I, I_{\mathcal{O}}, O, o, J)$ mit $I_{\mathcal{O}} = g^o_{\mathcal{O}}, O = h^o$ und $J = 0$ in einer Liste $\mathcal{U}_{\mathcal{A}}$ ab.

Simulation von $\mathcal{O}^{\mathsf{A}}_{\mathcal{K},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Anschließend speichert $\mathcal{B}$ einen Eintrag $(I, u, I_{\mathcal{O}}, O, o, J)$ mit $I = \mathfrak{g}^u, I_{\mathcal{O}} = g^o_{\mathcal{O}}, O = h^o$ und $J = 0$ in einer Liste $\mathcal{U}_H$ ab.

Simulation von $\mathcal{O}^{\mathsf{W}}_{\mathcal{O},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ führt das Schnorr-Identifikationsverfahren mit dem Signatur-Orakel durch und erhält ein Commitment $T \in \mathbb{G}_2$. Der Angreifer $\mathcal{B}$ definiert $T'_X := T$ und sendet T'_X an den Angreifer $\mathcal{A}$. Der Angreifer $\mathcal{A}$ sendet die Elemente C, E, A_0 sowie die Zahl ρ''_x an $\mathcal{B}$ und erstellt die Signature-of-Knowledge SoK_{W}. Dann berechnet $\mathcal{B}$ das Commitment $T_X = T'_X h^{\rho''_x}$ gemäß Protokoll und erzeugt den Hashwert $d = \mathsf{H}(T_X||I_{\mathcal{O}}||K)$ gemäß dem Random-Oracle. Der Angreifer $\mathcal{B}$ sendet nun den Hashwert d als Challenge an das Signatur-Orakel. Dieses antwortet mit einer Response $z \in \mathbb{Z}_p$ mit $T = h^z X^{-d}$. Der Angreifer $\mathcal{B}$ berechnet die Response $z_x = z + \rho''_x \mod p$, so dass wie gewünscht $T_X = T'_X h^{\rho''_x} = T h^{\rho''_x} = h^z X^{-d} h^{\rho''_x} = h^{z_x} X^{-d}$ gilt und (d, z_x) die korrekte Verteilung besitzt. Somit hat $\mathcal{B}$ die Signature-of-Knowledge SoK_X perfekt simuliert. Weiter simuliert der Angreifer $\mathcal{B}$ nun die BBS+A-Signatur analog zum Issue-Protokoll des BBS+A-Signaturverfahrens wie in Unterabschnitt 3.2.3 beschrieben, wobei zusätzlich die dem Angreifer $\mathcal{B}$ bekannte Zahl $o \in \mathbb{Z}_p$ Teil der zu signierenden Nachricht ist. Dann speichert $\mathcal{B}$ den Eintrag $(i_{\mathcal{A}}, \mathsf{view} = (\mathbb{T} = (I, I_{\mathcal{O}}, C, E, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K, d), \sigma' =$

$(\Sigma, e, s'', t'', z_x)), (s', u, t', o, \phi(\mathsf{x})))$ in einer Liste $\mathsf{view}_{\mathcal{A}}$ sowie einen Eintrag $(I_{\mathcal{O}}, K, T_X, SoK_X)$ in einer Liste $\sigma_{\mathcal{A}}$ ab, erhöht den Wert \$ um K und den Zähler $i_{\mathcal{A}}$ um 1. Falls $\mathcal{B}$ eine Anfrage nicht simulieren kann, bricht $\mathcal{B}$ ab. Dies passiert aber nur mit vernachlässigbarer Wahrscheinlichkeit (siehe Unterabschnitt 3.2.3).

Der Angreifer $\mathcal{B}$ benötigt insgesamt $q_{W,1}$ Anfragen an das Signatur-Orakel.

Nun erhält $\mathcal{B}$ (in der Rolle des Observers) eine Signature-of-Knowledge SoK_X. Falls $SoK_X \in \sigma_{\mathcal{A}} \cup \sigma_H$ ist (Definition von σ_H siehe Simulation von $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$), verhält sich $\mathcal{B}$ wie das Orakel. Falls $SoK_X \notin \sigma_{\mathcal{A}} \cup \sigma_H$ ist, wurde SoK_X nicht von $\mathcal{B}$ und somit vom Angreifer $\mathcal{A}$ erstellt. In diesem Fall verhält sich $\mathcal{B}$ wie das Orakel, außer dass $\mathcal{B}$ den Angreifer $\mathcal{A}$ während der Durchführung von SoK_X zurückspult, um den geheimen Schlüssel $x \in \mathbb{Z}_p^*$ zu extrahieren.

Somit wird nach erfolgreicher Durchführung der Wert J bzgl. des Kunden $\mathsf{pk}_{\mathcal{K}}$ um K erhöht. Des Weiteren wird nach erfolgreicher Durchführung der Eintrag $\left(I_{\mathcal{O}}, K, \widetilde{T_X}, SoK_X\right)$ in einer Liste $\mathbb{D}_\sigma$ gespeichert. (Falls sich der Eintrag also bereits in dieser Liste befindet, ist insbesondere $d \in \mathbb{D}$ und die Durchführung somit nicht erfolgreich, da der Observer im Protokoll dann abbricht.)

Es ist zu beachten, dass es auch möglich ist, dass sich der Angreifer $\mathcal{A}$ nicht gemäß Protokoll verhält und somit keine Signature-of-Knowledge SoK_X von $\mathcal{B}$ erhält, aber dennoch eine Signature-of-Knowledge SoK_X an $\mathcal{B}$ (in der Rolle des Observers) sendet.

Simulation von $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie der ehrliche Kunde und simuliert die BBS+A-Signatur perfekt mit Hilfe des Signatur-Orakels, da $\mathcal{B}$ die zu signierende Nachricht selbst wählt. Weiter simuliert $\mathcal{B}$ wie bei einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{O},\mathcal{B}}$-Anfrage die Signature-of-Knowledge SoK_X perfekt. Dann speichert $\mathcal{B}$ den Eintrag $(i, \mathbb{W}_i = (u, t, o, \Sigma, e, s', s'', \mathsf{info}), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$, einen Eintrag $(i, \mathsf{view} = (\mathbb{T} = (I, I_{\mathcal{O}}, C, E, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K, d), \sigma' = (\Sigma, e, s'', t'', z_x))$ in einer Liste view_H sowie einen Eintrag $(I_{\mathcal{O}}, K, T_X, SoK_X)$ in einer Liste σ_H ab und erhöht den Zähler i um 1. Der Angreifer $\mathcal{B}$ benötigt insgesamt $q_{W,2}$ Anfragen an das Signatur-Orakel.

Simulation von $\mathcal{O}^{\mathsf{S}}_{\mathcal{O},\mathcal{H}}$: Der Angreifer $\mathcal{B}$ verhält sich für die Simulation des Observers wie das Orakel.

Für die Simulation des Händlers geht der Angreifer $\mathcal{B}$ folgendermaßen vor: Nach den Fällen A bis C wurde die Münze coin nur mit vernachlässigbarer Wahrscheinlichkeit nicht von $\mathcal{A}$ erstellt. Somit verhält sich der Angreifer $\mathcal{B}$ wie das Orakel, außer dass $\mathcal{B}$ den Angreifer $\mathcal{A}$ während der Durchführung von SoK und SoK_σ zurückspult, um die Zahlen $c_1, c_2, c_3, c_4, c_5, c_6 \in \mathbb{Z}_p$ sowie das BBS+A Nachrichten-Signatur-Paar $((u, t, o, \phi(\mathsf{x})), (\Sigma, e, s))$ mit $O = h^o$ zu extrahieren. Dies ist (im Gegensatz zu Fall (2)) möglich, da der Angreifer $\mathcal{B}$ den entsprechenden privaten

Schlüssel $\mathsf{sk}_{\mathcal{O}}$ kennt und (in der Rolle des Observers) auf jede Challenge korrekt antworten kann (siehe unten).

Bei der Extraktion der Zahlen aus den beiden Signatures-of-Knowledge geht $\mathcal{B}$ folgendermaßen vor: Der Angreifer $\mathcal{A}$ wird zunächst nach der Challenge c für SoK und anschließend nach der Challenge c_σ für SoK_σ fragen, da c zur Berechnung von c_σ mit in die Hashfunktion einfließt. Dann wird $\mathcal{A}$ beide Signatures-of-Knowledge SoK und SoK_σ an $\mathcal{B}$ senden. Danach spult $\mathcal{B}$ den Angreifer $\mathcal{A}$ zu dem Zeitpunkt zurück, an dem $\mathcal{A}$ nach dem Wert c für SoK fragt. Jetzt antwortet $\mathcal{B}$ mit einer anderen Challenge $c' \neq c$. (Da $\mathcal{B}$ den privaten Schlüssel $\mathsf{sk}_{\mathcal{O}}$ kennt, kann $\mathcal{B}$ in der Rolle des Observers immer die korrekte Response $\dot{z_o}$ berechnen.) Erst danach wird $\mathcal{A}$ nach der Challenge c_σ fragen, die $\mathcal{B}$ gemäß dem Random-Orakel $\mathcal{O}^{\mathsf{H}}$ erstellt. Dann wird $\mathcal{A}$ erneut beide Signatures-of-Knowledge SoK und SoK_σ an $\mathcal{B}$ senden. Da $\mathcal{B}$ für SoK nun aber zwei verschiedene Challenge-Response-Tupel kennt, kann $\mathcal{B}$ die Zahlen aus SoK extrahieren. Nun wird $\mathcal{A}$ zu dem Zeitpunkt zurückgespult, an dem $\mathcal{A}$ nach dem Wert c_σ für SoK_σ fragt. Jetzt antwortet $\mathcal{B}$ mit einer anderen Challenge $c'_\sigma \neq c_\sigma$ und kann nach dem Erhalt von SoK und SoK_σ nun ebenfalls die Zahlen aus SoK_σ extrahieren.

Falls $c_3 \neq c_6$ mit $C_1 = g_0^{c_1} g_1^{c_2} u_0^{c_3} = g_0^{c_4} g_1^{c_5} u_0^{c_6}$ gilt, kennt $\mathcal{B}$ zwei verschiedene Darstellungen von C_1 bzgl. (g_0, g_1, u_0), was nur mit vernachlässigbarer Wahrscheinlichkeit möglich ist. Somit ist der Fall $c_3 \neq c_6$ vernachlässigbar. Dies wird für den folgenden Fall (2) benötigt.

Falls $O \notin \mathcal{U}_{\mathcal{A}}$ ist, gibt es die Fälle (1.1) $O \notin \mathcal{U}_{\mathcal{A}} \cup \mathcal{U}_H$ und (1.2) $O \in \mathcal{U}_H$. Der Fall (1.1) ist ein Widerspruch zur Unfälschbarkeit (Fall (1.1)) dieses Geldsystems, da dann das Nachrichten-Signatur-Paar $((u, t, o, \phi(\mathsf{x})), (\Sigma, e, s))$ nie während einer Orakel-Anfrage erstellt wurde. Im Fall (1.2) gibt es die Möglichkeiten, dass das Nachrichten-Signatur-Paar $((u, t, o, \phi(\mathsf{x})), (\Sigma, e, s))$ nie, oder mindestens einmal bei einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$-Anfrage erstellt wurde. Dies ist ein Widerspruch zur Unfälschbarkeit (Fall (1.1) oder (1.2)) dieses Geldsystems. Somit ist der Fall $O \notin \mathcal{U}_{\mathcal{A}}$ vernachlässigbar. Auch dies wird für den folgenden Fall (2) benötigt.

Nach erfolgreicher Durchführung wird der Wert \$* um k erhöht, der Wert J bzgl. des Kunden $\mathsf{pk}_{\mathcal{K}}$ um k verringert, ein Eintrag (j, coin) in einer Liste $\mathbb{C}_{\mathcal{A}}$ sowie ein Eintrag $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts})$ in einer Liste $\mathbb{S}$ gespeichert und der Zähler j um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird der Wert \$ um k erhöht, ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_{H,\mathcal{A}}$ gespeichert.

Simulation von $\mathcal{O}^{\mathsf{S}}_{\mathcal{K},\mathcal{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_H$, ein Eintrag (j, coin) in einer Liste $\mathbb{C}$ sowie ein Eintrag $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts})$ in einer Liste $\mathbb{S}$ gespeichert und der Zähler j um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$: Falls $\mathsf{coin} \in \mathbb{D}_{\mathcal{A}}$ verhält sich $\mathcal{B}$ wie das Orakel und gibt eine Fehlermeldung $\perp$ aus.

Falls $\mathsf{coin} \in \mathbb{C}_{H,\mathcal{A}} \setminus \mathbb{D}_{\mathcal{A}}$ ist, verhält sich $\mathcal{B}$ wie das Orakel. Somit wird nach erfolgreicher Durchführung der Wert \$* um k erhöht und die Münze coin in einer Liste $\mathbb{D}_{\mathcal{A}}$ gespeichert.

Falls $\mathsf{coin} \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{D}_{\mathcal{A}}$ ist, wurde die Münze coin nach den Fällen D bis F nur mit vernachlässigbarer Wahrscheinlichkeit nicht von $\mathcal{A}$ erstellt. Somit verhält sich der Angreifer $\mathcal{B}$ wie die ehrliche Bank, außer dass $\mathcal{B}$ den Angreifer $\mathcal{A}$ während der Durchführung von SoK und SoK_σ zurückspult (vgl. $\mathcal{O}^{\mathsf{S}}_{\mathcal{O},\mathcal{H}}$-Anfrage), um die Zahlen $c_1, c_2, c_3, c_4, c_5, c_6 \in \mathbb{Z}_p$ sowie das BBS+A Nachrichten-Signatur-Paar $((u, t, o, \phi(\mathsf{x})), (\Sigma, e, s))$ mit $O = h^o$ zu extrahieren. Falls die Werte nicht extrahiert werden können, bricht $\mathcal{B}$ ab. Dies passiert aber nur mit vernachlässigbarer Wahrscheinlichkeit. Anschließend wird nach erfolgreicher Durchführung der Wert \$* um k erhöht und die Münze coin in einer Liste $\mathbb{D}_{\mathcal{A}}$ gespeichert.

Wie bei der $\mathcal{O}^{\mathsf{S}}_{\mathcal{O},\mathcal{H}}$-Anfrage, sind die beiden Fälle $c_3 \neq c_6$ und $O \notin \mathcal{U}_{\mathcal{A}}$ erneut vernachlässigbar, was für den Fall (2) benötigt wird.

Simulation von $\mathcal{O}^{\mathsf{D}}_{\mathcal{H},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird ein Eintrag (j, coin) in einer Liste $\mathbb{D}_H$ gespeichert.

Simulation von $\mathcal{O}^{\mathsf{VW}}_{\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verwendet die Liste $\mathsf{view}_{\mathcal{A}}$ und simuliert die Signatur perfekt mit dem Signatur-Orakel. Der Angreifer $\mathcal{B}$ benötigt insgesamt q_V Anfragen an das Signatur-Orakel.

Simulation von $\mathcal{O}^{\mathsf{UT}}_{\mathcal{B},\mathcal{D}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den geheimen Schlüssel $y \in \mathbb{Z}_p^*$ kennt.

Simulation von $\mathcal{O}^{\mathsf{CT}}_{\mathcal{B},\mathcal{D}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den geheimen Schlüssel $z \in \mathbb{Z}_p^*$ kennt.

Spielende: Mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ gilt \$* > \$.

Nach Voraussetzung hat der Angreifer $\mathcal{A}$ jede Münze gemäß dem Bezahlprotokoll generiert. Daher konnte $\mathcal{A}$ durch die $\mathcal{O}^{\mathsf{W}}_{\mathcal{O},\mathcal{B}}$-Anfragen die Zähler der Observer aller korrupter Kunden mindestens um den Wert \$* erhöhen, obwohl $\mathcal{B}$ für diese korrupten Kunden nur Signatures-of-Knowledge für einen maximalen Wert von \$ erstellt hat. Mit anderen Worten befinden sich in der Liste $\sigma_{\mathcal{A}}$ Einträge $(I_{\mathcal{O},1}, K_1, T_{X,1}, SoK_{X,1}), \ldots, (I_{\mathcal{O},n_1}, K_{n_1}, T_{X,n_1}, SoK_{X,n_1})$ mit $\$ = \sum_{n=1}^{n_1} K_n$ und in der Liste $\mathbb{D}_\sigma$ Einträge $\left(I^\star_{\mathcal{O},1}, K^\star_1, \widetilde{T^\star_{X,1}}, SoK^\star_{X,1}\right), \ldots,$ $\left(I^\star_{\mathcal{O},n_2}, K^\star_{n_2}, \widetilde{T^\star_{X,n_2}}, SoK^\star_{X,n_2}\right)$ mit $\$^* = \sum_{n=1}^{n_2} K^\star_n > \$$, für zwei Zahlen $n_1, n_2 \in \mathbb{N}$.

Nach Ablauf der $\mathcal{O}^{\mathsf{W}}_{\mathcal{O},\mathcal{B}}$-Anfragen sind alle Einträge in der Liste $\mathbb{D}_\sigma$ paarweise verschieden. Folglich muss ein Eintrag $\left(I^\star_{\mathcal{O}}, K^\star, \widetilde{T^\star_X}, SoK^\star_X\right) \in \mathbb{D}_\sigma \setminus \sigma_{\mathcal{A}}$ mit $\widetilde{T^\star_X} = h^{z^\star_x} X^{-d^\star}$ und $d^\star = \mathsf{H}\left(\widetilde{T^\star_X} || I^\star_{\mathcal{O}} || K^\star\right)$ existieren, wobei $SoK^\star_X = (d^\star, z^\star_x)$ ist.

Angenommen, $SoK^\star_X$ wurde von $\mathcal{B}$ erzeugt. Dann gibt es einen Eintrag $\left(I'_{\mathcal{O}}, K', \widetilde{T^\star_X}, SoK^\star_X\right) \in \sigma_{\mathcal{A}} \cup \sigma_H$. Falls $\left(I'_{\mathcal{O}}, K', \widetilde{T^\star_X}, SoK^\star_X\right) \in \sigma_{\mathcal{A}}$ ist, folgt $(I'_{\mathcal{O}}, K') \neq (I^\star_{\mathcal{O}}, K^\star)$ aus

$\left(I_{\mathcal{O}}^{\star}, K^{\star}, \widetilde{T_X^{\star}}, SoK_X^{\star}\right) \notin \sigma_{\mathcal{A}}$. Falls $\left(I_{\mathcal{O}}', K', \widetilde{T_X^{\star}}, SoK_X^{\star}\right) \in \sigma_H$ ist, folgt $(I_{\mathcal{O}}', K') \neq (I_{\mathcal{O}}^{\star}, K^{\star})$ aus $\left(I_{\mathcal{O}}^{\star}, K^{\star}, \widetilde{T_X^{\star}}, SoK_X^{\star}\right) \in \mathbb{D}_{\sigma}$, da in $\mathbb{D}_{\sigma}$ nur Einträge von korrupten Kunden gespeichert werden. Folglich gilt in beiden Fällen $(I_{\mathcal{O}}', K') \neq (I_{\mathcal{O}}^{\star}, K^{\star})$ und $d^{\star} = \mathsf{H}(\cdot||I_{\mathcal{O}}^{\star}||K^{\star}) = \mathsf{H}(\cdot||I_{\mathcal{O}}'||K')$, was nur mit vernachlässigbarer Wahrscheinlichkeit möglich ist.

Somit wurde $SoK_X^{\star}$, außer mit vernachlässigbarer Wahrscheinlichkeit, von $\mathcal{A}$ erstellt und daher gilt $SoK_X^{\star} \notin \sigma_{\mathcal{A}} \cup \sigma_H$. Folglich hat $\mathcal{B}$ die Zahl $x \in \mathbb{Z}_p^*$ mit $X = h^x$ während einer $\mathcal{O}_{\mathcal{O},\mathcal{B}}^{\mathsf{W}}$-Anfrage aus $SoK_X^{\star}$ extrahiert. Damit kennt $\mathcal{B}$ den geheimen Signaturschlüssel und bricht die Unfälschbarkeit des BBS+A-Signaturverfahrens (und löst somit das q-SDH-Problem) mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist.

Fall (2): Wie in Abschnitt 9.2 beschrieben, muss bei jedem Bezahlprotokoll der jeweilige Observer an der Erstellung der Signature-of-Knowledge SoK beteiligt sein. Dies entspricht genau einem Honest-Verifier-Zero-Knowledge-Proof-of-Knowledge über die Kenntnis einer Zahl $o \in \mathbb{Z}_p^*$ mit $I_{\mathcal{O}} = g_{\mathcal{O}}^o \in \mathbb{G}_1$ und $O = h^o \in \mathbb{G}_2$ und damit dem Schnorr-Identifikationsverfahren (vgl. Unterunterabschnitt 2.10.1.1 und Abschnitt A.5). Da der im Folgenden konstruierte Angreifer $\mathcal{B}$ dies ohne Kenntnis des privaten Schlüssels o nicht simulieren kann, verwenden wir erneut die Annahme 9.1.

Der Angreifer $\mathcal{B}$ erhält den öffentlichen Schlüssel $\mathsf{pk} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, g, g_0, g_1, g_{\mathcal{O}}, I_{\mathcal{O}}^*, u_0, \ldots, u_K, h, h_1, \ldots, h_K, O^*)$ des BBS+A-Signaturverfahrens mit $I_{\mathcal{O}}^* = g_{\mathcal{O}}^{o^*}$ sowie $O^* = h^{o^*}$ und darf q Anfragen an das Signatur-Orakel stellen. Dies entspricht dem öffentlichen Schlüssel pk gemäß Korollar 3.2.1 mit $L = 1$, wobei der geheime Schlüssel die Zahl $o^* \in_{\mathcal{R}} \mathbb{Z}_p^*$ ist. Zusätzlich darf $\mathcal{B}$ erneut das Schnorr-Identifikationsverfahren polynomiell häufig mit dem Signatur-Orakel durchführen.

Simulation der Initialisierung: Der Angreifer $\mathcal{B}$ generiert eine zyklische Gruppe $\mathbb{G}_p$ der Ordnung p, wählt zufällig drei Zahlen $x, y, z \in_{\mathcal{R}} \mathbb{Z}_p^*$ und zwei Generatoren $g_2 \in_{\mathcal{R}} \mathbb{G}_1$ sowie $\mathfrak{g} \in_{\mathcal{R}} \mathbb{G}_p$ und berechnet die Generatoren $\mathfrak{g}_0 = \mathfrak{g}^y, X = h^x$ sowie $Z = h^z$. Anschließend sendet $\mathcal{B}$ die perfekt simulierten Systemparameter $\mathsf{sp} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \mathbb{G}_p', \hat{e}, g, g_0, g_1, g_2, g_{\mathcal{O}}, u_0, \ldots, u_K, h, h_1, \ldots, h_K, \mathfrak{g})$ sowie die beiden perfekt simulierten öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}} = X$ und $\mathsf{pk}_{\mathcal{D}} = (\mathfrak{g}_0, Z)$ an den Angreifer $\mathcal{A}$.

Simulation von $\mathcal{O}^{\mathsf{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}_{\mathcal{O},\mathcal{B}}^{\mathsf{A}}$: Der Angreifer $\mathcal{B}$ wählt zufällig eine Zahl $\gamma \in_{\mathcal{R}} \mathbb{Z}_p^*$ und definiert $I_{\mathcal{O}} := I_{\mathcal{O}}^* g_{\mathcal{O}}^{\gamma}$ sowie $O := O^* h^{\gamma}$, so dass $o = o^* + \gamma \mod p$ gilt. Dann sendet $\mathcal{B}$ die Elemente $(I_{\mathcal{O}}, O)$ an $\mathcal{A}$ und speichert einen Eintrag $(I, I_{\mathcal{O}}, O, \gamma, J)$ mit $J = 0$ in einer Liste $\mathcal{U}_{\mathcal{A}}$ ab.

Simulation von $\mathcal{O}_{\mathcal{K},\mathcal{B}}^{\mathsf{A}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Anschließend speichert $\mathcal{B}$ einen Eintrag $(I, u, I_{\mathcal{O}}, O, o, J)$ mit $I = \mathfrak{g}^u, I_{\mathcal{O}} = g_{\mathcal{O}}^o, O = h^o$ und $J = 0$ in einer Liste $\mathcal{U}_H$ ab.

Simulation von $\mathcal{O}^{\mathsf{W}}_{\mathcal{O},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ die Zahl $x \in \mathbb{Z}_p^*$ kennt. Somit wird nach erfolgreicher Durchführung ein Eintrag $(i_{\mathcal{A}}, \mathsf{view} = (\mathbb{T} = (I, I_{\mathcal{O}}, C, E, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K, d), \sigma' = (\Sigma, e, s'', t'', z_x)))$ in einer Liste $\mathsf{view}_{\mathcal{A}}$ gespeichert, der Wert \$ sowie der Wert J bzgl. des Kunden $\mathsf{pk}_{\mathcal{K}}$ jeweils um K erhöht, der Zähler $i_{\mathcal{A}}$ um 1 erhöht und der in der Liste $\mathcal{U}_{\mathcal{A}}$ vorhandene Eintrag $(I, I_{\mathcal{O}}, O, \gamma, J)$ aktualisiert.

Simulation von $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ die Zahl $x \in \mathbb{Z}_p^*$ kennt. Somit wird ein Eintrag $(i, \mathbb{W}_i = (u, t, o, \Sigma, e, s', s'', \mathsf{info}), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$ sowie ein Eintrag $(i, \mathsf{view} = (\mathbb{T} = (I, I_{\mathcal{O}}, C, E, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K, d), \sigma' = (\Sigma, e, s'', t'', z_x))$ in einer Liste view_H gespeichert und der Zähler i um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{S}}_{\mathcal{O},\mathcal{H}}$: Sei $\widetilde{I_{\mathcal{O}}}$ die Identität des Observers und $\widetilde{k}$ der Münzwert, den der Observer als Eingabe erhält. Sei $\left(\widetilde{I}, \widetilde{I_{\mathcal{O}}}, \widetilde{O}, \widetilde{\gamma}, \widetilde{J}\right)$ der entsprechende Eintrag in der Liste $\mathcal{U}_{\mathcal{A}}$. Falls $\widetilde{k} > \widetilde{J}$ ist, sendet $\mathcal{B}$ das Symbol $\perp$ an $\mathcal{A}$. Der Angreifer $\mathcal{B}$ erhält (in der Rolle des Observers) eine Zahl $\widetilde{R}$. Dann sendet $\mathcal{B}$ die Nachricht $m = (m_1, \phi(\mathsf{x})) = (\widetilde{R}, \widetilde{k})$ an das Signatur-Orakel. Der Angreifer $\mathcal{B}$ erhält nun eine BBS+A-Signatur (Σ^*, e^*, r^*) mit $\hat{e}\left(\Sigma^*, O^* h^{e^*}\right) = \hat{e}\left(g g_0^{r^*} g_1^{\widetilde{R}} u_0^{\widetilde{k}}, h\right)$. Nun muss diese Signatur bzgl. des Schlüssels O^* zu einer gültigen Signatur bzgl. des Schlüssels $\widetilde{O}$ transformiert werden. Dazu definiert der Angreifer $\mathcal{B}$ nun $\widetilde{\Sigma'} := \Sigma^*, \widetilde{r} := r^*$ sowie $\widetilde{e'} := e^* - \widetilde{\gamma} \mod p$. Somit gilt $O^* h^{e^*} = h^{o^* + \widetilde{\gamma}} h^{e^* - \widetilde{\gamma}} = \widetilde{O} h^{\widetilde{e'}}$ und daher $\hat{e}\left(\widetilde{\Sigma'}, \widetilde{O} h^{\widetilde{e'}}\right) = \hat{e}\left(g g_0^{\widetilde{r}} g_1^{\widetilde{R}} u_0^{\widetilde{k}}, h\right)$. Dann sendet $\mathcal{B}$ das perfekt simulierte Tripel $(\widetilde{\Sigma'}, \widetilde{e'}, \widetilde{r})$ an $\mathcal{A}$. Anschließend verringert $\mathcal{B}$ den Zähler $\widetilde{J}$ um $\widetilde{k}$, aktualisiert entsprechend den Eintrag $\left(\widetilde{I}, \widetilde{I_{\mathcal{O}}}, \widetilde{O}, \widetilde{\gamma}, \widetilde{J}\right)$ in der Liste $\mathcal{U}_{\mathcal{A}}$ und speichert einen Eintrag $\left(\widetilde{O}, \left(\widetilde{R}, \widetilde{k}\right), \left(\widetilde{\Sigma'}, \widetilde{e'}, \widetilde{r}\right)\right)$ in einer Liste $\sigma_{\mathcal{A}}$ sowie das erhaltene Nachrichten-Signatur-Paar $\left(\left(\widetilde{R}, \widetilde{k}\right), (\Sigma^*, e^*, r^*)\right)$ in einer Liste $\sigma^*_{\mathcal{A}}$ ab.

Weiter muss $\mathcal{B}$ einen Teil von SoK simulieren, indem $\mathcal{B}$ zwei Commitments $T_1 \in \mathbb{G}_1$ und $T_2 \in \mathbb{G}_2$ sowie nach Erhalt einer Challenge $c \in \mathbb{Z}_p$ eine Response $\dot{z_o} \in \mathbb{Z}_p$ mit $T_1 = g_{\mathcal{O}}^{\dot{z_o}} \widetilde{I_{\mathcal{O}}}^{-c}$ und $T_2 = h^{\dot{z_o}} \widetilde{O}^{-c}$ an $\mathcal{A}$ sendet.

Dazu führt $\mathcal{B}$ das Schnorr-Identifikationsverfahren mit dem Signatur-Orakel durch. Das Signatur-Orakel sendet zwei Commitments $T_1 \in \mathbb{G}_1$ und $T_2 \in \mathbb{G}_2$, die $\mathcal{B}$ weiter an den Angreifer $\mathcal{A}$ leitet. Anschließend erhält $\mathcal{B}$ vom Angreifer $\mathcal{A}$ eine Challenge $c \in \mathbb{Z}_p$, die $\mathcal{B}$ weiter an das Signatur-Orakel leitet. Dieses antwortet mit einer Response $z \in \mathbb{Z}_p$ mit $T_1 = g_{\mathcal{O}}^z \left(I_{\mathcal{O}}^*\right)^{-c}$ und $T_2 = h^z \left(O^*\right)^{-c}$. Der Angreifer $\mathcal{B}$ berechnet die Response $\dot{z_o} = z + c\widetilde{\gamma} \mod p$, so dass wie gewünscht $T_1 = g_{\mathcal{O}}^z \left(I_{\mathcal{O}}^*\right)^{-c} = g_{\mathcal{O}}^z g_{\mathcal{O}}^{c\widetilde{\gamma}} \left(I_{\mathcal{O}}^*\right)^{-c} \left(g_{\mathcal{O}}^{\widetilde{\gamma}}\right)^{-c} = g_{\mathcal{O}}^{\dot{z_o}} \widetilde{I_{\mathcal{O}}}^{-c}$ sowie $T_2 = h^z \left(O^*\right)^{-c} = h^z h^{c\widetilde{\gamma}} \left(O^*\right)^{-c} \left(h^{\widetilde{\gamma}}\right)^{-c} = h^{\dot{z_o}} \widetilde{O}^{-c}$ gelten und sendet $\dot{z_o}$ an den Angreifer $\mathcal{A}$.

Nun sendet $\mathcal{A}$ die Münze coin (ohne $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts})$) an $\mathcal{B}$ (in der Rolle des Händlers). Nach den Fällen A bis C wurde die Münze coin nur mit vernachlässigbarer Wahr-

scheinlichkeit nicht von $\mathcal{A}$ erstellt. Somit verhält sich der Angreifer $\mathcal{B}$ wie der ehrliche Händler, außer dass $\mathcal{B}$ den Angreifer $\mathcal{A}$ während der Durchführung von SoK_σ zurückspult, um die Signatur (Σ', e', r) auf die Nachricht (R, k) sowie die Zahl $c_6 \in \mathbb{Z}_p$ zu extrahieren. Beachte, dass möglicherweise $(R, k, \Sigma', e', r) \neq \left(\widetilde{R}, \widetilde{k}, \widetilde{\Sigma'}, \widetilde{e'}, \widetilde{r}\right)$ ist.

Wir zeigen ausführlich, dass die Extraktion möglich ist und nicht der Simulation von SoK widerspricht. Dazu wird der Angreifer $\mathcal{A}$ nach der ersten Durchführung von SoK_σ zum Beginn des Protokolls zurückgespult. Der Angreifer $\mathcal{B}$ wird erneut dieselbe Zahl $\bar{R}$ erhalten. Nun sendet $\mathcal{B}$ erneut dieselben Commitments T_1 und T_2 an den Angreifer $\mathcal{A}$. An dieser Stelle müssen zwei Fälle betrachtet werden. Entweder stellt der Angreifer $\mathcal{A}$ die Anfrage für den Hashwert c_σ (2.1) erst nachdem $\mathcal{A}$ die Challenge c der Signature-of-Knowledge SoK an $\mathcal{B}$ (in der Rolle des Observers) gesendet hat, oder (2.2) bereits vorher.

Wir zeigen zunächst, dass der Fall (2.2) vernachlässigbar ist. Da der Hashwert c zur Berechnung der Challenge c_σ mit in die Hashfunktion einfließt, wird der Angreifer $\mathcal{A}$ zunächst das Random-Oracle bzgl. des Hashwertes c befragen. Nun antwortet $\mathcal{B}$ mit einem anderen Hashwert $c' \neq c$ auf die $\mathcal{O}^{\mathsf{H}}$-Anfrage, wobei c der von $\mathcal{B}$ erzeugte Hashwert vor dem Zurückspulen war. Nach Voraussetzung befragt der Angreifer $\mathcal{A}$ das Random-Oracle bzgl. des Hashwertes c_σ, bevor $\mathcal{A}$ die geblendete Challenge c an $\mathcal{B}$ sendet. Da die gesamte Signature-of-Knowledge SoK zur Berechnung der Challenge c_σ mit in die Hashfunktion einfließt, muss der Angreifer $\mathcal{A}$ also SoK an $\mathcal{B}$ senden. Somit hat $\mathcal{B}$ insgesamt zwei Challenge-Response-Tupel der Signature-of-Knowledge SoK bzgl. verschiedener Challenges $c \neq c'$ erhalten und kann daher alle Zahlen aus SoK extrahieren. Insbesondere kann $\mathcal{B}$ daher die Zahl $\widetilde{o} \in \mathbb{Z}_p$ mit $\widetilde{O} = h^{\widetilde{o}}$ extrahieren. Daher kann $\mathcal{B}$ den geheimen Signaturschlüssel $o^* = \widetilde{o} - \widetilde{\gamma} \mod p$ berechnen und somit die Unfälschbarkeit des BBS+A-Signaturverfahrens brechen. Daher ist der Fall (2.2) vernachlässigbar und wir gehen im Folgenden davon aus, dass Fall (2.1) eintritt.

Wenn der Angreifer $\mathcal{A}$ den Hashwert für c anfragt, antwortet $\mathcal{B}$ mit demselben Hashwert c auf die $\mathcal{O}^{\mathsf{H}}$-Anfrage. Nach Voraussetzung wird $\mathcal{A}$ den Hashwert c_σ erst anfragen, nachdem $\mathcal{A}$ die Challenge c an $\mathcal{B}$ gesendet hat. Da sich somit bis zu diesem Zeitpunkt nichts im Vergleich zum Protokoll vor dem Zurückspulen geändert hat, wird $\mathcal{B}$ auch dieselbe Challenge c von $\mathcal{A}$ erhalten. Somit kann $\mathcal{B}$ erneut mit der korrekten Response $\dot{z_o}$ antworten. Wenn $\mathcal{A}$ nun den Hashwert c_σ für SoK_σ anfragt, antwortet $\mathcal{B}$ mit einer anderen Zahl $c'_\sigma \neq c_\sigma$ und kann folglich anschließend die Zahlen aus SoK_σ extrahieren. Folglich kann $\mathcal{B}$ einerseits den nötigen Teil von SoK simulieren und andererseits die nötigen Zahlen aus SoK_σ extrahieren. Aus diesem Grund wird deutlich, warum beide Signatures-of-Knowledge benötigt werden und eine einzige Signature-of-Knowledge nicht ausreichend ist.

In Fall (1) wurde bereits gezeigt, dass nur mit vernachlässigbarer Wahrscheinlich-

keit $c_3 \neq c_6$ ist, wobei $C_3 = Oh^{c_3}$ gilt. Daher berechnet $\mathcal{B}$ das Element $O = C_3 h^{-c_6}$ und speichert den Eintrag $(O, (R, k), (\Sigma', e', r))$ in einer Liste $\mathbb{D}_\sigma$. Falls die Werte nicht extrahiert werden können, bricht $\mathcal{B}$ ab. Dies passiert aber nur mit vernachlässigbarer Wahrscheinlichkeit. Dann erhöht $\mathcal{B}$ nach erfolgreicher Durchführung den Wert \$* um k, speichert einen Eintrag (j, coin) in einer Liste $\mathbb{C}_\mathcal{A}$ sowie einen Eintrag $(\mathsf{pk}_\mathcal{H}, \mathsf{ts})$ in einer Liste $\mathbb{S}$ ab und erhöht den Zähler j um 1.

Damit wurde die gesamte Anfrage perfekt simuliert, wenn $\mathcal{B}$ die genannten Werte aus SoK_D sowie SoK_σ extrahiert.

Simulation von $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird der Wert \$ um k erhöht und ein Eintrag $(i, \mathsf{pk}_\mathcal{K}, \mathsf{coin})$ in einer Liste $\mathbb{C}_{H,\mathcal{A}}$ gespeichert.

Simulation von $\mathcal{O}^{\mathsf{S}}_{\mathcal{K},\mathcal{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird ein Eintrag $(i, \mathsf{pk}_\mathcal{K}, \mathsf{coin})$ in einer Liste $\mathbb{C}_H$, ein Eintrag (j, coin) in einer Liste $\mathbb{C}$ sowie ein Eintrag $(\mathsf{pk}_\mathcal{H}, \mathsf{ts})$ in einer Liste $\mathbb{S}$ gespeichert und der Zähler j um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$: Falls $\mathsf{coin} \in \mathbb{D}_\mathcal{A}$ verhält sich $\mathcal{B}$ wie das Orakel und gibt eine Fehlermeldung $\perp$ aus.

Falls $\mathsf{coin} \in \mathbb{C}_{H,\mathcal{A}} \setminus \mathbb{D}_\mathcal{A}$ ist, verhält sich $\mathcal{B}$ wie das Orakel. Somit wird nach erfolgreicher Durchführung der Wert \$* um k erhöht und die Münze coin in einer Liste $\mathbb{D}_\mathcal{A}$ gespeichert.

Falls $\mathsf{coin} \notin \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{D}_\mathcal{A}$ ist, wurde die Münze coin nach den Fällen D bis F nur mit vernachlässigbarer Wahrscheinlichkeit nicht von $\mathcal{A}$ erstellt. Somit verhält sich der Angreifer $\mathcal{B}$ wie die ehrliche Bank, außer dass $\mathcal{B}$ den Angreifer $\mathcal{A}$ während der Durchführung von SoK_σ zurückspult, um die Signatur (Σ', e', r) auf die Nachricht (R, k) sowie sowie die Zahl $c_6 \in \mathbb{Z}_p$ zu extrahieren. In Fall (1) wurde bereits gezeigt, dass nur mit vernachlässigbarer Wahrscheinlichkeit $c_3 \neq c_6$ ist, wobei $C_3 = Oh^{c_3}$ gilt. Daher berechnet $\mathcal{B}$ das Element $O = C_3 h^{-c_6}$ und speichert den Eintrag $(O, (R, k), (\Sigma', e', r))$ in einer Liste $\mathbb{D}_\sigma$. Falls die Werte nicht extrahiert werden können, bricht $\mathcal{B}$ ab. Dies passiert aber nur mit vernachlässigbarer Wahrscheinlichkeit. Anschließend wird nach erfolgreicher Durchführung der Wert \$* um k erhöht und die Münze coin in einer Liste $\mathbb{D}_\mathcal{A}$ gespeichert.

