Neue Ansätze für die Sicherheit der Random-Oracle-Methodik

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

Björn Fay

aus Lich

November 2008

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Sicherheit spielt heute eine bedeutende Rolle in der elektronischen Datenverarbeitung, dadurch dass bei vielen Firmen die IT-Infrastruktur wächst und immer mehr Arbeitsschritte per Computer ausgeführt werden. Insbesondere die Datenablage und die Kommunikation stehen hier im Blickpunkt von Sicherheitsbetrachtungen. Große Bedeutung hat dabei die Kryptographie, die als Wissenschaft der Geheimschriften (von griechisch: kryptós – „verborgen“ – und gráphein – „schreiben“) viele nützliche Bausteine zur Verfügung stellt, mit deren Hilfe man z. B. eine vertrauliche Verbindung über das Internet aufbauen kann. Viele dieser Bausteine werden als *Protokolle* zwischen zwei oder mehr Teilnehmern beschrieben, wobei es auch oft vorkommt, dass nur ein Teilnehmer eine bestimmte Nachricht an einen anderen sendet, wie z. B. bei einer Verschlüsselung oder digitalen Signatur. Für die meisten dieser Protokolle existieren Sicherheitsbeweise, wobei viele auf bis jetzt unbewiesenen Annahmen beruhen, wie z. B. dass es im Allgemeinen schwierig ist, eine große natürliche Zahl in ihre Primfaktoren zu zerlegen. Diese Annahmen werden in den meisten Fällen schon längere Zeit untersucht, wobei es insbesondere bei den Versuchen die Standard-Annahmen zu widerlegen immer wieder kleine Fortschritte gab (was dann Anlass für größere Schlüssellängen gab), aber davon abgesehen haben sich die Annahmen als berechtigt erwiesen. Für eine der Standard-Annahmen, nämlich dass es in endlichen Gruppen, deren Ordnung einen großen Primteiler besitzt, schwer ist, den diskreten Logarithmus zu berechnen, konnte in [Sho97] sogar gezeigt werden, dass sie zutrifft, sofern man nur sogenannte generische Algorithmen verwendet, die nur auf den Gruppenoperationen beruhen und sich keine weiteren Eigenschaften der Gruppe zu Nutze machen.

Ein weiteres Problem der Sicherheitsbeweise in der Kryptographie ist, dass einige Protokolle sogenannte *Hashfunktionen* benutzten, die sich nur sehr schlecht theoretisch behandeln lassen. Solche Funktionen bilden beliebig lange Zeichenketten bzw. Nachrichten auf sehr kurze Zeichenketten fester Länge ab, die sich als eine Art Fingerabdruck ansehen lassen (dazu darf es unter anderem praktisch nicht möglich sein, zu einem Fingerabdruck eine passende Nachricht zu finden oder zwei Nachrichten, die denselben Fingerabdruck haben, was man *Einweg-Eigenschaft* bzw. *Kollisionsresistenz* nennt, siehe auch [BSW99]). Verwendung finden Hashfunktionen z. B. sehr oft bei digitalen Signaturen, wo zunächst der Hashwert der Nachricht berechnet wird und dann nur noch dieser signiert wird. Dies ist viel effizienter, als die ganze Nachricht zu signieren, da sich ein Hashwert einer Nachricht im Allgemeinen viel schneller berechnen lässt als eine Signatur auf diese Nachricht. Dass sich solche Hashfunktionen nur sehr schlecht theoretisch behandeln lassen, sieht man z. B. daran, dass erst vor kurzem (2004) für eine der meist-

benutzten Hashfunktionen (MD5) Kollisionen gefunden wurden (also Nachrichten mit dem gleichen Hashwert), obwohl diese Hashfunktion schon seit 1991 bekannt ist und auch vielfach untersucht wurde (siehe [WFLY04, LWW05, Kli06]). In manchen Fällen reicht zwar die Forderung nach Kollisionsresistenz einer Hashfunktion aus, aber oft auch nicht. Für diese Fälle hat sich ein durch [FS87] bzw. [BR93] eingeführtes Verfahren etabliert, das sich *Random-Oracle-Methodik* nennt. Dabei wird die verwendete Hashfunktion während des Beweises so behandelt als wäre sie eine echte Zufallsfunktion – ein sogenanntes *Random-Oracle*. Allerdings muss man bei einer Implementierung dieses Random-Oracle wieder durch eine Hashfunktion ersetzen, da es echte Zufallsfunktionen, wie sie für solche Protokolle benötigt würden, nicht gibt. Bei diesem Übergang aus der idealisierten theoretischen Welt, dem *Random-Oracle-Modell*, in die reale Welt hofft man, dass die Sicherheit nicht verloren geht, was aber eine reine Heuristik ist. Man kann zwar davon ausgehen, dass sich keine „strukturellen Fehler“ im Protokoll befinden, wenn der Beweis im Random-Oracle-Modell glückt, aber es kann immer noch sein, dass eine Schwachstelle der Hashfunktion ausgenutzt werden kann oder dass die Hashfunktion in irgendeiner Weise nicht mit dem restlichen Protokoll „verträglich“ ist. Dass dies durchaus möglich ist, wurde anhand eines beeindruckenden Beispiel-Protokolls [CGH98] gezeigt, das im Random-Oracle-Modell sicher ist, aber bei einer Implementierung mit einer beliebigen Hashfunktion unsicher wird.

Es ist daher von wesentlicher Bedeutung zu erkennen, wann die Random-Oracle-Methodik zu sicheren Protokollen führt. Wie das oben genannte Beispiel zeigt, reicht es dazu nicht, nur die Hashfunktionen zu untersuchen und geeignete Bedingungen an diese zu stellen. Vielmehr muss das vollständige Protokoll untersucht werden. Um aber nicht jedes Protokoll einzeln zu behandeln, ist es hilfreich, die Protokolle so allgemein wie möglich zu halten, und dann Bedingungen sowohl an die Protokolle als auch an die Hashfunktionen zu stellen. Genau dieses Vorgehen wird in der vorliegenden Arbeit angewendet und führt letztendlich zu einer besseren Absicherung der Random-Oracle-Methodik. Insbesondere auf die oben erwähnte und oft benutzte Methode, Hashwerte zu signieren, lassen sich die Ergebnisse dieser Arbeit anwenden, im Gegensatz zu dem Protokoll aus [CGH98], das die entsprechenden Bedingungen nicht erfüllt.

Die vorliegende Arbeit gliedert sich wie folgt: Zunächst werden in Kapitel 2 einige Grundlagen zusammenfassend dargestellt und Notationen angegeben. Außerdem wird neben dem bekannten Begriff der *„rechnerischen Ununterscheidbarkeit“* noch der neue Begriff der *„rechnerischen Unabhängigkeit“* eingeführt und dadurch die genauere Betrachtung von Verteilungen ermöglicht. Im Kapitel 3 wird definiert, was formal unter einem Random-Oracle zu verstehen ist und es wird verglichen, inwieweit sich unterschiedliche Arten von Random-Oracles (wie sie von verschiedenen Autoren benutzt werden) effizient gegenseitig simulieren lassen. Die Behandlung dieser Problematik wurde durch Herrn Prof. Dr. Jörg Schwenk angeregt, wobei sich herausgestellt hat, dass sich hier durchaus theoretisch Probleme ergeben können, diese aber für die Praxis eher von untergeordneter Bedeutung sind. Im Kapitel 4 werden dann die zur Implementierung benötigten Hashfunktionen behandelt, wobei zuerst bereits bekannte Hashfunktionen

dargestellt und anschließend neue definiert werden, nämlich *unvorhersehbare* und *zufallerhaltende*, welche zur besseren Absicherung der Random-Oracle-Methodik benötigt werden. Für eine Einordnung der neuen Hashfunktionen in das bereits bestehende Gefüge von Hashfunktionen werden die in dieser Arbeit vorgestellten Hashfunktionen noch miteinander verglichen, so dass sie sich am Ende in etwa der „Stärke“ nach anordnen lassen. Da wir die Sicherheit von Protokollen betrachten, muss geklärt werden, was genau unter einem solchen Protokoll zu verstehen ist. Dies wird in Kapitel 5 behandelt. Dabei wird die Struktur der Protokolle soweit angegeben, dass sie zwar eine sehr spezielle Form haben, aber immer noch so allgemein sind, dass sich viele Anwendungen damit realisieren lassen. Außerdem werden die möglichen Angriffe auf diese Protokolle spezifiziert sowie die möglichen Erfolge, die ein solcher Angriff haben kann. In Kapitel 6 wird schließlich die Random-Oracle-Methodik noch einmal genauer dargestellt und mit ähnlichen Verfahren verglichen sowie deren Unsicherheit aufgezeigt. Anschließend wird mit Hilfe der hier entwickelten neuen Ansätze gezeigt, wie sich die Random-Oracle-Methodik besser absichern lässt. Dabei wird im Wesentlichen eine Transformation der Protokolle durchgeführt und die Verteilung der an den Protokollen beteiligten Variablen im Random-Oracle-Modell und im Standard-Modell verglichen. Für zwei Vermutungen ließen sich leider keine streng formalen Beweise finden. Allerdings liefern die in dieser Arbeit entwickelten Methoden trotzdem gute und plausible Argumente für die Gültigkeit dieser Vermutungen. Abschließend werden in Kapitel 7 die wichtigsten Ergebnisse zusammengefasst und die sich aus dieser Arbeit ergebenden offenen Fragen diskutiert.

Ich möchte an dieser Stelle einigen Personen danken, die mir in der einen oder anderen Weise beim Anfertigen dieser Arbeit geholfen haben. Dies sind im Einzelnen mein Betreuer Prof. Dr. Albrecht Beutelspacher, durch den diese Arbeit überhaupt erst möglich wurde, Prof. Dr. Jörg Schwenk für seine Bereitschaft sich als Zweitgutachter zur Verfügung zu stellen und die Anregung für Abschnitt 3.2, Prof. Dr. Martin Kutrib für seine Hilfe bei Fragen der Informatik, Dr. Gerrit Eichner für seine Hilfe im Bereich der Stochastik, Dr. Henning Bordihn für seine Hinweise über Informationstheorie, Dr. Jörn Schweisgut, Dr. Christian Tobias und Thomas Schwarzpaul für Anmerkungen und das Korrekturlesen und schließlich noch meinen Eltern Anneliese und Walter Fay für ihre Unterstützung in vielen anderen Dingen.

2 Grundlagen und Notation

In diesem Kapitel werden die nicht-kryptographischen Grundlagen und deren Notation dargestellt. Die kryptographischen Grundlagen werden in den darauf folgenden Kapiteln behandelt.

Zunächst werden einige grundlegende Notationen erklärt. Die Menge der *natürlichen Zahlen* $\{0, 1, 2, \dots\}$ wird mit $\mathbb{N}$ bezeichnet und mit $\mathbb{N}^+$ die Menge aller *positiven natürlichen Zahlen*, also $\mathbb{N}^+ = \{n \in \mathbb{N} \mid n > 0\}$. Analog werden mit $\mathbb{R}$ die *reellen Zahlen* bezeichnet, sowie mit $\mathbb{R}^+$ alle *positiven reellen Zahlen*, also $\mathbb{R}^+ = \{r \in \mathbb{R} \mid r > 0\}$ und mit $\mathbb{R}_0^+$ alle *nicht-negativen reellen Zahlen*, also $\mathbb{R}_0^+ = \{r \in \mathbb{R} \mid r \geq 0\}$. Weiterhin seien $[a, b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}$ und $[a, b]_{\mathbb{N}} = \{x \in \mathbb{N} \mid a \leq x \leq b\} = [a, b] \cap \mathbb{N}$ *Intervalle* in $\mathbb{R}$ bzw. $\mathbb{N}$. Mit $\mathbb{Z}_n$ ist der *Restklassenring modulo n* gemeint und mit $\mathbb{Z}_n^*$ die *prime Restklassengruppe modulo n*.

Es bezeichnet $\lfloor a \rfloor$ die größte ganze Zahl, die kleiner oder gleich a ist, sowie $\lceil a \rceil$ die kleinste ganze Zahl, die größer oder gleich a ist. Für zwei ganze Zahlen a und b bedeutet $a \mid b$, dass a ein *Teiler* von b ist, also ein $c \in \mathbb{Z}$ existiert mit $b = ac$.

Für zwei Mengen A und B bezeichnet $M = A \dot\cup B$ die *disjunkte Vereinigung* von A und B, d. h. M ist die *Vereinigung* $A \cup B$ und der *Schnitt* $A \cap B$ ist die *leere Menge* $\emptyset$. Außerdem ist $A \setminus B = \{a \in A \mid a \notin B\}$ die *Differenz* von A und B sowie $|M|$ die Mächtigkeit der Menge M und $A \times B = \{(a, b) \mid a \in A, b \in B\}$ das *kartesische Produkt*.

Die logischen Operatoren *„und“* und *„oder“* werden mit $\wedge$ und $\vee$ bezeichnet und $\oplus$ ist die *bitweise XOR-Verknüpfung* (exklusives Oder). Die *Verknüpfung* bzw. *Hintereinanderausführung* zweier Funktionen f und g wird mit $f \circ g$ bezeichnet, also $(f \circ g)(x) = f(g(x))$.

2.1 Stochastische Grundlagen

Da in der Kryptographie meistens zufällig gewählte Schlüssel benötigt werden und auch potentielle Angreifer sich den Zufall zu Hilfe nehmen können, führt ein möglicher Angriff nicht immer zu einem Erfolg bzw. Misserfolg. Man hat es also mit einer Erfolgswahrscheinlichkeit für einen Angriff zu tun. Um diese abschätzen zu können, werden einige Begriffe und Ergebnisse aus der Stochastik benötigt.