Simulation von $\mathcal{O}^{\mathsf{D}}_{\mathcal{H},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird ein Eintrag (j, coin) in einer Liste $\mathbb{D}_H$ gespeichert.

Simulation von $\mathcal{O}^{\mathsf{VW}}_{\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ die Zahl $x \in \mathbb{Z}_p^*$ kennt und die entsprechenden Einträge in der Liste $\mathsf{view}_\mathcal{A}$ gespeichert hat.

Simulation von $\mathcal{O}^{\mathsf{UT}}_{\mathcal{B},\mathcal{D}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den geheimen Schlüssel $y \in \mathbb{Z}_p^*$ kennt.

Simulation von $\mathcal{O}^{\mathsf{CT}}_{\mathcal{B},\mathcal{D}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den geheimen Schlüssel $z \in \mathbb{Z}_p^*$ kennt.

Spielende: Mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ gilt $\$^* > \$$.

Nach Voraussetzung hat der Angreifer $\mathcal{A}$ mindestens eine Münze nicht gemäß dem Bezahlprotokoll erstellt. Da der Hashwert R bei jedem Bezahlvorgang, außer mit vernachlässigbarer Wahrscheinlichkeit, einmalig ist, kann jedes Nachrichten-Signatur-Paar $((R,k),(\Sigma',e',r))$ maximal einmal erfolgreich verwendet werden. Somit gibt es in der Liste $\sigma_{\mathcal{A}}$ Einträge $\left(\widetilde{O_1},\left(\widetilde{R_1},\widetilde{k_1}\right),\left(\widetilde{\Sigma'_1},\widetilde{e'_1},\widetilde{r_1}\right)\right),\ldots,\left(\widetilde{O_{n_1}},\left(\widetilde{R_{n_1}},\widetilde{k_{n_1}}\right),\left(\widetilde{\Sigma'_{n_1}},\widetilde{e'_{n_1}},\widetilde{r_{n_1}}\right)\right)$ mit $\sum_{n=1}^{n_1}\widetilde{k_n} \leq \$$ und in der Liste $\mathbb{D}_\sigma$ Einträge $(O_1,(R_1,k_1),(\Sigma'_1,e'_1,r_1)),\ldots,\Big(O_{n_2},(R_{n_2},k_{n_2}),\left(\Sigma'_{n_2},e'_{n_2},r_{n_2}\right)\Big)$ mit $\$^* = \sum_{n=1}^{n_2} k_n > \$$. Daher extrahiert $\mathcal{B}$ aus einer $\mathcal{O}^{\mathsf{S}}_{\mathcal{O},\mathcal{H}}$- oder $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$-Anfrage, außer mit vernachlässigbarer Wahrscheinlichkeit, einen Eintrag $(O^\star,(R^\star,k^\star),(\Sigma'^\star,e'^\star,r^\star)) \in \mathbb{D}_\sigma \setminus \sigma_{\mathcal{A}}$. Das heißt, die Signatur $(\Sigma'^\star,e'^\star,r^\star)$ wurde nicht von $\mathcal{B}$ (in der Rolle des Observers) auf die Nachricht $(R^\star,k^\star)$ bzgl. des Schlüssels $O^\star$ erstellt.

In Fall (1) wurde bereits gezeigt, dass der Fall $O^\star \notin \mathcal{U}_{\mathcal{A}}$ vernachlässigbar ist. Daher existiert in der Liste $\mathcal{U}_{\mathcal{A}}$ ein entsprechender Eintrag $(I^\star, I^\star_{\mathcal{O}}, O^\star, \gamma^\star, \cdot)$. Die Signatur $(\Sigma'^\star,e'^\star,r^\star)$ bzgl. des Schlüssels $O^\star$ muss nun in eine gültige Signatur bzgl. des Schlüssels O^* transformiert werden. Dazu definiert der Angreifer $\mathcal{B}$ nun $\Sigma^{**} := \Sigma'^\star, r^{**} := r^\star$ sowie $e^{**} := e'^\star + \gamma^\star \mod p$. Somit gilt $\hat{e}\left(\Sigma'^\star, O^\star h^{e'^\star}\right) = \hat{e}\left(\Sigma^{**}, O^* h^{\gamma^\star} h^{e'^\star}\right) = \hat{e}\left(\Sigma^{**}, O^* h^{e^{**}}\right)$. Folglich gilt $\hat{e}\left(\Sigma^{**}, O^* h^{e^{**}}\right) = \hat{e}\left(g g_0^{r^{**}} g_1^{R^\star} u_0^{k^\star}, h\right)$ und $((R^\star,k^\star),(\Sigma^{**},e^{**},r^{**}))$ ist damit ein gültiges BBS+A Nachrichten-Signatur-Paar.

Es bleibt zu zeigen, dass die Signatur $(\Sigma^{**},e^{**},r^{**})$ nicht vom Signatur-Orakel auf die Nachricht $(R^\star,k^\star)$ erstellt wurde, also dass $((R^\star,k^\star),(\Sigma^{**},e^{**},r^{**})) \notin \sigma^*_{\mathcal{A}}$ ist.

Angenommen, es gilt $((R^\star,k^\star),(\Sigma^{**},e^{**},r^{**})) \in \sigma^*_{\mathcal{A}}$. Dann existiert ein entsprechender Eintrag $(\widetilde{O},(R^\star,k^\star),(\widetilde{\Sigma'},\widetilde{e'},\widetilde{r})) \in \sigma_{\mathcal{A}}$ mit

$$\mathbb{D}_\sigma \setminus \sigma_{\mathcal{A}} \ni (O^\star,(R^\star,k^\star),(\Sigma'^\star,e'^\star,r^\star)) \neq (\widetilde{O},(R^\star,k^\star),(\widetilde{\Sigma'},\widetilde{e'},\widetilde{r})) \in \sigma_{\mathcal{A}}. \tag{9.1}$$

Falls $(\widetilde{O},(R^\star,k^\star),(\widetilde{\Sigma'},\widetilde{e'},\widetilde{r})) \notin \mathbb{D}_\sigma$ wäre, würde der Angreifer $\mathcal{A}$ bei einer Münze das Tupel $(O^\star,(R^\star,k^\star),(\Sigma'^\star,e'^\star,r^\star))$ anstelle des Tupels $(\widetilde{O},(R^\star,k^\star),(\widetilde{\Sigma'},\widetilde{e'},\widetilde{r}))$ verwenden. Dadurch kann $\mathcal{A}$ das Spiel aber nicht gewinnen, da dann nicht $\$^* > \$$ ist. (Der Angreifer $\mathcal{A}$ hätte dann lediglich eine Signatur $(\widetilde{\Sigma'},\widetilde{e'},\widetilde{r})$ auf die Nachricht $(R^\star,k^\star)$ bzgl. des Schlüssels $\widetilde{O}$ in eine Signatur $(\Sigma'^\star,e'^\star,r^\star)$ auf dieselbe Nachricht bzgl. des Schlüssels $O^\star$ transformiert.)

Da der Angreifer $\mathcal{A}$ nach Voraussetzung aber das Spiel gewinnt, muss folglich $(\widetilde{O},(R^\star,k^\star),(\widetilde{\Sigma'},\widetilde{e'},\widetilde{r})) \in \mathbb{D}_\sigma$ gelten. Somit gibt es die beiden verschiedenen Einträge aus (9.1) mit derselben Nachricht $(R^\star,k^\star)$ in der Liste $\mathbb{D}_\sigma$. Dann wurde jedoch bei zwei verschiedenen Münzen der gleiche Hashwert $R^\star \in \mathbb{Z}_p^*$ erzeugt, was nur mit vernachlässigbarer Wahrscheinlichkeit möglich ist.

Somit folgt schließlich $((R^\star,k^\star),(\Sigma^{**},e^{**},r^{**})) \notin \sigma^*_{\mathcal{A}}$. Der Angreifer $\mathcal{B}$ sendet das BBS+A Nachrichten-Signatur-Paar $((R^\star,k^\star),(\Sigma^{**},e^{**},r^{**}))$ an das Signatur-Orakel und

bricht damit die starke existentielle Unfälschbarkeit des BBS+A-Signaturverfahrens (und löst somit das q-SDH-Problem) mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist. □

9.3 Das faire teilbare elektronische Geldsystem mit vertrauenswürdigen Observern FTGvO-I

In diesem Abschnitt wird die Integration der Observer beschrieben, wenn diese, statt von der Bank, von einer vertrauenswürdigen dritten Partei (Trusted-Third-Party) an die Kunden verteilt werden. In fairen elektronischen Geldsystemen kann diese Partei auch der Deanonymisierer sein. Das Konzept, die Observer vom Deanonymisierer ausgeben zu lassen, wurde erstmals von T. Schwarzpaul [Sch09] vorgestellt und behandelt. Da in diesem Fall die Observer nach einer Kontoauflösung auch wieder bei der Trusted-Third-Party abgegeben werden könnten, können wir annehmen, dass die Bank keine auf dem Observer gespeicherten Daten auslesen kann. Des Weiteren kann davon ausgegangen werden, dass die Observer, ebenfalls wie die Trusted-Third-Party bzw. der Deanonymisierer, vertrauenswürdig sind und sich stets gemäß dem jeweiligen Protokoll verhalten. Aus diesem Grund können keine ein- und ausgehenden Informationen zwischen der Bank und dem Observer entstehen (vgl. auch [Sch09]).

Diese beiden Voraussetzungen ermöglichen es nun, dass der Observer Daten berechnen und speichern kann, mit denen die Bank ansonsten (wie bspw. im obigen Geldsystem FTGO-I) die Anonymität der Kunden aufheben könnte. Dadurch kann ein wesentlich effizienteres Geldsystem als im obigen Abschnitt 9.2 realisiert werden. Im Gegensatz zu [Sch09] werden allerdings die Observer zunächst so in das Geldsystem integriert, dass die bisherigen Sicherheitseigenschaften auch noch nach einer Manipulation eines Observers gewährleistet sind (wie bei FTGO-I). Im nächsten Abschnitt 9.4 wird schließlich ein effizienteres faires teilbares elektronisches Geldsystem mit Observern konstruiert, das für die bisherigen Sicherheitseigenschaften die Manipulationssicherheit der Observer voraussetzt.

Da die Observer nun von einer Trusted-Third-Party $\mathcal{TTP}$ ausgegeben werden, müssen sich die Kunden zunächst mit dem Protokoll Registry bei der Trusted-Third-Party $\mathcal{TTP}$ registrieren, bevor sie ein Konto bei der Bank $\mathcal{B}$ eröffnen können. Der öffentliche Schlüssel $\mathsf{pk}_{\mathcal{K}}$ eines Kunden wird nun bereits während Registry generiert, aber lediglich dazu benötigt, sich gegenüber seinem Observer und der Bank zu authentifizieren. Alle weiteren Protokolle laufen nun wie beim entsprechenden Geldsystem ohne Observer ab, außer dass für die Kontonummer $I := \mathsf{pk}_{\mathcal{O}}$ gilt und sämtliche Berechnungen vom Observer, anstatt vom Kunden, durchgeführt werden. Alle Daten, die der Observer speichern muss, müssen daher auf dem nichtauslesbaren Teil abgespeichert und alle anderen Daten gelöscht werden. Der Kunde verwaltet lediglich für jedes Abhebeprotokoll einen Zähler, der mit K initialisiert wird und entsprechend bei jedem Bezahlvorgang um den

Geldbetrag k verringert wird.

Wie auch in Abschnitt 9.2, wird die Integration der Observer exemplarisch an FTG-I gezeigt, kann aber völlig analog auf die übrigen elektronischen Geldsysteme angewandt werden. Die Trusted-Third-Party $\mathcal{TTP}$ kann dieselbe Partei wie der Deanonymisierer $\mathcal{D}$ oder eine weitere unabhängige Partei sein.

Im Folgenden werden hauptsächlich die Veränderungen im Vergleich zu FTG-I detailliert beschrieben. Die Algorithmen $\mathsf{Setup}, \mathsf{BGen}$ und DGen werden wie beim fairen teilbaren Geldsystem FTG-I durchgeführt. Da sich die Kontoeröffnungs-, Abhebe- und Bezahlprotokolle $\mathsf{Account}, \mathsf{Withdraw}$ und Spend verändern, werden die Algorithmen $\mathsf{Identify}, \mathsf{VerifyGuilt}$ und $\mathsf{VerifyWithdraw}$ sowie die Protokolle Spend-$K, \mathsf{Deposit}, \mathsf{UTrace}$ und CTrace dann analog zum fairen teilbaren Geldsystem FTG-I durchgeführt und aus diesem Grund nicht beschrieben.

Registry: Damit ein Kunde $\mathcal{K}$ dem Geldsystem beitreten kann, muss sich dieser zunächst bei der Trusted-Third-Party $\mathcal{TTP}$ registrieren und erhält anschließend einen Observer $\mathcal{O}$.

Schritt 1: Der Kunde $\mathcal{K}$ erzeugt sein Schlüsselpaar $(\mathsf{pk}_\mathcal{K}, \mathsf{sk}_\mathcal{K}) \in \mathbb{G}_p \times \mathbb{Z}_p^*$. Dazu wählt $\mathcal{K}$ zufällig seinen geheimen Schlüssel $\hat{u} \in_\mathcal{R} \mathbb{Z}_p^*$, berechnet seinen öffentlichen Schlüssel $\mathsf{pk}_\mathcal{K} = \mathfrak{g}^{\hat{u}}$ und sendet den öffentlichen Schlüssel $\mathsf{pk}_\mathcal{K}$ an die Trusted-Third-Party.

Schritt 2: Die Trusted-Third-Party $\mathcal{TTP}$ initialisiert nun einen Observer $\mathcal{O}$ für den Kunden $\mathcal{K}$. Dazu werden auf $\mathcal{O}$ die Systemparameter sp sowie die öffentlichen Schlüssel $\mathsf{pk}_\mathcal{B}, \mathsf{pk}_\mathcal{D}$ und $\mathsf{pk}_\mathcal{K}$ gespeichert. Diese öffentlichen Daten können auf dem auslesbaren Teil des Observers gespeichert werden. Anschließend wählt die Trusted-Third-Party $\mathcal{TTP}$ zufällig einen geheimen Schlüssel $u \in_\mathcal{R} \mathbb{Z}_p^*$, berechnet den öffentlichen Schlüssel $\mathsf{pk}_\mathcal{O} = \mathfrak{g}^u =: I$ und speichert $\mathsf{sk}_\mathcal{O} = u$ auf dem nichtauslesbaren Teil des Observers $\mathcal{O}$. Schließlich wird der Observer $\mathcal{O}$ zusammen mit dem öffentlichen Schlüssel $\mathsf{pk}_\mathcal{O}$ an den Kunden $\mathcal{K}$ übergeben. Weiter erstellt die Trusted-Third-Party $\mathcal{TTP}$ die Signature-of-Knowledge $SoK_{\mathsf{TTP}}\left[(y) : \mathfrak{g}_0 = \mathfrak{g}^y\right](\mathsf{pk}_\mathcal{O} || \mathsf{pk}_\mathcal{K})$ und sendet diese an den Kunden $\mathcal{K}$. Anhand von I kann $\mathcal{K}$ eindeutig identifiziert werden, weshalb I auch als Kontonummer oder Identität von $\mathcal{K}$ bezeichnet wird.

Schritt 3: Der Kunde $\mathcal{K}$ speichert sein Schlüsselpaar $(\mathsf{pk}_\mathcal{K}, \mathsf{sk}_\mathcal{K})$, den öffentlichen Schlüssel $\mathsf{pk}_\mathcal{O}$ sowie die Signature-of-Knowledge SoK_{TTP} ab.

Falls der öffentliche Schlüssel $\mathsf{pk}_\mathcal{K}$ oder $\mathsf{pk}_\mathcal{O}$ bereits in der Liste $\mathbb{D}_{\mathcal{K},\mathcal{O}}$ gespeichert ist, muss dieses Protokoll erneut durchgeführt werden. Dies ist allerdings nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Account: Damit ein Kunde $\mathcal{K}$ ein Konto bei der Bank $\mathcal{B}$ eröffnen kann, führen $\mathcal{K}$ und $\mathcal{B}$ das Kontoeröffnungsprotokoll $\mathsf{Account}$ durch. Zuvor muss sich der Kunde $\mathcal{K}$ gegenüber $\mathcal{B}$ ausweisen, beispielsweise mit einem Personalausweis.

Schritt 1: Der Kunde sendet das Paar $(\mathsf{pk}_{\mathcal{O}}, \mathsf{pk}_{\mathcal{K}})$ sowie die gespeicherte Signature-of-Knowledge SoK_{TTP} an $\mathcal{B}$. Anschließend wird die Signature-of-Knowledge SoK_{TTP} wieder gelöscht.

Schritt 2: Die Bank $\mathcal{B}$ verifiziert SoK_{TTP}. Anschließend speichert $\mathcal{B}$ die öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{O}}$ und $\mathsf{pk}_{\mathcal{K}}$ gemeinsam mit persönlichen Daten von $\mathcal{K}$ in ihrer Datenbank.

Withdraw: Damit ein Kunde $\mathcal{K}$ gemeinsam mit seinem Observer $\mathcal{O}$ eine Geldbörse mit dem Wert $K = 2^L$ bei der Bank $\mathcal{B}$ abheben kann, führen $\mathcal{O}, \mathcal{K}$ und $\mathcal{B}$ das Abhebeprotokoll Withdraw durch. Zuvor authentifizieren sich $\mathcal{K}$ und $\mathcal{B}$ sowie $\mathcal{K}$ und $\mathcal{O}$ gegenseitig. Danach gleicht $\mathcal{K}$ jeweils mit $\mathcal{B}$ und $\mathcal{O}$ den aktuellen Zeitpunkt ts ab (bzw. sendet ts an $\mathcal{O}$, falls der Observer über keine zeitlichen Informationen verfügt). Sei i die Anzahl der bisher durchgeführten Abhebeprotokolle des Kunden.

Schritt 1: Der Observer $\mathcal{O}$ übernimmt die Rolle des Kunden aus dem fairen teilbaren Geldsystem FTG-I. Somit sendet $\mathcal{O}$ die Elemente C, E, A_0 und die Signature-of-Knowledge SoK_{W} an den Kunden $\mathcal{K}$.

Schritt 2: Der Kunde $\mathcal{K}$ leitet die Elemente C, E, A_0 und die Signature-of-Knowledge SoK_{W} weiter an die Bank $\mathcal{B}$.

Schritt 3: Die Bank $\mathcal{B}$ verhält sich genau wie im fairen teilbaren Geldsystem und sendet das Tupel (Σ, e, s'', t'') an den Kunden $\mathcal{K}$.

Schritt 4: Der Kunde $\mathcal{K}$ verifiziert die Signatur durch Überprüfung der Gleichung $\hat{e}(\Sigma, Xh^e) \stackrel{?}{=} \hat{e}\left(gg_0^{s''}g_2^{t''}C, h\right)$, leitet (Σ, e, s'', t'') weiter an $\mathcal{O}$ und speichert das Paar $(i+1, J_{i+1})$ mit $J_{i+1} = K$ ab.

Schritt 5: Der Observer $\mathcal{O}$ berechnet $s = s' + s'' \mod p$ sowie $t = t' + t'' \mod p$, verifiziert die Signatur durch Überprüfung der Gleichung $\hat{e}(\Sigma, Xh^e) \stackrel{?}{=} \hat{e}\left(gg_0^{s''}g_2^{t''}C, h\right)$ und speichert $\mathbb{W} = (i+1, \Sigma, e, s, t, \mathsf{info})$ auf dem nichtauslesbaren Teil ab.

Spend: Damit ein Kunde $\mathcal{K}$, der im Besitz des Paares (i, J_i) ist, gemeinsam mit seinem Observer $\mathcal{O}$, der im Besitz der zugehörigen Geldbörse $\mathbb{W} = (i, \Sigma, e, s, t, \mathsf{info})$ ist, bei einem Händler $\mathcal{H}$ mit einer elektronischen Münze coin mit dem Wert $k = 2^\ell \leq J_i < K$ bezahlen kann, führen $\mathcal{O}, \mathcal{K}$ und $\mathcal{H}$ das Bezahlprotokoll Spend durch. Zuvor authentifizieren sich jedoch $\mathcal{O}$ und $\mathcal{K}$ gegenseitig und der Händler $\mathcal{H}$ authentifiziert sich gegenüber dem Kunden $\mathcal{K}$. Danach gleicht $\mathcal{K}$ jeweils mit $\mathcal{H}$ und $\mathcal{O}$ den aktuellen Zeitpunkt ts ab (bzw. sendet ts an $\mathcal{O}$, falls der Observer über keine zeitlichen Informationen verfügt). Weiter überprüft der Observer zunächst, ob der Wert k gültig ist, das heißt, ob noch mindestens k unbenutzte Serienschlüssel in der Geldbörse vorhanden sind. Andernfalls bricht $\mathcal{O}$ direkt ab und sendet das Symbol $\perp$ an $\mathcal{K}$.

Schritt 1: Der Händler $\mathcal{H}$ und der Observer $\mathcal{O}$ berechnen beide den Hashwert $R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}}||\mathsf{ts}||k)$.

Schritt 2: Der Observer $\mathcal{O}$ übernimmt die Rolle des Kunden aus dem fairen teilbaren Geldsystem FTG-I. Somit sendet $\mathcal{O}$ die Elemente S, T, T_C, A_1, B_1, B_2 und die Signature-of-Knowledge SoK an den Kunden $\mathcal{K}$.

Schritt 3: Der Kunde $\mathcal{K}$ leitet die Elemente S, T, T_C, A_1, B_1, B_2 sowie die Signature-of-Knowledge SoK weiter an den Händler $\mathcal{H}$ und verringert den Zähler J_i um k.

Schritt 4: Der Händler $\mathcal{H}$ geht wie im fairen teilbaren Geldsystem FTG-I vor.

9.3.1 Sicherheitsanalyse FTGvO-I

Um die Sicherheitseigenschaften zu beweisen, benötigen wir zunächst die folgende Annahme:

Annahme 9.2. *Jeder Observer ist vertrauenswürdig und verhält sich somit stets gemäß dem jeweiligen Protokoll. Weiter können die vom Observer gespeicherten Daten von niemandem ausgelesen werden.*

Satz 9.3.1. *Das faire teilbare elektronische Geldsystem mit vertrauenswürdigen Observern FTGvO-I ist im Random-Oracle-Modell unter der Annahme 9.2 ein sicheres faires elektronisches Geldsystem mit Observern, falls die q-SDH-Annahme, die q-polyDH-Annahme sowie die DBDH-Annahme für $\mathsf{Setup}_{\mathsf{Bil}}$ gelten und die DDH-Annahme für $\mathsf{Setup}_{\mathsf{G}}$ gilt. Dabei sind die Sicherheitseigenschaften Unfälschbarkeit, korrekte Kundenverfolgung und korrekte Münzverfolgung sogar dann erfüllt, wenn die Observer vom Angreifer $\mathcal{A}$ manipuliert werden.*

Beweis. Die drei Sicherheitsziele **Unfälschbarkeit**, **korrekte Kundenverfolgung** sowie **korrekte Münzverfolgung** folgen unmittelbar aus dem fairen teilbaren Geldsystem FTG-I, selbst wenn der Angreifer $\mathcal{A}$ den jeweiligen geheimen Schlüssel $\mathsf{sk}_{\mathcal{O}}$ erhält. Denn in diesem Fall handelt es sich gerade um das faire teilbare Geldsystem FTG-I.

Mit der Annahme 9.2 folgen die Sicherheitsziele **Anonymität**, **Double-Spending-Beschuldigung**, **Abhebe-Beschuldigung**, **Kundenverfolgung-Beschuldigung** sowie **Münzverfolgung-Beschuldigung** ebenfalls unmittelbar aus dem fairen teilbaren Geldsystem FTG-I, da sich aus der Sicht der Bank und der Händler nichts geändert hat. Des Weiteren existieren unter der Annahme 9.2 keine **eingehenden** und **ausgehenden Informationen** (vgl. auch [Sch09]).

Aus diesem Grund muss nur die Double-Spending-Prevention bewiesen werden.

Double-Spending-Prevention: Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Double-Spending-Prevention des Systems bricht.

Wir werden zeigen, dass die Erfolgswahrscheinlichkeit des Angreifers $\mathcal{A}$ unter der DLog-Annahme für $\mathsf{Setup}_\mathsf{G}$ und der Sicherheit des BBS+A-Signaturverfahrens vernachlässigbar ist.

Wenn die Observer bei jedem Protokoll involviert sind und der Angreifer $\mathcal{A}$ alle vom Observer erhaltenen Daten weiter an die Bank bzw. den Händler leitet, kann $\mathcal{A}$ das Spiel nicht gewinnen, da die Observer gemäß Protokoll ein Double-Spending verhindern. Folglich muss der Angreifer $\mathcal{A}$ bei mindestens einem Protokoll vom Protokollverlauf abweichen, um das Spiel zu gewinnen. Wenn $\mathcal{A}$ bei einem Protokoll vom Protokollverlauf abweicht und andere als die vom Observer erhaltenen Daten weiterleitet, muss $\mathcal{A}$ auch, außer mit vernachlässigbarer Wahrscheinlichkeit, die entsprechende Signature-of-Knowledge ohne den Observer erstellen. Somit gibt es die drei Möglichkeiten, dass der Angreifer (1) von einem Kontoeröffnungsprotokoll $\mathsf{Account}$ abweicht und die Signature-of-Knowledge SoK_{TTP} ohne eine entsprechende Orakel-Anfrage selbst erstellt, oder (2) von einem Abhebeprotokoll $\mathsf{Withdraw}$ abweicht und die Signature-of-Knowledge SoK_{W} ohne eine entsprechende Orakel-Anfrage selbst erstellt, oder (3) von einem Bezahlprotokoll Spend abweicht und die Signature-of-Knowledge SoK ohne eine entsprechende Orakel-Anfrage selbst erstellt.

Wir zeigen zunächst, dass die Fälle (1) und (2) vernachlässigbar sind, indem wir einen probabilistischen voraussichtlich-polynomiellen Angreifer $\mathcal{B}$ konstruieren, der den Angreifer $\mathcal{A}$ dazu verwendet, das DLog-Problem in der Gruppe $\mathbb{G}_p$ zu lösen.

Fall (1) und (2): Der Angreifer $\mathcal{B}$ erhält die Parameter $\left(p, \mathbb{G}'_p, \mathfrak{g}, \mathfrak{g}^a\right)$ und soll den diskreten Logarithmus $a \in \mathbb{Z}_p^*$ berechnen.

Simulation der Initialisierung: Der Angreifer $\mathcal{B}$ generiert die Parameter $\mathsf{sp}_{\mathsf{Bil}} = (p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \hat{e}, u_0, h)$, wobei $\mathbb{G}_1 = \langle u_0 \rangle, \mathbb{G}_2 = \langle h \rangle$ und $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ eine bilineare Abbildung ist. Dann wählt $\mathcal{B}$ zufällig drei Zahlen $\alpha, x, z \in_\mathcal{R} \mathbb{Z}_p^*$, vier Generatoren $g, g_0, g_1, g_2 \in_\mathcal{R} \mathbb{G}_1$ und definiert $\mathfrak{g}_0 := \mathfrak{g}^a$ mit $y := a$. Danach berechnet $\mathcal{B}$ die Elemente $u_i = u_0^{\alpha^i}$ sowie $h_i = h^{\alpha^i}$ für $1 \leq i \leq K$ und $X = h^x$ sowie $Z = h^z$. Anschließend sendet $\mathcal{B}$ die perfekt simulierten Systemparameter $\mathsf{sp} = \left(p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \mathbb{G}'_p, \hat{e}, g, g_0, g_1, g_2, u_0, \ldots, u_K, h, h_1, \ldots, h_K, \mathfrak{g}\right)$ sowie die perfekt simulierten öffentlichen Schlüssel $\mathsf{pk}_\mathcal{B} = X$ und $\mathsf{pk}_\mathcal{D} = (\mathfrak{g}_0, Z)$ an den Angreifer $\mathcal{A}$.

Simulation von $\mathcal{O}^{\mathsf{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{R}}_{\mathcal{TTP}}$: Der Angreifer $\mathcal{B}$ wählt zufällig eine Zahl $\gamma \in_\mathcal{R} \mathbb{Z}_p^*$ und definiert $I := \mathfrak{g}^a\mathfrak{g}^\gamma = \mathfrak{g}^u$, so dass $u = a + \gamma \mod p$ gilt. Weiter simuliert $\mathcal{B}$ die Signature-of-Knowledge SoK_{TTP} perfekt, da $\mathcal{B}$ die Zahl $y \in \mathbb{Z}_p^*$ nicht kennt. Anschließend speichert $\mathcal{B}$ den Eintrag (I, SoK_{TTP}) in einer Liste $\sigma_{\mathcal{A},\mathsf{TTP}}$ und das Tripel $(\mathsf{pk}_\mathcal{K}, I, \gamma)$ in einer Liste $\mathcal{U}_\mathcal{A}$.

Simulation von $\mathcal{O}^{\mathsf{R}}_{\mathcal{K},\mathcal{TTP}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, außer dass $\mathcal{B}$ die Signature-of-Knowledge SoK_{TTP} perfekt simuliert, da $\mathcal{B}$ die Zahl $y \in \mathbb{Z}_p^*$ nicht

kennt. Weiter speichert $\mathcal{B}$ den Eintrag (I, SoK_{TTP}) in einer Liste σ_{TTP} und das Tupel $(\mathsf{pk}_{\mathcal{K}}, \mathsf{sk}_{\mathcal{K}}, I, u)$ in einer Liste $\mathcal{U}_H$.

Simulation von $\mathcal{O}^{\mathsf{A}}_{\mathcal{O},\mathcal{B}}$: Der Observer ist beim Kontoeröffnungsprotokoll nicht beteiligt.

Für die Simulation der Bank geht $\mathcal{B}$ folgendermaßen vor: Falls $(I, SoK_{\mathsf{TTP}}) \in \sigma_{\mathcal{A},\mathsf{TTP}}$ ist, verhält sich der Angreifer $\mathcal{B}$ wie das Orakel.

Falls $(I, SoK_{\mathsf{TTP}}) \notin \sigma_{\mathcal{A},\mathsf{TTP}}$ ist, gibt es die beiden Möglichkeiten $(\cdot, SoK_{\mathsf{TTP}}) \notin \sigma_{\mathcal{A},\mathsf{TTP}} \cup \sigma_{\mathsf{TTP}}$ oder $(I', SoK_{\mathsf{TTP}}) \in \sigma_{\mathcal{A},\mathsf{TTP}} \cup \sigma_{\mathsf{TTP}}$ mit $I' \neq I$. Falls $(\cdot, SoK_{\mathsf{TTP}}) \notin \sigma_{\mathcal{A},\mathsf{TTP}} \cup \sigma_{\mathsf{TTP}}$ ist, wurde SoK_{TTP} vom Angreifer $\mathcal{A}$ erstellt. Somit verhält sich der Angreifer $\mathcal{B}$ wie das Orakel, außer dass $\mathcal{B}$ die Zahl $a \in \mathbb{Z}_p^*$ mit $\mathfrak{g}_0 = \mathfrak{g}^a$ aus SoK_{TTP} extrahiert. Der Fall $(I', SoK_{\mathsf{TTP}}) \in \sigma_{\mathcal{A},\mathsf{TTP}} \cup \sigma_{\mathsf{TTP}}$ mit $I' \neq I$ ist vernachlässigbar, da für die Challenge $\mathsf{H}(\cdots||I||\mathsf{pk}_{\mathcal{K}}) = \mathsf{H}(\cdots||I'||\mathsf{pk}'_{\mathcal{K}})$ gelten muss, wobei $\mathsf{pk}_{\mathcal{K}}$ und $\mathsf{pk}'_{\mathcal{K}}$ die beiden öffentlichen Schlüssel mit $(\mathsf{pk}_{\mathcal{K}}, I), (\mathsf{pk}'_{\mathcal{K}}, I') \in \mathcal{U}_{\mathcal{A}} \cup \mathcal{U}_H$ sind.

Simulation von $\mathcal{O}^{\mathsf{A}}_{\mathcal{K},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{W}}_{\mathcal{O},\mathcal{B}}$: Für die Simulation des Observers geht $\mathcal{B}$ folgendermaßen vor: Der Angreifer $\mathcal{B}$ wählt zufällig die Zahlen $\kappa_{0,0} \in_{\mathcal{R}} \mathbb{Z}_p^*, a_0, t' \in_{\mathcal{R}} \mathbb{Z}_p$ sowie das Element $C \in_{\mathcal{R}} \mathbb{G}_1$ und berechnet die Elemente E sowie A_0 gemäß Protokoll. Somit sind die Elemente C, E, A_0 perfekt simuliert. Anschließend simuliert $\mathcal{B}$ die Signature-of-Knowledge SoK_{W} perfekt und speichert diese in einer Liste σ_{W}.

Für die Simulation der Bank geht $\mathcal{B}$ folgendermaßen vor: Falls $SoK_{\mathsf{W}} \in \sigma_{\mathsf{W}}$ ist, verhält sich der Angreifer $\mathcal{B}$ wie das Orakel. Falls $SoK_{\mathsf{W}} \notin \sigma_{\mathsf{W}}$ ist, wurde SoK_{W}, außer mit vernachlässigbarer Wahrscheinlichkeit, von $\mathcal{A}$ erstellt. Somit verhält sich der Angreifer $\mathcal{B}$ wie das Orakel, außer dass $\mathcal{B}$ die Zahl $u \in \mathbb{Z}_p$ mit $I = \mathfrak{g}^u$ aus SoK_{W} extrahiert. Weiter sucht $\mathcal{B}$ das Tripel $(\mathsf{pk}_{\mathcal{K}}, I, \gamma) \in \mathcal{U}_{\mathcal{A}}$ und berechnet die Zahl $a = u - \gamma \mod p$, falls $(\cdot, I, \cdot) \in \mathcal{U}_{\mathcal{A}}$ ist. Beachte, dass für $(\cdot, I, \cdot) \notin \mathcal{U}_{\mathcal{A}}$ auch $(I, \cdot) \notin \sigma_{\mathcal{A},\mathsf{TTP}}$ bei einer $\mathcal{O}^{\mathsf{A}}_{\mathcal{O},\mathcal{B}}$-Anfrage gelten muss und $\mathcal{B}$ somit, außer mit vernachlässigbarer Wahrscheinlichkeit, bereits die Zahl $a \in Z_p^*$ extrahiert hat.

Nach erfolgreicher Durchführung wird der Eintrag $(i_{\mathcal{A}}, \mathbb{W}_{i_{\mathcal{A}}} = (\Sigma, e, s'', t', t'', \mathsf{info}), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_{\mathcal{O}}$ sowie der Eintrag $(i_{\mathcal{A}}, \mathsf{view})$ in einer Liste $\mathsf{view}_{\mathcal{A}}$ gespeichert und der Zähler $i_{\mathcal{A}}$ um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird der Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$ gespeichert und der Zähler i um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{S}}_{\mathcal{O},\mathcal{H}}$: Für die Simulation des Observers geht $\mathcal{B}$ folgendermaßen vor: Der Angreifer $\mathcal{B}$ verwendet den Eintrag $(i_{\mathcal{A}}, \mathbb{W}_{i_{\mathcal{A}}}, \mathsf{pk}_{\mathcal{K}}, \$_{i_{\mathcal{A}}}) \in \mathbb{W}_{\mathcal{O}}$, berechnet die Elemente S, T_C, A_1, B_1, B_2 gemäß Protokoll und berechnet das korrekte Sicherheitstag $T = I\mathfrak{g}_0^{R\kappa}$ ohne Kenntnis des privaten Schlüssels u, wobei $\kappa \in \mathbb{Z}_p^*$ der Schlüssel mit $S = \mathfrak{g}^{\kappa}$ ist. Dann simuliert $\mathcal{B}$ die Signature-of-Knowledge SoK perfekt. Anschließend wird der Eintrag $(i_{\mathcal{A}}, \mathbb{W}_{i_{\mathcal{A}}}, \mathsf{pk}_{\mathcal{K}}, \$_{i_{\mathcal{A}}})$ zu $(i_{\mathcal{A}}, \mathbb{W}'_{i_{\mathcal{A}}}, \mathsf{pk}_{\mathcal{K}}, \$_{i_{\mathcal{A}}} - k)$ aktualisiert.

Für die Simulation des Händlers verhält sich der Angreifer $\mathcal{B}$ wie das Orakel. Nach erfolgreicher Durchführung wird ein Eintrag (j, coin) in der Liste $\mathbb{C}_{\mathcal{A}}$ sowie ein Eintrag $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts})$ in der Liste $\mathbb{S}$ gespeichert, der Wert \$* um k erhöht und der Zähler j um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird der Wert \$ um k erhöht und ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_{H,\mathcal{A}}$ gespeichert.

Simulation von $\mathcal{O}^{\mathsf{S}}_{\mathcal{K},\mathcal{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel. Somit wird ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_H$, ein Eintrag (j, coin) in einer Liste $\mathbb{C}$ sowie ein Eintrag $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts})$ in der Liste $\mathbb{S}$ gespeichert und der Zähler j um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{D}}_{\mathcal{H},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{VW}}_{\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ die Zahl $x \in \mathbb{Z}_p^*$ kennt und die entsprechenden Einträge in der Liste $\mathsf{view}_{\mathcal{A}}$ gespeichert hat.

Simulation von $\mathcal{O}^{\mathsf{UT}}_{\mathcal{B},\mathcal{D}}$: Der Angreifer $\mathcal{B}$ erhält eine Münze $\mathsf{coin} = (k, S, T, T_C, R, \Phi = (A_1, B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$.

- Falls der Angreifer $\mathcal{B}$ die Münze coin zum Bezahlen verwendet hat, gibt es einen Eintrag $(\cdot, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin}) \in \mathbb{C}_{H,\mathcal{A}} \cup \mathbb{C}_H$. Somit kennt $\mathcal{B}$ die Kontonummer $I = \mathsf{pk}_{\mathcal{O}}$ mit $(\mathsf{pk}_{\mathcal{K}}, I) \in \mathcal{U}_H$ und gibt diese aus, wodurch die Anfrage perfekt simuliert ist.
- Falls der Angreifer $\mathcal{B}$ die Münze coin nicht zum Bezahlen verwendet hat, wurde SoK vom Angreifer $\mathcal{A}$ erstellt. Daher kann $\mathcal{B}$ die Zahlen $u, \kappa \in \mathbb{Z}_p$ mit $S = \mathfrak{g}^{\kappa}$ sowie $T = \mathfrak{g}^{u}\mathfrak{g}_0^{R\kappa}$ aus SoK extrahieren und die Kontonummer $I = \mathfrak{g}^{u}$ korrekt berechnen. Dann gibt $\mathcal{B}$ die Kontonummer I aus, wodurch die Anfrage perfekt simuliert ist. Falls $\mathcal{B}$ die Zahlen u, κ nicht extrahieren kann, bricht $\mathcal{B}$ ab. Dies ist aber nur mit vernachlässigbarer Wahrscheinlichkeit möglich.