Im Allgemeinen betrachtet man in der Stochastik Zufallsexperimente, die auf eine bestimmte Art und Weise ausgehen bzw. einen Ausgang ω haben. Die Menge aller Ausgänge $\Omega := \{\omega \mid \omega \text{ ist Ausgang des Experiments}\}$ heißt *Grundraum*. Manchmal interessiert man sich aber auch gar nicht für einzelne Ausgänge, sondern nur für ein bestimmtes *Ereignis* $A \subseteq \Omega$, das mehrere Ausgänge zusammenfasst. Die Ereignisse $\{\omega\}$ mit einzelnen

Ausgängen des Experiments werden auch *Elementarereignisse* genannt. Alle Ereignisse von Interesse werden in der Menge $\mathcal{A}$ zusammengefasst, die eine Teilmenge der Potenzmenge $\mathcal{P}(\Omega)$ des Grundraums ist. Bei abzählbarem Grundraum (endlich oder unendlich) betrachtet man alle nur erdenklichen Ereignisse, also $\mathcal{A} = \mathcal{P}(\Omega)$, bei überabzählbarem Grundraum meist nur eine echte Teilmenge. Diese muss allerdings bestimmte Bedingungen erfüllen: Es muss $\Omega \in \mathcal{A}$ sein (das sogenannte sichere Ereignis), und für alle Ereignisse $A \in \mathcal{A}$ muss auch das Gegenereignis $\Omega \setminus A \in \mathcal{A}$ sein. Außerdem sollten auch Ereignisse zusammengesetzt werden können, weswegen für eine Folge $(A_i)_{i\in\mathbb{N}}$ in $\mathcal{A}$ auch $\bigcup_{i\in\mathbb{N}} A_i \in \mathcal{A}$ sein muss. Eine Menge $\mathcal{A}$ mit diesen Eigenschaften nennt man *σ-Algebra*.

Jedes Ereignis $A \in \mathcal{A}$ hat eine gewisse *Wahrscheinlichkeit* $P(A) \in [0, 1]$. Für paarweise disjunkte Ereignisse $A_i, i \in \mathbb{N}$ soll dabei $P(\bigcup_{i\in\mathbb{N}} A_i) = \sum_{i\in\mathbb{N}} P(A_i)$ gelten und für das sichere Ereignis Ω, dass $P(\Omega) = 1$ ist. Für das unmögliche Ereignis $\emptyset$ folgt dann, dass $P(\emptyset) = 0$ ist. Eine Funktion $P\colon \mathcal{A} \to \mathbb{R}$ mit den obigen Eigenschaften heißt *Wahrscheinlichkeitsmaß*.

Ein Tripel $(\Omega, \mathcal{A}, P)$, bestehend aus Grundraum Ω, zugehöriger σ-Algebra $\mathcal{A}$ und Wahrscheinlichkeitsmaß P, heißt *Wahrscheinlichkeitsraum*. Nur für diese Räume lassen sich die Ergebnisse aus der Stochastik anwenden. Daher ist vorher zu klären, ob die in dieser Arbeit benutzten Konstrukte diese Bedingungen erfüllen. Eine genauere und ausführlichere Abhandlung findet sich z. B. in [Sch98] und [Bau91], auch für die noch folgenden Begriffe.

Eine wesentliche Rolle in der Stochastik spielen die sogenannten *Zufallsvariablen*, welche Abbildungen von einem Grundraum Ω eines Wahrscheinlichkeitsraumes $(\Omega, \mathcal{A}, P)$ in eine Menge M sind; man sagt dazu auch M-wertige Zufallsvariablen (über $\mathcal{A}$). Für diese Arbeit ist es ausreichend, abzählbare Mengen M zu betrachten. In diesem Fall muss für eine Zufallsvariable X nämlich nur $X^{-1}(m) := \{\omega \in \Omega \mid X(\omega) = m\} \in \mathcal{A}$ für alle $m \in M$ gelten. Dann folgt auch automatisch $X^{-1}(U) := \{\omega \in \Omega \mid X(\omega) \in U\} \in \mathcal{A}$ für alle $U \subseteq M$.

Wenn in einem Wahrscheinlichkeitsraum $(\Omega, \mathcal{A}, P)$ mit endlichem Grundraum Ω alle Elementarereignisse gleichwahrscheinlich sind, also $P(\{\omega\}) = \frac{1}{|\Omega|}$ für alle $\omega \in \Omega$, dann spricht man von einer *uniformen Verteilung* (oder auch *Gleichverteilung*). Wenn für einen beliebigen Wahrscheinlichkeitsraum $(\Omega, \mathcal{A}, P)$ und eine M-wertige Zufallsvariable X mit endlicher Menge M gilt, dass $P(X = m) := P(X^{-1}(m)) = \frac{1}{|M|}$ ist, dann sagt man, dass die Zufallsvariable *uniform verteilt* ist. Wenn für eine abzählbare Menge M, zwei M-wertige Zufallsvariablen X, Y und alle $m \in M$ gilt, dass $P(X = m) = P(Y = m)$ ist, dann sind X und Y *identisch verteilt* bzw. besitzen *dieselbe Verteilung*. Man schreibt dafür auch kurz $X \sim Y$.

In den meisten Fällen ist es nicht ausreichend, nur die Verteilung einzelner Zufallsvariablen zu kennen. Man interessiert sich oft für die gemeinsame Verteilung bzw. für die Abhängigkeit der Zufallsvariablen. Für diesen Zweck gibt es die *bedingte Wahrscheinlichkeit*. Für zwei Ereignisse A und B mit $P(B) > 0$ ist $P(A \mid B) := \frac{P(A\cap B)}{P(B)}$ die Wahrscheinlichkeit von A unter der Bedingung B. Für zwei Zufallsvariablen X und Y

wäre das dann z. B. $P(X = m \mid Y \in U) = P(X^{-1}(m) \mid Y^{-1}(U))$. Ereignisse $A_i \in \mathcal{A}$ mit $i \in I$ und einer beliebigen Menge I heißen *stochastisch unabhängig*, wenn für alle endlichen Teilmengen $J \subseteq I$ gilt, dass $P(\bigcap_{j\in J} A_j) = \prod_{j\in J} P(A_j)$ ist. Zufallsvariablen $X_i \colon \Omega \to M_i$ mit $i \in I$ und abzählbaren M_i heißen stochastisch unabhängig, wenn für alle möglichen $m_i \in M_i$ die Ereignisse $X_i^{-1}(m_i)$ mit $i \in I$ stochastisch unabhängig sind. Wenn zwei Ereignisse A und B stochastisch unabhängig sind, dann ist $P(A \mid B) = P(A)$, falls $P(B) > 0$ ist.

Da in dieser Arbeit Algorithmen behandelt werden, die sich beliebig oft des Zufalls bedienen können (siehe Abschnitt 2.3), müssen wir uns noch überlegen, wie dafür ein geeigneter Wahrscheinlichkeitsraum aussehen kann. In den meisten Fällen wird in der Literatur davon ausgegangen, dass diese Algorithmen Münzwürfe mit einer fairen Münze durchführen können. Wir werden das etwas verallgemeinern und abzählbar unendlich viele unabhängige Würfe eines „fairen Würfels“ mit Ergebnissen in $\Omega = \{\tilde{\omega}_1, \ldots, \tilde{\omega}_n\}$ betrachten, so dass wir uns nicht unnötiger Weise auf zwei Ergebnisse pro Wurf wie bei einer Münze einschränken. Formal betrachten wir also einen Grundraum, den wir Ω^∞ nennen werden, der aus Folgen $(\omega_i)_{i\in\mathbb{N}}$ mit $\omega_i \in \Omega$ besteht oder gleichwertig dazu aus Funktionen $\omega \colon \mathbb{N} \to \Omega$. Für einen einzelnen Wurf ist der Wahrscheinlichkeitsraum $(\Omega, \tilde{\mathcal{A}}, \tilde{P})$ mit $\tilde{\mathcal{A}} = \mathcal{P}(\Omega)$ und $\tilde{P}(\{\tilde{\omega}_i\}) = \frac{1}{n}$. Für eine abzählbar unendliche Folge von diesen Würfen suchen wir also einen Wahrscheinlichkeitsraum $(\Omega^\infty, \mathcal{A}, P)$, der sich für endliche Folgen „wie erwartet“ verhält. Für eine endliche Folge $\omega_0, \ldots, \omega_m \in \Omega$ soll die Wahrscheinlichkeit $\frac{1}{n^{m+1}}$ sein, wenn uns die restlichen Folgenglieder (der eigentlich unendlichen Folge) nicht interessieren und daher beliebige Werte annehmen dürfen. Dass solch ein Wahrscheinlichkeitsraum existiert und durch obige Forderung auch schon eindeutig bestimmt ist, wird in [Bau91] § 9 gezeigt. Wir bezeichnen diesen Grundraum wie bereits gesagt mit Ω^∞ und werden implizit auch immer den dazugehörigen Wahrscheinlichkeitsraum $(\Omega^\infty, \mathcal{A}, P)$ betrachten, ohne diesen jedesmal mit anzugeben. In den meisten Fällen wird $\Omega = \{0, 1\}$ sein (Münzwürfe). Wir werden den Grundraum Ω der einzelnen „Würfe“ meistens auch nicht mehr explizit angeben, er sei dann immer passend (aber endlich) gewählt.

Da später komplexe Wahrscheinlichkeitsbetrachtungen mit mehreren Zufallsvariablen bzw. Algorithmen durchgeführt werden, ist eine etwas mächtigere Notation sehr hilfreich, die sich am besten an einem einfachen Beispiel erklären lässt. Es seien $(\Omega_i, \mathcal{A}_i, P_i), i = 1, 2$, zwei Wahrscheinlichkeitsräume und $X_i \colon \Omega_i \to M, i = 1, 2$, zwei Zufallsvariablen, wobei M wie immer abzählbar ist. Wir konstruieren einen neuen Wahrscheinlichkeitsraum $(\Omega, \mathcal{A}, P)$ mit $P(A_1 \times A_2) = P_1(A_1) \cdot P_2(A_2)$ für alle $(A_1, A_2) \in \mathcal{A}_1 \times \mathcal{A}_2$ und mit $\mathcal{A}$ als der kleinsten σ-Algebra mit $\{A_1 \times A_2 \mid (A_1, A_2) \in \mathcal{A}_1 \times \mathcal{A}_2\} \subseteq \mathcal{A}$. Dieser Wahrscheinlichkeitsraum ist dadurch ebenfalls eindeutig bestimmt ([Bau91] § 9). Wir benutzen dann folgende Notation, die anschließend noch genauer erläutert wird:

$$P_{r,s}(a = b : a = X_1(r), b = X_2(s)) = P(\{(\omega_1, \omega_2) \in \Omega \mid X_1(\omega_1) = X_2(\omega_2)\}).$$

Es stehen also im Index von P Projektionen von Ω auf Ω_i (pro „Komponente“ eine Projektion, in der gleichen Reihenfolge, mit $r, s, t, \ldots$ bezeichnet) und nach dem Dop-

pelpunkt die durch Kommas getrennte „Vorschrift“ zum Berechnen der benutzten Hilfsvariablen (hier a und b). Rein formal könnte man den Doppelpunkt und die Kommas durch logische „und“ ersetzen, was aber nicht ganz so übersichtlich ist. Die Konstruktion ist dabei so gewählt, dass $X_1(r)$ und $X_2(s)$ beides M-wertige Zufallsvariablen über $\mathcal{A}$ sind, da z. B. für $A_1 = X_1^{-1}(m)$ gilt, dass

$$(X_1 \circ r)^{-1}(m) = r^{-1}(X_1^{-1}(m)) = r^{-1}(A_1) = A_1 \times \Omega_2 \in \mathcal{A}$$

ist. Später werden diese Zufallsvariablen meistens Algorithmen sein, die den Zufall benutzen (siehe Definition 2.3.2). Dabei werden die Projektionen bzw. der von den Algorithmen benutzte Zufall nicht mehr als Argument in runden Klammern angegeben, sondern als Index des Algorithmus, passend zur Notation der Algorithmen (siehe Beispiel 2.3.3). Im Allgemeinen wird bei dieser Notation der Wahrscheinlichkeitsraum nicht mehr explizit angegeben, sondern immer entsprechend konstruiert sein (siehe z. B. Definition 2.4.1). Wir werden die Notation mit den Indizes aber nicht benutzen, wenn es auch ohne sie möglich ist, die Wahrscheinlichkeiten eindeutig und verständlich auszudrücken. Wir benutzen die Notation also nur, wenn sie zur formalen Absicherung oder dem Verständnis beiträgt. Wenn wir die Notation nicht benutzen und eine Zufallsvariable (oder ein Random-Oracle, siehe Definition 3.1.1) mehrfach in einem Ausdruck auftaucht, dann ist damit immer der gleiche Wert (bzw. die gleiche Instanz eines Random-Oracles) gemeint, also z. B.

$$P(X = X) = P_r(X(r) = X(r)) = 1.$$

Ein weiterer wichtiger Begriff ist der *Erwartungswert* einer $\mathbb{N}$-wertigen Zufallsvariablen X (in dieser Arbeit die Laufzeit von Algorithmen). Dieser wird durch

$$E(X) = \sum_{n \in \mathbb{N}} n \cdot P(X = n)$$

berechnet. Auch beim Erwartungswert wird gelegentlich zur Verdeutlichung die erweiterte Notation mit den Projektionen im Index benutzt werden.

2.2 Asymptotisches Wachstum

Da Sicherheit in der Kryptographie in den meisten Fällen eher eine Frage der Zeit ist, nämlich wie lange braucht ein Angreifer, um ein Verfahren zu brechen, wird in diesem Abschnitt zunächst das asymptotische Wachstum behandelt (für eine genauere Abhandlung siehe [Rei99]). Damit ist es möglich, formal zu beschreiben, was es bedeutet, dass ein Angreifer bzw. ein Algorithmus effizient ist oder nicht.