Simulation von $\mathcal{O}^{\mathsf{CT}}_{\mathcal{B},\mathcal{D}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den geheimen Schlüssel $z \in \mathbb{Z}_p^*$ kennt.

Spielende: Der Angreifer $\mathcal{A}$ gibt mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ zwei Münzen coin und coin' aus, so dass die Ausgabe von $\mathsf{Identify}(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{D}}, \mathsf{coin}, \mathsf{coin}')$ ein öffentlicher Schlüssel $\mathsf{pk}_{\mathcal{O}} \in \mathcal{U}_{\mathcal{A}}$ ist.

Nach Voraussetzung hat der Angreifer $\mathcal{A}$ eine Signature-of-Knowledge SoK_{TTP} oder SoK_{W} ohne eine entsprechende Observer-Anfrage erstellt. Somit gilt $SoK_{\mathsf{TTP}} \notin \sigma_{\mathcal{A},\mathsf{TTP}}$ oder $SoK_{\mathsf{W}} \notin \sigma_{\mathsf{W}}$ und daher hat $\mathcal{B}$ den diskreten Logarithmus $a \in \mathbb{Z}_p^*$, außer mit vernachlässigbarer Wahrscheinlichkeit, bei einer $\mathcal{O}^{\mathsf{A}}_{\mathcal{O},\mathcal{B}}$- oder $\mathcal{O}^{\mathsf{W}}_{\mathcal{O},\mathcal{B}}$-Anfrage extrahiert.

Somit löst $\mathcal{B}$ das DLog-Problem in $\mathbb{G}_p$ mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist.

Fall (3): Es bleibt zu zeigen, dass der Angreifer $\mathcal{A}$ ebenfalls nur mit vernachlässigbarer Wahrscheinlichkeit von einem Bezahlprotokoll abweichen und die Signature-of-Knowledge SoK ohne eine entsprechende Orakel-Anfrage selbst erstellen kann.

Unabhängig davon, wie $\mathcal{B}$ die Orakel-Anfragen simuliert, werden zunächst die verschiedenen Fälle aufgelistet, in denen der Angreifer $\mathcal{A}$ das Spiel gewinnt.

Falls der Angreifer $\mathcal{A}$ von einem Bezahlprotokoll abweicht, muss $\mathcal{A}$ bei der Münze coin oder coin' die Signature-of-Knowledge SoK über ein Nachrichten-Signatur-Paar $((u^\star, t^\star, \phi^\star(\mathsf{x})), (\Sigma^\star, e^\star, s^\star))$ selbst erstellen. Daher kann $\mathcal{B}$ dieses Nachrichten-Signatur-Paar extrahieren. Dann gibt es die beiden Möglichkeiten, dass das Nachrichten-Signatur-Paar (3.1) nie oder (3.2) mindestens einmal bei der Simulation einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{O},\mathcal{B}}$- oder $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$-Anfrage erzeugt wurde. Diese Fälle können nun analog zu den Fällen (1.1) bzw. (1.2) der Unfälschbarkeit behandelt werden.

Fall (3.1): Der Angreifer $\mathcal{B}$ erhält den öffentlichen Schlüssel $(p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \hat{e}, g, g_0, g_1, g_2, u_0, \ldots, u_K, h, h_1, \ldots, h_K, X)$ des BBS+A-Signaturverfahrens und darf q Anfragen an das Signatur-Orakel stellen.

Simulation der Initialisierung: Der Angreifer $\mathcal{B}$ generiert eine zyklische Gruppe $\mathbb{G}_p$ der Ordnung p, wählt zufällig zwei Zahlen $y, z \in_{\mathcal{R}} \mathbb{Z}^*_p$ sowie einen Generator $\mathfrak{g} \in_{\mathcal{R}} \mathbb{G}_p$ und berechnet die Generatoren $\mathfrak{g}_0 = \mathfrak{g}^y$ sowie $Z = h^z$. Dann sendet $\mathcal{B}$ die perfekt simulierten Systemparameter $\mathsf{sp} = (p, \mathbb{G}'_1, \mathbb{G}'_2, \mathbb{G}'_T, \mathbb{G}'_p, \hat{e}, g, g_0, g_1, g_2, u_0, \ldots, u_K, h, h_1, \ldots, h_K, \mathfrak{g})$ sowie die perfekt simulierten öffentlichen Schlüssel $\mathsf{pk}_{\mathcal{B}} = X$ und $\mathsf{pk}_{\mathcal{D}} = (\mathfrak{g}_0, Z)$ an den Angreifer $\mathcal{A}$.

Simulation von $\mathcal{O}^{\mathsf{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{R}}_{\mathcal{TTP}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{R}}_{\mathcal{K},\mathcal{TTP}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{A}}_{\mathcal{O},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{A}}_{\mathcal{K},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{W}}_{\mathcal{O},\mathcal{B}}$: Für die Simulation des Observers verhält sich $\mathcal{B}$ wie das Orakel.

Für die Simulation der Bank geht $\mathcal{B}$ folgendermaßen vor: Es wurde bereits in Fall (2) gezeigt, dass nur mit vernachlässigbarer Wahrscheinlichkeit die Signature-of-Knowledge SoK_{W} von $\mathcal{A}$ erstellt wurde. Daher leitet $\mathcal{A}$ die Daten weiter, die $\mathcal{A}$ soeben von $\mathcal{B}$ (in der Rolle des Observers) erhalten hat. Somit kennt $\mathcal{B}$ das Tupel $(s', u, t', \phi(\mathsf{x}))$ mit $C = g_0^{s'} g_1^u g_2^{t'} u_0^{\phi(\alpha)}$. Der Angreifer $\mathcal{B}$ wählt zufällig eine Zahl $t'' \in_{\mathcal{R}} \mathbb{Z}_p$, berechnet $t = t' + t'' \mod p$, leitet die Nachricht $(u, t, \phi(\mathsf{x}))$ an das Signatur-Orakel und erhält eine Signatur $\sigma = (\Sigma, e, s)$. Dann sendet $\mathcal{B}$ das perfekt simulierte Tupel (Σ, e, s'', t'') mit $s'' := s - s' \mod p$ an den Angreifer $\mathcal{A}$.

Nach erfolgreicher Durchführung wird der Eintrag $(i_{\mathcal{A}}, \mathbb{W}_{i_{\mathcal{A}}} = (\Sigma, e, s, u, t, \mathsf{info}), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_{\mathcal{O}}$ sowie der Eintrag $(i_{\mathcal{A}}, \mathsf{view} = (\mathbb{T} = (\mathsf{pk}_{\mathcal{K}}, I, C, E, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s'', t'')), (s', u, t', \phi(\mathsf{x})))$ in einer Liste $\mathsf{view}_{\mathcal{A}}$ gespeichert und der Zähler $i_{\mathcal{A}}$ um 1 erhöht. Der Angreifer $\mathcal{B}$ benötigt insgesamt $q_{W,1}$ Anfragen an das Signatur-Orakel.

Simulation von $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie der ehrliche Kunde und simuliert die Bank perfekt mit Zugriff auf das Signatur-Orakel, da $\mathcal{B}$ die zu signierende Nachricht selbst wählt. Somit wird der Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$ gespeichert und der Zähler i um 1 erhöht. Der Angreifer $\mathcal{B}$ benötigt insgesamt $q_{W,2}$ Anfragen an das Signatur-Orakel.

Simulation von $\mathcal{O}^{\mathsf{S}}_{\mathcal{O},\mathcal{H}}$: Der Angreifer $\mathcal{B}$ verwendet den Eintrag $(i_{\mathcal{A}}, \mathbb{W}_{i_{\mathcal{A}}}, \mathsf{pk}_{\mathcal{K}}, K) \in \mathbb{W}_{\mathcal{O}}$ und verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$: Der Angreifer $\mathcal{B}$ verwendet den Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, K) \in \mathbb{W}_H$ und verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{S}}_{\mathcal{K},\mathcal{H}}$: Der Angreifer $\mathcal{B}$ verwendet den Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, K) \in \mathbb{W}_H$ und verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{D}}_{\mathcal{H},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{VW}}_{\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verwendet die Liste $\mathsf{view}_{\mathcal{A}}$ und simuliert die Signatur perfekt mit dem Signatur-Orakel. Der Angreifer $\mathcal{B}$ benötigt insgesamt q_V Anfragen an das Signatur-Orakel.

Simulation von $\mathcal{O}^{\mathsf{UT}}_{\mathcal{B},\mathcal{D}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den geheimen Schlüssel $y \in \mathbb{Z}_p^*$ kennt.

Simulation von $\mathcal{O}^{\mathsf{CT}}_{\mathcal{B},\mathcal{D}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den geheimen Schlüssel $z \in \mathbb{Z}_p^*$ kennt.

Spielende: Der Angreifer $\mathcal{A}$ gibt mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ zwei Münzen coin und coin' aus, so dass die Ausgabe von $\mathsf{Identify}(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{D}}, \mathsf{coin}, \mathsf{coin}')$ ein öffentlicher Schlüssel $\mathsf{pk}_{\mathcal{O}} \in \mathcal{U}_{\mathcal{A}}$ ist.

Nach Voraussetzung wurde eine Signature-of-Knowledge der Münze coin bzw. coin' nicht bei einer $\mathcal{O}^{\mathsf{S}}_{\mathcal{O},\mathcal{H}}$-Anfrage von $\mathcal{B}$ (in der Rolle des Observers), sondern vom Angreifer $\mathcal{A}$ erstellt. Dies sei o.B.d.A. die Signature-of-Knowledge SoK der Münze coin. Dann kann der Angreifer $\mathcal{B}$ das Nachrichten-Signatur-Paar $((u^\star, t^\star, \phi^\star(\mathsf{x})), (\Sigma^\star, e^\star, s^\star))$ aus SoK extrahieren. Falls $\mathcal{B}$ das Nachrichten-Signatur-Paar nicht extrahieren kann, bricht $\mathcal{B}$ ab. Dies ist jedoch nur mit vernachlässigbarer Wahrscheinlichkeit möglich. Weiter wurde nach

Voraussetzung das Nachrichten-Signatur-Paar $((u^\star, t^\star, \phi^\star(\mathsf{x})), (\Sigma^\star, e^\star, s^\star))$ niemals bei einer Abhebe-Anfrage erstellt. Daher sendet der Angreifer $\mathcal{B}$ das Nachrichten-Signatur-Paar $((u^\star, t^\star, \phi^\star(\mathsf{x})), (\Sigma^\star, e^\star, s^\star))$ an das Signatur-Orakel und bricht damit die existentielle Unfälschbarkeit des BBS+A-Signaturverfahrens mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist.

Fall (3.2): Der Angreifer $\mathcal{B}$ erhält eine zufällige 2-SDH-Probleminstanz $(p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \hat{e}, u_0, u_0^a, h, h^a)$.

Simulation der Initialisierung: Der Angreifer $\mathcal{B}$ generiert eine zyklische Gruppe $\mathbb{G}_p$ der Ordnung p, wählt zufällig fünf Zahlen $\alpha, \beta, x, y, z \in_{\mathcal{R}} \mathbb{Z}_p^*$, einen Generator $\mathfrak{g} \in_{\mathcal{R}} \mathbb{G}_p$ sowie drei Generatoren $g, g_1, g_2 \in_{\mathcal{R}} \mathbb{G}_1$. Dann berechnet $\mathcal{B}$ die Generatoren $\mathfrak{g}_0 = \mathfrak{g}^y, g_0 = u_0^\beta, X = h^x, Z = h^z$ sowie $u_i = u_0^{\alpha^i}$ und $h_i = h^{\alpha^i}$ für $1 \leq i \leq K$. Anschließend sendet $\mathcal{B}$ die perfekt simulierten Systemparameter $\mathsf{sp} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \mathbb{G}_p', \hat{e}, g, g_0, g_1, g_2, u_0, \ldots, u_K, h, h_1, \ldots, h_K, \mathfrak{g})$ sowie die perfekt simulierten Schlüssel $\mathsf{pk}_{\mathcal{B}} = X$ und $\mathsf{pk}_{\mathcal{D}} = (\mathfrak{g}_0, Z)$ an den Angreifer $\mathcal{A}$.

Simulation von $\mathcal{O}^{\mathsf{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{R}}_{\mathcal{TTP}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{R}}_{\mathcal{K},\mathcal{TTP}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{A}}_{\mathcal{O},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{A}}_{\mathcal{K},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{W}}_{\mathcal{O},\mathcal{B}}$: Für die Simulation des Observers geht $\mathcal{B}$ folgendermaßen vor: Der Angreifer $\mathcal{B}$ verhält sich wie der ehrliche Observer, außer bei der Berechnung von C und der Erstellung von SoK_{W}. Hier wählt $\mathcal{B}$ zufällig eine Zahl $\delta \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Commitment $C = (u_0^a)^\beta\, g_0^\delta g_1^u g_2^{t'} u_0^{\phi(\alpha)} = g_0^a g_0^\delta g_1^u g_2^{t'} u_0^{\phi(\alpha)}$ mit $s' := a + \delta \mod p$ und simuliert SoK_{W} perfekt. Nach Fall (2) leitet der Angreifer $\mathcal{A}$ nur mit vernachlässigbarer Wahrscheinlichkeit andere Elemente und eine andere Signature-of-Knowledge weiter an $\mathcal{B}$ (in der Rolle der Bank).

Der Angreifer $\mathcal{B}$ verhält sich für die Simulation der Bank wie das Orakel, da $\mathcal{B}$ den privaten Schlüssel x kennt. Dann wird ein Eintrag $(i_{\mathcal{A}}, \mathbb{W}_{i_{\mathcal{A}}} = (\Sigma, e, \delta, s'', t, \mathsf{info}), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_{\mathcal{O}}$, ein Eintrag $(i_{\mathcal{A}}, \mathsf{view} = (\mathbb{T} = (I, C, E, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s'', t'')))$ in einer Liste $\mathsf{view}_{\mathcal{A}}$ sowie das Nachrichten-Signatur-Paar $((u, t, \phi(\mathsf{x})), (\Sigma, e, \delta, s''))$ in einer Liste $\sigma_{\mathcal{A}}$ gespeichert und der Zähler $i_{\mathcal{A}}$ um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie der ehrliche Kunde (bzw. Observer), außer bei der Berechnung von C und der Erstellung von SoK_{W}. Hier wählt $\mathcal{B}$ wie bei einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{O},\mathcal{B}}$-Anfrage zufällig eine Zahl $\delta \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet

das Commitment $C = (u_0^a)^\beta \, g_0^\delta g_1^u g_2^{t'} u_0^{\phi(\alpha)} = g_0^a g_0^\delta g_1^u g_2^{t'} u_0^{\phi(\alpha)}$ mit $s' := a + \delta \mod p$ und simuliert SoK_{W}. Da $\mathcal{B}$ den privaten Schlüssel x kennt, kann $\mathcal{B}$ eine gültige Signatur erstellen. Insgesamt hat $\mathcal{B}$ somit die Anfrage perfekt simuliert. Dann wird ein Eintrag $(i, \mathbb{W}_i = (\Sigma, e, \delta, s'', t, \mathsf{info}), \mathsf{pk}_{\mathcal{K}}, K)$ in einer Liste $\mathbb{W}_H$, ein Eintrag $(i, \mathsf{view} = (\mathbb{T} = (I, C, E, A_0, SoK_{\mathsf{W}}, \mathsf{ts}, K), \sigma' = (\Sigma, e, s'', t'')))$ in einer Liste view_H sowie das Nachrichten-Signatur-Paar $((u, t, \phi(\mathsf{x})), (\Sigma, e, \delta, s''))$ in einer Liste σ_H gespeichert und der Zähler i um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{S}}_{\mathcal{O},\mathcal{H}}$: Für die Simulation des Observers geht $\mathcal{B}$ folgendermaßen vor: Der Angreifer $\mathcal{B}$ verhält sich wie der ehrliche Observer mit Geldbörse $\mathbb{W}_{i_{\mathcal{A}}}$, außer dass $\mathcal{B}$ die Signature-of-Knowledge SoK perfekt simulieren muss, da $\mathcal{B}$ die Signatur (die Zahl $s = a + \delta + s'' \mod p$) nicht kennt.

Für die Simulation des Händlers verhält sich $\mathcal{B}$ wie das Orakel. Anschließend wird der Eintrag $(i_{\mathcal{A}}, \mathbb{W}_{i_{\mathcal{A}}}, \mathsf{pk}_{\mathcal{K}}, \$_{i_{\mathcal{A}}})$ in der Liste $W_{\mathcal{O}}$ zu $\left(i_{\mathcal{A}}, \mathbb{W}'_{i_{\mathcal{A}}}, \mathsf{pk}_{\mathcal{K}}, \$_{i_{\mathcal{A}}} - k\right)$ aktualisiert, ein Eintrag (j, coin) in einer Liste $\mathbb{C}_{\mathcal{A}}$ sowie ein Eintrag $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts})$ in einer Liste $\mathbb{S}$ gespeichert und der Zähler j um 1 erhöht.

Simulation von $\mathcal{O}^{\mathsf{S}}_{\mathcal{K}}$: Der Angreifer $\mathcal{B}$ verhält sich wie der ehrliche Kunde mit Geldbörse $\mathbb{W}_i$, außer dass $\mathcal{B}$ die Signature-of-Knowledge SoK perfekt simulieren muss, da $\mathcal{B}$ die Signatur (die Zahl $s = a + \delta + s'' \mod p$) nicht kennt. Anschließend wird der Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i)$ zu $(i, \mathbb{W}'_i, \mathsf{pk}_{\mathcal{K}}, \$_i - k)$ aktualisiert und ein Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_{H,\mathcal{A}}$ gespeichert.

Simulation von $\mathcal{O}^{\mathsf{S}}_{\mathcal{K},\mathcal{H}}$: Der Angreifer $\mathcal{B}$ verhält sich wie der ehrliche Händler und wie der ehrliche Kunde mit Geldbörse $\mathbb{W}_i$, außer dass $\mathcal{B}$ die Signature-of-Knowledge SoK perfekt simulieren muss, da $\mathcal{B}$ die Signatur (die Zahl $s = a + \delta + s'' \mod p$) nicht kennt. Dann aktualisiert $\mathcal{B}$ den Eintrag $(i, \mathbb{W}_i, \mathsf{pk}_{\mathcal{K}}, \$_i)$ zu $(i, \mathbb{W}'_i, \mathsf{pk}_{\mathcal{K}}, \$_i - k)$, speichert einen Eintrag $(i, \mathsf{pk}_{\mathcal{K}}, \mathsf{coin})$ in einer Liste $\mathbb{C}_H$, einen Eintrag (j, coin) in einer Liste $\mathbb{C}$ sowie einen Eintrag $(\mathsf{pk}_{\mathcal{H}}, \mathsf{ts})$ in einer Liste $\mathbb{S}$ ab und erhöht den Zähler j um 1.

Simulation von $\mathcal{O}^{\mathsf{D}}_{\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{D}}_{\mathcal{H},\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel.

Simulation von $\mathcal{O}^{\mathsf{VW}}_{\mathcal{B}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ die Zahl $x \in \mathbb{Z}_p^*$ kennt und die entsprechenden Einträge in der Liste $\mathsf{view}_{\mathcal{A}}$ gespeichert hat.

Simulation von $\mathcal{O}^{\mathsf{UT}}_{\mathcal{B},\mathcal{D}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den geheimen Schlüssel $y \in \mathbb{Z}_p^*$ kennt.

Simulation von $\mathcal{O}^{\mathsf{CT}}_{\mathcal{B},\mathcal{D}}$: Der Angreifer $\mathcal{B}$ verhält sich wie das Orakel, da $\mathcal{B}$ den geheimen Schlüssel $z \in \mathbb{Z}_p^*$ kennt.

Spielende: Der Angreifer $\mathcal{A}$ gibt mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda)$ zwei Münzen coin und coin' aus, so dass die Ausgabe von $\mathsf{Identify}(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{D}}, \mathsf{coin}, \mathsf{coin}')$ ein öffentlicher Schlüssel $\mathsf{pk}_{\mathcal{O}} \in \mathcal{U}_{\mathcal{A}}$ ist.

Nach Voraussetzung wurde eine Signature-of-Knowledge der Münze coin bzw. coin' nicht bei einer $\mathcal{O}^{\mathsf{S}}_{\mathcal{O},\mathcal{B}}$-Anfrage von $\mathcal{B}$ (in der Rolle des Observers), sondern vom Angreifer $\mathcal{A}$ erstellt. Dies sei o.B.d.A. die Signature-of-Knowledge SoK der Münze coin. Dann kann der Angreifer $\mathcal{B}$ das Nachrichten-Signatur-Paar $((u^\star, t^\star, \phi^\star(\mathsf{x})), (\Sigma^\star, e^\star, s^\star))$ aus SoK extrahieren. Falls $\mathcal{B}$ das Nachrichten-Signatur-Paar nicht extrahieren kann, bricht $\mathcal{B}$ ab. Dies ist jedoch nur mit vernachlässigbarer Wahrscheinlichkeit möglich. Weiter wurde nach Voraussetzung das Nachrichten-Signatur-Paar $((u^\star, t^\star, \phi^\star(\mathsf{x})), (\Sigma^\star, e^\star, s^\star))$ bei der Simulation einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{O},\mathcal{B}}$- oder $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$-Anfrage erstellt. Somit existiert ein Eintrag $((u^\star, t^\star, \phi^\star(\mathsf{x})), (\Sigma^\star, e^\star, \delta^\star, s''^\star)) \in \sigma_{\mathcal{A}} \cup \sigma_H$.

Der Angreifer $\mathcal{B}$ sucht diesen Eintrag und berechnet den diskreten Logarithmus $a = s^\star - s''^\star - \delta^\star \mod p$ und löst damit das 2-SDH-Problem mit nicht vernachlässigbarer Wahrscheinlichkeit $\epsilon(\lambda) - \nu(\lambda)$, wobei ν eine vernachlässigbare Funktion ist. □

9.4 Das faire teilbare elektronische Geldsystem mit vertrauenswürdigen, manipulationssicheren Observern FTGvmO-I

Wenn ein hohes Maß an Vertrauen in die Manipulationssicherheit der Observer gesetzt wird, kann eine noch wesentlich effizientere Integration der Observer ermöglicht werden. Allerdings sind in diesem Fall sämtliche Sicherheitseigenschaften nur dann garantiert, wenn der Angreifer die Observer nicht manipulieren kann.

Die Grundidee der Integration ist die folgende: Wie im obigen Abschnitt 9.3 werden die Observer von einer Trusted-Third-Party an die Kunden verteilt, besitzen das Schlüsselpaar $(\mathsf{pk}_{\mathcal{O}}, \mathsf{sk}_{\mathcal{O}}) = (I := \mathfrak{g}^u, u) \in \mathbb{G}_p \times \mathbb{Z}_p^*$ und übernehmen in den Abhebe- und Bezahlprotokollen sämtliche Berechnungen. Die Kunden leiten die vom Observer erhaltenen Daten erneut lediglich an die Bank bzw. den Händler weiter. Da ein hohes Maß an Vertrauen an die Manipulationssicherheit der Observer gesetzt wird, müssen alle Sicherheitseigenschaften nur dann garantiert sein, wenn der Angreifer die Observer nicht manipulieren kann. Aus diesem Grund muss eine Münze keine Seriennummer und kein Sicherheitstag enthalten. Denn solange kein Observer manipuliert werden kann, wird sich jeder Observer gemäß Protokoll verhalten und kein Double-Spending begehen. Folglich muss im Abhebeprotokoll weder ein Binär-Baum generiert noch der entsprechende Akkumulator der Serienschlüssel berechnet und signiert werden. Daher werden die Generatoren $u_0, \ldots, u_K \in \mathbb{G}_1$ sowie $h_1, \ldots, h_K \in \mathbb{G}_2$ nicht benötigt. Ein weiterer enormer Vorteil ist, dass im Abhebeprotokoll stets ein beliebiger Geldbetrag K' abgehoben werden kann, da nichts über einen Akkumulator bewiesen werden muss.

Stattdessen wird im Abhebeprotokoll lediglich vom Observer eine Signature-of-Knowledge bzw. eine Schnorr-Signatur bzgl. $\mathsf{pk}_{\mathcal{O}}$ (vgl. Unterabschnitt 2.10.2) und von der Bank eine BBS+-Signatur auf den geheimen Schlüssel $\mathsf{sk}_{\mathcal{O}}$ erstellt.

Wie auch in Abschnitt 9.2 wird die Integration der Observer exemplarisch an FTG-I gezeigt, kann aber völlig analog auf die übrigen elektronischen Geldsysteme angewandt werden. Die Trusted-Third-Party $\mathcal{TTP}$ kann erneut dieselbe Partei wie der Deanonymisierer $\mathcal{D}$ oder eine weitere unabhängige Partei sein.

Im Folgenden werden hauptsächlich die Veränderungen im Vergleich zu FTGvO-I detailliert beschrieben. Die Algorithmen BGen und DGen sowie die Protokolle $\mathsf{Registry}$ und $\mathsf{Account}$ werden wie beim fairen teilbaren Geldsystem mit vertrauenswürdigem Observer FTGvO-I durchgeführt. Da sich die Kontoeröffnungs-, Abhebe- und Bezahlprotokolle $\mathsf{Account}, \mathsf{Withdraw}$ und Spend verändern, werden der Algorithmus $\mathsf{VerifyWithdraw}$ sowie die Protokolle Spend-$K, \mathsf{Deposit}, \mathsf{UTrace}$ und CTrace dann analog zum fairen teilbaren Geldsystem mit vertrauenswürdigem Observer FTGvO-I durchgeführt und aus diesem Grund nicht beschrieben. Die Algorithmen $\mathsf{Identify}$ und $\mathsf{VerifyGuilt}$ werden bei diesem Geldsystem nicht benötigt, da ein Double-Spending unter der Annahme 9.2 nur mit vernachlässigbarer Wahrscheinlichkeit möglich ist (siehe Unterabschnitt 9.4.1).

Setup geht wie beim fairen teilbaren Geldsystem FTG-I (und somit wie bei FTGvO-I) vor, außer dass die Generatoren $g_2, u_0, \ldots, u_K \in \mathbb{G}_1$ sowie $h_1, \ldots, h_K \in \mathbb{G}_2$ nicht benötigt werden. Die Ausgabe sind die Systemparameter

$$\mathsf{sp} = (p, \mathbb{G}_1', \mathbb{G}_2', \mathbb{G}_T', \mathbb{G}_p', \hat{e}, g, g_0, g_1, h, \mathfrak{g}, \mathsf{H}) \leftarrow \mathsf{Setup}\left(1^{\lambda}\right).$$

Withdraw: Damit ein Kunde $\mathcal{K}$ gemeinsam mit seinem Observer $\mathcal{O}$ eine Geldbörse mit einem beliebigen Wert K' bei der Bank $\mathcal{B}$ abheben kann, führen $\mathcal{O}, \mathcal{K}$ und $\mathcal{B}$ das Abhebeprotokoll $\mathsf{Withdraw}$ durch. Zuvor authentifizieren sich $\mathcal{K}$ und $\mathcal{B}$ sowie $\mathcal{K}$ und $\mathcal{O}$ gegenseitig. Danach gleicht $\mathcal{K}$ jeweils mit $\mathcal{B}$ und $\mathcal{O}$ den aktuellen Zeitpunkt ts ab (bzw. sendet ts an $\mathcal{O}$, falls der Observer über keine zeitlichen Informationen verfügt).Sei i die Anzahl der bisher durchgeführten Abhebeprotokolle des Kunden.

Schritt 1: Der Observer $\mathcal{O}$ wählt zufällig zwei Zahlen $s', t \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Commitment $C = g_0^{s'} g_1^{u}$ sowie das Element $E = h^t$. Dann erstellt $\mathcal{O}$ die folgende Signature-of-Knowledge:

$$SoK_{\mathsf{W}}\left[(u) : I = \mathfrak{g}^u\right](C||E||\mathsf{ts}||K').$$

Anschließend sendet $\mathcal{O}$ die Elemente C, E und die Signature-of-Knowledge SoK_{W} an den Kunden $\mathcal{K}$.

Schritt 2: Der Kunde $\mathcal{K}$ leitet die Elemente C, E und die Signature-of-Knowledge SoK_{W} weiter an die Bank $\mathcal{B}$.

Schritt 3: Die Bank $\mathcal{B}$ verifiziert SoK_{W}, wählt zufällig zwei Zahlen $e, s'' \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Element $\Sigma = (g g_0^{s''} C)^{\frac{1}{x+e}}$. Dann sendet $\mathcal{B}$ das Tripel

(Σ, e, s'') an $\mathcal{K}$ und bucht dem Kunden $\mathcal{K}$ den Betrag K' vom Konto mit der Kontonummer I ab. Anschließend speichert $\mathcal{B}$ die Abhebeinformationen $\mathbb{T} = (I, C, E, SoK_{\mathsf{W}}, \mathsf{ts}, K')$ in der Datenbank $\mathbb{D}_W$ ab.

Schritt 4: Der Kunde $\mathcal{K}$ verifiziert die Signatur durch Überprüfung der Gleichung $\hat{e}(\Sigma, Xh^e) \stackrel{?}{=} \hat{e}\left(gg_0^{s''}C, h\right)$, speichert das Paar $(i+1, J_{i+1})$ mit $J_{i+1} = K'$ und leitet das Tripel (Σ, e, s'') weiter an $\mathcal{O}$.

Schritt 5: Der Observer $\mathcal{O}$ berechnet $s = s' + s'' \mod p$, verifiziert die Signatur durch Überprüfung der Gleichung $\hat{e}(\Sigma, Xh^e) \stackrel{?}{=} \hat{e}\left(gg_0^{s''}C, h\right)$ und speichert $\mathbb{W} = (i+1, \Sigma, e, s, t, J)$ auf dem nichtauslesbaren Teil ab, wobei $J = K'$ der Zähler der Geldbörse ist.

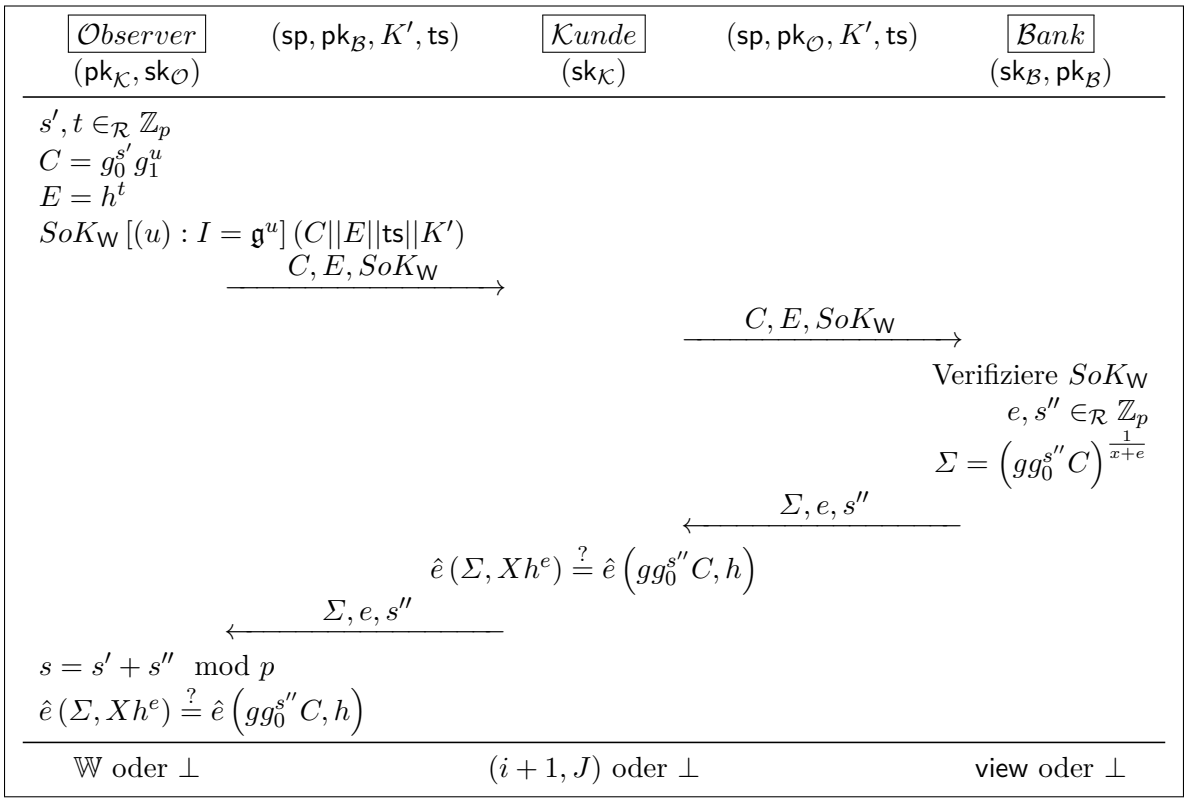

Abbildung 9.4: Abhebeprotokoll Withdraw von FTGvmO-I

Spend: Damit ein Kunde $\mathcal{K}$, der im Besitz des Paares (i, J) ist, gemeinsam mit seinem Observer $\mathcal{O}$, der im Besitz der Geldbörse $\mathbb{W} = (i, \Sigma, e, s, t, J)$ ist, bei einem Händler $\mathcal{H}$ mit einer elektronischen Münze coin mit einem beliebigen Wert $k \leq J$ bezahlen kann, führen $\mathcal{O}, \mathcal{K}$ und $\mathcal{H}$ das Bezahlprotokoll Spend durch, welches in Abbildung 9.5 dargestellt ist. Zuvor authentifizieren sich jedoch $\mathcal{O}$ und $\mathcal{K}$ gegenseitig und der Händler $\mathcal{H}$ authentifiziert sich gegenüber dem Kunden $\mathcal{K}$. Danach

gleicht $\mathcal{K}$ jeweils mit $\mathcal{H}$ und $\mathcal{O}$ den aktuellen Zeitpunkt ts ab (bzw. sendet ts an $\mathcal{O}$, falls der Observer über keine zeitlichen Informationen verfügt). Weiter überprüft der Observer zunächst, ob die Bedingung $k \leq J$ erfüllt ist. Andernfalls bricht $\mathcal{O}$ direkt ab und sendet das Symbol $\perp$ an $\mathcal{K}$.

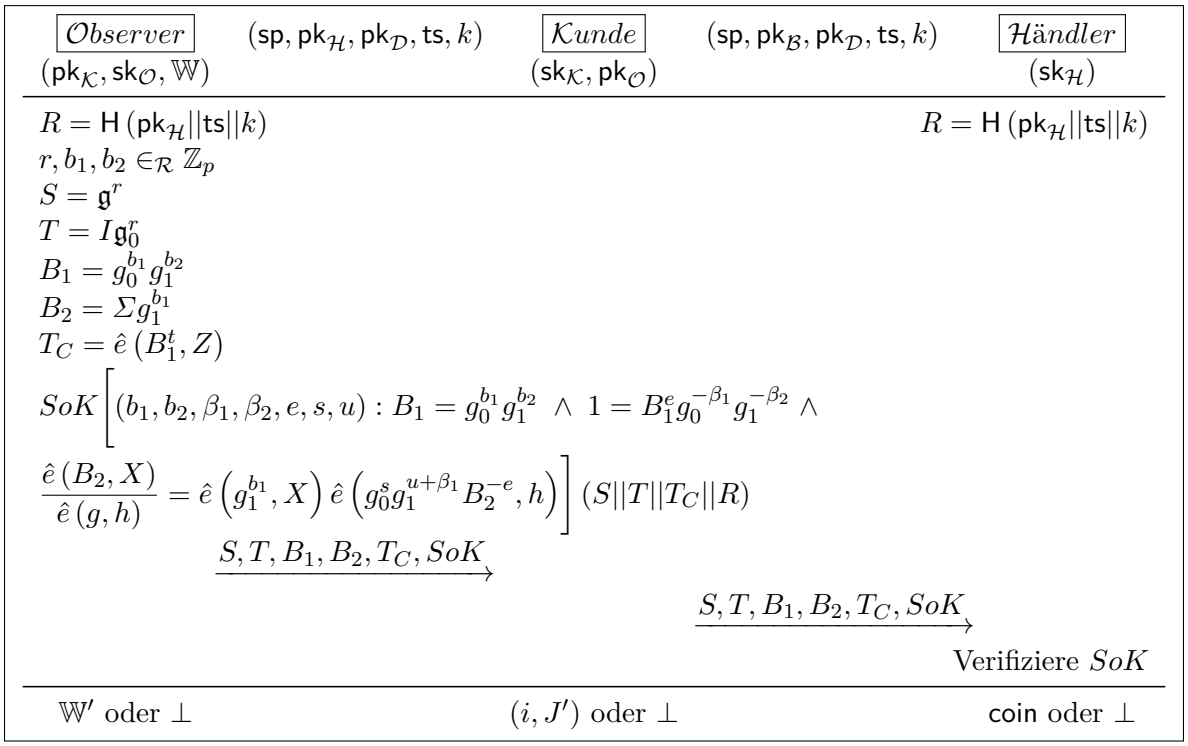

Abbildung 9.5: Bezahlprotokoll Spend von FTGvmO-I

Schritt 1: Der Händler $\mathcal{H}$ und der Observer $\mathcal{O}$ berechnen beide den Hashwert $R = \mathsf{H}(\mathsf{pk}_{\mathcal{H}}||\mathsf{ts}||k)$.

Schritt 2: Der Observer wählt zufällig drei Zahlen $r, b_1, b_2 \in_{\mathcal{R}} \mathbb{Z}_p$ und erstellt eine ElGamal-Verschlüsselung der Kontonummer I durch $S = \mathfrak{g}^r$ und $T = I\mathfrak{g}_0^r$, da dies für die Kundenverfolgung benötigt wird. (Die Elemente S und T sind nun allerdings keine Seriennummern bzw. keine Sicherheitstags, sondern lediglich eine ElGamal-Verschlüsselung von I.) Weiter berechnet $\mathcal{O}$ wie üblich die Elemente $B_1 = g_0^{b_1} g_1^{b_2}$ und $B_2 = \Sigma g_1^{b_1}$ sowie den Münz-Identifikator $T_C = \hat{e}(B_1^t, Z)$. Dann erstellt $\mathcal{O}$ die folgende Signature-of-Knowledge:

$$SoK\Big[(b_1, b_2, \beta_1, \beta_2, e, s, u,) : B_1 = g_0^{b_1} g_1^{b_2} \ \wedge \ 1 = B_1^e g_0^{-\beta_1} g_1^{-\beta_2} \wedge \frac{\hat{e}(B_2, X)}{\hat{e}(g, h)} = \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(g_0^s g_1^{u+\beta_1} B_2^{-e}, h\right)\Big] (S||T||T_C||R)\,.$$

Schließlich sendet $\mathcal{O}$ die Elemente S, T, B_1, B_2, T_C sowie die Signature-of-Knowledge SoK an den Kunden $\mathcal{K}$ und verringert den Zähler J um k.

Schritt 3: Der Kunde $\mathcal{K}$ leitet die Elemente S, T, B_1, B_2, T_C sowie die Signature-of-Knowledge SoK weiter an den Händler $\mathcal{H}$ und verringert den Zähler J um k.

Schritt 4: Der Händler $\mathcal{H}$ verifiziert SoK und speichert die Münze $\mathsf{coin} = (k, S, T, T_C, R, \Phi = (B_1, B_2, SoK, \mathsf{ts}, \mathsf{pk}_{\mathcal{H}}))$ ab.

Beachte, dass die Integration der Observer in das teilbare Geldsystem TG-I identisch zur Integration der Observer in das teilbare Geldsystem TG-II realisiert werden kann.