Zunächst einige Notationen. Mit $\mathcal{N}\colon \mathbb{N} \to \mathbb{N}, n \mapsto n$ wird die Identität auf den natürlichen Zahlen bezeichnet. Für zwei Funktionen $f, g\colon \mathbb{N} \to \mathbb{R}_0^+$ bedeutet $f \leq_{ae} g$, dass f fast überall (engl. almost everywhere) kleiner oder gleich g ist, also

$$f \leq_{ae} g \quad :\Leftrightarrow \quad \exists n_0 \in \mathbb{N}\colon \forall n \geq n_0\colon f(n) \leq g(n).$$

Für die übrigen Vergleichsoperatoren wird das entsprechend definiert. Für eine reelle Funktion f bedeutet f^n die n-te Potenz, also $f^n(x) := (f(x))^n$ und nicht die n-fache Hintereinanderausführung von f. Für zwei reelle Funktionen f, g bezeichnen $f+g$ und fg die beiden Funktionen $x \mapsto f(x)+g(x)$ und $x \mapsto f(x) \cdot g(x)$. Außerdem wird $\frac{1}{0} := \infty > r$ für alle $r \in \mathbb{R}$ gesetzt.

Die Standard-Notation für asymptotisches Wachstum ist die *Groß-O-Notation* und deren Verwandte.

Definition 2.2.1. *Es sei $g\colon \mathbb{N} \to \mathbb{R}_0^+$ eine Funktion, dann ist*

$$O(g) := \{f\colon \mathbb{N} \to \mathbb{R}_0^+ \mid \exists k \in \mathbb{N}^+\colon f \leq_{ae} k \cdot g\},$$
$$\omega(g) := \{f\colon \mathbb{N} \to \mathbb{R}_0^+ \mid \forall k \in \mathbb{N}^+\colon f \geq_{ae} k \cdot g\}.$$

Wie anhand der Definition zu sehen ist (vgl. [Rei99]), spielen konstante Faktoren keine Rolle, d. h. es gilt z. B. $O(g) = O(cg)$ für eine Konstante $c \in \mathbb{R}^+$. Wenn man die beiden Notationen mit den normalen Vergleichsoperatoren vergleicht, kann man sich das etwa so vorstellen: $f \in O(g)$ bedeutet im Wesentlichen $f \leq g$ und $f \in \omega(g)$ entspricht $f > g$. Es gelten weiter die folgenden Aussagen:

Satz 2.2.2. *Es seien $f, g\colon \mathbb{N} \to \mathbb{R}_0^+$ Funktionen, dann gilt:*

$$f \in \omega(g) \quad \Leftrightarrow \quad \lim_{n\to\infty} \frac{g(n)}{f(n)} = 0,$$
$$O(g) \cap \omega(g) = \emptyset.$$

Beweis. Siehe [Rei99]. □

Bei der Betrachtung der Effizienz von Algorithmen stellt man sich im Wesentlichen die Frage, ob die benötigte Zeit eine polynomiale Funktion der Eingabelänge ist (meistens in binärer Notation, nicht zu verwechseln mit der Größe einer Zahl, so würde z. B. die Zahl 16 nur eine Eingabelänge von 5 ergeben und die Zahl 1024 eine Länge von 11). Wenn dies der Fall ist, sagt man, dass ein Algorithmus effizient ist. Es gibt nämlich viele (schwere) Probleme, für die man keine Algorithmen kennt, die in polynomialer Zeit eine Lösung berechnen können. Im Gegensatz dazu gibt es (einfache) Probleme wie z. B. Addieren, Multiplizieren und Potenzieren für die man Algorithmen kennt, die eine polynomiale Laufzeit haben. Wir bezeichnen die Menge dieser Laufzeiten mit poly oder etwas genauer:

Definition 2.2.3. *Es ist*

$$\mathsf{poly} := \bigcup_{k\in\mathbb{N}} O(\mathcal{N}^k).$$

Man findet öfter auch eine andere, aber dazu gleichwertige Definition.

Lemma 2.2.4. *Für alle Funktionen $f\colon \mathbb{N} \to \mathbb{R}_0^+$ gilt*

$$f \in \mathsf{poly} \quad \Leftrightarrow \quad \exists k \in \mathbb{N}\colon f \leq_{ae} \mathcal{N}^k.$$

Beweis. Es gilt

$$\begin{aligned} f \in \mathsf{poly} \quad &\Rightarrow \quad \exists k \in \mathbb{N}\colon f \in O(\mathcal{N}^k) \\ &\Rightarrow \quad \exists k \in \mathbb{N}, k' \in \mathbb{N}^+\colon f \leq_{ae} k'\mathcal{N}^k \\ &\Rightarrow \quad \exists k \in \mathbb{N}\colon f \leq_{ae} \mathcal{N}^{k+1} \qquad (\text{für } \mathcal{N} \geq k') \\ &\Rightarrow \quad \exists k \in \mathbb{N}\colon f \leq_{ae} \mathcal{N}^k \\ &\Rightarrow \quad \exists k \in \mathbb{N}\colon f \in O(\mathcal{N}^k) \\ &\Rightarrow \quad f \in \mathsf{poly}. \end{aligned}$$

□

Wie zu erwarten liegen alle Polynome in poly.

Lemma 2.2.5. *Für $f\colon \mathbb{N} \to \mathbb{R}_0^+, x \mapsto \sum_{i=0}^n a_i x^i$ mit $a_i \in \mathbb{R}$ gilt $f \in \mathsf{poly}$.*

Beweis. Es ist also $f = \sum_{i=0}^n a_i \mathcal{N}^i$ und damit

$$\begin{aligned} f &\leq_{ae} \sum_{i=0}^{n} a_i \mathcal{N}^i \\ &\leq_{ae} \sum_{i=0}^{n} |a_i| \mathcal{N}^n \\ &\leq_{ae} \left(\sum_{i=0}^{n} |a_i| \right) \mathcal{N}^n \end{aligned}$$

also $f \in O(\mathcal{N}^n) \subseteq \mathsf{poly}$. □

Außerdem gilt für zwei Funktionen $f, g \in \mathsf{poly}$, dass auch die Hintereinanderausführung, Produkt und Summe der beiden Funktionen in poly liegt.

Lemma 2.2.6. *Es seien $f, g \in \mathsf{poly}$, wobei g nur Werte in $\mathbb{N}$ besitzt. Dann gilt*

- $f \circ g \in \mathsf{poly}$,
- $f \cdot g \in \mathsf{poly}$,
- $f + g \in \mathsf{poly}$.

Beweis. Es gilt

$$\begin{aligned} f, g \in \mathsf{poly} &\Rightarrow \exists k, l \in \mathbb{N}\colon f \leq_{ae} \mathcal{N}^k, g \leq_{ae} \mathcal{N}^l \\ &\Rightarrow \exists k, l \in \mathbb{N}\colon f \circ g \leq_{ae} \left(\mathcal{N}^l\right)^k = \mathcal{N}^{lk} \\ &\Rightarrow \exists k \in \mathbb{N}\colon f \circ g \leq_{ae} \mathcal{N}^k \\ &\Rightarrow f \circ g \in \mathsf{poly} \end{aligned}$$

und

$$\begin{aligned} f, g \in \mathsf{poly} &\Rightarrow \exists k, l \in \mathbb{N}\colon f \leq_{ae} \mathcal{N}^k, g \leq_{ae} \mathcal{N}^l \\ &\Rightarrow \exists k, l \in \mathbb{N}\colon f \cdot g \leq_{ae} \mathcal{N}^k \cdot \mathcal{N}^l \leq \mathcal{N}^{l+k} \\ &\Rightarrow \exists k \in \mathbb{N}\colon f \cdot g \leq_{ae} \mathcal{N}^k \\ &\Rightarrow f \cdot g \in \mathsf{poly} \end{aligned}$$

sowie

$$\begin{aligned} f, g \in \mathsf{poly} &\Rightarrow \exists k, l \in \mathbb{N}\colon f \leq_{ae} \mathcal{N}^k, g \leq_{ae} \mathcal{N}^l \\ &\Rightarrow \exists k, l \in \mathbb{N}\colon f + g \leq_{ae} \mathcal{N}^k + \mathcal{N}^l \leq 2\mathcal{N}^{l+k} \\ &\Rightarrow \exists k \in \mathbb{N}\colon f + g \leq_{ae} 2\mathcal{N}^k \\ &\Rightarrow f + g \in \mathsf{poly}. \end{aligned}$$

□

Aber nicht nur die Zeit ist ein Kriterium für die Sicherheit in der Kryptographie, auch die Wahrscheinlichkeit mit der ein System gebrochen werden kann, ist von Interesse. Hierbei möchte man natürlich eine möglichst kleine Wahrscheinlichkeit haben. Da man immer die Möglichkeit hat, einen geheimen Schlüssel oder sonst ein benötigtes Geheimnis zu raten, bzw. zufällig das Gleiche zu machen wie ein legitimer Teilnehmer, wird diese Wahrscheinlichkeit nicht gleich null sein. Man betrachtet hierbei die Wahrscheinlichkeit in Abhängigkeit der Eingabelänge bzw. des Sicherheitsparameters und möchte, dass sie möglichst schnell so klein wird, dass sie nicht mehr von einem effizienten Algorithmus „ausgenutzt“ werden kann. Wenn dies der Fall ist, sagt man, dass die Wahrscheinlichkeit *vernachlässigbar* ist (vgl. [Gol03]).

Definition 2.2.7. *Eine Funktion* $f\colon \mathbb{N} \to \mathbb{R}_0^+$ *ist genau dann* vernachlässigbar*, wenn*

$$\forall p \in \mathsf{poly}\colon f \leq_{ae} \frac{1}{p}$$

gilt. Die Menge aller vernachlässigbaren Funktionen heißt negl.

Man hätte sich hierbei auch auf alle positiven $p \in \mathsf{poly}$ beschränken können, was zum gleichen Ergebnis geführt hätte. Auch hierzu findet man häufig andere, aber gleichwertige Definitionen.

Lemma 2.2.8. *Für eine Funktion $f\colon \mathbb{N} \to \mathbb{R}_0^+$ sind die folgenden Aussagen äquivalent:*

(i) Die Funktion f ist vernachlässigbar.

(ii) Es gilt

$$\forall k \in \mathbb{N}\colon f \leq_{ae} \frac{1}{\mathcal{N}^k}.$$

(iii) Es gilt

$$f \in \bigcap_{k\in\mathbb{N}} O\left(\frac{1}{\mathcal{N}^k}\right).$$

Beweis. (i) $\Rightarrow$ (ii) gilt, da $\forall k \in \mathbb{N}\colon \mathcal{N}^k \in \mathsf{poly}$.

(ii) $\Rightarrow$ (iii) gilt, da aus $f \leq_{ae} \frac{1}{\mathcal{N}^k}$ folgt, dass $f \in O\left(\frac{1}{\mathcal{N}^k}\right)$ ist.

(iii) $\Rightarrow$ (i) gilt, denn für $p \in \mathsf{poly}$ gibt es nach Lemma 2.2.4 ein $k \in \mathbb{N}$ mit $p \leq_{ae} \mathcal{N}^k$ und nach (iii) ist $f \in \bigcap_{j\in\mathbb{N}} O\left(\frac{1}{\mathcal{N}^j}\right) \subseteq O\left(\frac{1}{\mathcal{N}^{k+1}}\right)$. Also gibt es ein $k' \in \mathbb{N}^+$ mit

$$\begin{aligned} f &\leq_{ae} k' \frac{1}{\mathcal{N}^{k+1}} \\ &\leq_{ae} \frac{1}{\mathcal{N}^k} \qquad (\text{für } \mathcal{N} \geq k') \\ &\leq_{ae} \frac{1}{p}. \end{aligned}$$

□

Auch hier spielen konstante positive Faktoren keine Rolle. Es ist also sozusagen das Gegenstück zur polynomialen Laufzeit von Algorithmen. Für einen Ausdruck $f(n)$ sagt man auch, dass er vernachlässigbar in n ist, wenn f vernachlässigbar ist. Eine vernachlässigbare Funktion ist sogar so klein, dass Summen und Produkte von vernachlässigbaren Funktionen auch vernachlässigbar sind, also genauso wie bei poly.

Lemma 2.2.9. *Es seien $f, g\colon \mathbb{N} \to \mathbb{R}_0^+$ vernachlässigbare Funktionen. Dann sind $f + g$ und fg vernachlässigbar.*

Beweis. Zunächst ist $f \leq_{ae} 1$ und damit $fg \leq_{ae} g$, was vernachlässigbar ist. Damit ist auch fg vernachlässigbar.

Es gilt $p \in \mathsf{poly} \Leftrightarrow 2p \in \mathsf{poly}$. Damit gilt für alle $p \in \mathsf{poly}$, dass $f \leq_{ae} \frac{1}{2p}$ und $g \leq_{ae} \frac{1}{2p}$, woraus

$$f + g \leq_{ae} \frac{1}{2p} + \frac{1}{2p} = \frac{2}{2p} = \frac{1}{p}$$

folgt. Damit ist auch $f + g$ vernachlässigbar. □

Außerdem ist auch das Produkt einer vernachlässigbaren Funktion mit einer aus poly wieder vernachlässigbar. Dies besagt im Wesentlichen, dass die Vereinigung polynomial vieler Ereignisse mit vernachlässigbarer Wahrscheinlichkeit immer noch eine vernachlässigbare Wahrscheinlichkeit besitzt. Daraus folgt, dass ein effizienter Algorithmus keinen Nutzen aus Ereignissen mit vernachlässigbaren Wahrscheinlichkeiten ziehen kann, da die daraus resultierende Ausgabe eine vernachlässigbare Wahrscheinlichkeit besitzen würde.