9.4.1 Sicherheitsanalyse von FTGvmO-I

Satz 9.4.1. *Das faire teilbare elektronische Geldsystem mit vertrauenswürdigen, manipulationssicheren Observern FTGvmO-I ist im Random-Oracle-Modell unter der Annahme 9.2 ein sicheres faires elektronisches Geldsystem mit Observern, falls die q-SDH-Annahme sowie die DBDH-Annahme für* $\mathsf{Setup}_{\mathsf{Bil}}$ *gelten und die DDH-Annahme für* $\mathsf{Setup}_{\mathsf{G}}$ *gilt.*

Beweis. Da es die Algorithmen $\mathsf{Identify}$ und $\mathsf{VerifyGuilt}$ nicht gibt, ist die **Double-Spending-Beschuldigung** trivialer Weise erfüllt. Die Sicherheitsziele **Anonymität**, **Abhebe-Beschuldigung**, **Kundenverfolgung-Beschuldigung** und **Münzverfolgung-Beschuldigung** folgen mit der Annahme 9.2 analog zum fairen teilbaren Geldsystem mit vertrauenswürdigem Observer FTGvO-I. Des Weiteren existieren unter der Annahme 9.2 keine **eingehenden** und **ausgehenden Informationen** (vgl. auch [Sch09]).

Unfälschbarkeit, korrekte Kundenverfolgung, korrekte Münzverfolgung sowie Double-Spending-Prevention: Sei $\mathcal{A}$ ein polynomieller Angreifer, der die Unfälschbarkeit, die korrekte Kundenverfolgung, die korrekte Münzverfolgung oder die Double-Spending-Prevention des Systems bricht.

Wir werden analog zu FTGvO-I zeigen, dass die Erfolgswahrscheinlichkeit des Angreifers $\mathcal{A}$ unter der DLog-Annahme für $\mathsf{Setup}_{\mathsf{G}}$ und der Sicherheit des BBS+-Signaturverfahrens vernachlässigbar ist.

Wenn die Observer bei jedem Protokoll involviert sind und der Angreifer $\mathcal{A}$ alle vom Observer erhaltenen Daten weiter an die Bank bzw. den Händler leitet, kann $\mathcal{A}$ kein Spiel gewinnen, da sich die Observer gemäß dem jeweiligen Protokoll verhalten. Folglich muss der Angreifer $\mathcal{A}$ bei mindestens einem Protokoll vom Protokollverlauf abweichen, um ein Spiel zu gewinnen. Wenn $\mathcal{A}$ bei einem Protokoll vom Protokollverlauf abweicht und andere als die vom Observer erhaltenen Daten weiterleitet, muss $\mathcal{A}$ auch, außer mit vernachlässigbarer Wahrscheinlichkeit, die entsprechende Signature-of-Knowledge ohne den Observer erstellen. Somit gibt es wie bei FTGvO-I die drei Möglichkeiten, dass der

Angreifer (1) von einem Kontoeröffnungsprotokoll Account abweicht und die Signature-of-Knowledge SoK_{TTP} ohne eine entsprechende Orakel-Anfrage selbst erstellt, oder (2) von einem Abhebeprotokoll Withdraw abweicht und die Signature-of-Knowledge SoK_{W} ohne eine entsprechende Orakel-Anfrage selbst erstellt, oder (3) von einem Bezahlprotokoll Spend abweicht und die Signature-of-Knowledge SoK ohne eine entsprechende Orakel-Anfrage selbst erstellt.

Die beiden Fälle (1) und (2) sind jedoch analog zum in Abschnitt 9.3 beschriebenen Geldsystem FTGvO-I vernachlässigbar.

Es bleibt zu zeigen, dass der Angreifer $\mathcal{A}$ ebenfalls nur mit vernachlässigbarer Wahrscheinlichkeit von einem Bezahlprotokoll abweichen und die Signature-of-Knowledge SoK ohne eine entsprechende Orakel-Anfrage selbst erstellen kann.

Falls der Angreifer $\mathcal{A}$ von einem Bezahlprotokoll abweicht, muss $\mathcal{A}$ bei der Münze coin oder coin$'$ die Signature-of-Knowledge SoK über ein Nachrichten-Signatur-Paar $(u^\star, (\Sigma^\star, e^\star, s^\star))$ selbst erstellen. Daher kann $\mathcal{B}$ dieses Nachrichten-Signatur-Paar extrahieren. Dann gibt es die beiden Möglichkeiten, dass das Nachrichten-Signatur-Paar (3.1) nie oder (3.2) mindestens einmal bei der Simulation einer $\mathcal{O}^{\mathsf{W}}_{\mathcal{O},\mathcal{B}}$- oder $\mathcal{O}^{\mathsf{W}}_{\mathcal{K},\mathcal{B}}$-Anfrage erzeugt wurde. Der Fall (3.1) ist jedoch ebenfalls analog zum in Abschnitt 9.3 beschriebenen Geldsystem FTGvO-I vernachlässigbar. Es ist lediglich zu beachten, dass es sich nun um BBS+-Signaturen auf die Nachricht $u \in \mathbb{Z}_p$ handelt. Somit ist der Fall (3.1) ein Widerspruch zur Sicherheit des BBS+-Signaturverfahrens.

Der Fall (3.2) ist ebenfalls analog zum in Abschnitt 9.3 beschriebenen Geldsystem FTGvO-I vernachlässigbar. □

9.5 Effizienzvergleich der teilbaren elektronischen Geldsysteme mit Observern

Analog zu Tabelle 7.5 werden in der folgenden Tabelle 9.1 die fairen teilbaren elektronischen Geldsysteme mit Observern FTGO-I und FTGvO-I mit dem ihnen zugrunde liegenden fairen teilbaren elektronischen Geldsystem FTG-I für eine 256-Bit Primzahl p verglichen. Dabei wird lediglich der Mehraufwand der fairen teilbaren Geldsysteme mit Observern angegeben. Abschließend werden in der Tabelle 9.2 die beiden fairen teilbaren elektronischen Geldsysteme FTG-I und FTGvmO-I miteinander verglichen.

	FTGO-I	FTGvO-I
sp	$\mathbb{G}_1$	
Bytes	33	
Registry	/	$\mathbb{G}_p^2 \times \mathbb{Z}_p^2$
Bytes	/	130
$\mathcal{TTP}$	/	2 ME
Account	$\mathbb{G}_1 \times \mathbb{G}_2$	$\mathbb{G}_p \times \mathbb{Z}_p^2$
Bytes	98	97
$\mathcal{B}ank$	2 ME	1 ME
Daten in $\mathbb{D}_{\mathcal{K}}$	$\mathbb{Z}_p$	$\mathbb{G}_p$
Bytes	32	33
Withdraw $\mathcal{O} \leftrightarrow \mathcal{K}$	$\mathbb{G}_2 \times \mathbb{Z}_p^2$	$\mathbb{G}_1^3 \times \mathbb{G}_2 \times \mathbb{Z}_p^9$
Bytes	129	452
Withdraw $\mathcal{K} \leftrightarrow \mathcal{B}$	$\mathbb{Z}_p^2$	
Bytes	32	
$\mathcal{O}bserver$ und $\mathcal{K}unde$	3 ME	2 ME + 2 P
$\mathcal{B}ank$	2 ME	
$(\mathbb{T}, \sigma')$	$\mathbb{G}_1 \times \mathbb{Z}_p^2$	
Bytes	97	
Spend $\mathcal{O} \leftrightarrow \mathcal{K}$	$\mathbb{G}_1^2 \times \mathbb{G}_2 \times \mathbb{Z}_p^5$	$\mathbb{G}_p^2 \times \mathbb{G}_1^3 \times \mathbb{G}_T \times \mathbb{Z}_p^{12}$
Bytes	291	677
Spend $\mathcal{K} \leftrightarrow \mathcal{H}$	$\mathbb{G}_1^2 \times \mathbb{G}_2 \times \mathbb{Z}_p^{13}$	
Bytes	515	
$\mathcal{O}bserver$ und $\mathcal{K}unde$	15 ME + 4 P	
$\mathcal{H}ändler$	6 ME + 2 P	
Deposit	$\mathbb{G}_1^2 \times \mathbb{G}_2 \times \mathbb{Z}_p^{13}$	
Bytes	515	
$\mathcal{B}ank$	6 ME + 2 P	
coin $\in \mathbb{D}_C$	$\mathbb{G}_1^2 \times \mathbb{G}_2 \times \mathbb{Z}_p^{13}$	
Bytes	515	

Tabelle 9.1: Mehraufwand für elektronische Geldsysteme mit Observern

	FTG-I	FTGmvO-I
sp	$\mathbb{G}_p \times \mathbb{G}_1^{K+5} \times \mathbb{G}_2^{K+1}$	$\mathbb{G}_p \times \mathbb{G}_1^3 \times \mathbb{G}_2$
Bytes	100.615	197
Registry	/	$\mathbb{G}_p^2 \times \mathbb{Z}_p^2$
Bytes	/	130
$\mathcal{K}unde$	/	1 ME
$\mathcal{TTP}$	/	2 ME
Account	$\mathbb{G}_p$	$\mathbb{G}_p^2 \times \mathbb{Z}_p^2$
Bytes	33	130
$\mathcal{K}unde$	1 ME	
$\mathcal{B}ank$		1 ME
Daten in $\mathbb{D}_\mathcal{K}$	$\mathbb{G}_p$	$\mathbb{G}_p^2$
Bytes	33	66
Withdraw $\mathcal{O} \leftrightarrow \mathcal{K}$	/	$\mathbb{G}_1^2 \times \mathbb{G}_2 \times \mathbb{Z}_p^4$
Bytes	/	259
Withdraw $\mathcal{K} \leftrightarrow \mathcal{B}$	$\mathbb{G}_1^3 \times \mathbb{G}_2 \times \mathbb{Z}_p^9$	$\mathbb{G}_1^2 \times \mathbb{G}_2 \times \mathbb{Z}_p^4$
Bytes	452	259
$\mathcal{O}bserver$	/	5 ME + 2 P
$\mathcal{K}unde$	$(2K+7)$ ME + 3 P	2 ME + 2 P
$\mathcal{B}ank$	5 ME + 2 P	2 ME
tree	$\mathbb{Z}_p^{3K-2}$	
Bytes	98.240	0
$\mathbb{W}$	$\mathbb{G}_1 \times \mathbb{Z}_p^4$	$\mathbb{G}_1 \times \mathbb{Z}_p^3 \times \log(K')$
Bytes	161	131
$(\mathbb{T}, \sigma')$	$\mathbb{G}_p \times \mathbb{G}_1^2 \times \mathbb{G}_2 \times \mathbb{Z}_p^7$	$\mathbb{G}_p \times \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{Z}_p^2 \times \log(K')$
Bytes	388	197
Spend $\mathcal{O} \leftrightarrow \mathcal{K}$	/	$\mathbb{G}_p^2 \times \mathbb{G}_1^2 \times \mathbb{G}_T \times \mathbb{Z}_p^8$
Bytes	/	516
Spend $\mathcal{K} \leftrightarrow \mathcal{H}$	$\mathbb{G}_p^2 \times \mathbb{G}_1^3 \times \mathbb{G}_T \times \mathbb{Z}_p^{12}$	$\mathbb{G}_p^2 \times \mathbb{G}_1^2 \times \mathbb{G}_T \times \mathbb{Z}_p^8$
Bytes	677	516
$\mathcal{O}bserver$	/	8 ME + 2 P
$\mathcal{K}unde$	13 ME + 4 P	
$\mathcal{H}ändler$	$(2k+6)$ ME + 4 P	4 ME + 2 P
Deposit	$\mathbb{G}_p^2 \times \mathbb{G}_1^3 \times \mathbb{G}_T \times \mathbb{Z}_p^{13}$	$\mathbb{G}_p^2 \times \mathbb{G}_1^2 \times \mathbb{G}_T \times \mathbb{Z}_p^9 \times \log(k)$
Bytes	709	550
$\mathcal{B}ank$	$(2k+6)$ ME + 4 P	4 ME + 2 P
coin $\in \mathbb{D}_C$	$\mathbb{G}_p^3 \times \mathbb{G}_1^3 \times \mathbb{G}_T \times \mathbb{Z}_p^{k+13}$	$\mathbb{G}_p^3 \times \mathbb{G}_1^2 \times \mathbb{G}_T \times \mathbb{Z}_p^9 \times \log(k)$
Bytes	4.838	583

Tabelle 9.2: Effizienzvergleich zwischen FTG-I und FTGmvO-I

KAPITEL 10

ZUSAMMENFASSUNG

In Kapitel 3 wurde ein neues Signaturverfahren (BBS+A) konstruiert, das speziell für elektronische Geldsysteme entwickelt wurde, in denen das Akkumulator-Schema von Nguyen eingesetzt wird. Im Gegensatz zum bisher einzig vergleichbaren ESS+-Signaturverfahren ist die BBS+A-Signatur in allen Pairing Typen unter einer Standard-Annahme stark existentiell unfälschbar unter einem adaptiven Angriff mit gewählten Nachrichten. Die teilbaren elektronischen Geldsysteme TG-II (Abschnitt 6.6), FTG-0 (Abschnitt 7.2) und FTG-II (Abschnitt 7.4) sowie die Erweiterung der Geldbörsen mit unterschiedlichen Geldbeträgen (Abschnitt 8.3) wurden erst durch dieses neue Signaturverfahren ermöglicht.

Da in der Literatur verschiedene Sicherheitsdefinitionen elektronischer Geldsysteme vorhanden sind, wurden in Kapitel 4 neue spielbasierende Definitionen entwickelt, die die in der Literatur bestehenden Sicherheitsziele umfassen.

In Kapitel 5 wurde das kompakte elektronische Geldsystem von Au. *et al.* [ASM07] modifiziert und weiterentwickelt, damit Kunden im Abhebeprotokoll einen beliebigen Geldbetrag $K' \leq K$ abheben können und die Komplexität des Bezahlvorgangs konstant und somit insbesondere unabhängig vom Wert der abgehobenen Geldbörse ist. In der Literatur ist nach bestem Wissen bisher kein kompaktes elektronisches Geldsystem mit diesen Eigenschaften zu finden.

Um faire teilbare elektronische Geldsysteme mit Observern zu konstruieren, wurde in Kapitel 6 ein neues teilbares elektronisches Geldsystem (TG-I) entwickelt. Dieses Geldsystem ist nach bestem Wissen das erste teilbare elektronische Geldsystem, das die zentralen Sicherheitsziele gewährleistet und bei dem der Datentransfer der Abhebe- und Bezahlprotokolle unabhängig vom Wert der abgehobenen Geldbörse und der ausgegebenen Münze ist. Somit ist dieses Geldsystem effizienter als alle in der Literatur

vorhandenen, vergleichbaren teilbaren elektronischen Geldsysteme mit denselben zentralen Sicherheitszielen. Das Geldsystem TG-I wurde anschließend zu einem teilbaren elektronischen Geldsystem (TG-II) modifiziert, um noch effizientere Bezahlprotokolle zu erhalten. Insgesamt können bei beiden teilbaren Geldsystemen Geldbörsen im Wert von $K = 2^L$ abgehoben und Münzen im Wert von $k = 2^\ell$ mit $0 \leq \ell \leq L$ zum Bezahlen verwendet werden, wobei L ein festgelegter Wert ist.

Zudem wurde in Kapitel 6 ein Angriff auf das teilbare elektronische Geldsystem [CG10] entwickelt, mit dem die Unfälschbarkeit gebrochen werden kann.

Anschließend wurden die beiden teilbaren elektronischen Geldsysteme TG-I und TG-II in Kapitel 7 zu fairen Geldsystemen erweitert, indem ein Deanonymisierer sowie zwei Protokolle zur Kunden- und Münzverfolgung integriert wurden. Im Gegensatz zu der von M. Au in [Au09, Abschnitt 5.6] vorgeschlagenen Methode, wie jedes teilbare Geldsystem zu einem fairen teilbaren Geldsystem erweitert werden kann, wurden in Kapitel 7

1. die Kundenverfolgung ohne eine zusätzliche Verschlüsselung (der Identität) während des Bezahlprotokolls und

2. die Münzverfolgung ohne eine verifizierbare Verschlüsselung (des Münz-Identifikationsschlüssels), die die eher ineffiziente Cut-and-Choose-Technik verwendet, während des Abhebeprotokolls realisiert.

Da die Komplexität der Münzverfolgung bei diesen fairen teilbaren Geldsystemen allerdings von der Anzahl aller eingelösten Münzen abhängt, wurde in Abschnitt 7.6 eine alternative Methode vorgestellt, aus der allerdings ein ineffizienterer Abhebevorgang resultiert. Des Weiteren wurde in Abschnitt 7.7 die bisher kaum behandelte Möglichkeit thematisiert, ob und wie Kunden nach einem Verlust oder Diebstahl ihr elektronisches Geld sperren und zurückerstatten lassen können, ohne dabei die Anonymität ihrer bereits ausgegebenen Münzen zu verlieren. Es wurde bewiesen, dass dies in keinem elektronischen Geldsystem möglich ist.

In Kapitel 8 wurden verschiedene Modifizierungen der zuvor entwickelten elektronischen Geldsysteme analysiert. So wurde in Abschnitt 8.1 ein effizientes Bezahlprotokoll entwickelt, welches das Bezahlen einer Münze mit einem beliebigen Wert $k \leq K$ ermöglicht, so dass lediglich eine Relation über einen Akkumulator und eine Signatur in zero-knowledge bewiesen werden muss. Der beschriebene Ansatz kann zwar entsprechend auf die teilbaren Geldsysteme [ASM08] und [CG10] übertragen werden, jedoch müssen in diesen Systemen dennoch Relationen über $O(\log(k))$ Akkumulatoren und $O(\log(k))$ Signaturen in zero-knowledge bewiesen werden. In den Abschnitten 8.2 und 8.3 wurden anschließend zwei unterschiedliche Möglichkeiten vorgestellt, wie eine Geldbörse mit einem beliebigen Wert $K' \leq K$ abgehoben werden kann. Wie bereits erwähnt, kann die in Abschnitt 8.3 beschriebene Methode nicht allein durch das ESS+-Signaturverfahren realisiert werden.

Schließlich wurden in Kapitel 9 insgesamt drei Konzepte angegeben, wie Observer effizient in die zuvor entwickelten (fairen) teilbaren elektronischen Geldsysteme integriert werden können, um eine mehrfache Ausgabe der Münzen unmittelbar während des Bezahlvorgangs zu verhindern. Die erste Variante benötigt keine Trusted-Third-Party, verhindert aber lediglich ein Over-Spending (Abschnitt 9.2). Werden die Observer von einer Trusted-Third-Party an die Kunden verteilt, können die Observer auch ein Double-Spending verhindern (Abschnitt 9.3). Unter der Annahme, dass alle Observer manipulationssicher sind, wurde in Abschnitt 9.4 ein noch effizienteres faires teilbares elektronisches Geldsystem mit Observern entwickelt. Durch die Integration der Observer wird zudem in diesem System der Datentransfer zwischen Kunde und Bank bzw. Kunde und Händler im Vergleich zum ursprünglichen fairen teilbaren elektronischen Geldsystem verringert. Nach bestem Wissen sind keine Veröffentlichungen bekannt, die sich mit einem Observereinsatz bei teilbaren oder kompakten Geldsystemen beschäftigen. Des Weitern wurde ebenfalls bewiesen, dass, im Gegensatz zur bisher falschen Annahme, die Anonymität eines elektronischen Geldsystems mit Observern gebrochen werden kann, wenn die Bank wieder in den Besitz der von ihr ausgehändigten manipulierten Observer kommt.

In der folgenden Tabelle 10.1 werden die Komplexitäten der in dieser Arbeit konstruierten (fairen) teilbaren elektronischen Geldsysteme (mit Observern) festgehalten. Dabei bezeichnet $\mathsf{Withdraw}^*$ das modifizierte Abhebeprotokoll aus Abschnitt 8.2 und Spend^* das modifizierte Bezahlprotokoll aus Abschnitt 8.1.

Zur besseren Übersicht werden Felder leer gelassen, wenn sich durch die Integration der Fairness, die Modifizierung des Abhebeprotokolls oder die Integration der Observer im Vergleich zum ursprünglichen (teilbaren bzw. fairen teilbaren) Geldsystem keine Änderungen ergeben.

In der dritten Spalte werden die Komplexitäten der fairen Geldsysteme FTG-I und FTG-II aufgelistet, wobei auch das Sperren von Münzen berücksichtigt wird und das Symbol # die Anzahl aller Kunden des Geldsystems bezeichnet. In der vierten Spalte sind die Komplexitäten der Geldsysteme mit der Modifizierung aus Abschnitt 8.3 dargestellt, wobei im Abhebeprotokoll einer von $n \leq K - 1$ verschiedenen Geldbeträgen $K_1, \ldots, K_n$ abgehoben werden kann. In der letzten Spalte werden die Komplexitäten der Geldsysteme mit der Integration der Observer aus Abschnitt 9.4 beschrieben. Dabei ist zu beachten, dass im Abhebeprotokoll jeder beliebige Betrag K' abgehoben und im Bezahlprotokoll mit einer Münze mit einem beliebigen Wert $k \leq K'$ bezahlt werden kann.

Die Komplexitäten $O(\lambda + \log(k))$ bzw. $O(\lambda + \log(K))$ kommen zustande, da der Wert k bzw. K als $\log(k)$-Bit bzw. $\log(K)$-Bit String gespeichert werden muss, wenn unterschiedliche Geldbeträge abgehoben und ausgegeben werden können. Wird nur das Zahlen einer Münze vom Wert $k = 2^\ell$ für $0 \leq \ell \leq L$ unterstützt, ist ein $\log(\ell)$-Bit String ausreichend, um den Wert $k = 2^\ell$ zu repräsentieren (vgl. auch Abschnitt 6.8).

	TG-I / TG-II	FTG-I / FTG-II	Modifizierung Abschnitt 8.3	Observer Abschnitt 9.4
sp	$O(\lambda \cdot K)$			$O(\lambda)$
$\mathsf{pk}_{\mathcal{B}}$	$O(\lambda)$		$O(\lambda \cdot K \cdot n)$	
$\mathsf{sk}_{\mathcal{B}}$	$O(\lambda)$		$O(\lambda \cdot n)$	
$(\mathsf{pk}_{\mathcal{D}}, \mathsf{sk}_{\mathcal{D}})$	/	$O(\lambda)$		
Withdraw	$O(\lambda)$			
$\mathcal{K}unde$	$O(\lambda \cdot K)$		$O(\lambda \cdot K_i)$	$O(\lambda)$
$\mathcal{B}ank$	$O(\lambda)$			
$\mathbb{W} + \mathsf{tree}$	$O(\lambda \cdot K)$		$O(\lambda \cdot K_i)$	$O(\lambda + \log(K'))$
$(\mathbb{T}, \sigma')$	$O(\lambda)$		$O(\lambda + \log(K_i))$	$O(\lambda + \log(K'))$
Spend	$O(\lambda)$			
$\mathcal{K}unde$	$O(\lambda)$			
$\mathcal{H}ändler$	$O(\lambda \cdot k)$	$O(\lambda \cdot (k + \#))$		$O(\lambda)$
coin	$O(\lambda + \log(\ell))$			$O(\lambda + \log(k))$
Deposit	$O(\lambda + \log(\ell))$			$O(\lambda + \log(k))$
$\mathcal{B}ank$	$O(\lambda \cdot k)$			$O(\lambda)$
$\mathsf{coin} \in \mathbb{D}_C$	$O(\lambda \cdot k)$			$O(\lambda + \log(k))$
$\mathsf{Withdraw}^*$	$O(\lambda \cdot \log(K))$		/	/
$\mathcal{K}unde$	$O(\lambda \cdot K)$		/	/
$\mathcal{B}ank$	$O(\lambda \cdot \log(K))$		/	/
$\mathbb{W} + \mathsf{tree}$	$O(\lambda \cdot K)$		/	/
$(\mathbb{T}, \sigma')$	$O(\lambda \cdot \log(K))$		/	/
Spend^*	$O(\lambda \cdot \log(k))$			/
$\mathcal{K}unde$	$O(\lambda \cdot \log(k))$			/
$\mathcal{H}ändler$	$O(\lambda \cdot k)$	$O(\lambda \cdot (k + \#))$		/
coin	$O(\lambda \cdot \log(k))$			/
UTrace	/	$O(\lambda + \log(k))$		
$\mathcal{B}ank$	/	$O(\#)$		
$\mathcal{D}ean.$	/	$O(\lambda)$		
CTrace	/	$O(\lambda \cdot K)$		
$\mathcal{B}ank$	/	$O(\lambda \cdot K \cdot \#)$		
$\mathcal{D}ean.$	/	$O(\lambda \cdot K)$		

Tabelle 10.1: Komplexitäten der teilbaren elektronischen Geldsysteme

10.1 Offene Fragestellungen

Die teilbaren elektronischen Geldsysteme in Kapitel 6 sind so konzipiert, dass der Datentransfer im Abhebe- und Bezahlprotokoll unabhängig vom entsprechenden Wert der Geldbörse bzw. der Münze ist. Durch den Einsatz des Akkumulator-Schemas von Nguyen sind allerdings die öffentlichen Systemparameter, der Berechnungsaufwand der Binär-Bäume und somit der Speicherplatz der Geldbörsen linear abhängig von der Größe der Geldbörsen. Somit ist offen, ob bei einem teilbaren elektronischen Geldsystem in den genannten Bereichen eine Komplexität von $O(\lambda \cdot \log(K))$ realisiert werden kann, wie bspw. bei dem kompakten Geldsystem [CHL05]. Des Weiteren muss bereits der Händler während des Bezahlprotokolls alle Serienschlüssel berechnen, wodurch die Berechnungskomplexität $O(\lambda \cdot \log(k))$ beträgt. Im teilbaren Geldsystem [ASM08] ist der Rechenaufwand des Händlers jedoch unabhängig vom Wert der Münze. Daher ist eine weitere Fragestellung, ob die Händler bei einem teilbaren Geldsystem, das die zentralen Sicherheitsziele gewährleistet, ebenfalls eine Berechnungskomplexität von $O(\lambda)$ erreichen können, ohne dass die Effizienz des Abhebevorgangs reduziert wird.

Während in Kapitel 7 eine Kundenverfolgung sehr effizient in teilbare elektronische Geldsysteme integriert werden kann, sind die in dieser Arbeit vorgestellten Methoden zur Münzverfolgung abhängig von der Anzahl aller Kunden. Die in [CHL05] entwickelte Münzverfolgung ist zwar unabhängig von der Anzahl der Kunden, jedoch wird im Bezahlprotokoll ein verifizierbares Verschlüsselungsverfahren verwendet, das eine eher ineffiziente Cut-and-Choose-Technik einsetzt. Folglich sollte in Zukunft untersucht werden, ob eine effiziente Münzverfolgung in teilbare elektronische Geldsysteme integriert werden kann, die unabhängig von der Anzahl der Kunden ist und die Komplexität der bestehenden Protokolle beibehält.

Die durch die Bank ausgehändigten Observer in Kapitel 9 gewährleisten keine Double-Spending-Prevention, sondern lediglich eine Over-Spending-Prevention. Somit ist es offen, ob ein teilbares elektronisches Geldsystem mit Observern auch eine Double-Spending-Prevention realisieren kann, wenn die Observer von der Bank ausgehändigt werden, ohne dass dies Auswirkungen auf die Sicherheit und die Komplexität des Geldsystems hat.

ANHANG A

ABLAUF DER SIGNATURES-OF-KNOWLEDGE

Im Anhang werden die benötigten Signatures-of-Knowledge (SoK) der Protokolle detailliert beschrieben und deren Sicherheit bewiesen.

A.1 Die SoK des Issue-Protokolls

Der Prover $\mathcal{P}(\mathsf{pk}, C, l, A_0, s', m_1, \ldots, m_L, a_0, \phi(l))$ erstellt für den Verifier $\mathcal{V}(\mathsf{pk}, C, l, A_0)$ im BBS+A-Signaturerstellungsprotokoll Issue die folgende Signature-of-Knowledge:

$$SoK\Big[(s', m_1, \ldots, m_L, a_0, \phi(l)) : \hat{e}(C, h)\,\hat{e}\left(A_0, h_1 h^{-l}\right)^{-1} = \hat{e}\left(g_0^{s'} g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(l)+a_0 l} u_1^{-a_0}, h\right)\Big].$$

Commitment: Der Prover $\mathcal{P}$ wählt zufällig die Zahlen $\rho_s, \rho_{m_1}, \ldots, \rho_{m_L}, \rho_{a_0}, \rho_\phi \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet das Commitment

$$T = \hat{e}\left(g_0^{\rho_s} g_1^{\rho_{m_1}} \cdots g_L^{\rho_{m_L}} u_0^{\rho_\phi + \rho_{a_0} l} u_1^{-\rho_{a_0}}, h\right).$$

Challenge: Der Prover $\mathcal{P}$ berechnet die Challenge

$$c = \mathsf{H}(C||A_0||T).$$

Response: Der Prover $\mathcal{P}$ berechnet in $\mathbb{Z}_p$ die Responses

$$z_s = \rho_s + cs', \qquad z_{m_1} = \rho_{m_1} + cm_1, \ \ldots, \qquad z_{m_L} = \rho_{m_L} + cm_L,$$

$$z_{a_0} = \rho_{a_0} + ca_0, \qquad z_\phi = \rho_\phi + c\phi(k)$$

und sendet diese mit der Challenge an $\mathcal{V}$.

Verify: Der Verifier $\mathcal{V}$ berechnet das Commitment

$$\widetilde{T} = \hat{e}\left(g_0^{z_s} g_1^{z_{m_1}} \cdots g_L^{z_{m_L}} u_0^{z_\phi + z_{a_0} l} u_1^{-z_{a_0}} C^{-c} A_0^{-lc}, h\right) \hat{e}\left(A_0^c, h_1\right).$$

und überprüft

$$c \stackrel{?}{=} \mathsf{H}\left(C||A_0||\widetilde{T}\right).$$

Satz A.1.1. *Die oben beschriebene Signature-of-Knowledge erfüllt die Durchführbarkeit, die spezielle Korrektheit und die Special-Honest-Verifier-Zero-Knowledge-Eigenschaft.*

Beweis. Wir zeigen zunächst die Durchführbarkeit, dann die spezielle Korrektheit und anschließend die Special-Honest-Verifier-Zero-Knowledge-Eigenschaft.

Durchführbarkeit: Verhält sich der Prover $\mathcal{P}$ gemäß dem Protokoll, dann gilt:

$$\begin{aligned}
\widetilde{T} &= \hat{e}\Big(g_0^{z_s} g_1^{z_{m_1}} \cdots g_L^{z_{m_L}} u_0^{z_\phi + z_{a_0} l} u_1^{-z_{a_0}} C^{-c} A_0^{-lc}, h\Big) \hat{e}\left(A_0^c, h_1\right) \\
&= \hat{e}\left(g_0^{\rho_s + cs'} g_1^{\rho_{m_1} + cm_1} \cdots g_L^{\rho_{m_L} + cm_L} u_0^{\rho_\phi + c\phi(l) + \rho_{a_0} l + ca_0 l} u_1^{-\rho_{a_0} - ca_0} C^{-c} A_0^{-lc}, h\right) \hat{e}\left(A_0^c, h_1\right) \\
&= \hat{e}\left(g_0^{\rho_s} g_1^{\rho_{m_1}} \cdots g_L^{\rho_{m_L}} u_0^{\rho_\phi + \rho_{a_0} l} u_1^{-\rho_{a_0}}, h\right) \hat{e}\left(g_0^{cs'} g_1^{cm_1} \cdots g_L^{cm_L} u_0^{c\phi(l) + ca_0 l} u_1^{-ca_0} C^{-c} A_0^{-lc}, h\right) \\
&\quad \hat{e}\left(A_0^c, h_1\right) \\
&= T\Bigg(\hat{e}\left(g_0^{s'} g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(l) + a_0 l} u_1^{-a_0} \left(g_0^{s'} g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}\right)^{-1} \left(W_l u_0^{a_0}\right)^{-l}, h\right) \\
&\quad \hat{e}\left(W_l u_0^{a_0}, h_1\right)\Bigg)^c \\
&= T\left(\hat{e}\left(u_0^{\phi(l)} u_0^{-\phi(\alpha)} W_l^{-l}, h\right) \hat{e}\left(W_l, h_1\right)\right)^c \\
&= T\left(\hat{e}\left(u_0^{\phi(\alpha)}, h\right)^{-1} \hat{e}\left(W_l, h_1 h^{-l}\right) \hat{e}\left(u_0^{\phi(l)}, h\right)\right)^c = T,
\end{aligned}$$

da für den korrekt berechneten Zeugen W_l die Gleichung $\hat{e}\left(u_0^{\phi(\alpha)}, h\right) = \hat{e}\left(W_l, h_1 h^{-l}\right) \hat{e}\left(u_0^{\phi(l)}, h\right)$ gilt (vgl. Unterabschnitt 2.9.2).

Spezielle Korrektheit: Der Knowledge-Extractor $\mathcal{E}$ erhält die Parameter (pk, C, l, A_0), das Commitment T und zwei Challenge-Response-Tupel $(c, z_s, z_r, z_{m_1}, \ldots, z_{m_L}, z_{a_0}, z_\phi)$ und $\left(c', z_s', z_r', z_{m_1}', \ldots, z_{m_L}', z_{a_0}', z_\phi'\right)$ mit $c \neq c'$ und

$$T = \hat{e}\left(g_0^{z_s} g_1^{z_{m_1}} \cdots g_L^{z_{m_L}} u_0^{z_\phi + z_{a_0} l} u_1^{-z_{a_0}} C^{-c} A_0^{-lc}, h\right) \hat{e}\left(A_0^c, h_1\right)$$
$$= \hat{e}\left(g_0^{z'_s} g_1^{z'_{m_1}} \cdots g_L^{z'_{m_L}} u_0^{z'_\phi + z'_{a_0} l} u_1^{-z'_{a_0}} C^{-c'} A_0^{-lc'}, h\right) \hat{e}\left(A_0^{c'}, h_1\right).$$

Daraus folgt:

$$\frac{\hat{e}\left(C, h\right)}{\hat{e}\left(A_0, h_1 h^{-l}\right)} = \hat{e}\left(g_0^{\frac{z_s - z'_s}{c - c'}} g_1^{\frac{z_{m_1} - z'_{m_1}}{c - c'}} \cdots g_L^{\frac{z_{m_L} - z'_{m_L}}{c - c'}} u_0^{\frac{z_\phi - z'_\phi}{c - c'} + \frac{z_{a_0} - z'_{a_0}}{c - c'} l} u_1^{-\frac{z_{a_0} - z'_{a_0}}{c - c'}}, h\right).$$

Daher kann der Knowledge-Extractor $\mathcal{E}$ die folgenden Zahlen in $\mathbb{Z}_p$ berechnen:

$$s' = \frac{z_s - z'_s}{c - c'}, \qquad m_1 = \frac{z_{m_1} - z'_{m_1}}{c - c'}, \ \ldots, \qquad m_L = \frac{z_{m_L} - z'_{m_L}}{c - c'},$$
$$a_0 = \frac{z_{a_0} - z'_{a_0}}{c - c'}, \qquad \phi\left(l\right) = \frac{z_\phi - z'_\phi}{c - c'}.$$

Somit gilt:

$$\hat{e}\left(C, h\right) \hat{e}\left(A_0, h_1 h^{-l}\right)^{-1} = \hat{e}\left(g_0^{s'} g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(l) + a_0 l} u_1^{-a_0}, h\right).$$

Special-Honest-Verifier-Zero-Knowledge: Der Simulator $\mathcal{SIM}$ erhält die Parameter (pk, C, l, A_0) sowie die Challenge c und simuliert die Signature-of-Knowledge folgendermaßen:

Der Simulator $\mathcal{SIM}$ wählt zufällig die Responses $z_s, z_{m_1}, \ldots, z_{m_L}, z_{a_0}, z_\phi \in_{\mathcal{R}} \mathbb{Z}_p$, berechnet das Commitment

$$T = \hat{e}\left(g_0^{z_s} g_1^{z_{m_1}} \cdots g_L^{z_{m_L}} u_0^{z_\phi + z_{a_0} l} u_1^{-z_{a_0}} C^{-c} A_0^{-lc}, h\right) \hat{e}\left(A_0^c, h_1\right)$$

und definiert $\mathsf{H}(C||A_0||T) := c$.

Somit ist die Verifikation von $\mathcal{V}$ trivialer Weise erfüllt. Weiter sind alle Responses unabhängig und identisch verteilt und somit besitzt auch das Commitment T die gleiche Verteilung wie im realen Protokoll. Insgesamt hat $\mathcal{S}$ die Signature-of-Knowledge perfekt simuliert. □

A.2 Die SoK des Prove-Protokolls

Der Prover $\mathcal{P}(\mathsf{pk}, D_m, D_n, D_\sigma, A_1, A_2, B_1, B_2, l, r_m, r_n, r_\sigma, r_a, a_1, \delta_1, \delta_2, \delta_r, b_1, b_2, \beta_1, \beta_2, e, s, m_1, \ldots, m_L, n_1, \phi_1\left(l\right))$ erstellt für den Verifier $\mathcal{V}(\mathsf{pk}, D_m, D_n, D_\sigma, A_1, A_2, B_1, B_2, l)$ im Signaturbeweisprotokoll Prove die folgende Signature-of-Knowledge, wobei $\delta_1 = a_1 n_1, \delta_2 = \phi_1(l) n_1, \delta_r = r_a n_1, \beta_1 = b_1 e, \beta_2 = b_2 e$ ist.

$$SoK\Big[(r_m, r_n, r_\sigma, r_a, a_1, \delta_1, \delta_2, \delta_r, b_1, b_2, \beta_1, \beta_2, e, s, m_1, \ldots, m_L, n_1, \phi_1\left(l\right)) :$$

$$A_2 = g_0^{r_a} g_1^{a_1} g_2^{\phi_1(l)} \wedge 1 = A_2^{n_1} g_0^{-\delta_r} g_1^{-\delta_1} g_2^{-\delta_2} \wedge B_1 = g_0^{b_1} g_1^{b_2} \wedge 1 = B_1^{e} g_0^{-\beta_1} g_1^{-\beta_2}$$
$$\wedge D_m = g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L} \wedge D_n = g_0^{r_n} g_1^{n_1} \wedge D_\sigma = g_0^{r_\sigma} g_1^{e} g_2^{s} \wedge$$
$$\hat{e}(B_2, X) \left(\hat{e}(g,h)\, \hat{e}\left(A_1, h_2 h_1^{-l}\right)\right)^{-1} = \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(A_1^{n_1}, h_1\right)$$
$$\hat{e}\left(g_0^{s} g_1^{m_1+\beta_1} g_2^{m_2} \cdots g_L^{m_L} u_0^{l\delta_1+\delta_2} u_1^{la_1-\delta_1+\phi_1(l)} u_2^{-a_1} A_1^{-ln_1} B_2^{-e}, h\right)\Big].$$

Commitment: Der Prover $\mathcal{P}$ wählt zufällig die Zahlen $\rho_{r_m}, \rho_{r_n}, \rho_{r_\sigma}, \rho_{r_a}, \rho_{a_1}, \rho_{\delta_1}, \rho_{\delta_2}, \rho_{\delta_r}, \rho_{b_1}, \rho_{b_2}, \rho_{\beta_1}, \rho_{\beta_2}, \rho_e, \rho_s, \rho_{m_1}, \ldots, \rho_{m_L}, \rho_n, \rho_\phi \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Commitments

$$\begin{aligned}
T_{A_2} &= g_0^{\rho_{r_a}} g_1^{\rho_{a_1}} g_2^{\rho_\phi},\\
T_{1_A} &= A_2^{\rho_n} g_0^{-\rho_{\delta_r}} g_1^{-\rho_{\delta_1}} g_2^{-\rho_{\delta_2}},\\
T_{B_1} &= g_0^{\rho_{b_1}} g_1^{\rho_{b_2}},\\
T_{1_B} &= B_1^{\rho_e} g_0^{-\rho_{\beta_1}} g_1^{-\rho_{\beta_2}},\\
T_{D_m} &= g_0^{\rho_{r_m}} g_1^{\rho m_1} \cdots g_L^{\rho_{m_L}},\\
T_{D_n} &= g_0^{\rho_{r_n}} g_1^{\rho_n},\\
T_{D_\sigma} &= g_0^{\rho_{r_\sigma}} g_1^{\rho_e} g_2^{\rho_s},\\
T &= \hat{e}\left(g_1^{\rho_{b_1}}, X\right) \hat{e}\left(A_1^{\rho_n}, h_1\right)\\
&\quad \hat{e}\left(g_0^{\rho_s} g_1^{\rho_{m_1}+\rho_{\beta_1}} g_2^{\rho_{m_2}} \cdots g_L^{\rho_{m_L}} u_0^{l\rho_{\delta_1}+\rho_{\delta_2}} u_1^{l\rho_{a_1}-\rho_{\delta_1}+\rho_\phi} u_2^{-\rho_{a_1}} A_1^{-l\rho_n} B_2^{-\rho_e}, h\right).
\end{aligned}$$

Challenge: Der Prover $\mathcal{P}$ berechnet die Challenge

$$c = \mathsf{H}\left(D_m||D_n||D_\sigma||A_1||A_2||B_1||B_2||T_{A_2}||T_{1_A}||T_{B_1}||T_{1_B}||T_{D_m}||T_{D_n}||T_{D_\sigma}||T\right).$$

Response: Der Prover $\mathcal{P}$ berechnet in $\mathbb{Z}_p$ die Responses

$$\begin{aligned}
z_{r_m} &= \rho_{r_m} + c r_m, & z_{r_n} &= \rho_{r_n} + c r_n, & z_{r_\sigma} &= \rho_{r_\sigma} + c r_\sigma,\\
z_{r_a} &= \rho_{r_a} + c r_a, & z_{a_1} &= \rho_{a_1} + c a_1, & z_{\delta_1} &= \rho_{\delta_1} + c\delta_1,\\
z_{\delta_2} &= \rho_{\delta_2} + c\delta_2, & z_{\delta_r} &= \rho_{\delta_r} + c\delta_r, & z_{b_1} &= \rho_{b_1} + c b_1,\\
z_{b_2} &= \rho_{b_2} + c b_2, & z_{\beta_1} &= \rho_{\beta_1} + c\beta_1, & z_{\beta_2} &= \rho_{\beta_2} + c\beta_2,\\
z_e &= \rho_e + ce, & z_s &= \rho_s + cs, & z_{m_1} &= \rho_{m_1} + c m_1, \ \ldots,\\
z_{m_L} &= \rho_{m_L} + c m_L, & z_n &= \rho_n + c n_1, & z_\phi &= \rho_\phi + c\phi_1(l)
\end{aligned}$$

und sendet diese mit der Challenge an $\mathcal{V}$.