Lemma 2.2.10. *Es seien $f \in \mathsf{negl}$ und $g \in \mathsf{poly}$. Dann ist $fg \in \mathsf{negl}$.*

Beweis. Es sei $p \in \mathsf{poly}$. Dann ist auch $gp \in \mathsf{poly}$. Weil $f \in \mathsf{negl}$ ist, gilt $f \leq_{ae} \frac{1}{gp}$, also

$$fg \leq_{ae} \frac{1}{gp} g = \frac{1}{p},$$

womit $fg \in \mathsf{negl}$ folgt, da dies für alle $p \in \mathsf{poly}$ gilt. □

Folgender Sachverhalt wird später oft benötigt, wenn es darum geht, dass ein polynomial beschränkter Algorithmus zufällig Werte raten oder bestimmen muss, aber ihm dies nur mit einer vernachlässigbaren Wahrscheinlichkeit gelingen darf. Der folgende Satz kann hier auch nochmal als Anwendungsbeispiel für die verschiedenen Begriffe dienen.

Satz 2.2.11. *Es sei $f \in \mathsf{poly}$ und $g \in \omega(\log_2(\mathcal{N}))$. Dann ist $\frac{f(n)}{2^{g(n)}}$ vernachlässigbar in n.*

Beweis. Nach Voraussetzung existiert ein $k_0 \in \mathbb{N}$ mit $f \leq_{ae} \mathcal{N}^{k_0}$ und für alle $k \in \mathbb{N}$ gilt $g \geq_{ae} k \log_2(\mathcal{N})$. Dann gilt für alle $k \in \mathbb{N}$, dass ein n_0 existiert, so dass für alle $n \geq n_0$ die Ungleichung

$$\frac{f(n)}{2^{g(n)}} \leq \frac{n^{k_0}}{2^{k \log_2(n)}} = \frac{n^{k_0}}{n^k} = \frac{1}{n^{k-k_0}}$$

gilt. Dies gilt insbesondere für alle $k \geq k_0$, womit $\frac{f(n)}{2^{g(n)}}$ vernachlässigbar in n ist. □

2.3 Turing-Maschinen und Algorithmen

Nachdem im letzten Abschnitt öfter von Algorithmen und deren Laufzeiten die Rede war, bleibt noch zu klären, was man genau unter einem *Algorithmus* versteht und wie dessen *Laufzeit* definiert ist. Insbesondere, wenn man auch noch zufällige Algorithmen zulässt, ist die Frage nach der Laufzeit interessant, da es wegen des Zufalls passieren kann, dass ein Algorithmus bei derselben Eingabe mal eine kurze Laufzeit hat, mal eine lange oder vielleicht auch gar nicht anhält. Gemäß der *Churchschen These* (siehe [Rei99]) können mit *Turing-Maschinen* alle intuitiv berechenbaren Funktionen berechnet werden. Insbesondere lassen sich alle Algorithmen, die man auf modernen Computern implementieren kann durch Turing-Maschinen beschreiben bzw. berechnen.

Bei der Definition von Turing-Maschinen dienen [Rei99] und [Goo97] als Grundlage. Für diese Arbeit reicht eine relativ einfache Definition aus. Ein *Alphabet* Σ ist die Menge

der Symbole mit denen eine Turing-Maschine umgehen kann. Im Folgenden wird meist nur $\Sigma = \{0, 1\}$ benutzt werden, was wegen Theorem 1.4.10 in [Rei99] keine Einschränkung ist. Allerdings müssen dann für die Ein- und Ausgaben evtl. geeignete Codierungen gewählt werden. Mit Σ^* und Σ^n werden endliche Folgen bzw. Folgen der Länge n von Elementen aus Σ bezeichnet, die man dann *Worte* oder auch *Zeichenketten* nennt. In diesem Zusammenhang ist 1^n das Wort der Länge n, das aus der n-fachen Wiederholung von 1 besteht. Die *Länge eines Wortes* w wird mit $|w|$ bezeichnet und die Verkettung von zwei Wörtern w und v wird mit wv bezeichnet. Es ist außerdem ε das *leere Wort* der Länge 0, das auch in Σ^* enthalten ist.

Eine Turing-Maschine besteht aus einem *Eingabeband*, einem *Ausgabeband* und mehreren *Arbeitsbändern* (hier kann man sich wegen Theorem 1.4.2 in [Rei99] auf eines beschränken), sowie einer endlichen *Zustandsmenge* (siehe Abbildung 2.1). Die Maschine beginnt in einem genau definierten *Anfangszustand* und kann in jedem Schritt auf die Bänder lesend oder schreibend zugreifen (jeweils ein Zeichen). Von dem Eingabeband kann nur gelesen werden und auf das Ausgabeband nur geschrieben werden. In Abhängigkeit der gelesen Zeichen und des aktuellen Zustandes nimmt die Maschine dann einen neuen Zustand an, was formal durch eine *Übergangsrelation* beschrieben wird. Das geht solange weiter bis die Maschine sich in einem vorher definierten *Endzustand* befindet (sie hält). Das muss aber nicht passieren, d. h. es könnte sein, dass die Maschine ewig weiterläuft. Für eine ausführliche und formalere Definition sowie Behandlung dieses Themas siehe [Rei99].

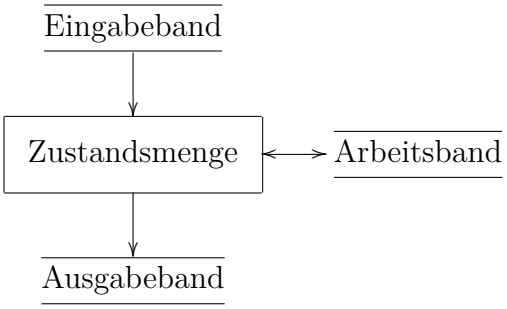

Abbildung 2.1: Turing-Maschine

Da eine Turing-Maschine pro Schritt immer nur ein Zeichen pro Band lesen oder schreiben kann, braucht sie maximal so viel Platz auf dem Arbeits- bzw. Ausgabeband, wie sie Schritte benötigt, um bis in den Endzustand zu kommen. Für eine Definition von *effizienten* Algorithmen bzw. Turing-Maschinen reicht es also die Laufzeit zu betrachten.

Definition 2.3.1. *Eine Turing-Maschine ist genau dann* effizient *oder auch* polynomial beschränkt*, wenn ein* $f \in \mathsf{poly}$ *existiert, so dass sie sich bei jeder Eingabe der Länge* n

in weniger als $f(n)$ Schritten im Endzustand befindet. Ein Algorithmus ist genau dann effizient, wenn er durch eine effiziente Turing-Maschine beschrieben werden kann.

Eine solche Maschine oder Algorithmus wird kurz *PT-Turing-Maschine* bzw. *PT-Algorithmus* genannt (engl. polynomial time). Im Rest der Arbeit wird kein Unterschied zwischen Algorithmus und Turing-Maschine gemacht, da eine Turing-Maschine nur eine streng formale Methode ist einen Algorithmus aufzuschreiben. Des Weiteren wird eine Funktion, deren Werte durch einen PT-Algorithmus berechnet werden können, *PT-Funktion* genannt.