Verify: Der Verifier $\mathcal{V}$ berechnet die Commitments

$$\begin{aligned}
\widetilde{T_{A_2}} &= g_0^{z_{r_a}} g_1^{z_{a_1}} g_2^{z_\phi} A_2^{-c},\\
\widetilde{T_{1_A}} &= A_1^{z_n} g_0^{-z_{\delta_r}} g_1^{-z_{\delta_1}} g_2^{-z_{\delta_2}},
\end{aligned}$$

$$
\begin{aligned}
\widetilde{T_{B_1}} &= g_0^{z_{b_1}} g_1^{z_{b_2}} B_1^{-c},\\
\widetilde{T_{1_B}} &= B_1^{z_e} g_0^{-z_{\beta_1}} g_1^{-z_{\beta_2}},\\
\widetilde{T_{D_m}} &= g_0^{z_{r_m}} g_1^{z_{m_1}} \cdots g_L^{z_{m_L}} D_m^{-c},\\
\widetilde{T_{D_n}} &= g_0^{z_{r_n}} g_1^{z_n} D_n^{-c},\\
\widetilde{T_{D_\sigma}} &= g_0^{z_{r_\sigma}} g_1^{z_e} g_2^{z_s} D_\sigma^{-c},\\
\widetilde{T} &= \hat{e}\left(g_1^{z_{b_1}} B_2^{-c}, X\right) \hat{e}\left(A_1^{z_n - lc}, h_1\right) \hat{e}\left(A_1^c, h_2\right)\\
&\quad \hat{e}\left(g^c g_0^{z_s} g_1^{z_{m_1}+z_{\beta_1}} g_2^{z_{m_2}} \cdots g_L^{z_{m_L}} u_0^{lz_{\delta_1}+z_{\delta_2}} u_1^{lz_{a_1}-z_{\delta_1}+z_\phi} u_2^{-z_{a_1}} A_1^{-lz_n} B_2^{-z_e}, h\right)
\end{aligned}
$$

und überprüft

$$
c \stackrel{?}{=} \mathsf{H}\left(D_m||D_n||D_\sigma||A_1||A_2||B_1||B_2||\widetilde{T_{A_2}}||\widetilde{T_{1_A}}||\widetilde{T_{B_1}}||\widetilde{T_{1_B}}||\widetilde{T_{D_m}}||\widetilde{T_{D_n}}||\widetilde{D_\sigma}||\widetilde{T}\right).
$$

Satz A.2.1. *Die oben beschriebene Signature-of-Knowledge erfüllt die Durchführbarkeit, die spezielle Korrektheit und die Special-Honest-Verifier-Zero-Knowledge-Eigenschaft.*

Beweis. Wir zeigen zunächst die Durchführbarkeit, dann sie spezielle Korrektheit und anschließend die Special-Honest-Verifier-Zero-Knowledge-Eigenschaft.

Durchführbarkeit: Verhält sich der Prover $\mathcal{P}$ gemäß dem Protokoll, dann gilt:

$$
\begin{aligned}
\widetilde{T_{A_2}} &= g_0^{z_{r_a}} g_1^{z_{a_1}} g_2^{z_\phi} A_2^{-c} = g_0^{\rho_{r_a}+cr_a} g_1^{\rho_{a_1}+ca_1} g_2^{\rho_\phi + c\phi_1(l)} A_2^{-c}\\
&= g_0^{\rho_{r_a}} g_1^{\rho_{a_1}} g_2^{\rho_\phi} g_0^{cr_a} g_1^{ca_1} g_2^{c\phi_1(l)} \left(g_0^{r_a} g_1^{a_1} g_2^{\phi_1(l)}\right)^{-c} = g_0^{\rho_{r_a}} g_1^{\rho_{a_1}} g_2^{\rho_\phi} = T_{A_2},\\
\widetilde{T_{1_A}} &= A_2^{z_n} g_0^{-z_{\delta_r}} g_1^{-z_{\delta_1}} g_2^{-z_{\delta_2}} = A_2^{\rho_n+cn_1} g_0^{-\rho_{\delta_r}-c\delta_r} g_1^{-\rho_{\delta_1}-c\delta_1} g_2^{-\rho_{\delta_2}-c\delta_2}\\
&= A_2^{\rho_n} g_0^{-\rho_{\delta_r}} g_1^{-\rho_{\delta_1}} g_2^{-\rho_{\delta_2}} \left(g_0^{r_a} g_1^{a_1} g_2^{\phi_1(l)}\right)^{cn_1} g_0^{-cr_a n_1} g_1^{-ca_1 n_1} g_2^{-c\phi_1(l) n_1}\\
&= A_2^{\rho_n} g_0^{-\rho_{\delta_r}} g_1^{-\rho_{\delta_1}} g_2^{-\rho_{\delta_2}} = T_{1_A},\\
\widetilde{T_{B_1}} &= g_0^{z_{b_1}} g_1^{z_{b_2}} B_1^{-c} = g_0^{\rho_{b_1}+cb_1} g_1^{\rho_{b_2}+cb_2} B_1^{-c} = g_0^{\rho_{b_1}} g_1^{\rho_{b_2}} g_0^{cb_1} g_1^{cb_2} \left(g_0^{b_1} g_1^{b_2}\right)^{-c} = g_0^{\rho_{b_1}} g_1^{\rho_{b_2}} = T_{B_1},\\
\widetilde{T_{1_B}} &= B_1^{z_e} g_0^{-z_{\beta_1}} g_1^{-z_{\beta_2}} = B_1^{\rho_e+ce} g_0^{-\rho_{\beta_1}-c\beta_1} g_1^{-\rho_{\beta_2}-c\beta_2}\\
&= B_1^{\rho_e} g_0^{-\rho_{\beta_1}} g_1^{-\rho_{\beta_2}} \left(g_0^{b_1} g_1^{b_2}\right)^{ce} g_0^{-cb_1 e} g_1^{-cb_2 e} = B_1^{\rho_e} g_0^{-\rho_{\beta_1}} g_1^{-\rho_{\beta_2}} = T_{1_B},\\
\widetilde{T_{D_m}} &= g_0^{z_{r_m}} g_1^{z_{m_1}} \cdots g_L^{z_{m_L}} D_m^{-c} = g_0^{\rho_{r_m}+cr_m} g_1^{\rho_{m_1}+cm_1} \cdots g_L^{\rho_{m_L}+cm_L} D_m^{-c}\\
&= g_0^{\rho_{r_m}} g_1^{\rho_{m_1}} \cdots g_L^{\rho_{m_L}} g_0^{cr_m} g_1^{cm_1} \cdots g_L^{cm_L} \left(g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L}\right)^{-c} = g_0^{\rho_{r_m}} g_1^{\rho_{m_1}} \cdots g_L^{\rho_{m_L}} = T_{D_m},\\
\widetilde{T_{D_n}} &= g_0^{z_{r_n}} g_1^{z_n} D_n^{-c} = g_0^{\rho_{r_n}+cr_n} g_1^{\rho_n+cn_1} D_n^{-c} = g_0^{\rho_{r_n}} g_1^{\rho_n} g_0^{cr_n} g_1^{cn_1} \left(g_0^{r_n} g_1^{n_1}\right)^{-c} = g_0^{\rho_{r_n}} g_1^{\rho_n} = T_{D_n},\\
\widetilde{T_{D_\sigma}} &= g_0^{z_{r_\sigma}} g_1^{z_e} g_2^{z_s} D_\sigma^{-c} = g_0^{\rho_{r_\sigma}+cr_\sigma} g_1^{\rho_e+ce} g_2^{\rho_s+cs} D_\sigma^{-c}\\
&= g_0^{\rho_{r_\sigma}} g_1^{\rho_e} g_2^{\rho_s} g_0^{cr_\sigma} g_1^{ce} g_2^{cs} \left(g_0^{r_\sigma} g_1^{e} g_2^{s}\right)^{-c} = g_0^{\rho_{r_\sigma}} g_1^{\rho_e} g_2^{\rho_s} = T_{D_\sigma},\\
\widetilde{T} &= \hat{e}\left(g_1^{z_{b_1}} B_2^{-c}, X\right) \hat{e}\left(A_1^{z_n - lc}, h_1\right) \hat{e}\left(A_1^c, h_2\right)
\end{aligned}
$$

$$
\begin{aligned}
&\hat{e}\left(g^c g_0^{z_s} g_1^{z_{m_1}+z_{\beta_1}} g_2^{z_{m_2}} \cdots g_L^{z_{m_L}} u_0^{lz_{\delta_1}+z_{\delta_2}} u_1^{lz_{a_1}-z_{\delta_1}+z_\phi} u_2^{-z_{a_1}} A_1^{-lz_n} B_2^{-z_e}, h\right)\\
&= \hat{e}\left(g_1^{\rho_{b_1}+cb_1} B_2^{-c}, X\right) \hat{e}\left(A_1^{\rho_n+cn_1-lc}, h_1\right) \hat{e}\left(A_1^c, h_2\right)\\
&\quad \hat{e}\Big(g^c g_0^{\rho_s+cs} g_1^{\rho_{m_1}+cm_1+\rho_{\beta_1}+c\beta_1} g_2^{\rho_{m_2}+cm_2} \cdots g_L^{\rho_{m_L}+cm_L} u_0^{l\rho_{\delta_1}+lc\delta_1+\rho_{\delta_2}+c\delta_2}\\
&\qquad u_1^{l\rho_{a_1}+lca_1-\rho_{\delta_1}-c\delta_1+\rho_\phi+c\phi_1(l)} u_2^{-\rho_{a_1}-ca_1} A_1^{-l\rho_n-lcn_1} B_2^{-\rho_e-ce}, h\Big)\\
&= \hat{e}\left(g_1^{\rho_{b_1}}, X\right) \hat{e}\left(A_1^{\rho_n}, h_1\right) \hat{e}\Big(g_0^{\rho_s} g_1^{\rho_{m_1}+\rho_{\beta_1}} g_2^{\rho_{m_2}} \cdots g_L^{\rho_{m_L}} u_0^{l\rho_{\delta_1}+\rho_{\delta_2}} u_1^{l\rho_{a_1}-\rho_{\delta_1}+\rho_\phi} u_2^{-\rho_{a_1}} A_1^{-l\rho_n}\\
&\quad B_2^{-\rho_e}, h\Big) \hat{e}\left(g_1^{cb_1} B_2^{-c}, X\right) \hat{e}\left(A_1^{cn_1-lc}, h_1\right) \hat{e}\left(A_1^c, h_2\right)\\
&\quad \hat{e}\left(g^c g_0^{cs} g_1^{cm_1+c\beta_1} g_2^{cm_2} \cdots g_L^{cm_L} u_0^{lc\delta_1+c\delta_2} u_1^{lca_1-c\delta_1+c\phi_1(l)} u_2^{-ca_1} A_1^{-lcn_1} B_2^{-ce}, h\right)\\
&= T\Big(\hat{e}\left(g_1^{b_1}\left(\Sigma g_1^{b_1}\right)^{-1}, X\right) \hat{e}\left(\left(W_{1,l} u_0^{a_1}\right)^{n_1-l}, h_1\right) \hat{e}\left(W_{1,l} u_0^{a_1}, h_2\right) \hat{e}\Big(g g_0^s g_1^{m_1+b_1 e}\\
&\quad g_2^{m_2} \cdots g_L^{m_L} u_0^{la_1 n_1+\phi_1(l)n_1} u_1^{la_1-a_1 n_1+\phi_1(l)} u_2^{-a_1} \left(W_{1,l} u_0^{a_1}\right)^{-ln_1} \left(\Sigma g_1^{b_1}\right)^{-e}, h\Big)\Big)^c\\
&= T\Big(\hat{e}\left(\Sigma, X h^e\right)^{-1} \hat{e}\left(W_{1,l}^{n_1-l}, h_1\right) \hat{e}\left(W_{1,l}, h_2\right)\\
&\quad \hat{e}\left(g g_0^s g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi_1(l)n_1} u_1^{\phi_1(l)} W_{1,l}^{-ln_1}, h\right)\Big)^c\\
&= T\left(\hat{e}\left(\Sigma, X h^e\right)^{-1} \hat{e}\left(g g_0^s g_1^{m_1} \cdots g_L^{m_L}, h\right) \hat{e}\left(W_{1,l}^{\alpha-l} u_0^{\phi_1(l)}, h^{\alpha+n_1}\right)\right)^c\\
&= T\left(\hat{e}\left(\Sigma, X h^e\right)^{-1} \hat{e}\left(g g_0^s g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}, h\right)\right)^c = T,
\end{aligned}
$$

da für $W_{1,l}$ die Gleichung $\hat{e}\left(u_0^{\phi(\alpha)}, h\right) = \hat{e}\left(W_{1,l}, h^{(\alpha-l)(\alpha+n_1)}\right) \hat{e}\left(u_0^{\phi_1(l)}, h^{\alpha+n_1}\right)$ und für Σ die Gleichung $\hat{e}\left(\Sigma, X h^e\right) = \hat{e}\Big(g g_0^s g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}, h\Big)$ gilt.

Spezielle Korrektheit: Der Knowledge-Extractor $\mathcal{E}$ erhält die Parameter $(\mathsf{pk}, D_m, D_n, D_\sigma, A_1, A_2, B_1, B_2, l)$, die Commitments $(T_{A_2}, T_{1_A}, T_{B_1}, T_{1_B}, T_{D_m}, T_{D_n}, T_{D_\sigma}, T)$ sowie die beiden Challenge-Response Tupel $(c, z_{r_m}, z_{r_n}, z_{r_\sigma}, z_{r_a}, z_{a_1}, z_{\delta_1}, z_{\delta_2}, z_{\delta_r}, z_{b_1}, z_{b_2}, z_{\beta_1}, z_{\beta_2}, z_e, z_s, z_{m_1}, \ldots, z_{m_L}, z_n, z_\phi)$ und $\left(c', z'_{r_m}, z'_{r_n}, z'_{r_\sigma}, z'_{r_a}, z'_{a_1}, z'_{\delta_1}, z'_{\delta_2}, z'_{\delta_r}, z'_{b_1}, z'_{b_2}, z'_{\beta_1}, z'_{\beta_2}, z'_e, z'_s, z'_{m_1}, \ldots, z'_{m_L}, z'_n, z'_\phi\right)$ mit

$$
\begin{aligned}
T_{A_2} &= g_0^{z_{r_a}} g_1^{z_{a_1}} g_2^{z_\phi} A_2^{-c} = g_0^{z'_{r_a}} g_1^{z'_{a_1}} g_2^{z'_\phi} A_2^{-c'},\\
T_{1_A} &= A_2^{z_n} g_0^{-z_{\delta_r}} g_1^{-z_{\delta_1}} g_2^{-z_{\delta_2}} = A_2^{z'_n} g_0^{-z'_{\delta_r}} g_1^{-z'_{\delta_1}} g_2^{-z'_{\delta_2}},\\
T_{B_1} &= g_0^{z_{b_1}} g_1^{z_{b_2}} B_1^{-c} = g_0^{z'_{b_1}} g_1^{z'_{b_2}} B_1^{-c'},\\
T_{1_B} &= B_1^{z_e} g_0^{-z_{\beta_1}} g_1^{-z_{\beta_2}} = B_1^{z'_e} g_0^{-z'_{\beta_1}} g_1^{-z'_{\beta_2}},\\
T_{D_m} &= g_0^{z_{rm}} g_1^{z_{m_1}} \cdots g_L^{z_{m_L}} D_m^{-c} = g_0^{z'_{rm}} g_1^{z'_{m_1}} \cdots g_L^{z'_{m_L}} D_m^{-c'},\\
T_{D_n} &= g_0^{z_{r_n}} g_1^{z_n} D_n^{-c} = g_0^{z'_{r_n}} g_1^{z'_n} D_n^{-c'},
\end{aligned}
$$

$$
\begin{aligned}
T_{D_\sigma} &= g_0^{z_{r_\sigma}} g_1^{z_e} g_2^{z_s} D_\sigma^{-c} = g_0^{z'_{r_\sigma}} g_1^{z'_e} g_2^{z'_s} D_\sigma^{-c'},\\
T &= \hat{e}\left(g_1^{z_{b_1}} B_2^{-c}, X\right) \hat{e}\left(A_1^{z_n - lc}, h_1\right) \hat{e}\left(A_1^{c}, h_2\right)\\
&\quad \hat{e}\left(g^c g_0^{z_s} g_1^{z_{m_1}+z_{\beta_1}} g_2^{z_{m_2}} \cdots g_L^{z_{m_L}} u_0^{lz_{\delta_1}+z_{\delta_2}} u_1^{lz_{a_1}-z_{\delta_1}+z_\phi} u_2^{-z_{a_1}} A_1^{-lz_n} B_2^{-z_e}, h\right)\\
&= \hat{e}\left(g_1^{z'_{b_1}} B_2^{-c'}, X\right) \hat{e}\left(A_1^{z'_n - lc'}, h_1\right) \hat{e}\left(A_1^{c'}, h_2\right)\\
&\quad \hat{e}\left(g^{c'} g_0^{z'_s} g_1^{z'_{m_1}+z'_{\beta_1}} g_2^{z'_{m_2}} \cdots g_L^{z'_{m_L}} u_0^{lz'_{\delta_1}+z'_{\delta_2}} u_1^{lz'_{a_1}-z'_{\delta_1}+z'_\phi} u_2^{-z'_{a_1}} A_1^{-lz'_n} B_2^{-z'_e}, h\right).
\end{aligned}
$$

Daraus folgt:

$$
\begin{aligned}
A_2 &= g_0^{\frac{z_{r_a}-z'_{r_a}}{c-c'}} g_1^{\frac{z_{a_1}-z'_{a_1}}{c-c'}} g_2^{\frac{z_\phi - z'_\phi}{c-c'}} = g_0^{\frac{z_{\delta_r}-z'_{\delta_r}}{z_n-z'_n}} g_1^{\frac{z_{\delta_1}-z'_{\delta_1}}{z_n-z'_n}} g_2^{\frac{z_{\delta_2}-z'_{\delta_2}}{z_n-z'_n}},\\
B_1 &= g_0^{\frac{z_{b_1}-z'_{b_1}}{c-c'}} g_1^{\frac{z_{b_2}-z'_{b_2}}{c-c'}} = g_0^{\frac{z_{\beta_1}-z'_{\beta_1}}{z_e-z'_e}} g_1^{\frac{z_{\beta_2}-z'_{\beta_2}}{z_e-z'_e}},\\
D_m &= g_0^{\frac{z_{r_m}-z'_{r_m}}{c-c'}} g_1^{\frac{z_{m_1}-z'_{m_1}}{c-c'}} \cdots g_L^{\frac{z_{m_L}-z'_{m_L}}{c-c'}},\\
D_n &= g_0^{\frac{z_{r_n}-z'_{r_n}}{c-c'}} g_1^{\frac{z_n-z'_n}{c-c'}},\\
D_\sigma &= g_0^{\frac{z_{r_\sigma}-z'_{r_\sigma}}{c-c'}} g_1^{\frac{z_e-z'_e}{c-c'}} g_2^{\frac{z_s-z'_s}{c-c'}},\\
\frac{\hat{e}(B_2,X)}{\hat{e}(g,h)\,\hat{e}\left(A_1, h_2 h_1^{-l}\right)} &= \hat{e}\left(g_1^{\frac{z_{b_1}-z'_{b_1}}{c-c'}}, X\right) \hat{e}\left(A_1^{\frac{z_n-z'_n}{c-c'}}, h_1\right) \hat{e}\Bigg(g_0^{\frac{z_s-z'_s}{c-c'}} g_1^{\frac{z_{m_1}-z'_{m_1}}{c-c'}+\frac{z_{\beta_1}-z'_{\beta_1}}{c-c'}}\\
&\quad g_2^{\frac{z_{m_2}-z'_{m_2}}{c-c'}} \cdots g_L^{\frac{z_{m_L}-z'_{m_L}}{c-c'}} u_0^{l\frac{z_{\delta_1}-z'_{\delta_1}}{c-c'}+\frac{z_{\delta_2}-z'_{\delta_2}}{c-c'}} u_1^{l\frac{z_{a_1}-z'_{a_1}}{c-c'}-\frac{z_{\delta_1}-z'_{\delta_1}}{c-c'}+\frac{z_\phi-z'_\phi}{c-c'}}\\
&\quad u_2^{-\frac{z_{a_1}-z'_{a_1}}{c-c'}} A_1^{-l\frac{z_n-z'_n}{c-c'}} B_2^{-\frac{z_e-z'_e}{c-c'}}, h\Bigg).
\end{aligned}
$$

Daher kann der Knowledge-Extractor $\mathcal{E}$ die folgenden Zahlen in $\mathbb{Z}_p$ berechnen:

$$
\begin{aligned}
r_m &= \frac{z_{r_m}-z'_{r_m}}{c-c'}, & r_n &= \frac{z_{r_n}-z'_{r_n}}{c-c'}, & r_\sigma &= \frac{z_{r_\sigma}-z'_{r_\sigma}}{c-c'},\\
r_a &= \frac{z_{r_a}-z'_{r_a}}{c-c'}, & a_1 &= \frac{z_{a_1}-z'_{a_1}}{c-c'}, & \delta_1 &= \frac{z_{\delta_1}-z'_{\delta_1}}{c-c'},\\
\delta_2 &= \frac{z_{\delta_2}-z'_{\delta_2}}{c-c'}, & \delta_r &= \frac{z_{\delta_r}-z'_{\delta_r}}{c-c'}, & b_1 &= \frac{z_{b_1}-z'_{b_1}}{c-c'},\\
b_2 &= \frac{z_{b_2}-z'_{b_2}}{c-c'}, & \beta_1 &= \frac{z_{\beta_1}-z'_{\beta_1}}{c-c'}, & \beta_2 &= \frac{z_{\beta_2}-z'_{\beta_2}}{c-c'},\\
e &= \frac{z_e-z'_e}{c-c'}, & s &= \frac{z_s-z'_s}{c-c'}, & m_1 &= \frac{z_{m_1}-z'_{m_1}}{c-c'}, \ldots,
\end{aligned}
$$

$$m_L = \frac{z_{m_L} - z'_{m_L}}{c - c'}, \qquad n_1 = \frac{z_n - z'_n}{c - c'}, \qquad \phi_1(l) = \frac{z_\phi - z'_\phi}{c - c'}.$$

Somit gilt:

$$\begin{aligned}
A_2 &= g_0^{r_a} g_1^{a_1} g_2^{\phi_1(l)} = g_0^{\frac{\delta_r}{n_1}} g_1^{\frac{\delta_1}{n_1}} g_2^{\frac{\delta_2}{n_1}}, \\
B_1 &= g_0^{b_1} g_1^{b_2} = g_0^{\frac{\beta_1}{e}} g_1^{\frac{\beta_2}{e}}, \\
D_m &= g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L}, \\
D_n &= g_0^{r_n} g_1^{n_1}, \\
D_\sigma &= g_0^{r_\sigma} g_1^{e} g_2^{s}, \\
\frac{\hat{e}(B_2, X)}{\hat{e}(g, h)\,\hat{e}\left(A_1, h_2 h_1^{-l}\right)} &= \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(A_1^{n_1}, h_1\right) \\
&\quad \hat{e}\left(g_0^{s} g_1^{m_1+\beta_1} g_2^{m_2} \cdots g_L^{m_L} u_0^{l\delta_1+\delta_2} u_1^{la_1-\delta_1+\phi_1(l)} u_2^{-a_1} A_1^{-ln_1} B_2^{-e}, h\right).
\end{aligned}$$

Special-Honest-Verifier-Zero-Knowledge: Der Simulator $\mathcal{SIM}$ erhält die Parameter $(\mathsf{pk}, D_m, D_n, D_\sigma, A_1, A_2, B_1, B_2, l)$ sowie die Challenge c und simuliert die Signature-of-Knowledge folgendermaßen:

Der Simulator $\mathcal{SIM}$ wählt zufällig die Responses $z_{r_m}, z_{r_n}, z_{r_\sigma}, z_{r_a}, z_{a_1}, z_{\delta_1}, z_{\delta_2}, z_{\delta_r}, z_{b_1}, z_{b_2}, z_{\beta_1}, z_{\beta_2}, z_e, z_s, z_{m_1}, \ldots, z_{m_L}, z_n, z_\phi \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Commitments

$$\begin{aligned}
T_{A_2} &= g_0^{z_{r_a}} g_1^{z_{a_1}} g_2^{z_\phi} A_2^{-c}, \\
T_{1_A} &= A_2^{z_n} g_0^{-z_{\delta_r}} g_1^{-z_{\delta_1}} g_2^{-z_{\delta_2}}, \\
T_{B_1} &= g_0^{z_{b_1}} g_1^{z_{b_2}} B_1^{-c}, \\
T_{1_B} &= B_1^{z_e} g_0^{-z_{\beta_1}} g_1^{-z_{\beta_2}}, \\
T_{D_m} &= g_0^{z_{r_m}} g_1^{z_{m_1}} \cdots g_L^{z_{m_L}} D_m^{-c}, \\
T_{D_n} &= g_0^{z_{r_n}} g_1^{z_n} D_n^{-c}, \\
T_{D_\sigma} &= g_0^{z_{r_\sigma}} g_1^{z_e} g_2^{z_s} D_\sigma^{-c}, \\
T &= \hat{e}\left(g_1^{z_{b_1}} B_2^{-c}, X\right) \hat{e}\left(A_1^{z_n - lc}, h_1\right) \hat{e}\left(A_1^{c}, h_2\right) \\
&\quad \hat{e}\left(g^c g_0^{z_s} g_1^{z_{m_1}+z_{\beta_1}} g_2^{z_{m_2}} \cdots g_L^{z_{m_L}} u_0^{lz_{\delta_1}+z_{\delta_2}} u_1^{lz_{a_1}-z_{\delta_1}+z_\phi} u_2^{-z_{a_1}} A_1^{-lz_n} B_2^{-z_e}, h\right).
\end{aligned}$$

Somit ist die Verifikation von $\mathcal{V}$ trivialer Weise erfüllt. Weiter sind alle Responses unabhängig und identisch verteilt und somit besitzen auch alle Commitments die gleiche Verteilung wie im realen Protokoll. Insgesamt hat $\mathcal{S}$ die Signature-of-Knowledge perfekt simuliert. □

A.3 Die SoK des Prove-k-Protokolls

Der Prover $\mathcal{P}(\mathsf{pk}, D_m, D_\sigma, A_1, B_1, B_2, n_1, \ldots, n_k, l, r_m, r_\sigma, a_1, b_1, b_2, \beta_1, \beta_2, e, s, m_1, \ldots, m_L, \phi_I(l))$ erstellt für den Verifier $\mathcal{V}(\mathsf{pk}, D_m, D_\sigma, A_1, B_1, B_2, n_1, \ldots, n_k, l)$ im Signaturbeweisprotokoll Prove-k die folgende Signature-of-Knowledge, wobei $\beta_1 = b_1 e, \beta_2 = b_2 e$, $u_I = u_0^{\prod_{j=1}^{k}(\alpha+n_j)}$, $u_{l,I} = u_0^{(\alpha-l)\prod_{j=1}^{k}(\alpha+n_j)}$ und $h_{l,I} = h^{(\alpha-l)\prod_{j=1}^{k}(\alpha+n_j)}$ ist.

$$SoK\Bigg[(r_m, r_\sigma, a_1, b_1, b_2, \beta_1, \beta_2, e, s, m_1, \ldots, m_L, \phi_I(l)) : B_1 = g_0^{b_1} g_1^{b_2} \wedge$$
$$1 = B_1^e g_0^{-\beta_1} g_1^{-\beta_2} \ \wedge \ D_m = g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L} \ \wedge \ D_\sigma = g_0^{r_\sigma} g_1^e g_2^s \wedge$$
$$\frac{\hat{e}(B_2, X)}{\hat{e}(g, h)\,\hat{e}(A_1, h_{l,I})} = \hat{e}\left(g_1^{b_1}, X\right)\hat{e}\left(g_0^s g_1^{m_1+\beta_1} g_2^{m_2} \ldots g_L^{m_L} u_{l,I}^{-a_1} u_I^{\phi_I(l)} B_2^{-e}, h\right)\Bigg].$$

Commitment: Der Prover $\mathcal{P}$ wählt zufällig die Zahlen $\rho_{r_m}, \rho_{r_\sigma}, \rho_{a_1}, \rho_{b_1}, \rho_{b_2}, \rho_{\beta_1}, \rho_{\beta_2}, \rho_e, \rho_s, \rho_{m_1}, \ldots, \rho_{m_L}, \rho_\phi \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Commitments

$$\begin{aligned}
T_{B_1} &= g_0^{\rho_{b_1}} g_1^{\rho_{b_2}},\\
T_{1_B} &= B_1^{\rho_e} g_0^{-\rho_{\beta_1}} g_1^{-\rho_{\beta_2}},\\
T_{D_m} &= g_0^{\rho_{r_m}} g_1^{\rho m_1} \cdots g_L^{\rho_{m_L}},\\
T_{D_\sigma} &= g_0^{\rho_{r_\sigma}} g_1^{\rho_e} g_2^{\rho_s},\\
T &= \hat{e}\left(g_1^{\rho_{b_1}}, X\right)\hat{e}\left(g_0^{\rho_s} g_1^{\rho_{m_1}+\rho_{\beta_1}} g_2^{\rho_{m_2}} \ldots g_L^{\rho_{m_L}} u_{l,I}^{-\rho_{a_1}} u_I^{\rho_\phi} B_2^{-\rho_e}, h\right).
\end{aligned}$$

Challenge: Der Prover $\mathcal{P}$ berechnet die Challenge

$$c = \mathsf{H}(D_m||D_\sigma||A_1||B_1||B_2||T_{B_1}||T_{1_B}||T_{D_m}||T_{D_\sigma}||T).$$

Response: Der Prover $\mathcal{P}$ berechnet in $\mathbb{Z}_p$ die Responses

$$\begin{aligned}
z_{r_m} &= \rho_{r_m} + c r_m, & z_{r_\sigma} &= \rho_{r_\sigma} + c r_\sigma, & z_{a_1} &= \rho_{a_1} + c a_1,\\
z_{b_1} &= \rho_{b_1} + c b_1, & z_{b_2} &= \rho_{b_2} + c b_2, & z_{\beta_1} &= \rho_{\beta_1} + c\beta_1,\\
z_{\beta_2} &= \rho_{\beta_2} + c\beta_2, & z_e &= \rho_e + ce, & z_s &= \rho_s + cs,\\
z_{m_1} &= \rho_{m_1} + c m_1, \ldots, & z_{m_L} &= \rho_{m_L} + c m_L, & z_\phi &= \rho_\phi + c\phi_I(l)
\end{aligned}$$

und sendet diese mit der Challenge an $\mathcal{V}$.

Verify: Der Verifier $\mathcal{V}$ berechnet die Commitments

$$\begin{aligned}
\widetilde{T_{B_1}} &= g_0^{z_{b_1}} g_1^{z_{b_2}} B_1^{-c},\\
\widetilde{T_{1_B}} &= B_1^{z_e} g_0^{-z_{\beta_1}} g_1^{-z_{\beta_2}},\\
\widetilde{T_{D_m}} &= g_0^{z_{r_m}} g_1^{z_{m_1}} \cdots g_L^{z_{m_L}} D_m^{-c},
\end{aligned}$$

$$\widetilde{T_{D_\sigma}} = g_0^{z_{r\sigma}} g_1^{z_e} g_2^{z_s} D_\sigma^{-c},$$
$$\widetilde{T} = \hat{e}\left(g_1^{z_{b_1}} B_2^{-c}, X\right) \hat{e}\left(g^c g_0^{z_s} g_1^{z_{m_1}+z_{\beta_1}} g_2^{z_{m_2}} \dots g_L^{z_{m_L}} u_{l,I}^{-z_{a_1}} u_I^{z_\phi} B_2^{-z_e}, h\right) \hat{e}\left(A_1^c, h_{l,I}\right)$$

und überprüft

$$c \stackrel{?}{=} \mathsf{H}\left(D_m||D_\sigma||A_1||B_1||B_2||\widetilde{T_{B_1}}||\widetilde{T_{1_B}}||\widetilde{T_{D_m}}||\widetilde{T_{D_\sigma}}||\widetilde{T}\right).$$

Satz A.3.1. *Die oben beschriebene Signature-of-Knowledge erfüllt die Durchführbarkeit, die spezielle Korrektheit und die Special-Honest-Verifier-Zero-Knowledge-Eigenschaft.*

Beweis. Wir zeigen zunächst die Durchführbarkeit, dann sie spezielle Korrektheit und anschließend die Special-Honest-Verifier-Zero-Knowledge-Eigenschaft.

Durchführbarkeit: Verhält sich der Prover $\mathcal{P}$ gemäß dem Protokoll, dann gilt:

$$\begin{aligned}
\widetilde{T_{B_1}} &= g_0^{z_{b_1}} g_1^{z_{b_2}} B_1^{-c} = g_0^{\rho_{b_1}+cb_1} g_1^{\rho_{b_2}+cb_2} B_1^{-c} = g_0^{\rho_{b_1}} g_1^{\rho_{b_2}} g_0^{cb_1} g_1^{cb_2} \left(g_0^{b_1} g_1^{b_2}\right)^{-c} = g_0^{\rho_{b_1}} g_1^{\rho_{b_2}} = T_{B_1},\\
\widetilde{T_{1_B}} &= B_1^{z_e} g_0^{-z_{\beta_1}} g_1^{-z_{\beta_2}} = B_1^{\rho_e+ce} g_0^{-\rho_{\beta_1}-c\beta_1} g_1^{-\rho_{\beta_2}-c\beta_2}\\
&= B_1^{\rho_e} g_0^{-\rho_{\beta_1}} g_1^{-\rho_{\beta_2}} \left(g_0^{b_1} g_1^{b_2}\right)^{ce} g_0^{-cb_1 e} g_1^{-cb_2 e} = B_1^{\rho_e} g_0^{-\rho_{\beta_1}} g_1^{-\rho_{\beta_2}} = T_{1_B},\\
\widetilde{T_{D_m}} &= g_0^{z_{r_m}} g_1^{z_{m_1}} \cdots g_L^{z_{m_L}} D_m^{-c} = g_0^{\rho_{r_m}+cr_m} g_1^{\rho_{m_1}+cm_1} \cdots g_L^{\rho_{m_L}+cm_L} D_m^{-c}\\
&= g_0^{\rho_{r_m}} g_1^{\rho_{m_1}} \cdots g_L^{\rho_{m_L}} g_0^{cr_m} g_1^{cm_1} \cdots g_L^{cm_L} \left(g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L}\right)^{-c} = g_0^{\rho_{r_m}} g_1^{\rho_{m_1}} \cdots g_L^{\rho_{m_L}} = T_{D_m},\\
\widetilde{T_{D_\sigma}} &= g_0^{z_{r\sigma}} g_1^{z_e} g_2^{z_s} D_\sigma^{-c} = g_0^{\rho_{r\sigma}+cr_\sigma} g_1^{\rho_e+ce} g_2^{\rho_s+cs} D_\sigma^{-c}\\
&= g_0^{\rho_{r\sigma}} g_1^{\rho_e} g_2^{\rho_s} g_0^{cr_\sigma} g_1^{ce} g_2^{cs} \left(g_0^{r_\sigma} g_1^{e} g_2^{s}\right)^{-c} = g_0^{\rho_{r\sigma}} g_1^{\rho_e} g_2^{\rho_s} = T_{D_\sigma},\\
\widetilde{T} &= \hat{e}\left(g_1^{z_{b_1}} B_2^{-c}, X\right) \hat{e}\left(g^c g_0^{z_s} g_1^{z_{m_1}+z_{\beta_1}} g_2^{z_{m_2}} \dots g_L^{z_{m_L}} u_{l,I}^{-z_{a_1}} u_I^{z_\phi} B_2^{-z_e}, h\right) \hat{e}\left(A_1, h_{l,I}^c\right)\\
&= \hat{e}\left(g_1^{\rho_{b_1}+cb_1} B_2^{-c}, X\right) \hat{e}\Big(g^c g_0^{\rho_s+cs} g_1^{\rho_{m_1}+cm_1+\rho_{\beta_1}+c\beta_1} g_2^{\rho_{m_2}+cm_2} \dots g_L^{\rho_{m_L}+cm_L}\\
&\quad u_{l,I}^{-\rho_{a_1}-ca_1} u_I^{\rho_\phi+c\phi_I(l)} B_2^{-\rho_e-ce}, h\Big) \hat{e}\left(A_1^c, h_{l,I}\right)\\
&= \hat{e}\left(g_1^{\rho_{b_1}}, X\right) \hat{e}\left(g_0^{\rho_s} g_1^{\rho_{m_1}+\rho_{\beta_1}} g_2^{\rho_{m_2}} \dots g_L^{\rho_{m_L}} u_{l,I}^{-\rho_{a_1}} u_I^{\rho_\phi} B_2^{-\rho_e}, h\right) \hat{e}\left(g_1^{cb_1} B_2^{-c}, X\right)\\
&\quad \hat{e}(g^c g_0^{cs} g_1^{cm_1+c\beta_1} g_2^{cm_2} \dots g_L^{cm_L} u_{l,I}^{-ca_1} u_I^{c\phi_I(l)} B_2^{-ce}, h) \hat{e}\left(A_1^c, h_{l,I}\right)\\
&= T\Bigg(\hat{e}\left(g_1^{b_1}\left(\Sigma g_1^{b_1}\right)^{-1}, X\right) \hat{e}\left(g g_0^{s} g_1^{m_1+b_1 e} g_2^{m_2} \dots g_L^{m_L} u_{l,I}^{-a_1} u_I^{\phi_I(l)} \left(\Sigma g_1^{b_1}\right)^{-e}, h\right)\\
&\quad \hat{e}\left(W_{I,l} u_0^{a_1}, h_{l,I}\right)\Bigg)^c\\
&= T\left(\hat{e}\left(\Sigma, X h^e\right)^{-1} \hat{e}\left(g g_0^s g_1^{m_1} \dots g_L^{m_L} u_I^{\phi_I(l)}, h\right) \hat{e}\left(W_{I,l}, h_{l,I}\right)\right)^c\\
&= T\left(\hat{e}\left(\Sigma, X h^e\right)^{-1} \hat{e}\left(g g_0^s g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}, h\right)\right)^c = T,
\end{aligned}$$

da für $W_{I,l}$ die Gleichung $\hat{e}\left(u_0^{\phi(\alpha)}, h\right) = \hat{e}\left(W_{I,l}, h_{l,I}\right) \hat{e}\left(u_0^{\phi_I(l)}, h_I\right)$ und für Σ die Gleichung $\left(\Sigma, X h^e\right) = \hat{e}\left(g g_0^s g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}, h\right)$ gilt.

Spezielle Korrektheit: Der Knowledge-Extractor $\mathcal{E}$ erhält die Parameter $(\mathsf{pk}, D_m, D_\sigma, A_1, B_1, B_2, l)$, die Commitments $(T_{B_1}, T_{1_B}, T_{D_m}, T_{D_\sigma}, T)$ sowie die beiden Challenge-Response Tupel $(c, z_{r_m}, z_{r_\sigma}, z_{a_1}, z_{b_1}, z_{b_2}, z_{\beta_1}, z_{\beta_2}, z_e, z_s, z_{m_1}, \ldots, z_{m_L}, z_\phi)$ und $\left(c', z'_{r_m}, z'_{r_\sigma}, z'_{a_1}, z'_{b_1}, z'_{b_2}, z'_{\beta_1}, z'_{\beta_2}, z'_e, z'_s, z'_{m_1}, \ldots, z'_{m_L}, z'_\phi\right)$ mit

$$
\begin{aligned}
T_{B_1} &= g_0^{z_{b_1}} g_1^{z_{b_2}} B_1^{-c} = g_0^{z'_{b_1}} g_1^{z'_{b_2}} B_1^{-c'},\\
T_{1_B} &= B_1^{z_e} g_0^{-z_{\beta_1}} g_1^{-z_{\beta_2}} = B_1^{z'_e} g_0^{-z'_{\beta_1}} g_1^{-z'_{\beta_2}},\\
T_{D_m} &= g_0^{z_{r_m}} g_1^{z_{m_1}} \cdots g_L^{z_{m_L}} D_m^{-c} = g_0^{z'_{r_m}} g_1^{z'_{m_1}} \cdots g_L^{z'_{m_L}} D_m^{-c'},\\
T_{D_\sigma} &= g_0^{z_{r\sigma}} g_1^{z_e} g_2^{z_s} D_\sigma^{-c} = g_0^{z'_{r\sigma}} g_1^{z'_e} g_2^{z'_s} D_\sigma^{-c'},\\
T &= \hat{e}\left(g_1^{z_{b_1}} B_2^{-c}, X\right) \hat{e}\left(g^c g_0^{z_s} g_1^{z_{m_1}+z_{\beta_1}} g_2^{z_{m_2}} \ldots g_L^{z_{m_L}} u_{l,I}^{-z_{a_1}} u_I^{z_\phi} B_2^{-z_e}, h\right) \hat{e}\left(A_1^c, h_{l,I}\right)\\
&= \hat{e}\left(g_1^{z'_{b_1}} B_2^{-c'}, X\right) \hat{e}\left(g^{c'} g_0^{z'_s} g_1^{z'_{m_1}+z'_{\beta_1}} g_2^{z'_{m_2}} \ldots g_L^{z'_{m_L}} u_{l,I}^{-z'_{a_1}} u_I^{z'_\phi} B_2^{-z'_e}, h\right) \hat{e}\left(A_1^{c'}, h_{l,I}\right).
\end{aligned}
$$

Daraus folgt:

$$
\begin{aligned}
B_1 &= g_0^{\frac{z_{b_1}-z'_{b_1}}{c-c'}} g_1^{\frac{z_{b_2}-z'_{b_2}}{c-c'}} = g_0^{\frac{z_{\beta_1}-z'_{\beta_1}}{z_e-z'_e}} g_1^{\frac{z_{\beta_2}-z'_{\beta_2}}{z_e-z'_e}},\\
D_m &= g_0^{\frac{z_{r_m}-z'_{r_m}}{c-c'}} g_1^{\frac{z_{m_1}-z'_{m_1}}{c-c'}} \cdots g_L^{\frac{z_{m_L}-z'_{m_L}}{c-c'}},\\
D_\sigma &= g_0^{\frac{z_{r\sigma}-z'_{r\sigma}}{c-c'}} g_1^{\frac{z_e-z'_e}{c-c'}} g_2^{\frac{z_s-z'_s}{c-c'}},\\
\frac{\hat{e}\left(B_2, X\right)}{\hat{e}\left(g, h\right) \hat{e}\left(A_1, h_{l,I}\right)} &= \hat{e}\left(g_1^{\frac{z_{b_1}-z'_{b_1}}{c-c'}}, X\right) \hat{e}\Bigg(g_0^{\frac{z_s-z'_s}{c-c'}} g_1^{\frac{z_{m_1}-z'_{m_1}}{c-c'}+\frac{z_{\beta_1}-z'_{\beta_1}}{c-c'}} g_2^{\frac{z_{m_2}-z'_{m_2}}{c-c'}} \cdots\\
&\qquad g_L^{\frac{z_{m_L}-z'_{m_L}}{c-c'}} u_{l,I}^{-\frac{z_{a_1}-z'_{a_1}}{c-c'}} u_I^{\frac{z_\phi-z'_\phi}{c-c'}} B_2^{-\frac{z_e-z'_e}{c-c'}}, h\Bigg).
\end{aligned}
$$

Daher kann der Knowledge-Extractor $\mathcal{E}$ die folgenden Zahlen in $\mathbb{Z}_p$ berechnen:

$$
\begin{aligned}
r_m &= \frac{z_{r_m}-z'_{r_m}}{c-c'}, & r_\sigma &= \frac{z_{r_\sigma}-z'_{r_\sigma}}{c-c'}, & a_1 &= \frac{z_{a_1}-z'_{a_1}}{c-c'},\\
b_1 &= \frac{z_{b_1}-z'_{b_1}}{c-c'}, & b_2 &= \frac{z_{b_2}-z'_{b_2}}{c-c'}, & \beta_1 &= \frac{z_{\beta_1}-z'_{\beta_1}}{c-c'},\\
\beta_2 &= \frac{z_{\beta_2}-z'_{\beta_2}}{c-c'}, & e &= \frac{z_e-z'_e}{c-c'}, & s &= \frac{z_s-z'_s}{c-c'},\\
m_1 &= \frac{z_{m_1}-z'_{m_1}}{c-c'}, \ldots, & m_L &= \frac{z_{m_L}-z'_{m_L}}{c-c'}, & \phi_I\left(l\right) &= \frac{z_\phi-z'_\phi}{c-c'}.
\end{aligned}
$$

Somit gilt:

$$
B_1 = g_0^{b_1} g_1^{b_2} = g_0^{\frac{\beta_1}{e}} g_1^{\frac{\beta_2}{e}},
$$

$$
\begin{aligned}
D_m &= g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L}, \\
D_\sigma &= g_0^{r_\sigma} g_1^{e} g_2^{s}, \\
\frac{\hat{e}\left(B_2, X\right)}{\hat{e}\left(g, h\right) \hat{e}\left(A_1, h_{l,I}\right)} &= \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(g_0^{s} g_1^{m_1+\beta_1} g_2^{m_2} \ldots g_L^{m_L} u_{l,I}^{-a_1} u_I^{\phi_I(l)} B_2^{-e}, h\right).
\end{aligned}
$$

Special-Honest-Verifier-Zero-Knowledge: Der Simulator $\mathcal{SIM}$ erhält die Parameter $(\mathsf{pk}, D_m, D_\sigma, A_1, B_1, B_2, l)$ sowie die Challenge c und simuliert die Signature-of-Knowledge folgendermaßen:

Der Simulator $\mathcal{SIM}$ wählt zufällig die Responses $z_{r_m}, z_{r_\sigma}, z_{a_1}, z_{b_1}, z_{b_2}, z_{\beta_1}, z_{\beta_2}, z_e, z_s, z_{m_1}, \ldots, z_{m_L}, z_\phi \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Commitments

$$
\begin{aligned}
T_{B_1} &= g_0^{z_{b_1}} g_1^{z_{b_2}} B_1^{-c}, \\
T_{1_B} &= B_1^{z_e} g_0^{-z_{\beta_1}} g_1^{-z_{\beta_2}}, \\
T_{D_m} &= g_0^{z_{r_m}} g_1^{z_{m_1}} \cdots g_L^{z_{m_L}} D_m^{-c}, \\
T_{D_\sigma} &= g_0^{z_{r_\sigma}} g_1^{z_e} g_2^{z_s} D_\sigma^{-c}, \\
T &= \hat{e}\left(g_1^{z_{b_1}} B_2^{-c}, X\right) \hat{e}\left(g^c g_0^{z_s} g_1^{z_{m_1}+z_{\beta_1}} g_2^{z_{m_2}} \ldots g_L^{z_{m_L}} u_{l,I}^{-z_{a_1}} u_I^{z_\phi} B_2^{-z_e}, h\right) \hat{e}\left(A_1^c, h_{l,I}\right).
\end{aligned}
$$

Somit ist die Verifikation von $\mathcal{V}$ trivialer Weise erfüllt. Weiter sind alle Responses unabhängig und identisch verteilt und somit besitzen auch alle Commitments die gleiche Verteilung wie im realen Protokoll. Insgesamt hat $\mathcal{SIM}$ die Signature-of-Knowledge perfekt simuliert. □

A.4 Die SoK des Prove-K-Protokolls

Der Prover $\mathcal{P}\left(\mathsf{pk}, D_m, B_1, B_2, n_1, \ldots, n_K, r_m, b_1, b_2, \beta_1, \beta_2, e, s, m_1, \ldots, m_L\right)$ erstellt für den Verifier $\mathcal{V}\left(\mathsf{pk}, D_m, B_1, B_2, n_1, \ldots, n_K\right)$ im Signaturbeweisprotokoll $\mathsf{Prove}\text{-}K$ die folgende Signature-of-Knowledge, wobei $\beta_1 = b_1 e, \beta_2 = b_2 e$ und $V = u_0^{\prod_{j=1}^{K}(\alpha+n_j)}$ ist.

$$
\begin{aligned}
SoK\Big[(r_m, b_1, b_2, \beta_1, \beta_2, e, s, m_1, \ldots, m_L) : B_1 = g_0^{b_1} g_1^{b_2} \ \wedge\ 1 = B_1^{e} g_0^{-\beta_1} g_1^{-\beta_2} \\
\wedge\ D_m = g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L} \ \wedge \\
\hat{e}\left(B_2, X\right) \hat{e}\left(gV, h\right)^{-1} = \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(g_0^{s} g_1^{m_1+\beta_1} g_2^{m_2} \cdots g_L^{m_L} B_2^{-e}, h\right)\Big].
\end{aligned}
$$

Commitment: Der Prover $\mathcal{P}$ wählt zufällig die Zahlen $\rho_{r_m}, \rho_{b_1}, \rho_{b_2}, \rho_{\beta_1}, \rho_{\beta_2}, \rho_e, \rho_s, \rho_{m_1}, \ldots, \rho_{m_L} \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Commitments

$$
\begin{aligned}
T_{B_1} &= g_0^{\rho_{b_1}} g_1^{\rho_{b_2}}, \\
T_{1_B} &= B_1^{\rho_e} g_0^{-\rho_{\beta_1}} g_1^{-\rho_{\beta_2}}, \\
T_{D_m} &= g_0^{\rho_{r_m}} g_1^{\rho m_1} \cdots g_L^{\rho_{m_L}}, \\
T &= \hat{e}\left(g_1^{\rho_{b_1}}, X\right) \hat{e}\left(g_0^{\rho_s} g_1^{\rho_{m_1}+\rho_{\beta_1}} g_2^{\rho_{m_2}} \ldots g_L^{\rho_{m_L}} B_2^{-\rho_e}, h\right).
\end{aligned}
$$

Challenge: Der Prover $\mathcal{P}$ berechnet die Challenge

$$c = \mathsf{H}\left(D_m||B_1||B_2||T_{B_1}||T_{1_B}||T_{D_m}||T\right).$$

Response: Der Prover $\mathcal{P}$ berechnet in $\mathbb{Z}_p$ die Responses

$$\begin{aligned}
z_{r_m} &= \rho_{r_m} + cr_m, & z_{b_1} &= \rho_{b_1} + cb_1, & z_{b_2} &= \rho_{b_2} + cb_2,\\
z_{\beta_1} &= \rho_{\beta_1} + c\beta_1, & z_{\beta_2} &= \rho_{\beta_2} + c\beta_2, & z_e &= \rho_e + ce,\\
z_s &= \rho_s + cs, & z_{m_1} &= \rho_{m_1} + cm_1, \ \ldots, & z_{m_L} &= \rho_{m_L} + cm_L
\end{aligned}$$

und sendet diese mit der Challenge an $\mathcal{V}$.

Verify: Der Verifier $\mathcal{V}$ berechnet die Commitments

$$\begin{aligned}
\widetilde{T_{B_1}} &= g_0^{z_{b_1}} g_1^{z_{b_2}} B_1^{-c},\\
\widetilde{T_{1_B}} &= B_1^{z_e} g_0^{-z_{\beta_1}} g_1^{-z_{\beta_2}},\\
\widetilde{T_{D_m}} &= g_0^{z_{r_m}} g_1^{z_{m_1}} \cdots g_L^{z_{m_L}} D_m^{-c},\\
\widetilde{T} &= \hat{e}\left(g_1^{z_{b_1}} B_2^{-c}, X\right) \hat{e}\left(g^c g_0^{z_s} g_1^{z_{m_1}+z_{\beta_1}} g_2^{z_{m_2}} \cdots g_L^{z_{m_L}} B_2^{-z_e} V^c, h\right)
\end{aligned}$$

und überprüft

$$c \stackrel{?}{=} \mathsf{H}\left(D_m||B_1||B_2||\widetilde{T_{B_1}}||\widetilde{T_{1_B}}||\widetilde{T_{D_m}}||\widetilde{T}\right).$$

Satz A.4.1. *Die oben beschriebene Signature-of-Knowledge erfüllt die Durchführbarkeit, die spezielle Korrektheit und die Special-Honest-Verifier-Zero-Knowledge-Eigenschaft.*

Beweis. Wir zeigen zunächst die Durchführbarkeit, dann die spezielle Korrektheit und anschließend die Special-Honest-Verifier-Zero-Knowledge-Eigenschaft.

Durchführbarkeit: Verhält sich der Prover $\mathcal{P}$ gemäß dem Protokoll, dann gilt:

$$\begin{aligned}
\widetilde{T_{B_1}} &= g_0^{z_{b_1}} g_1^{z_{b_2}} B_1^{-c} = g_0^{\rho_{b_1}+cb_1} g_1^{\rho_{b_2}+cb_2} B_1^{-c} = g_0^{\rho_{b_1}} g_1^{\rho_{b_2}} g_0^{cb_1} g_1^{cb_2} B_1^{-c} = g_0^{\rho_{b_1}} g_1^{\rho_{b_2}} = T_{B_1},\\
\widetilde{T_{1_B}} &= B_1^{z_e} g_0^{-z_{\beta_1}} g_1^{-z_{\beta_2}} = B_1^{\rho_e+ce} g_0^{-\rho_{\beta_1}-c\beta_1} g_1^{-\rho_{\beta_2}-c\beta_2}\\
&= B_1^{\rho_e} g_0^{-\rho_{\beta_1}} g_1^{-\rho_{\beta_2}} \left(g_0^{b_1} g_1^{b_2}\right)^{ce} g_0^{-cb_1 e} g_1^{-cb_2 e} = B_1^{\rho_e} g_0^{-\rho_{\beta_1}} g_1^{-\rho_{\beta_2}} = T_{1_B},\\
\widetilde{T_{D_m}} &= g_0^{z_{r_m}} g_1^{z_{m_1}} \cdots g_L^{z_{m_L}} D_m^{-c} = g_0^{\rho_{r_m}+cr_m} g_1^{\rho_{m_1}+cm_1} \cdots g_L^{\rho_{m_L}+cm_L} D_m^{-c}\\
&= g_0^{\rho_{r_m}} g_1^{\rho_{m_1}} \cdots g_L^{\rho_{m_L}} g_0^{cr_m} g_1^{cm_1} \cdots g_L^{cm_L} D_m^{-c} = g_0^{\rho_{r_m}} g_1^{\rho_{m_1}} \cdots g_L^{\rho_{m_L}} = T_{D_m},\\
\widetilde{T} &= \hat{e}\left(g_1^{z_{b_1}} B_2^{-c}, X\right) \hat{e}\left(g^c g_0^{z_s} g_1^{z_{m_1}+z_{\beta_1}} g_2^{z_{m_2}} \cdots g_L^{z_{m_L}} B_2^{-z_e} V^c, h\right)\\
&= \hat{e}\left(g_1^{\rho_{b_1}+cb_1} B_2^{-c}, X\right) \hat{e}(g^c g_0^{\rho_s+cs} g_1^{\rho_{m_1}+cm_1+\rho_{\beta_1}+c\beta_1} g_2^{\rho_{m_2}+cm_2} \cdots\\
&\quad g_L^{\rho_{m_L}+cm_L} B_2^{-\rho_e-ce} V^c, h)
\end{aligned}$$

$$
\begin{aligned}
&= \hat{e}\left(g_1^{\rho_{b_1}}, X\right) \hat{e}\left(g_0^{\rho_s} g_1^{\rho_{m_1}+\rho_{\beta_1}} g_2^{\rho_{m_2}} \cdots g_L^{\rho_{m_L}} B_2^{-\rho_e}, h\right) \hat{e}\left(g_1^{cb_1} B_2^{-c}, X\right) \\
&\quad \hat{e}\left(g^c g_0^{cs} g_1^{cm_1+c\beta_1} g_2^{cm_2} \cdots g_L^{cm_L} B_2^{-ce} V^c, h\right) \\
&= T\left(\hat{e}\left(g_1^{b_1}\left(\Sigma g_1^{b_1}\right)^{-1}, X\right) \hat{e}\left(g g_0^{s} g_1^{m_1+b_1 e} g_2^{m_2} \cdots g_L^{m_L}\left(\Sigma g_1^{b_1}\right)^{-e} V, h\right)\right)^c \\
&= T\left(\hat{e}\left(\Sigma, X h^e\right)^{-1} \hat{e}\left(g g_0^{s} g_1^{m_1} \cdots g_L^{m_L} u_0^{\phi(\alpha)}, h\right)\right)^c = T.
\end{aligned}
$$

Spezielle Korrektheit: Der Knowledge-Extractor $\mathcal{E}$ erhält die Parameter $(\mathsf{pk}, D_m, B_1, B_2, n_1, \ldots, n_K)$, die Commitments $(T_{B_1}, T_{1_B}, T_{D_m}, T)$ und beiden Challenge-Response Tupel $(c, z_{r_m}, z_{b_1}, z_{b_2}, z_{\beta_1}, z_{\beta_2}, z_e, z_s, z_{m_1}, \ldots, z_{m_L})$ und $\left(c', z'_{r_m}, z'_{b_1}, z'_{b_2}, z'_{\beta_1}, z'_{\beta_2}, z'_e, z'_s, z'_{m_1}, \ldots, z'_{m_L}\right)$ mit

$$
\begin{aligned}
T_{B_1} &= g_0^{z_{b_1}} g_1^{z_{b_2}} B_1^{-c} = g_0^{z'_{b_1}} g_1^{z'_{b_2}} B_1^{-c'}, \\
T_{1_B} &= B_1^{z_e} g_0^{-z_{\beta_1}} g_1^{-z_{\beta_2}} = B_1^{z'_e} g_0^{-z'_{\beta_1}} g_1^{-z'_{\beta_2}}, \\
T_{D_m} &= g_0^{z_{r_m}} g_1^{z_{m_1}} \cdots g_L^{z_{m_L}} D_m^{-c} = g_0^{z'_{r_m}} g_1^{z'_{m_1}} \cdots g_L^{z'_{m_L}} D_m^{-c'}, \\
T &= \hat{e}\left(g_1^{z_{b_1}} B_2^{-c}, X\right) \hat{e}\left(g^c g_0^{z_s} g_1^{z_{m_1}+z_{\beta_1}} g_2^{z_{m_2}} \cdots g_L^{z_{m_L}} B_2^{-z_e} V^c, h\right) \\
&= \hat{e}\left(g_1^{z'_{b_1}} B_2^{-c'}, X\right) \hat{e}\left(g^{c'} g_0^{z'_s} g_1^{z'_{m_1}+z'_{\beta_1}} g_2^{z'_{m_2}} \cdots g_L^{z'_{m_L}} B_2^{-z'_e} V^{c'}, h\right).
\end{aligned}
$$

Daraus folgt:

$$
\begin{aligned}
B_1 &= g_0^{\frac{z_{b_1}-z'_{b_1}}{c-c'}} g_1^{\frac{z_{b_2}-z'_{b_2}}{c-c'}} = g_0^{\frac{z_{\beta_1}-z'_{\beta_1}}{z_e-z'_e}} g_1^{\frac{z_{\beta_2}-z'_{\beta_2}}{z_e-z'_e}}, \\
D_m &= g_0^{\frac{z_{r_m}-z'_{r_m}}{c-c'}} g_1^{\frac{z_{m_1}-z'_{m_1}}{c-c'}} \cdots g_L^{\frac{z_{m_L}-z'_{m_L}}{c-c'}}, \\
\frac{\hat{e}\left(B_2, X\right)}{\hat{e}\left(gV, h\right)} &= \hat{e}\left(g_1^{\frac{z_{b_1}-z'_{b_1}}{c-c'}}, X\right) \\
&\quad \hat{e}\left(B_2^{-\frac{z_e-z'_e}{c-c'}} g_0^{\frac{z_s-z'_s}{c-c'}} g_1^{\frac{z_{m_1}-z'_{m_1}}{c-c'}+\frac{z_{\beta_1}-z'_{\beta_1}}{c-c'}} g_2^{\frac{z_{m_2}-z'_{m_2}}{c-c'}} \cdots g_L^{\frac{z_{m_L}-z'_{m_L}}{c-c'}}, h\right).
\end{aligned}
$$

Daher kann der Knowledge-Extractor $\mathcal{E}$ die folgenden Zahlen in $\mathbb{Z}_p$ berechnen:

$$
\begin{aligned}
r_m &= \frac{z_{r_m}-z'_{r_m}}{c-c'}, & b_1 &= \frac{z_{b_1}-z'_{b_1}}{c-c'}, & b_2 &= \frac{z_{b_2}-z'_{b_2}}{c-c'}, \\
\beta_1 &= \frac{z_{\beta_1}-z'_{\beta_1}}{c-c'}, & \beta_2 &= \frac{z_{\beta_2}-z'_{\beta_2}}{c-c'}, & e &= \frac{z_e-z'_e}{c-c'}, \\
s &= \frac{z_s-z'_s}{c-c'}, & m_1 &= \frac{z_{m_1}-z'_{m_1}}{c-c'}, \ \ldots, & m_L &= \frac{z_{m_L}-z'_{m_L}}{c-c'}.
\end{aligned}
$$

Somit gilt:

$$\begin{aligned} B_1 &= g_0^{b_1} g_1^{b_2} = g_0^{\frac{\beta_1}{e}} g_1^{\frac{\beta_2}{e}}, \\ D_m &= g_0^{r_m} g_1^{m_1} \cdots g_L^{m_L}, \\ \hat{e}\left(B_2, X\right) \hat{e}\left(gV, h\right)^{-1} &= \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(g_0^{s} g_1^{m_1+\beta_1} g_2^{m_2} \cdots g_L^{m_L} B_2^{-e}, h\right). \end{aligned}$$

Special-Honest-Verifier-Zero-Knowledge: Der Simulator $\mathcal{SIM}$ erhält die Parameter $(\mathsf{pk}, D_m, B_1, B_2, n_1, \ldots, n_K)$ sowie die Challenge c und simuliert die Signature-of-Knowledge folgendermaßen:

Der Simulator $\mathcal{SIM}$ wählt zufällig die Responses $z_{r_m}, z_{b_1}, z_{b_2}, z_{\beta_1}, z_{\beta_2}, z_e, z_s, z_{m_1}, \ldots, z_{m_L} \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Commitments

$$\begin{aligned} T_{B_1} &= g_0^{z_{b_1}} g_1^{z_{b_2}} B_1^{-c}, \\ T_{1_B} &= B_1^{z_e} g_0^{-z_{\beta_1}} g_1^{-z_{\beta_2}}, \\ T_{D_m} &= g_0^{z_{r_m}} g_1^{z_{m_1}} \cdots g_L^{z_{m_L}} D_m^{-c}, \\ T &= \hat{e}\left(g_1^{z_{b_1}} B_2^{-c}, X\right) \hat{e}\left(g^{c} g_0^{z_s} g_1^{z_{m_1}+z_{\beta_1}} g_2^{z_{m_2}} \cdots g_L^{z_{m_L}} B_2^{-z_e} V^{c}, h\right). \end{aligned}$$

Somit ist die Verifikation von $\mathcal{V}$ trivialer Weise erfüllt. Weiter sind alle Responses unabhängig und identisch verteilt und somit besitzt auch alle Commitments die gleiche Verteilung wie im realen Protokoll. Insgesamt hat $\mathcal{SIM}$ die Signature-of-Knowledge perfekt simuliert. □

A.5 Die SoK des Bezahlprotokolls Spend von FTGO-I

Die zwei Prover $\mathcal{P}(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{D}}, \mathsf{pk}_{\mathcal{H}}, \mathsf{pk}_{\mathcal{O}}, S, T, T_C, A_1, B_1, B_2, C_1, C_3, l, a_1, b_1, b_2, \beta_1, \beta_2, c_1, c_2, c_3, e, o, s, t, u, \kappa, \phi_I(l))$ und $\mathcal{P}'(\mathsf{sp}, \mathsf{pk}_{\mathcal{K}}, o)$ erstellen für den Verifier $\mathcal{V}(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{D}}, S, T, T_C, A_1, B_1, B_2, C_1, C_3, l)$ im Bezahlprotokoll Spend die folgende Signature-of-Knowledge, $\beta_1 = b_1 e, \beta_2 = b_2 e, u_I = u_0^{\prod_{j\in I}(\alpha+\kappa_{L+1,j})}, u_{l,I} = u_0^{(\alpha-l)\prod_{j\in I}(\alpha+\kappa_{L+1,j})}$ und $h_{l,I} = h^{(\alpha-l)\prod_{j\in I}(\alpha+\kappa_{L+1,j})}$ ist.

$$\begin{aligned} SoK\Big[(a_1, b_1, b_2, \beta_1, \beta_2, c_1, c_2, c_3, e, o, s, t, u, \kappa, \phi_I(l)) : \\ B_1 = g_0^{b_1} g_1^{b_2} \ \wedge\ 1 = B_1^{e} g_0^{-\beta_1} g_1^{-\beta_2} \ \wedge\ S = \mathfrak{g}^{\kappa} \ \wedge\ T = \mathfrak{g}^{u} \mathfrak{g}_0^{R\kappa} \ \wedge \\ \frac{\hat{e}\left(B_2, X\right)}{\hat{e}\left(g, h\right) \hat{e}\left(A_1, h_{l,I}\right)} = \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(g_0^{s} g_1^{u+\beta_1} g_2^{t} g_{\mathcal{O}}^{o} u_{l,I}^{-a_1} u_I^{\phi_I(l)} B_2^{-e}, h\right) \ \wedge \\ T_C = \hat{e}\left(B_1^{t}, Z\right) \ \wedge\ C_1 = g_0^{c_1} g_1^{c_2} u_0^{c_3} \ \wedge\ C_3 = h^{o+c_3}\Big](R). \end{aligned}$$

Commitment $\mathcal{P}'$: Der Prover $\mathcal{P}'$ wählt zufällig eine Zahl $\dot{\rho}_o \in_{\mathcal{R}} \mathbb{Z}_p$, berechnet die Commitments

$$T_1 = g_{\mathcal{O}}^{\dot{\rho}_o}, \qquad T_2 = h^{\dot{\rho}_o}$$

und sendet T_1 sowie T_2 an $\mathcal{P}$.

Commitment: Der Prover $\mathcal{P}$ wählt zufällig die Zahlen $\rho_{a_1}, \rho_{b_1}, \rho_{b_2}, \rho_{\beta_1}, \rho_{\beta_2}, \rho_{c_1}, \rho_{c_2}, \rho_{c_3}, \rho_e, \ddot{\rho}_o, \rho_s, \rho_t, \rho_u, \rho_\kappa, \rho_\phi, \delta \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Commitments

$$\begin{aligned}
T_{B_1} &= g_0^{\rho_{b_1}} g_1^{\rho_{b_2}}, \\
T_{1_B} &= B_1^{\rho_e} g_0^{-\rho_{\beta_1}} g_1^{-\rho_{\beta_2}}, \\
T_S &= \mathfrak{g}^{\rho_\kappa}, \\
T_T &= \mathfrak{g}^{\rho_u} \mathfrak{g}_0^{R\rho_\kappa}, \\
T_{T_C} &= \hat{e}\left(B_1^{\rho_t}, Z\right), \\
T_{C_1} &= g_0^{\rho_{c_1}} g_1^{\rho_{c_2}} u_0^{\rho_{c_3}}, \\
T_{C_3} &= h^{\ddot{\rho}_o} T_2 O^{\delta} h^{\rho_{c_3}}, \\
T_X &= \hat{e}\left(g_1^{\rho_{b_1}}, X\right) \hat{e}\left(g_0^{\rho_s} g_1^{\rho_u + \rho_{\beta_1}} g_2^{\rho_t} g_{\mathcal{O}}^{\ddot{\rho}_o} T_1 I_{\mathcal{O}}^{\delta} u_{l,I}^{-\rho_{a_1}} u_I^{\rho_\phi} B_2^{-\rho_e}, h\right).
\end{aligned}$$

Challenge: Der Prover $\mathcal{P}$ berechnet die Challenge

$$c = \mathsf{H}\left(S||T||T_C||A_1||B_1||B_2||C_1||C_3||T_{B_1}||T_{1_B}||T_S||T_T||T_{T_C}||T_{C_1}||T_{C_3}||T_X||R\right)$$

und sendet c an $\mathcal{P}'$.

Response $\mathcal{P}'$: Der Prover $\mathcal{P}'$ berechnet die Response

$$\dot{z_o} = \dot{\rho}_o + co \mod p$$

und sendet $\dot{z_o}$ an $\mathcal{P}$.

Verify $\mathcal{P}'$: Der Prover $\mathcal{P}$ verifiziert die Gleichungen

$$T_1 \stackrel{?}{=} g_{\mathcal{O}}^{\dot{z_o}} I_{\mathcal{O}}^{-c}, \qquad T_2 \stackrel{?}{=} h^{\dot{z_o}} O^{-c}.$$

Response: Der Prover $\mathcal{P}$ berechnet in $\mathbb{Z}_p$ die Responses

$$\begin{aligned}
z_{a_1} &= \rho_{a_1} + ca_1, & z_{b_1} &= \rho_{b_1} + cb_1, & z_{b_2} &= \rho_{b_2} + cb_2, \\
z_{\beta_1} &= \rho_{\beta_1} + c\beta_1, & z_{\beta_2} &= \rho_{\beta_2} + c\beta_2, & z_{c_1} &= \rho_{c_1} + cc_1, \\
z_{c_2} &= \rho_{c_2} + cc_2, & z_{c_3} &= \rho_{c_3} + cc_3, & z_e &= \rho_e + ce, \\
z_o &= \dot{z_o} + \ddot{\rho}_o, & z_s &= \rho_s + cs, & z_t &= \rho_t + ct, \\
z_u &= \rho_u + cu, & z_\kappa &= \rho_\kappa + c\kappa, & z_\phi &= \rho_\phi + c\phi_I\left(l\right)
\end{aligned}$$

und sendet diese mit der Challenge an $\mathcal{V}$.

Verify: Der Verifier $\mathcal{V}$ berechnet die Commitments

$$\begin{aligned}
\widetilde{T_{B_1}} &= g_0^{z_{b_1}} g_1^{z_{b_2}} B_1^{-c},\\
\widetilde{T_{1_B}} &= B_1^{z_e} g_0^{-z_{\beta_1}} g_1^{-z_{\beta_2}},\\
\widetilde{T_S} &= \mathfrak{g}^{z_\kappa} S^{-c},\\
\widetilde{T_T} &= \mathfrak{g}^{z_u} \mathfrak{g}_0^{R z_\kappa} T^{-c},\\
\widetilde{T_{T_C}} &= \hat{e}\left(B_1^{z_t}, Z\right) T_C^{-c},\\
\widetilde{T_{C_1}} &= g_0^{z_{c_1}} g_1^{z_{c_2}} u_0^{z_{c_3}} C_1^{-c},\\
\widetilde{T_{C_3}} &= h^{z_o + z_{c_3}} C_3^{-c},\\
\widetilde{T_X} &= \hat{e}\left(g_1^{z_{b_1}} B_2^{-c}, X\right) \hat{e}\left(g^c g_0^{z_s} g_1^{z_u + z_{\beta_1}} g_2^{z_t} g_{\mathcal{O}}^{z_o} u_{l,I}^{-z_{a_1}} u_I^{z_\phi} B_2^{-z_e}, h\right) \hat{e}\left(A_1^c, h_{l,I}\right).
\end{aligned}$$

und überprüft

$$c \stackrel{?}{=} \mathsf{H}\left(S||T||T_C||A_1||B_1||B_2||C_1||C_2||C_3||\widetilde{T_{B_1}}||\widetilde{T_{1_B}}||\widetilde{T_S}||\widetilde{T_T}||\widetilde{T_{T_C}}||\widetilde{T_{C_1}}||\widetilde{T_{C_3}}||\widetilde{T_X}||R\right).$$

Satz A.5.1. *Die oben beschriebene Signature-of-Knowledge erfüllt die Durchführbarkeit, die spezielle Korrektheit und die Special-Honest-Verifier-Zero-Knowledge-Eigenschaft.*

Beweis. Wir zeigen zunächst die Durchführbarkeit, dann die spezielle Korrektheit und anschließend die Special-Honest-Verifier-Zero-Knowledge-Eigenschaft.

Durchführbarkeit: Verhalten sich die Prover $\mathcal{P}$ und $\mathcal{P}'$ gemäß dem Protokoll, dann gilt $z_o = \dot{z_o} + \ddot{\rho_o} = \dot{\rho_o} + co + \ddot{\rho_o} \in \mathbb{Z}_p$ und damit:

$$\begin{aligned}
\widetilde{T_{B_1}} &= g_0^{z_{b_1}} g_1^{z_{b_2}} B_1^{-c} = g_0^{\rho_{b_1} + cb_1} g_1^{\rho_{b_2} + cb_2} B_1^{-c} = g_0^{\rho_{b_1}} g_1^{\rho_{b_2}} g_0^{cb_1} g_1^{cb_2} \left(g_0^{b_1} g_1^{b_2}\right)^{-c} = g_0^{\rho_{b_1}} g_1^{\rho_{b_2}} = T_{B_1},\\
\widetilde{T_{1_B}} &= B_1^{z_e} g_0^{-z_{\beta_1}} g_1^{-z_{\beta_2}} = B_1^{\rho_e + ce} g_0^{-\rho_{\beta_1} - c\beta_1} g_1^{-\rho_{\beta_2} - c\beta_2}\\
&= B_1^{\rho_e} g_0^{-\rho_{\beta_1}} g_1^{-\rho_{\beta_2}} \left(g_0^{b_1} g_1^{b_2}\right)^{ce} g_0^{-cb_1 e} g_1^{-cb_2 e} = B_1^{\rho_e} g_0^{-\rho_{\beta_1}} g_1^{-\rho_{\beta_2}} = T_{1_B},\\
\widetilde{T_S} &= \mathfrak{g}^{z_\kappa} S^{-c} = \mathfrak{g}^{\rho_\kappa + c\kappa} S^{-c} = \mathfrak{g}^{\rho_\kappa} \mathfrak{g}^{c\kappa} \left(\mathfrak{g}^{\kappa}\right)^{-c} = \mathfrak{g}^{\rho_\kappa} = T_S,\\
\widetilde{T_T} &= \mathfrak{g}^{z_u} \mathfrak{g}_0^{R z_\kappa} T^{-c} = \mathfrak{g}^{\rho_u + cu} \mathfrak{g}_0^{R\rho_\kappa + Rc\kappa} T^{-c} = \mathfrak{g}^{\rho_u} \mathfrak{g}_0^{R\rho_\kappa} \mathfrak{g}^{cu} \mathfrak{g}_0^{Rc\kappa} \left(\mathfrak{g}^{u} \mathfrak{g}_0^{R\kappa}\right)^{-c} = \mathfrak{g}^{\rho_u} \mathfrak{g}_0^{R\rho_\kappa} = T_T,\\
\widetilde{T_{T_C}} &= \hat{e}\left(B_1^{z_t}, Z\right) T_C^{-c} = \hat{e}\left(B_1^{\rho_t + ct}, Z\right) T_C^{-c} = \hat{e}\left(B_1^{\rho_t}, Z\right) \hat{e}\left(B_1^{ct}, Z\right) \hat{e}\left(B_1^{t}, Z\right)^{-c}\\
&= \hat{e}\left(B_1^{\rho_t}, Z\right) = T_{T_C}\\
\widetilde{T_{C_1}} &= g_0^{z_{c_1}} g_1^{z_{c_2}} u_0^{z_{c_3}} C_1^{-c} = g_0^{\rho_{c_1} + cc_1} g_1^{\rho_{c_2} + cc_2} u_0^{\rho_{c_3} + cc_3} C_1^{-c} = g_0^{\rho_{c_1}} g_1^{\rho_{c_2}} u_0^{\rho_{c_3}} g_0^{cc_1} g_1^{cc_2} u_0^{cc_3} \left(g_0^{c_1} g_1^{c_2} u_0^{c_3}\right)^{-c}\\
&= g_0^{\rho_{c_1}} g_1^{\rho_{c_2}} u_0^{\rho_{c_3}} = T_{C_1}\\
\widetilde{T_{C_3}} &= h^{z_o + z_{c_3}} C_3^{-c} = h^{\dot{\rho_o} + (c+\delta)o + \ddot{\rho_o} + \rho_{c_3} + cc_3} C_3^{-c} = h^{\ddot{\rho_o} + \dot{\rho_o} + \delta o + \rho_{c_3}} h^{co + cc_3} C_3^{-c}\\
&= h^{\ddot{\rho_o}} T_2 O^{\delta} h^{\rho_{c_3}} h^{co + cc_3} \left(h^{o + c_3}\right)^{-c} = h^{\ddot{\rho_o}} T_2 O^{\delta} h^{\rho_{c_3}} = T_{C_3},\\
\widetilde{T_X} &= \hat{e}\left(g_1^{z_{b_1}} B_2^{-c}, X\right) \hat{e}\left(g^c g_0^{z_s} g_1^{z_u + z_{\beta_1}} g_2^{z_t} g_{\mathcal{O}}^{z_o} u_{l,I}^{-z_{a_1}} u_I^{z_\phi} B_2^{-z_e}, h\right) \hat{e}\left(A_1, h_{l,I}^{c}\right)
\end{aligned}$$

$$
\begin{aligned}
&= \hat{e}\left(g_1^{\rho_{b_1}+cb_1} B_2^{-c}, X\right) \hat{e}\Big(g^c g_0^{\rho_s+cs} g_1^{\rho_u+cu+\rho_{\beta_1}+c\beta_1} g_2^{\rho_t+ct} g_{\mathcal{O}}^{\dot{\rho}_o+(c+\delta)o+\ddot{\rho}_o} \\
&\quad u_{l,I}^{-\rho_{a_1}-ca_1} u_I^{\rho_\phi+c\phi_I(l)} B_2^{-\rho_e-ce}, h\Big) \hat{e}\left(A_1^c, h_{l,I}\right) \\
&= \hat{e}\left(g_1^{\rho_{b_1}}, X\right) \hat{e}\left(g_0^{\rho_s} g_1^{\rho_u+\rho_{\beta_1}} g_2^{\rho_t} g_{\mathcal{O}}^{\ddot{\rho}_o+\dot{\rho}_o+\delta o} u_{l,I}^{-\rho_{a_1}} u_I^{\rho_\phi} B_2^{-\rho_e}, h\right) \hat{e}\left(g_1^{cb_1} B_2^{-c}, X\right) \\
&\quad \hat{e}(g^c g_0^{cs} g_1^{cu+c\beta_1} g_2^{ct} g_{\mathcal{O}}^{co} u_{l,I}^{-ca_1} u_I^{c\phi_I(l)} B_2^{-ce}, h) \hat{e}\left(A_1^c, h_{l,I}\right) \\
&= \hat{e}\left(g_1^{\rho_{b_1}}, X\right) \hat{e}\left(g_0^{\rho_s} g_1^{\rho_u+\rho_{\beta_1}} g_2^{\rho_t} g_{\mathcal{O}}^{\ddot{\rho}_o} T_1 I_{\mathcal{O}}^{\delta} u_{l,I}^{-\rho_{a_1}} u_I^{\rho_\phi} B_2^{-\rho_e}, h\right) \hat{e}\left(g_1^{cb_1} B_2^{-c}, X\right) \\
&\quad \hat{e}(g^c g_0^{cs} g_1^{cu+c\beta_1} g_2^{ct} g_{\mathcal{O}}^{co} u_{l,I}^{-ca_1} u_I^{c\phi_I(l)} B_2^{-ce}, h) \hat{e}\left(A_1^c, h_{l,I}\right) \\
&= T_X \left(\hat{e}\left(g_1^{b_1}\left(\Sigma g_1^{b_1}\right)^{-1}, X\right) \hat{e}\left(g g_0^s g_1^{u+b_1 e} g_2^t g_{\mathcal{O}}^o u_{l,I}^{-a_1} u_I^{\phi_I(l)} \left(\Sigma g_1^{b_1}\right)^{-e}, h\right) \hat{e}\left(W_{I,l} u_0^{a_1}, h_{l,I}\right)\right)^c \\
&= T_X \left(\hat{e}\left(\Sigma, X h^e\right)^{-1} \hat{e}\left(g g_0^s g_1^u g_2^t g_{\mathcal{O}}^o u_I^{\phi_I(l)}, h\right) \hat{e}\left(W_{I,l}, h_{l,I}\right)\right)^c \\
&= T_X \left(\hat{e}\left(\Sigma, X h^e\right)^{-1} \hat{e}\left(g g_0^s g_1^u g_2^t g_{\mathcal{O}}^o u_0^{\phi(\alpha)}, h\right)\right)^c = T_X,
\end{aligned}
$$

da für $W_{I,l}$ die Gleichung $\hat{e}\left(u_0^{\phi(\alpha)}, h\right) = \hat{e}\left(W_{I,l}, h_{l,I}\right) \hat{e}\left(u_o^{\phi_I(l)}, h_I\right)$ und für Σ die Gleichung $\left(\Sigma, X h^e\right) = \hat{e}\left(g g_0^s g_1^u g_2^t g_{\mathcal{O}}^o u_0^{\phi(\alpha)}, h\right)$ gilt.

Spezielle Korrektheit: Der Knowledge-Extractor $\mathcal{E}$ erhält die Parameter $(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{D}}, S, T, T_C, A_1, B_1, B_2, C_1, C_3, l)$, die Commitments $(T_{B_1}, T_{1_B}, T_S, T_T, T_{T_C}, T_{C_1}, T_{C_3}, T_X)$ sowie die beiden Challenge-Response Tupel $(c, z_{a_1}, z_{b_1}, z_{b_2}, z_{\beta_1}, z_{\beta_2}, z_{c_1}, z_{c_2}, z_{c_3}, z_e, z_o, z_s, z_t, z_u, z_\kappa, z_\phi)$ und $\left(c', z'_{a_1}, z'_{b_1}, z'_{b_2}, z'_{\beta_1}, z'_{\beta_2}, z'_{c_1}, z'_{c_2}, z'_{c_3}, z'_e, z'_o, z'_s, z'_t, z'_u, z'_\kappa, z'_\phi\right)$ mit

$$
\begin{aligned}
T_{B_1} &= g_0^{z_{b_1}} g_1^{z_{b_2}} B_1^{-c} = g_0^{z'_{b_1}} g_1^{z'_{b_2}} B_1^{-c'}, \\
T_{1_B} &= B_1^{z_e} g_0^{-z_{\beta_1}} g_1^{-z_{\beta_2}} = B_1^{z'_e} g_0^{-z'_{\beta_1}} g_1^{-z'_{\beta_2}}, \\
T_S &= \mathfrak{g}^{z_\kappa} S^{-c} = \mathfrak{g}^{z'_\kappa} S^{-c'}, \\
T_T &= \mathfrak{g}^{z_u} \mathfrak{g}_0^{R z_\kappa} T^{-c} = \mathfrak{g}^{z'_u} \mathfrak{g}_0^{R z'_\kappa} T^{-c'}, \\
T_{T_C} &= \hat{e}\left(B_1^{z_t}, Z\right) T_C^{-c} = \hat{e}\left(B_1^{z'_t}, Z\right) T_C^{-c'}, \\
T_{C_1} &= g_0^{z_{c_1}} g_1^{z_{c_2}} u_0^{z_{c_3}} C_1^{-c} = g_0^{z'_{c_1}} g_1^{z'_{c_2}} u_0^{z'_{c_3}} C_1^{-c'}, \\
T_{C_3} &= h^{z_o+z_{c_3}} C_3^{-c} = h^{z'_o+z'_{c_3}} C_3^{-c'}, \\
T_X &= \hat{e}\left(g_1^{z_{b_1}} B_2^{-c}, X\right) \hat{e}\left(g^c g_0^{z_s} g_1^{z_u+z_{\beta_1}} g_2^{z_t} g_{\mathcal{O}}^{z_o} u_{l,I}^{-z_{a_1}} u_I^{z_\phi} B_2^{-z_e}, h\right) \hat{e}\left(A_1^c, h_{l,I}\right) \\
&= \hat{e}\left(g_1^{z'_{b_1}} B_2^{-c'}, X\right) \hat{e}\left(g^{c'} g_0^{z'_s} g_1^{z'_u+z'_{\beta_1}} g_2^{z'_t} g_{\mathcal{O}}^{z'_o} u_{l,I}^{-z'_{a_1}} u_I^{z'_\phi} B_2^{-z'_e}, h\right) \hat{e}\left(A_1^{c'}, h_{l,I}\right).
\end{aligned}
$$

Daraus folgt:

$$
B_1 = g_0^{\frac{z_{b_1}-z'_{b_1}}{c-c'}} g_1^{\frac{z_{b_2}-z'_{b_2}}{c-c'}} = g_0^{\frac{z_{\beta_1}-z'_{\beta_1}}{z_e-z'_e}} g_1^{\frac{z_{\beta_2}-z'_{\beta_2}}{z_e-z'_e}},
$$

$$
\begin{aligned}
S &= \mathfrak{g}^{\frac{z_\kappa - z'_\kappa}{c-c'}}, \\
T &= \mathfrak{g}^{\frac{z_u - z'_u}{c-c'}} \mathfrak{g}_o^{R\frac{z_\kappa - z'_\kappa}{c-c'}}, \\
T_C &= \hat{e}\left(B_1^{\frac{z_t - z'_t}{c-c'}}, Z\right), \\
C_1 &= g_0^{\frac{z_{c_1} - z'_{c_1}}{c-c'}} g_1^{\frac{z_{c_2} - z'_{c_2}}{c-c'}} u_0^{\frac{z_{c_3} - z'_{c_3}}{c-c'}}, \\
C_3 &= h^{\frac{z_o - z'_o}{c-c'} + \frac{z_{c_3} - z'_{c_3}}{c-c'}}, \\
\frac{\hat{e}\left(B_2, X\right)}{\hat{e}\left(g,h\right)\hat{e}\left(A_1, h_{l,I}\right)} &= \hat{e}\left(g_1^{\frac{z_{b_1} - z'_{b_1}}{c-c'}}, X\right) \hat{e}\Bigg(g_0^{\frac{z_s - z'_s}{c-c'}} g_1^{\frac{z_u - z'_u}{c-c'} + \frac{z_{\beta_1} - z'_{\beta_1}}{c-c'}} g_2^{\frac{z_t - z'_t}{c-c'}} \\
&\qquad g_{\mathcal{O}}^{\frac{z_o - z'_o}{c-c'}} u_{l,I}^{-\frac{z_{a_1} - z'_{a_1}}{c-c'}} u_I^{\frac{z_\phi - z'_\phi}{c-c'}} B_2^{-\frac{z_e - z'_e}{c-c'}}, h\Bigg).
\end{aligned}
$$

Daher kann der Knowledge-Extractor $\mathcal{E}$ die folgenden Zahlen in $\mathbb{Z}_p$ berechnen:

$$
\begin{aligned}
a_1 &= \frac{z_{a_1} - z'_{a_1}}{c-c'}, & b_1 &= \frac{z_{b_1} - z'_{b_1}}{c-c'}, & b_2 &= \frac{z_{b_2} - z'_{b_2}}{c-c'}, \\
\beta_1 &= \frac{z_{\beta_1} - z'_{\beta_1}}{c-c'}, & \beta_2 &= \frac{z_{\beta_2} - z'_{\beta_2}}{c-c'}, & c_1 &= \frac{z_{c_1} - z'_{c_1}}{c-c'}, \\
c_2 &= \frac{z_{c_2} - z'_{c_2}}{c-c'}, & c_3 &= \frac{z_{c_3} - z'_{c_3}}{c-c'}, & e &= \frac{z_e - z'_e}{c-c'}, \\
o &= \frac{z_o - z'_o}{c-c'}, & s &= \frac{z_s - z'_s}{c-c'}, & t &= \frac{z_t - z'_t}{c-c'}, \\
u &= \frac{z_u - z'_u}{c-c'}, & \kappa &= \frac{z_\kappa - z'_\kappa}{c-c'}, & \phi_I\left(l\right) &= \frac{z_\phi - z'_\phi}{c-c'}.
\end{aligned}
$$

Somit gilt:

$$
\begin{aligned}
B_1 &= g_0^{b_1} g_1^{b_2} = g_0^{\frac{\beta_1}{e}} g_1^{\frac{\beta_2}{e}}, \\
S &= \mathfrak{g}^{\kappa}, \\
T &= \mathfrak{g}^{u} \mathfrak{g}_o^{R\kappa}, \\
T_C &= \hat{e}\left(B_1^t, Z\right), \\
C_1 &= g_0^{c_1} g_1^{c_2} u_0^{c_3}, \\
C_3 &= h^{o+c_3}, \\
\frac{\hat{e}\left(B_2, X\right)}{\hat{e}\left(g,h\right)\hat{e}\left(A_1, h_{l,I}\right)} &= \hat{e}\left(g_1^{b_1}, X\right) \hat{e}\left(g_0^s g_1^{u+\beta_1} g_2^t g_{\mathcal{O}}^o u_{l,I}^{-a_1} u_I^{\phi_I(l)} B_2^{-e}, h\right).
\end{aligned}
$$

Special-Honest-Verifier-Zero-Knowledge: Der Simulator $\mathcal{SIM}$ erhält die Parameter $(\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{D}}, S, T, T_C, A_1, B_1, B_2, C_1, C_3, l)$ sowie die Challenge c und simuliert die Signature-of-Knowledge folgendermaßen:

Der Simulator $\mathcal{SIM}$ wählt zufällig die Responses $z_{a_1}, z_{b_1}, z_{b_2}, z_{\beta_1}, z_{\beta_2}, z_{c_1}, z_{c_2}, z_{c_3}, z_e, z_o, z_s, z_t, z_u, z_\kappa, z_\phi \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Commitments

$$\begin{aligned}
T_{B_1} &= g_0^{z_{b_1}} g_1^{z_{b_2}} B_1^{-c},\\
T_{1_B} &= B_1^{z_e} g_0^{-z_{\beta_1}} g_1^{-z_{\beta_2}},\\
T_S &= \mathfrak{g}^{z_\kappa} S^{-c},\\
T_T &= \mathfrak{g}^{z_u} \mathfrak{g}_0^{R z_\kappa} T^{-c},\\
T_{T_C} &= \hat{e}\left(B_1^{z_t}, Z\right) T_C^{-c},\\
T_{C_1} &= g_0^{z_{c_1}} g_1^{z_{c_2}} u_0^{z_{c_3}} C_1^{-c},\\
T_{C_3} &= h^{z_o + z_{c_3}} C_3^{-c},\\
T_X &= \hat{e}\left(g_1^{z_{b_1}} B_2^{-c}, X\right) \hat{e}\left(g^c g_0^{z_s} g_1^{z_u + z_{\beta_1}} g_2^{z_t} g_{\mathcal{O}}^{z_o} u_{l,I}^{-z_{a_1}} u_I^{z_\phi} B_2^{-z_e}, h\right) \hat{e}\left(A_1^c, h_{l,I}\right).
\end{aligned}$$

Somit ist die Verifikation von $\mathcal{V}$ trivialer Weise erfüllt. Weiter sind alle Responses unabhängig und identisch verteilt und somit besitzen auch alle Commitments die gleiche Verteilung wie im realen Protokoll. Insgesamt hat $\mathcal{SIM}$ die Signature-of-Knowledge perfekt simuliert. □

Weiter entspricht die Kommunikation zwischen $\mathcal{P}'$ und $\mathcal{P}$ genau dem folgenden Proof-of-Knowledge über einen diskreten Logarithmus

$$PoK\left[(o) : I_{\mathcal{O}} = g_{\mathcal{O}}^o \wedge O = h^o\right]$$

(vgl. auch Abbildung 9.2) und damit dem Schnorr Identifikationsverfahren in den Gruppen $\mathbb{G}_1$ und $\mathbb{G}_2$.

Satz A.5.2. *Die Kommunikation zwischen $\mathcal{P}'$ und $\mathcal{P}$ entspricht genau dem Proof-of-Knowledge*

$$PoK\left[(o) : I_{\mathcal{O}} = g_{\mathcal{O}}^o \wedge O = h^o\right]$$

und erfüllt die Durchführbarkeit, die spezielle Korrektheit und die Special-Honest-Verifier-Zero-Knowledge-Eigenschaft.

Beweis. Wir zeigen zunächst, dass es sich um einen Proof-of-Knowledge handelt.

Aus dem Ablauf obiger Signature-of-Knowledge entnehmen wir die drei Phasen

Commitment $\mathcal{P}'$: Der Prover $\mathcal{P}'$ wählt zufällig eine Zahl $\dot{\rho_o} \in_{\mathcal{R}} \mathbb{Z}_p$, berechnet die Commitments

$$T_1 = g_{\mathcal{O}}^{\dot{\rho_o}}, \qquad T_2 = h^{\dot{\rho_o}}$$

und sendet T_1 sowie T_2 an $\mathcal{P}$.

Challenge: Der Prover $\mathcal{P}$ berechnet die Challenge

$$c = \mathsf{H}\left(S||T||T_C||A_1||B_1||B_2||C_1||C_2||C_3||T_{B_1}||T_{1_B}||T_S||T_T||T_{T_C}||T_{C_1}||T_{C_3}||T_X||R\right)$$

und sendet c an $\mathcal{P}'$.

Response $\mathcal{P}'$: Der Prover $\mathcal{P}'$ berechnet die Response

$$\dot{z_o} = \dot{\rho_o} + co \mod p$$

und sendet diese an $\mathcal{P}$.

Verify $\mathcal{P}'$: Der Prover $\mathcal{P}$ verifiziert die Gleichungen

$$T_1 \stackrel{?}{=} g_{\mathcal{O}}^{\dot{z_o}} I_{\mathcal{O}}^{-c}, \qquad T_2 \stackrel{?}{=} h^{\dot{z_o}} O^{-c}.$$

Somit handelt es sich genau um ein ein Σ-Protokoll und damit um einen Proof-of-Knowledge (vgl. Unterabschnitt 2.10.1).

Analog zu den Σ-Protokollen in Unterunterabschnitt 2.10.1.1 folgen die Durchführbarkeit, die spezielle Korrektheit und die Special-Honest-Verifier-Zero-Knowledge-Eigenschaft. □

A.6 Die zweite SoK des Bezahlprotokolls Spend von FTGO-I

Der Prover $\mathcal{P}(\mathsf{sp}, C_1, C_2, C_3, c_4, c_5, c_6, \gamma_1, \gamma_2, \gamma_3, e', r)$ erstellt für den Verifier $\mathcal{V}(\mathsf{sp}, C_1, C_2, C_3)$ im Bezahlprotokoll Spend die folgende Signature-of-Knowledge SoK_σ, wobei $\gamma_1 = c_4(e' - c_6), \gamma_2 = c_5(e' - c_6)$ und $\gamma_3 = c_6(e' - c_6)$ ist.

$$SoK_\sigma\Bigg[(c_4, c_5, c_6, \gamma_1, \gamma_2, \gamma_3, e', r) : C_1 = g_0^{c_4} g_1^{c_5} u_0^{c_6} \ \wedge \ 1 = C_1^{e'-c_6} g_0^{-\gamma_1} g_1^{-\gamma_2} u_0^{-\gamma_3} \ \wedge$$
$$\frac{\hat{e}\left(C_2, C_3\right)}{\hat{e}\left(g g_1^R u_0^k, h\right)} = \hat{e}\left(g_1^{c_4}, C_3\right) \hat{e}\left(g_0^r g_1^{\gamma_1} C_2^{c_6 - e'}, h\right)\Bigg]\left(R||SoK\right).$$

Commitment: Der Prover $\mathcal{P}$ wählt zufällig die Zahlen $\rho_{c_4}, \rho_{c_5}, \rho_{c_6}, \rho_{\gamma_1}, \rho_{\gamma_2}, \rho_{\gamma_3}, \rho_e, \rho_r \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Commitments

$$\begin{aligned} T_{C_1} &= g_0^{\rho_{c_4}} g_1^{\rho_{c_5}} u_0^{\rho_{c_6}}, \\ T_{1_C} &= C_1^{\rho_e - \rho_{c_6}} g_0^{-\rho_{\gamma_1}} g_1^{-\rho_{\gamma_2}} u_0^{-\rho_{\gamma_3}}, \\ T_O &= \hat{e}\left(g_1^{\rho_{c_4}}, C_3\right) \hat{e}\left(g_0^{\rho_r} g_1^{\rho_{\gamma_1}} C_2^{\rho_{c_6} - \rho_e}, h\right). \end{aligned}$$

Challenge: Der Prover $\mathcal{P}$ berechnet die Challenge

$$c_\sigma = \mathsf{H}\left(C_1||C_2||C_3||T_{C_1}||T_{1_C}||T_O||R||SoK\right),$$

wobei das Challenge-Response Tupel der Signature-of-Knowledge SoK zur Berechnung der Challenge c_σ mit in die Hashfunktion einfließt.

Response: Der Prover $\mathcal{P}$ berechnet in $\mathbb{Z}_p$ die Responses

$$\begin{aligned}
z_{c_4} &= \rho_{c_4} + c_\sigma c_4, & z_{c_5} &= \rho_{c_5} + c_\sigma c_5, & z_{c_6} &= \rho_{c_6} + c_\sigma c_6,\\
z_{\gamma_1} &= \rho_{\gamma_1} + c_\sigma \gamma_1, & z_{\gamma_2} &= \rho_{\gamma_2} + c_\sigma \gamma_2, & z_{\gamma_3} &= \rho_{\gamma_3} + c_\sigma \gamma_3,\\
z_e &= \rho_e + c_\sigma e', & z_r &= \rho_r + c_\sigma r
\end{aligned}$$

und sendet diese mit der Challenge an $\mathcal{V}$.

Verify: Der Verifier $\mathcal{V}$ berechnet die Commitments

$$\begin{aligned}
\widetilde{T_{C_1}} &= g_0^{z_{c_4}} g_1^{z_{c_5}} u_0^{z_{c_6}} C_1^{-c_\sigma},\\
\widetilde{T_{1_C}} &= C_1^{z_e - z_{c_6}} g_0^{-z_{\gamma_1}} g_1^{-z_{\gamma_2}} u_0^{-z_{\gamma_3}},\\
\widetilde{T_O} &= \hat{e}\left(g_1^{z_{c_4}} C_2^{-c_\sigma}, C_3\right) \hat{e}\left(g^{c_\sigma} g_0^{z_r} g_1^{c_\sigma R + z_{\gamma_1}} u_0^{c_\sigma k} C_2^{z_{c_6} - z_e}, h\right)
\end{aligned}$$

und überprüft

$$c_\sigma \stackrel{?}{=} \mathsf{H}\left(C_1||C_2||C_3||\widetilde{T_{C_1}}||\widetilde{T_{1_C}}||\widetilde{T_O}||R||c\right).$$

Satz A.6.1. *Die oben beschriebene Signature-of-Knowledge erfüllt die Durchführbarkeit, die spezielle Korrektheit und die Special-Honest-Verifier-Zero-Knowledge-Eigenschaft.*

Beweis. Wir zeigen zunächst die Durchführbarkeit, dann die spezielle Korrektheit und anschließend die Special-Honest-Verifier-Zero-Knowledge-Eigenschaft.

Durchführbarkeit: Verhält sich der Prover $\mathcal{P}$ gemäß dem Protokoll, dann gilt $c_4 = c_1, c_5 = c_2, c_6 = c_3$ und damit:

$$\begin{aligned}
\widetilde{T_{C_1}} &= g_0^{z_{c_4}} g_1^{z_{c_5}} u_0^{z_{c_6}} C_1^{-c_\sigma} = g_0^{\rho_{c_4} + c_\sigma c_4} g_1^{\rho_{c_5} + c_\sigma c_5} u_0^{\rho_{c_6} + c_\sigma c_6} C_1^{-c_\sigma}\\
&= g_0^{\rho_{c_4}} g_1^{\rho_{c_5}} u_0^{\rho_{c_6}} g_0^{c_\sigma c_4} g_1^{c_\sigma c_5} u_0^{c_\sigma c_6} \left(g_0^{c_4} g_1^{c_5} u_0^{c_6}\right)^{-c_\sigma} = g_0^{\rho_{c_4}} g_1^{\rho_{c_5}} u_0^{\rho_{c_6}} = T_{C_1},\\
\widetilde{T_{1_C}} &= C_1^{z_e - z_{c_6}} g_0^{-z_{\gamma_1}} g_1^{-z_{\gamma_2}} u_0^{-z_{\gamma_3}} = C_1^{\rho_e + c_\sigma e' - \rho_{c_6} - c_\sigma c_6} g_0^{-\rho_{\gamma_1} - c_\sigma \gamma_1} g_1^{-\rho_{\gamma_2} - c_\sigma \gamma_2} u_0^{-\rho_{\gamma_3} - c_\sigma \gamma_3}\\
&= C_1^{\rho_e - \rho_{c_6}} g_0^{-\rho_{\gamma_1}} g_1^{-\rho_{\gamma_2}} u_0^{-\rho_{\gamma_3}} \left(g_0^{c_4} g_1^{c_5} u_0^{c_6}\right)^{c_\sigma e' - c_\sigma c_6} g_0^{-c_\sigma c_4 (e' - c_6)} g_1^{-c_\sigma c_5 (e' - c_6)} u_0^{-c_\sigma c_6 (e' - c_6)}\\
&= C_1^{\rho_e - \rho_{c_6}} g_0^{-\rho_{\gamma_1}} g_1^{-\rho_{\gamma_2}} u_0^{-\rho_{\gamma_3}} = T_{1_C},\\
\widetilde{T_O} &= \hat{e}\left(g_1^{z_{c_4}} C_2^{-c_\sigma}, C_3\right) \hat{e}\left(g^{c_\sigma} g_0^{z_r} g_1^{c_\sigma R + z_{\gamma_1}} u_0^{c_\sigma k} C_2^{z_{c_6} - z_e}, h\right)\\
&= \hat{e}\left(g_1^{\rho_{c_4} + c_\sigma c_4} C_2^{-c_\sigma}, C_3\right) \hat{e}\left(g^{c_\sigma} g_0^{\rho_r + c_\sigma r} g_1^{c_\sigma R + \rho_{\gamma_1} + c_\sigma \gamma_1} u_0^{c_\sigma k} C_2^{\rho_{c_6} + c_\sigma c_6 - \rho_e - c_\sigma e'}, h\right)
\end{aligned}$$

$$
\begin{aligned}
&= \hat{e}\left(g_1^{\rho_{c_4}}, C_3\right) \hat{e}\left(g_0^{\rho_r} g_1^{\rho_{\gamma_1}} C_2^{\rho_{c_6}-\rho_e}, h\right) \\
&\quad \hat{e}\left(g_1^{c_\sigma c_4} C_2^{-c_\sigma}, C_3\right) \hat{e}\left(g^{c_\sigma} g_0^{c_\sigma r} g_1^{c_\sigma R + c_\sigma \gamma_1} u_0^{c_\sigma k} C_2^{c_\sigma c_6 - c_\sigma e'}, h\right) \\
&= T_O \left(\hat{e}\left(g_1^{c_4}\left(\Sigma' g_1^{c_4}\right)^{-1}, Oh^{c_6}\right) \hat{e}\left(g g_0^r g_1^{R+c_4(e'-c_6)} u_0^k \left(\Sigma' g_1^{c_4}\right)^{c_6-e'}, h\right)\right)^{c_\sigma} \\
&= T_O \left(\hat{e}\left(\Sigma', Oh^{c_6}\right)^{-1} \hat{e}\left(g g_0^r g_1^R u_0^k \Sigma'^{c_6-e'}, h\right)\right)^{c_\sigma} \\
&= T_O \left(\hat{e}\left(\Sigma', Oh^{e'}\right)^{-1} \hat{e}\left(g g_0^r g_1^R u_0^k, h\right)\right)^{c_\sigma} = T_O,
\end{aligned}
$$

da für Σ' die Gleichung $\left(\Sigma', Oh^{e'}\right) = \hat{e}\left(g g_0^r g_1^R u_0^k, h\right)$ gilt.

Spezielle Korrektheit: Der Knowledge-Extractor $\mathcal{E}$ erhält die Parameter $(\mathsf{sp}, C_1, C_2, C_3)$, die Commitments (T_{C_1}, T_{1_C}, T_O) sowie die beiden Challenge-Response Tupel $(c_\sigma, z_{c_4}, z_{c_5}, z_{c_6}, z_{\gamma_1}, z_{\gamma_2}, z_{\gamma_3}, z_e, z_r)$ und $\left(c'_\sigma, z'_{c_4}, z'_{c_5}, z'_{c_6}, z'_{\gamma_1}, z'_{\gamma_2}, z'_{\gamma_3}, z'_e, z'_r\right)$ mit

$$
\begin{aligned}
T_{C_1} &= g_0^{z_{c_4}} g_1^{z_{c_5}} u_0^{z_{c_6}} C_1^{-c_\sigma} = g_0^{z'_{c_4}} g_1^{z'_{c_5}} u_0^{z'_{c_6}} C_1^{-c'_\sigma}, \\
T_{1_C} &= C_1^{z_e - z_{c_6}} g_0^{-z_{\gamma_1}} g_1^{-z_{\gamma_2}} u_0^{-z_{\gamma_3}} = C_1^{z'_e - z'_{c_6}} g_0^{-z'_{\gamma_1}} g_1^{-z'_{\gamma_2}} u_0^{-z'_{\gamma_3}}, \\
T_O &= \hat{e}\left(g_1^{z_{c_4}} C_2^{-c_\sigma}, C_3\right) \hat{e}\left(g^{c_\sigma} g_0^{z_r} g_1^{c_\sigma R + z_{\gamma_1}} u_0^{c_\sigma k} C_2^{z_{c_6} - z_e}, h\right) \\
&= \hat{e}\left(g_1^{z'_{c_4}} C_2^{-c'_\sigma}, C_3\right) \hat{e}\left(g^{c'_\sigma} g_0^{z'_r} g_1^{c'_\sigma R + z'_{\gamma_1}} u_0^{c'_\sigma k} C_2^{z'_{c_6} - z'_e}, h\right).
\end{aligned}
$$

Daraus folgt:

$$
C_1 = g_0^{\frac{z_{c_4}-z'_{c_4}}{c_\sigma - c'_\sigma}} g_1^{\frac{z_{c_5}-z'_{c_5}}{c_\sigma - c'_\sigma}} u_0^{\frac{z_{c_6}-z'_{c_6}}{c_\sigma - c'_\sigma}} = g_0^{\frac{z_{\gamma_1}-z'_{\gamma_1}}{z_e - z_{c_6} - \left(z'_e - z'_{c_6}\right)}} g_1^{\frac{z_{\gamma_2}-z'_{\gamma_2}}{z_e - z_{c_6} - \left(z'_e - z'_{c_6}\right)}} u_0^{\frac{z_{\gamma_3}-z'_{\gamma_3}}{z_e - z_{c_6} - \left(z'_e - z'_{c_6}\right)}},
$$

$$
\frac{\hat{e}\left(C_2, C_3\right)}{\hat{e}\left(g g_1^R u_0^k, h\right)} = \hat{e}\left(g_1^{\frac{z_{c_4}-z'_{c_4}}{c_\sigma - c'_\sigma}}, C_3\right) \hat{e}\left(g_0^{\frac{z_r - z'_r}{c_\sigma - c'_\sigma}} g_1^{\frac{z_{\gamma_1}-z'_{\gamma_1}}{c_\sigma - c'_\sigma}} C_2^{\frac{z_{c_6}-z'_{c_6}}{c_\sigma - c'_\sigma} - \frac{z_e - z'_e}{c_\sigma - c'_\sigma}}, h\right).
$$

Daher kann der Knowledge-Extractor $\mathcal{E}$ die folgenden Zahlen in $\mathbb{Z}_p$ berechnen:

$$
\begin{aligned}
c_4 &= \frac{z_{c_4} - z'_{c_4}}{c_\sigma - c'_\sigma}, & c_5 &= \frac{z_{c_5} - z'_{c_5}}{c_\sigma - c'_\sigma}, & c_6 &= \frac{z_{c_6} - z'_{c_6}}{c_\sigma - c'_\sigma}, \\
\gamma_1 &= \frac{z_{\gamma_1} - z'_{\gamma_1}}{c_\sigma - c'_\sigma}, & \gamma_2 &= \frac{z_{\gamma_2} - z'_{\gamma_2}}{c_\sigma - c'_\sigma}, & \gamma_3 &= \frac{z_{\gamma_3} - z'_{\gamma_3}}{c_\sigma - c'_\sigma}, \\
e' &= \frac{z_e - z'_e}{c_\sigma - c'_\sigma}, & r &= \frac{z_r - z'_r}{c_\sigma - c'_\sigma}.
\end{aligned}
$$

Somit gilt:

$$
C_1 = g_0^{c_4} g_1^{c_5} u_0^{c_6} = g_0^{\frac{\gamma_1}{e'-c_6}} g_1^{\frac{\gamma_2}{e'-c_6}} u_0^{\frac{\gamma_3}{e'-c_6}},
$$

$$\frac{\hat{e}\left(C_2, C_3\right)}{\hat{e}\left(gg_1^R u_0^k, h\right)} = \hat{e}\left(g_1^{c_4}, C_3\right) \hat{e}\left(g_0^r g_1^{\gamma_1} C_2^{c_6 - e'}, h\right).$$

Special-Honest-Verifier-Zero-Knowledge: Der Simulator $\mathcal{SIM}$ erhält die Parameter $(\mathsf{sp}, C_1, C_2, C_3)$ sowie die Challenge c_σ und simuliert die Signature-of-Knowledge folgendermaßen:

Der Simulator $\mathcal{SIM}$ wählt zufällig die Responses $z_{c_1}, z_{c_2}, z_{c_3}, z_{\gamma_1}, z_{\gamma_2}, z_{\gamma_3}, z_e, z_r \in_{\mathcal{R}} \mathbb{Z}_p$ und berechnet die Commitments

$$\begin{aligned}
T_{C_1} &= g_0^{z_{c_4}} g_1^{z_{c_5}} u_0^{z_{c_6}} C_1^{-c_\sigma}, \\
T_{1_C} &= C_1^{z_e - z_{c_6}} g_0^{-z_{\gamma_1}} g_1^{-z_{\gamma_2}} u_0^{-z_{\gamma_3}}, \\
T_O &= \hat{e}\left(g_1^{z_{c_4}} C_2^{-c_\sigma}, C_3\right) \hat{e}\left(g^{c_\sigma} g_0^{z_r} g_1^{c_\sigma R + z_{\gamma_1}} u_0^{c_\sigma k} C_2^{z_{c_6} - z_e}, h\right).
\end{aligned}$$

Somit ist die Verifikation von $\mathcal{V}$ trivialer Weise erfüllt. Weiter sind alle Responses unabhängig und identisch verteilt und somit besitzen auch alle Commitments die gleiche Verteilung wie im realen Protokoll. Insgesamt hat $\mathcal{SIM}$ die Signature-of-Knowledge perfekt simuliert.

□

LITERATURVERZEICHNIS

[ACHM05] Ateniese, Giuseppe ; Camenisch, Jan ; Hohenberger, Susan ; Medeiros, Breno de: *Practical Group Signatures without Random Oracles*. Cryptology ePrint Archive, Report 2005/385. `http://eprint.iacr.org/2005/385`. Version: Oktober 2005

[ACM05] Ateniese, Giuseppe ; Camenisch, Jan ; Medeiros, Breno de: Untraceable RFID tags via Insubvertible Encryption. In: *12th ACM Conference on Computer and Communications Security – CCS '05*, ACM, 2005 (CCS), S. 92–101

[ADR02] An, Jee H. ; Dodis, Yevgeniy ; Rabin, Tal: On the Security of Joint Signature and Encryption. In: *Advances in Cryptology – EUROCRYPT '02* Bd. 2332, Springer, 2002 (Lecture Notes in Computer Science), S. 83–107

[ASM06] Au, Man H. ; Susilo, Willy ; Mu, Yi: Constant-Size Dynamic k-TAA. In: *Security and Cryptography for Networks – SCN '06* Bd. 4116, Springer, 2006 (Lecture Notes in Computer Science), S. 111–125

[ASM07] Au, Man H. ; Susilo, Willy ; Mu, Yi: Practical Compact E-Cash. In: *Information Security and Privacy – ACISP '07* Bd. 4586, Springer, 2007 (Lecture Notes in Computer Science), S. 431–445

[ASM08] Au, Man H. ; Susilo, Willy ; Mu, Yi: Practical Anonymous Divisible E-Cash From Bounded Accumulators. In: *Financial Cryptography and Data Security – FC '08* Bd. 5143, Springer, 2008 (Lecture Notes in Computer Science), S. 287–301

[ATSM09] Au, Man H. ; Tsang, Patrick P. ; Susilo, Willy ; Mu, Yi: Dynamic Universal Accumulators for DDH Groups and Their Application to Attribute-Based Anonymous Credential Systems. In: *Topics in Cryptology – CT-RSA*

'09 Bd. 5473, Springer, 2009 (Lecture Notes in Computer Science), S. 295–308

[Au09] Au, Man H.: *Contribution to Privacy-Preserving Cryptographic Techniques*, School of Computer Science and Software Engineering, University of Wollongong, Diss., Mai 2009

[AWSM07] Au, Man H. ; Wu, Qianhong ; Susilo, Willy ; Mu, Yi: Compact E-Cash from Bounded Accumulator. In: *Topics in Cryptology – CT-RSA '07* Bd. 4377, Springer, 2007 (Lecture Notes in Computer Science), S. 178–195

[BB04] Boneh, Dan ; Boyen, Xavier: Short Signatures Without Random Oracles. In: *Advances in Cryptology – EUROCRYPT '04* Bd. 3027, Springer, 2004 (Lecture Notes in Computer Science), S. 56–73

[BB08] Boneh, Dan ; Boyen, Xavier: Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups. In: *Journal of Cryptology* 21 (2008), Nr. 2, S. 149–177

[BBS04] Boneh, Dan ; Boyen, Xavier ; Shacham, Hovav: Short Group Signatures. In: *Advances in Cryptology – CRYPTO '04* Bd. 3152, Springer, 2004 (Lecture Notes in Computer Science), S. 41–55

[BCKL09] Belenkiy, Mira ; Chase, Melissa ; Kohlweiss, Markulf ; Lysyanskaya, Anna: Compact E-Cash and Simulatable VRFs Revisited. In: *Pairing-Based Cryptography – Pairing '09* Bd. 5671, Springer, 2009 (Lecture Notes in Computer Science), S. 114–131

[BF01] Boneh, Dan ; Franklin, Matthew: Identity-Based Encryption from the Weil Pairing. In: *Advances in Cryptology – CRYPTO '01* Bd. 2139, Springer, 2001 (Lecture Notes in Computer Science), S. 213–229

[BF03] Boneh, Dan ; Franklin, Matthew: Identity-Based Encryption from the Weil Pairing. In: *SIAM Journal on Computing* 32 (2003), Nr. 3, S. 586–615

[BG92] Bellare, Mihir ; Goldreich, Oded: On Defining Proofs of Knowledge. In: *Advances in Cryptology – CRYPTO '92* Bd. 740, Springer, 1992 (Lecture Notes in Computer Science), S. 390–420

[BGK95] Brickell, Ernie ; Gemmell, Peter ; Kravitz, David: Trustee-based Tracing Extensions to Anonymous Cash and the Making of Anonymous Change. In: *Sixth annual ACM-SIAM Symposium on Discrete Algorithms – SODA '95*, ACM, 1995 (SODA), S. 457–466

[BGLS03] Boneh, Dan ; Gentry, Craig ; Lynn, Ben ; Shacham, Hovav: Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In: *Advances in Cryptology – EUROCRYPT '03* Bd. 2656, Springer, 2003 (Lecture Notes in Computer Science), S. 416–432

[BGMM05] Ballard, Lucas ; Green, Matthew ; Medeiros, Breno de ; Monrose, Fabian: *Correlation-Resistant Storage via Keyword-Searchable Encryption.* Cryptology ePrint Archive, Report 2005/417. `http://eprint.iacr.org/2005/417`. Version: November 2005

[BIT07a] BITKOM: *EU-Vergleich: Deutschlands Jugendliche beim Online-Shopping auf Platz 1.* `http://www.bitkom.org/files/documents/BITKOM_PI_Online-Shopping_Jugendliche_22.05.2007.pdf`. Version: Mai 2007

[BIT07b] BITKOM: *Online-Shopping: Die meisten zahlen per Lastschrift.* `http://www.bitkom.org/files/documents/bitkom-pi_online-bezahlen_04.10.2007.pdf`. Version: Oktober 2007

[BIT08] BITKOM: *Deutsche kaufen gerne im Internet ein.* `http://www.bitkom.org/files/documents/BITKOM_Presseinfo_Online-Shopping_24_01_2008.pdf`. Version: Januar 2008

[BIT09a] BITKOM: *Im Internet kaufen Verbraucher am liebsten Fahrkarten.* `http://www.bitkom.org/files/documents/BITKOM_Presseinfo_Online-Shopping_27_07_2009_v3.pdf`. Version: Juli 2009

[BIT09b] BITKOM: *Sicherheitsbedenken verhindern Transaktionen im Web.* `http://www.bitkom.org/files/documents/PI_BITKOM_Sicherheit_03_03_2009_V8_Headline_Verzicht_auf_Transaktionen.pdf`. Version: März 2009

[BIT10] BITKOM: *Sechs von zehn Deutschen kaufen im Internet ein.* `http://www.bitkom.org/files/documents/BITKOM_Presseinfo_Online-Shopping.pdf`. Version: Mai 2010

[BIT11] BITKOM: *Jeder zweite shoppt im Web.* `http://www.bitkom.org/files/documents/BITKOM_Presseinfo_Online-Shopping_Bezahlverfahren.pdf`. Version: Mai 2011

[BIT14] BITKOM: *51 Millionen Deutsche kaufen Waren im Internet.* `http://www.bitkom.org/files/documents/BITKOM_Presseinfo_Trends_im_Online-Shopping_08_05_2014.pdf`. Version: Mai 2014

[BLS01] Boneh, Dan ; Lynn, Ben ; Shacham, Hovav: Short Signatures from the Weil Paring. In: *Advances in Cryptology – ASIACRYPT '01* Bd. 2248, Springer, 2001 (Lecture Notes in Computer Science), S. 514–532

[BLS04] Boneh, Dan ; Lynn, Ben ; Shacham, Hovav: Short Signatures from the Weil Pairing. In: *Journal of Cryptology* 17 (2004), Nr. 4, S. 297–319

[Blu83] Blum, Manuel: Coin Flipping By Telephone a Protocol for Solving Impossible Problems. In: *ACM SIGACT News* 15 (1983), Nr. 1, S. 23–27

[BM94] Benaloh, Josh ; Mare, Michael de: One-Way Accumulators: A Decentralized Alternative to Digital Signatures (Extended Abstract). In: *Advances in Cryptology – EUROCRYPT '93* Bd. 765, Springer, 1994 (Lecture Notes in Computer Science), S. 274–285

[BN06] Barreto, Paulo ; Naehrig, Michael: Pairing-Friendly Elliptic Curves of Prime Order. In: *Selected Areas in Cryptography – SAC '05* Bd. 3897, Springer, 2006 (Lecture Notes in Computer Science), S. 319–331

[BNS10] Beutelspacher, Albrecht ; Neumann, Heike ; Schwarzpaul, Thomas: *Kryptografie in Theorie und Praxis*. Vieweg + Teubner, 2010

[Bon98] Boneh, Dan: The Decision Diffie-Hellman Problem. In: *Third Algorithmic Number Theory Symposium* Bd. 1423, Springer, 1998 (Lecture Notes in Computer Science), S. 48–63

[Bou00] Boudot, Fabrice: Efficient Proofs that a Committed Number Lies in an Interval. In: *Advances in Cryptology – EUROCRYPT '00* Bd. 1807, Springer, 2000 (Lecture Notes in Computer Science), S. 431–444

[BP97] Barić, Niko ; Pfitzmann, Birgit: Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees. In: *Advances in Cryptology – EUROCRYPT '97* Bd. 1233, Sp, 1997 (Lecture Notes in Computer Science), S. 480–494

[BR93] Bellare, Mihir ; Rogaway, Phillip: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In: *1st ACM Conference on Computer and Communications Security – CCS '93*, ACM, 1993 (CCS), S. 62–73

[Bra93] Brands, Stefan: An Efficient Off-line Electronic Cash System Based On The Representation Problem / Centrum voor Wiskunde en Informatica. 1993. – Forschungsbericht. – Technical Report

[Bra94] Brands, Stefan: Untraceable Off-line Cash in Wallets with Observers. In: *Advances in Cryptology – CRYPTO '93* Bd. 773, Springer, 1994 (Lecture Notes in Computer Science), S. 302–318

[BS04] Boneh, Dan ; Shacham, Hovav: Group Signatures with Verifier-Local Revocation. In: *11th ACM Conference on Computer and Communications Security – CCS '04*, ACM, 2004 (CCM), S. 168–177

[BSS05] Blake, Ian (Hrsg.) ; Seroussi, Gadiel (Hrsg.) ; Smart, Nigel (Hrsg.): *Advances in Elliptic Curve Cryptography*. Cambridge University Press, 2005 (London Mathematical Society Lecture Note Series 317)

[BSW06] Beutelspacher, Albrecht ; Schwenk, Jörg ; Wolfenstetter, Klaus-Dieter: *Moderne Verfahren der Kryptographie: von RSA zu Zero-Knowledge*. Vieweg, 2006

[Cam98] Camenisch, Jan: *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem*, Swiss Federal Institute of Technology Zürich, Diss., February 1998

[Can13] Canard, Sébastien: -. Oktober 2013. – Persönliche Kommunikation

[CCS07] Chen, Liqun ; Cheng, Zhaohui ; Smart, Nigel: Identity-based key agreement protocols from pairings. In: *International Journal of Information Security* 6 (2007), Nr. 4, S. 213–241

[CDG+09] Canard, Sébastien ; Delerablée, Cécile ; Gouget, Aline ; Hufschmitt, Emeline ; Laguillaumie, Fabien ; Sibert, Hervé ; Traoré, Jacques ; Vergnaud, Damien: Fair E-cash: Be Compact, Spend Faster. In: *Information Security – ISC '09* Bd. 5735, Springer, 2009 (Lecture Notes in Computer Science), S. 294–309

[CDM00] Cramer, Roland ; Damgård, Ivan ; MacKenzie, Philip: Efficient Zero-Knowledge Proofs of Knowledge Without Intractability Assumptions. In: *Public Key Cryptography – PKC '00* Bd. 1751, Springer, 2000 (Lecture Notes in Computer Science), S. 354–371

[CDS94] Cramer, Roland ; Damgård, Ivan ; Schoenmakers, Berry: Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In: *Advances in Cryptology – CRYPTO '94* Bd. 839, Springer, 1994 (Lecture Notes in Computer Science), S. 174–187

[CFN89] Chaum, David ; Fiat, Amos ; Naor, Moni: Untraceable Electronic Cash. In: *Advances in Cryptology – CRYPTO '88* Bd. 403, Springer, 1989 (Lecture Notes in Computer Science), S. 319–327

[CFT98] Chan, Agnes ; Frankel, Yair ; Tsiounis, Yiannis: Easy Come - Easy Go Divisible Cash. In: *Advances in Cryptology – EUROCRYPT '98* Bd. 1403, Springer, 1998 (Lecture Notes in Computer Science), S. 561–575

[CG07] Canard, Sébastien ; Gouget, Aline: Divisible E-cash Systems can be Truly Anonymous. In: *Advances in Cryptology – EUROCRYPT '07* Bd. 4514, Springer, 2007 (Lecture Notes in Computer Science), S. 482–497

[CG08] Canard, Sébastien ; Gouget, Aline: Anonymity in Transferable E-cash. In: *Applied Cryptography and Network Security – ACNS '08* Bd. 5037, Springer, 2008 (Lecture Notes in Computer Science), S. 207–223

[CG10] Canard, Sébastien ; Gouget, Aline: Multiple Denominations in E-cash with Compact Transaction Data. In: *Financial Cryptography and Data Security – FC '10* Bd. 6052, Springer, 2010 (Lecture Notes in Computer Science), S. 82–97

[CGH98] Canetti, Ran ; Goldreich, Oded ; Halevi, Shai: The Random Oracle Methodology, Revisited (preliminary version). In: *Thirtieth annual ACM Symposium on Theory of Computing – STOC '98*, ACM, 1998 (STOC), S. 209–218

[CGH04] Canetti, Ran ; Goldreich, Oded ; Halevi, Shai: The Random Oracle Methodology, Revisited. In: *Journal ACM* 51 (2004), Nr. 4, S. 557–594

[CH10] Camacho, Philippe ; Hevia, Alejandro: On the Impossibility of Batch Update for Cryptographic Accumulators. In: *Progress in Cryptology - LATINCRYPT '10* Bd. 6212, Springer, 2010 (Lecture Notes in Computer Science), S. 178–188

[Cha83] Chaum, David: Blind Signatures For Untraceable Payments. In: *Advances in Cryptology – CRYPTO '82*, Plenum Press, 1983, S. 199–203

[Cha84] Chaum, David: Blind Signature System. In: *Advances in Cryptology – CRYPTO '83*, Plenum Press, 1984, S. 153

[CHKM09] Chatterjee, Sanjit ; Hankerson, Darrel ; Knapp, Edward ; Menezes, Alfred: *Comparing Two Pairing-Based Aggregate Signature Schemes*. Cryptology ePrint Archive, Report 2009/060. `http://eprint.iacr.org/2009/060`. Version: September 2009

[CHL05] Camenisch, Jan ; Hohenberger, Susan ; Lysyanskaya, Anna: Compact E-Cash. In: *Advances in Cryptology – EUROCRYPT '05* Bd. 3494, Springer, 2005 (Lecture Notes in Computer Science), S. 302–321

[CHL06a] Camenisch, Jan ; Hohenberger, Susan ; Lysyanskaya, Anna: Balancing Accountability and Privacy Using E-Cash. In: *Security and Cryptography for Networks – SCN '06* Bd. 4116, Springer, 2006 (Lecture Notes in Computer Science), S. 141–155

[CHL06b] Camenisch, Jan ; Hohenberger, Susan ; Lysyanskaya, Anna: *Compact E-Cash*. Cryptology ePrint Archive, Report 2005/060. `http://eprint.iacr.org/2005/060`. Version: März 2006

[CHM10] Chatterjee, Sanjit ; Hankerson, Darrel ; Menezes, Alfred: On the Efficiency and Security of Pairing-Based Protocols in the Type 1 and Type 4 Settings. In: *Arithmetic of Finite Fields – WAIFI '10* Bd. 6087, Springer, 2010 (Lecture Notes in Computer Science), S. 114–134

[CKOS01] Crescenzo, Giovanni D. ; Katz, Jonathan ; Ostrovsky, Rafail ; Smith, Adam: Efficient and Non-Interactive Non-Malleable Commitment. In: *Advances in Cryptology – EUROCRYPT '01* Bd. 2045, Sp, 2001 (Lecture Notes in Computer Science), S. 40–59

[CL02a] Camenisch, Jan ; Lysyanskaya, Anna: Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In: *Advances in Cryptology – CRYPTO '02* Bd. 2442, Springer, 2002 (Lecture Notes in Computer Science), S. 61–76

[CL02b] Camenisch, Jan ; Lysyanskaya, Anna: A Signature Scheme with Efficient Protocols. In: *Security in Communication Networks – SCN '02* Bd. 2576, Springer, 2002 (Lecture Notes in Computer Science), S. 268–289

[CL04] Camenisch, Jan ; Lysyanskaya, Anna: Signature Schemes and Anonymous Credentials from Bilinear Maps. In: *Advances in Cryptology – CRYPTO '04* Bd. 3152, Springer, 2004 (Lecture Notes in Computer Science), S. 56–72

[CM09] Chatterjee, Sanjit ; Menezes, Alfred: *On Cryptographic Protocols Employing Asymmetric Pairings – The Role of Ψ Revisited*. Cryptology ePrint Archive, Report 2009/480. `http://eprint.iacr.org/2009/480`. Version: September 2009

[CMS96] Camenisch, Jan ; Maurer, Ueli ; Stadler, Markus: Digital Payment Systems with Passive Anonymity-Revoking Trustees. In: *Computer Security – ESORICS '96* Bd. 1146, Springer, 1996 (Lecture Notes in Computer Science), S. 33–43

[CP93] Chaum, David ; Pedersen, Torben: Wallet Databases With Observers. In: *Advances in Cryptology – CRYPTO '92* Bd. 740, Springer, 1993 (Lecture Notes in Computer Science), S. 89–105

[CP94] Cramer, Roland ; Pedersen, Torben: Improved Privacy in Wallets with Observers. In: *Advances in Cryptology – EUROCRYPT '93* Bd. 765, Springer, 1994 (Lecture Notes in Computer Science), S. 329–343

[CPS94] Camenisch, Jan ; Pivetau, Jean-Marc ; Stadler, Markus: An Efficient Electronic Payment System Protecting Privacy. In: *Computer Security – ESORICS '94* Bd. 875, Springer, 1994 (Lecture Notes in Computer Science), S. 205–215

[CPS96] Camenisch, Jan ; Pivetau, Jean-Marc ; Stadler, Markus: An Efficient Fair Payment System. In: *3rd ACM Conference on Computer and Communications Security – CCS '96*, ACM, 1996 (CCS), S. 88–94

[Cra96] Cramer, Roland: *Modular Design of Secure yet Practical Cryptographic Protocols*, Centrum Wiskunde & Informatica and University of Amsterdam, Diss., November 1996

[CS97] Camenisch, Jan ; Stadler, Markus: Efficient Group Signature Schemes for Large Groups. In: *Advances in Cryptology – CRYPTO '97* Bd. 1294, Springer, 1997 (Lecture Notes in Computer Science), S. 410–424

[CV05] Catalano, Dario ; Visconti, Ivan: Hybrid Commitments and Their Applications. In: *Automata, Languages and Programming – ICALP '05* Bd. 3580, Springer, 2005 (Lecture Notes in Computer Science), S. 298–310

[Dam88] Damgård, Ivan: Collision Free Hash Functions and Public Key Signature Schemes. In: *Advances in Cryptology – EUROCRYPT '87* Bd. 304, Springer, 1988 (Lecture Notes in Computer Science), S. 203–216

[Dam90] Damgård, Ivan: On the Existence of Bit Commitment Schemes and Zero-Knowledge Proofs. In: *Advances in Cryptology – CRYPTO '89* Bd. 435, Springer, 1990 (Lecture Notes in Computer Science), S. 17–27

[Dam99] Damgård, Ivan: Commitment Schemes and Zero-Knowledge Protocols. In: *Lectures on Data Security: Modern Cryptology in Theory and Practice* Bd. 1561, Springer, 1999 (Lecture Notes in Computer Science), S. 63–86

[Dam11] Damgård, Ivan: *On Σ-protocols*. Cryptologic Protocol Theory. `https://services.brics.dk/java/courseadmin/CPT/documents/getDocument/Sigma.pdf?d=53899`. Version: 2011

[DDN91] Dolev, Danny ; Dwork, Cynthia ; Naor, Moni: Non-Malleable Cryptography. In: *Twenty-third annual ACM Symposium on Theory of Computing – STOC '91*, ACM, 1991 (STOC), S. 542–552

[DH76] Diffie, Whitfield ; Hellman, Martin: New Directions in Cryptography. In: *IEEE Transactions on Information Theory* 22 (1976), Nr. 6, S. 644–654

[DN02] Damgård, Ivan ; Nielsen, Jesper: Perfect Hiding and Perfect Binding Universally Composable Commitment Schemes with Constant Expansion Factor. In: *Advances in Cryptology – CRYPTO '02* Bd. 2442, Springer, 2002 (Lecture Notes in Computer Science), S. 3–42

[DY05] Dodis, Yevgeniy ; Yampolskiy, Aleksandr: A Verifiable Random Function With Short Proofs and Keys. In: *Public-Key Cryptography – PKC '05* Bd. 3386, Springer, 2005 (Lecture Notes in Computer Science), S. 416–431

[ElG85a] ElGamal, Taher: A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms. In: *Advances in Cryptology – CRYPTO '84* Bd. 196, Springer, 1985 (Lecture Notes in Computer Science), S. 10–18

[ElG85b] ElGamal, Taher: A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms. In: *IEEE Transactions on Information Theory* 31 (1985), Nr. 4, S. 469–472

[EO94] Eng, Tony ; Okamoto, Tatsuaki: Single-Term Divisible Electronic Coins. In: *Advances in Cryptology – EUROCRYPT '94* Bd. 950, Springer, 1994 (Lecture Notes in Computer Science), S. 306–319

[FS86] Fiat, Amos ; Shamir, Adi: How to Prove Yourself: Practical Solutions of Identification and Signature Problems. In: *Advances in Cryptology – CRYPTO '86* Bd. 263, Springer, 1986 (Lecture Notes in Computer Science), S. 186–194

[FS89] Feige, Uriel ; Shamir, Adi: Zero Knowledge Proofs of Knowledge in Two Rounds. In: *Advances in Cryptology – CRYPTO '89* Bd. 435, Springer, 1989 (Lecture Notes in Computer Science), S. 526–544

[FTY96] Frankel, Yair ; Tsiounis, Yiannis ; Yung, Moti: Indirect Discourse Proofs: Achieving Efficient Fair Off-Line E-Cash. In: *Advances in Cryptology – ASIACRYPT '96* Bd. 1163, Springer, 1996 (Lecture Notes in Computer Science), S. 286–300

[FTY98] Frankel, Yair ; Tsiounis, Yiannis ; Yung, Moti: Fair Off-Line e-Cash made easy. In: *Advances in Cryptology – ASIACRYPT '98* Bd. 1514, Springer, 1998 (Lecture Notes in Computer Science), S. 257–270

[Gal05] Galindo, David: Boneh-Franklin Identity Based Encryption Revisited. In: *Automata, Language and Programming – ICALP '05* Bd. 3580, Springer, 2005 (Lecture Notes in Computer Science), S. 791–802

[GFK09] GFK: *GfK-Studie: Privater E-Commerce-Umsatz in Deutschland wächst weiter*. `http://www.gfk.com/group/press_information/press_releases/003717/index.de.html`. Version: 2009

[GM82] Goldwasser, Shafi ; Micali, Silvio: Probabilistic Encryption & How To Play Mental Poker Keeping Secret All Partial Information. In: *Fourteenth annual ACM Symposium on Theory of Computing – STOC '82*, ACM, 1982 (STOC), S. 365–377

[GM84] Goldwasser, Shafi ; Micali, Silvio: Probabilistic Encryption. In: *Journal of Computer and System Science* 28 (1984), Nr. 2, S. 207–299

[GMR84] Goldwasser, Shafi ; Micali, Silvio ; Rivest, Ronald: A "Paradoxical" Solution to the Signature Problem (Extended Abstract). In: *25th Annual IEEE Symposium on Foundations of Computer Science – FOCS '84*, IEEE Computer Society, 1984 (FOCS), S. 441–448

[GMR85] Goldwasser, Shafi ; Micali, Silvio ; Rackoff, Charles: The Knowledge Complexity of Interactive Proof-Systems. In: *Seventeenth annual ACM Symposium on Theory of Computing – STOC '85*, ACM, 1985 (STOC), S. 291–304

[GMR88] Goldwasser, Shafi ; Micali, Silvio ; Rivest, Ronald: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. In: *SIAM Journal on Computing* 17 (1988), Nr. 2, S. 281–308

[GMR89] Goldwasser, Shafi ; Micali, Silvio ; Rackoff, Charles: The Knowledge Complexity of Interactive Proof Systems. In: *SIAM Journal on Computing* 18 (1989), Nr. 1, S. 186–208

[GMW87] Goldreich, Oded ; Micali, Silvio ; Wigderson, Avi: How to Play ANY Mental Game. In: *Nineteenth annual ACM Symposium on Theory and Computing – STOC '87*, ACM, 1987 (STOC), S. 218–229

[GMW91] Goldreich, Oded ; Micali, Silvio ; Wigderson, Avi: Proofs that Yield Nothing But Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems. In: *Journal ACM* 38 (1991), Nr. 3, S. 690–730

[Gol01] Goldreich, Oded: *Foundations of Cryptography - Volume 1 (Basic Tools)*. Cambridge University Press, 2001

[Gol04] Goldreich, Oded: *Foundations of Cryptography - Volume 2 (Basic Applications)*. Cambridge University Press, 2004

[GPS06] Galbraith, Steven ; Paterson, Kenneth ; Smart, Nigel: *Pairings for Cryptographers*. Cryptology ePrint Archive, Report 2006/165. `http://eprint.iacr.org/2006/165`. Version: Mai 2006

[GPS08] Galbraith, Steven ; Paterson, Kenneth ; Smart, Nigel: Pairings for Cryptographers. In: *Discrete Applied Mathematics* 156 (2008), Nr. 16, S. 3113–3121

[GR04] Galbraith, Steven ; Rotger, Victor: Easy Decision-Diffie-Hellman Groups. In: *LMS Journal of Computation and Mathematics* 7 (2004), S. 201–218

[GT03] Gaud, Matthieu ; Traoré, Jacques: On the Anonymity of Fair Offline E-Cash Systems. In: *Financial Cryptography and Data Security – FC '03* Bd. 2742, Springer, 2003 (Lecture Notes in Computer Science), S. 34–50

[Hoh06] Hohenberger, Susan: *Advances in Signatures, Encryption, and E-Cash from Bilinear Groups*, The Department of Electrical Engineering and Computer Science, Massachusetts Insitute of Technology, Diss., Mai 2006

[HPS08] Hoffstein, Jeffrey ; Pipher, Jill ; Silverman, Joseph: *An Introduction to Mathematical Cryptography*. Springer, 2008

[HSV06] Hess, Florian ; Smart, Nigel ; Vercauteren, Frederik: The Eta Pairing Revisited. In: *IEEE Transactions on Information Theory* 52 (2006), Nr. 10, S. 4595–4602

[HTY99] Handschuh, Helena ; Tsiounis, Yiannis ; Yung, Moti: Decision Oracles Are Equivalent to Matching Oracles. In: *Public-Key Cryptography – PKC '99* Bd. 1560, 1999 (Lecture Notes in Computer Science), S. 276–289

[HU94] Hopcroft, John E. ; Ullman, Jeffrey D.: *Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie*. Addison Wesley, 1994

[IL13] Izabacène, Malika ; Libert, Benoît: Divisible E-Cash in the Standard Model. In: *Advances in Cryptology – Pairing '12* Bd. 7708, Springer, 2013 (Lecture Notes in Computer Science), S. 314–322

[JMV01] Johnson, Don ; Menezes, Alfred ; Vanstone, Scott: The Elliptic Curve Digital Signature Algorithm (ECDSA). In: *International Journal of Information Security* 1 (2001), Nr. 1, S. 36–63

[JN01] Joux, Antoine ; Nguyen, Kim: *Separating Decision Diffie-Hellman from Diffie-Hellman in cryptographic groups*. Cryptology ePrint Archive, Report 2001/003. `http://eprint.iacr.org/2001/003`. Version: Januar 2001

[KLL06] Käsper, Emilia ; Laur, Sven ; Lipmaa, Helger: *Black-Box Knowledge Extraction Revisited*. Cryptology ePrint Archive, Report 2006/356. `http://eprint.iacr.org/2006/356`. Version: Oktober 2006

[Kun91] Kunz, Ernst: *Algebra*. Vieweg Studium, 1991

[KV02] Kügler, Dennis ; Vogt, Holger: Fair Tracing without Trustees. In: *Financial Cryptography – FC '01* Bd. 2339, Springer, 2002 (Lecture Notes in Computer Science), S. 136–148

[KZG10a] Kate, Aniket ; Zaverucha, Gregory ; Goldberg, Ian: Constant-Size Commitments to Polynomials and Their Applications. In: *Advances in Cryptology – ASIACRYPT '10* Bd. 6477, Springer, 2010 (Lecture Notes in Computer Science), S. 177–194

[KZG10b] Kate, Aniket ; Zaverucha, Gregory ; Goldberg, Ian: Polynomial Commitments / Centre for Applied Cryptographic Research, University of Waterloo. 2010. – Forschungsbericht

[LLP09] Lee, Eunjeong ; Lee, Hyang-Sook ; Park, Cheol-Min: Efficient and Generalized Pairing Computation on Abelian Varieties. In: *IEEE Transactions on Information Theory* 55 (2009), Nr. 4, S. 1793–1803

[LTW05] Liu, Joseph K. ; Tsang, Patrick P. ; Wong, Duncan S.: Recoverable and Untraceable E-Cash. In: *Public Key Infrastructure – PKI '05* Bd. 3545, Springer, 2005 (Lecture Notes in Computer Science), S. 206–214

[Lyn07] Lynn, Ben: *On the Implementation of Pairing-Based Cryptosystems*, The Department of Computer Science, Stanford University, Diss., Juni 2007

[Mao03] Mao, Wenbo: *Modern Cryptography: Theory and Practice*. Prentice Hall Professional Technical Reference, 2003

[Mau94] Maurer, Ueli: Towards the Equivalence of Breaking the Diffie-Hellman Protocol and Computing Discrete Logarithms. In: *Advances in Cryptology – CRYPTO '94* Bd. 839, Springer, 1994 (Lecture Notes in Computer Science), S. 271–281

[Mil85] Miller, Victor S.: Use of Elliptic Curves in Cryptography. In: *Advances in Cryptology – CRYPTO '85* Bd. 218, Springer, 1985 (Lecture Notes in Computer Science), S. 417–426

[MRS88] Micali, Silvio ; Rackoff, Charles ; Sloan, Bob: The Notion of Security for Probabilistic Cryptosystems. In: *SIAM Journal on Computing* 17 (1988), Nr. 2, S. 412–426

[MW99] Maurer, Ueli ; Wolf, Stefan: The Relationship Between Breaking the Diffie–Hellman Protocol and Computing Discrete Logarithms. In: *SIAM Journal on Computing* 28 (1999), Nr. 4, S. 1689–1721

[Ngu05] Nguyen, Lan: Accumulators from Bilinear Pairings and Applications to ID-based Ring Signatures and Group Membership Revocation. In: *Topics in Cryptology – CT-RSA '05* Bd. 3376, Springer, 2005 (Lecture Notes in Computer Science), S. 275–292

[Ngu06] Nguyen, Lan: Privacy-Protecting Coupon System Revisited. In: *Financial Cryptography and Data Security – FC '06* Bd. 4107, Springer, 2006 (Lecture Notes in Computer Science), S. 266–280

[NS00] Nakanishi, Toru ; Sugiyama, Yuji: Unlinkable Divisible Electronic Cash. In: *Information Security – ISW '00* Bd. 1975, Springer, 2000 (Lecture Notes in Computer Science), S. 121–134

[NS03] Neumann, Heike ; Schwarzpaul, Thomas: Fairness in Wallets with Observer. In: *14th International Workshop on Database and Expert Systems Applications – DEXA '03*, IEEE Computer Society, 2003 (DEXA). – ISBN 0–7695–1993–8, S. 364–368

[NS06] Neumann, Heike ; Schwarzpaul, Thomas: Digital Coins: Fairness Implemented by Observer. In: *Journal of Theoretical and Applied Electronic Commerce Research* 1 (2006), Nr. 1, S. 1–15

[NSA09] NSA: *The Case for Elliptic Curve Cryptography*. `http://www.nsa.gov/business/programs/elliptic_curve.shtml`. Version: Januar 2009

[NSS02] Nakanishi, Toru ; Shiota, Mitsuaki ; Sugiyama, Yuji: An Unlinkable Divisible Electronic Cash with User's Less Computations Using Active Trustees. In: *International Symposium on Information Theory and Its Applications*, 2002

[NY89] Naor, Moni ; Yung, Moti: Universal One-Way Hash Functions and their Cryptographic Applications. In: *Twenty-first annual ACM Symposium on Theory of Computing – STOC '89*, ACM, 1989 (STOC), S. 33–43

[NY90] Naor, Moni ; Yung, Moti: Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attacks. In: *Twenty-second annual ACM Symposium on Theory of Computing – STOC '90*, ACM, 1990 (STOC), S. 427–437

[Oka95] Okamoto, Tatsuaki: An Efficient Divisible Electronic Cash System. In: *Advances in Cryptology – CRYPTO '95* Bd. 963, Springer, 1995 (Lecture Notes in Computer Science), S. 438–451

[OO91] Okamoto, Tatsuaki ; Ohta, Kazuo: Universal Electronic Cash. In: *Advances in Cryptology – CRYPTO '91* Bd. 576, Springer, 1991 (Lecture Notes in Computer Science), S. 324–337

[Ped91] Pedersen, Torben: Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In: *Advances in Cryptology – CRYPTO '91* Bd. 576, Springer, 1991 (Lecture Notes in Computer Science), S. 129–140

[PS96] Pointcheval, David ; Stern, Jacques: Security Proofs for Signature Schemes. In: *Advances in Cryptology – EUROCRYPT '96* Bd. 1070, Springer, 1996 (Lecture Notes in Computer Science), S. 387–398

[PS00] Pointcheval, David ; Stern, Jacques: Security Arguments for Digital Signatures and Blind Signatures. In: *Journal Of Cryptology* 13 (2000), Nr. 3, S. 361–396

[PW91] Pfitzmann, Birgit ; Waidner, Michael: How to Break and Repair a „Provably Secure" Untraceable Payment System. In: *Advances in Cryptology – CRYPTO '91* Bd. 576, Springer, 1991 (Lecture Notes in Computer Science), S. 338–350

[Rei99] Reischuk, K. R.: *Komplexitätstheorie Band 1: Grundlagen*. B. G. Teubner, 1999

[RSA78] Rivest, Ronald ; Shamir, Adi ; Adleman, Leonard: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. In: *Communications of the ACM* 21 (1978), Nr. 2, S. 120–126

[SB04] Scott, Michael ; Barreto, Paulo: Compressed Pairings. In: *Advances in Cryptology – CRYPTO '04* Bd. 3152, Springer, 2004 (Lecture Notes in Computer Science), S. 140–156

[Sch91] Schnorr, Claus P.: Efficient Signature Generation by Smart Cards. In: *Journal of Cryptology* 4 (1991), Nr. 3, S. 161–174

[Sch09] Schwarzpaul, Thomas: *Langfristig sichere elektronische Geldsysteme mit Observern*, Mathematisches Institut, Justus-Liebig-Universität Gießen, Diss., März 2009

[Sch11] Schäge, Sven: Tight Proofs for Signature Schemes without Random Oracles. In: *Advances in Cryptology – EUROCRYPT '11* Bd. 6632, Springer, 2011 (Lecture Notes in Computer Science), S. 189–206

[Sha49] Shannon, Claude: Communication Theory of Secrecy Systems. In: *Bell Systems Technical Journal* 28 (1949), S. 656–715

[Sha79] Shamir, Adi: How to Share a Secret. In: *Communications of the ACM* 22 (1979), Nr. 11, S. 612–613

[SN92] Solms, Sebastiaan von ; Naccache, David: On Blind Signatures and Perfect Crimes. In: *Computers and Security* 11 (1992), Nr. 6, S. 581–583

[SPC95] Stadler, Markus ; Pivetau, Jean-Marc ; Camenisch, Jan: Fair Blind Signatures. In: *Advances in Cryptology – EUROCRYPT '95* Bd. 921, Springer, 1995 (Lecture Notes in Computer Science), S. 209–219

[Str98] Stroth, Gernot: *Algebra*. De Gruyter, 1998

[STS99a] Sander, Tomas ; Ta-Shma, Amnon: Auditable, Anonymous Electronic Cash. In: *Advances in Cryptology – CRYPTO '99* Bd. 1666, Springer, 1999 (Lecture Notes in Computer Science), S. 555–572

[STS99b] Sander, Tomas ; Ta-Shma, Amnon: Flow Control: A New Approach for Anonymity Control in Electronic Cash Systems. In: *Financial Cryptography – FC '99* Bd. 1648, Springer, 1999 (Lecture Notes in Computer Science), S. 46–61

[STS99c] Sander, Tomas ; Ta-Shma, Amnon: On Anonymous Electronic Cash and Crime. In: *Information Security – ISW '99* Bd. 1729, Springer, 1999 (Lecture Notes in Computer Science), S. 202–206

[Tro06] Trolin, Mårten: *A Stronger Definition for Anonymous Electronic Cash*. Cryptology ePrint Archive, Report 2006/241. `http://eprint.iacr.org/2006/241`. Version: August 2006

[Tsi97] Tsiounis, Yiannis: *Efficient Electronic Cash: New Notions and Techniques*, The Department of Computer Science, Northeastern University, Diss., Juni 1997

[TY98] Tsiounis, Yiannis ; Yung, Moti: On the Security of ElGamal Based Encryption. In: *Public Key Cryptography – PKC '98* Bd. 1431, Springer, 1998 (Lecture Notes in Computer Science), S. 117–134

[Ver01] Verheul, Eric: Evidence that XTR is more secure than supersingular elliptic curve cryptosystems. In: *Advances in Cryptology – EUROCRYPT '01* Bd. 2045, Springer, 2001 (Lecture Notes in Computer Science), S. 195–210

[Wei05] Wei, Victor: *More Compact E-Cash with Efficient Coin Tracing*. Cryptology ePrint Archive, Report 2005/411. `http://eprint.iacr.org/2005/411`. Version: Mai 2005

[XY03] Xu, Shouhuai ; Yung, Moti: Retrofitting Fairness on the Original RSA-Based E-cash. In: *Financial Cryptography – FC '03* Bd. 2742, Springer, 2003 (Lecture Notes in Computer Science), S. 51–68

INDEX

Abhebe-Beschuldigung, 146
Akkumulator-Schema, 60
  beschränkt, 61
  dynamisch, 63
  kollisionsfrei, 61
  Nguyen, 62
Algorithmus, 13
  deterministisch, 13
  polynomiell, 13
  probabilistisch, 13
  voraussichtlich-polynomiell, 13
Angreifer
  beschränkt, 5
  unbeschränkt, 5
Angriff
  adaptiv, 36, 39
  mit bekannten Signaturen, 39
  mit gewählten Geheimtexten, 34
  mit gewählten Klartexten, 34
  mit gewählten Nachrichten, 39
  ohne bekannte Signaturen, 39
Annahme
  Computational-Diffie-Hellman, 27
  Decisional-Bilinear-Diffie-Hellman, 30
  Decisional-Diffie-Hellman, 27
  Decisional-Linear-Diffie-Hellman, 30
  Diskreter-Logarithmus, 25
  External-Decisional-Diffie-Hellman, 29
  q-polynomial-Diffie-Hellman, 32
  q-Strong-Diffie-Hellman, 31
  Symmetric-External-Decisional-Diffie-Hellman, 29
Anonymität, 2, 144
  aufhebbar, 6
  bei Erstattungsfähigkeit, 287
  perfekt, 5
  rechnerisch, 5

Bank, 2, 129
Bilineare Abbildung, 21
Binär-Baum, 173

Commitment-Schema, 44
  berechnungsbindend, 48
    stark, 88
  beschränkt, 48
  Pedersen, 46
  perfekt verbergend, 45
  Polynom, 47
  rechnerisch bindend, 45

Darstellung, 26
Deanonymisierer, 6, 133, 227
Digitales Signaturverfahren, 39
  BBS, 43
  existentiell unfälschbar, 41
    stark, 41

mit effizienten Protokollen, 66
BBS+, 4, 68
CL, 4
ESS, 73
ESS+, 4, 73
Schnorr-Signatur, 42, 59
Digitales-Signaturverfahren
mit effizienten Protokollen
BBS+A, 94
Diskreter-Logarithmus, 25
Double-Spending, 3, 135
Beschuldigung, 146
Detection, 3
Double-Spender, 4, 135
Prevention, 4, 150, 309

Elliptische Kurve, 18
nichtsingulär, 18
Erstattungsfähigkeit, 284
Existentielle Unfälschbarkeit, 41
stark, 41

Fiat-Shamir-Heuristik, 17, 58

Geheimtext, 33
Geldbörse, 5, 130
Geldsystem
elektronisch, 2, 131
fair, 6, 134, 227
kompakt, 5, 130, 157
mit Observer, 4, 135, 309
offline, 3
online, 3
teilbar, 5, 130, 173
Generator, 12
Generatortupel, 26
zufällig, 26
Gruppe, 11
additiv, 12
endlich, 12
kommutativ, 12
multiplikativ, 12
Ordnung, 12
zyklisch, 12

Händler, 2, 129
Hashfunktion, 16
Einweg, 16
kollisionsresistent
schwach, 16
stark, 17
kryptografisch, 17

Informationen
ausgehend, 151
eingehend, 151
Interaktives Beweissystem, 51

Klartext, 33
Konkatenation, 12
Kunde, 2, 129
Kundenverfolgung, 6, 134, 227
Beschuldigung, 147
korrekt, 147

Münz-Identifikator, 228, 238
Münze, 2, 130
teilbar, 5, 130
Münzverfolgung, 6, 134, 227
Beschuldigung, 149
korrekt, 148

Observer, 4, 135
Over-Spending, 311
Prevention, 150, 311

Pairing, 21
ate-Pairing, 23
R-ate-Pairing, 23
Tate-Pairing, 22, 23
Weil-Pairing, 22
Polynomielle Ununterscheidbarkeit, 35
Problem
Computational-Diffie-Hellman, 27
Darstellungsproblem, 26
Decisional-Bilinear-Diffie-Hellman, 30
Decisional-Diffie-Hellman, 27

Decisional-Linear-Diffie-Hellman, 30
Diskreter-Logarithmus, 25
External-Decisional-Diffie-Hellman, 29
q-polynomial-Diffie-Hellman, 32
q-Strong-Diffie-Hellman, 31
Symmetric-External-Decisional-Diffie-Hellman, 29
Proof-of-Knowledge, 52
Protokoll, 13
interaktiv, 13, 54
nichtinteraktiv, 58
Protokollansicht, 13
Sigma-Protokoll, 54
Public-Key-Verschlüsselungsverfahren, 34
ElGamal, 36
linear, 37
polynomiell ununterscheidbar, 35
semantisch sicher, 34, 35

Random-Oracle-Modell, 17

Secret-Sharing, 3, 90
Seriennummer, 131, 180
Serienschlüssel, 130, 180
Sicherheitstag, 131, 180
Signatur
blind, 3
digital, 2
Signature-of-Knowledge, 59
Signaturverfahren, *siehe* Digitales Signaturverfahren
Signer, 66

Trusted-Third-Party (TTP), 6, 133

Unfälschbarkeit, 143
ununterscheidbar, 53
perfekt, 53
rechnerisch, 53
User, 66

Verifier, 67
Verschlüsselungsverfahren, *siehe* Public-Key-Verschlüsselungsverfahren

Zero-Knowledge, 54
Honest-Verifier, 54
perfekt, 54
rechnerisch, 54
perfekt, 54
Prover, 51
rechnerisch, 54
Verifier, 51
Zero-Knowledge-Proof-of-Knowledge, 51