Im Folgenden werden noch Algorithmen behandelt, die zufällige Entscheidungen treffen dürfen. Man kann sich natürlich fragen, was es einem Algorithmus bzw. in unserem Fall einem Angreifer bringt, sich des Zufalls zu bedienen. Zunächst wird er dadurch mächtiger, weil er ein weiteres Mittel zur Hand hat, das er benutzen kann. Es war z. B. bis 2002 nicht klar, dass man deterministisch (also ohne den Zufall zu benutzen) in polynomialer Zeit feststellen kann, ob eine Zahl eine Primzahl ist oder nicht (siehe [AKS04]). Bis zu diesem Zeitpunkt kannte man nur effiziente Primzahltests, die den Zufall zur Hilfe nehmen, aber deren Ergebnis nur mit einer bestimmten kontrollierbaren Wahrscheinlichkeit richtig ist. Man benutzt sie auch heute noch, da sie immer noch schneller sind als deterministische Tests (vgl. [Coh00] Algorithm 8.2.2 (Rabin-Miller)).

Formal wird ein solcher Algorithmus durch eine Turing-Maschine beschrieben, die ein weiteres Eingabeband mit unendlich vielen Zufallsbits erhält (vgl. [Gol03] und [Weg03]). Man nennt solche Algorithmen *probabilistisch* (engl. probabilistic). Wenn probabilistische Algorithmen effizient gemäß obiger Definition sind, nennt man sie *Monte-Carlo-Algorithmen* oder auch kurz *PPT-Algorithmen* (eng. probabilistic polynomial time). Das bedeutet, für einen PPT-Algorithmus gibt es ein $f \in \mathsf{poly}$, so dass für alle Eingaben x und Zufalls-Bit-Folgen die Laufzeit kleiner als $f(|x|)$ ist. Es sei hier kurz bemerkt, dass solche Algorithmen bei derselben Eingabe nicht immer dasselbe Ergebnis produzieren, insbesondere nicht unbedingt ein „richtiges“, falls es ein solches überhaupt gibt. Die Schreibweise für probabilistische Algorithmen und den von ihnen benutzten Zufall wird in folgender Definition beschrieben:

Definition 2.3.2. *Für einen* probabilistischen Algorithmus *A bezeichnet $A_r(x)$ die Ausgabe von A bei der Eingabe von x, wenn auf dem* Zufallsband *die Folge $r = (r_i)_{i \in \mathbb{N}}$ steht. Mit $A(x)$ wird zum einen die Menge aller möglichen Ausgaben von A bei der Eingabe von x bezeichnet und zum anderen die Funktion (bzw. Zufallsvariable), die r auf $A_r(x)$ abbildet. Die jeweilige Verwendungsart ergibt sich aus dem Zusammenhang.*

Die Menge aller möglichen Inhalte des Zufallsbandes ist dann Ω^∞ mit endlicher Menge $\Omega \subset \{0,1\}^*$. In den meisten Fällen kann man von $\Omega = \{0,1\}$ ausgehen, also dass das Zufallsband eine Folge von Nullen und Einsen ist, wobei die Wahrscheinlichkeit für eine Null und eine Eins jeweils $\frac{1}{2}$ beträgt und außerdem jedes Folgenglied stochastisch unabhängig von allen anderen ist.

Im Gegensatz zu Monte-Carlo-Algorithmen gibt es noch *Las-Vegas-Algorithmen*, die zwar immer das „richtige“ Ergebnis berechnen, dafür aber nur eine *erwartete polynomiale Laufzeit* haben. Das bedeutet, für einen effizienten Las-Vegas-Algorithmus A gibt es ein $f \in \mathsf{poly}$, so dass $E_r(\text{Laufzeit von } A_r(x)) \leq f(|x|)$ für alle Eingaben x gilt. Diese spielen in dieser Arbeit aber nur eine untergeordnete Rolle.

Als weitere Variante der Turing-Maschine gibt es noch die sogenannte *Orakel-Maschine* (siehe Abbildung 2.2). Dies ist eine Turing-Maschine, die ein weiteres Arbeitsband (das Orakel-Band) hat, auf das ein Orakel Zugriff hat, so dass die Turing-Maschine Anfragen an dieses Orakel stellen kann. Eine solche Anfrage dauert genau einen Schritt und wird durch einen besonderen Anfragezustand eingeleitet. Während eines solchen Schrittes verarbeitet das Orakel die Eingabe auf dem Orakel-Band und stellt das Ergebnis wieder auf diesem Band bereit. Im nächsten Schritt befindet sich die Turing-Maschine dann in einem Antwortzustand (vgl. [Gol03] und [Weg03]) und kann die Antwort des Orakels bearbeiten. Die Bezeichnung für eine solche Orakel-Maschine A mit Zugriff auf ein Orakel B ist A^B. Diese Orakel-Maschinen gibt es auch jeweils mit und ohne Zufallsband, sowie

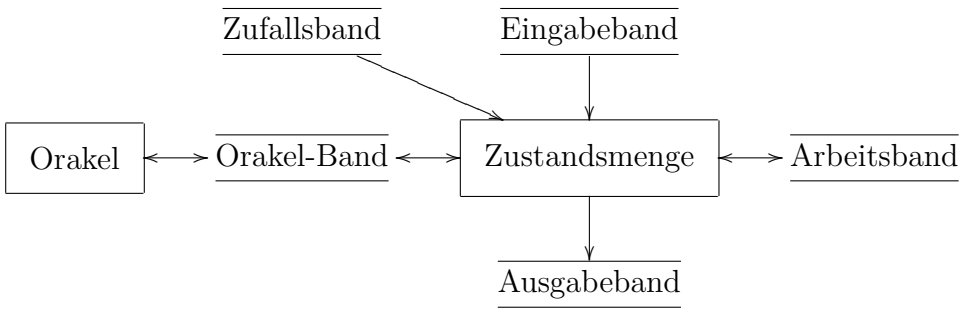

Abbildung 2.2: Orakel-Maschine mit Zufallsband

mit beliebigen Laufzeiten, also insbesondere auch als PT- und PPT-Maschinen.

Zum Abschluss des Kapitels noch ein Beispiel, das die Notation und ihre Stärke verdeutlichen soll:

Beispiel 2.3.3. *Es sei A ein PPT-Algorithmus, der bei Eingabe von (a, b) jeweils mit einer Wahrscheinlichkeit von $\frac{1}{2}$ entweder $a + b$ oder $a - b$ ausgibt. Er entscheidet dies anhand des ersten Bits auf seinem Zufallsband, bei 1 gibt er $a + b$ aus, bei 0 ist die Ausgabe $a - b$. Es ist in diesem Fall also $\Omega = \{0, 1\}$ und Ω^∞ die Menge aller binären Folgen, wobei uns aber nur das erste Glied der Folge interessiert.*

Dann ist einerseits

$$P_{r,s}(A_r(3, 2) = A_s(5, 4)) = \frac{1}{4},$$

aber andererseits

$$P_r(A_r(3, 2) = A_r(5, 4)) = \frac{1}{2}.$$

Ohne die Schreibweise mit Indizes wäre nicht ganz klar, welche Variante gemeint ist. Außerdem könnten mit dieser Notation auch zwei Algorithmen dasselbe Zufallsband benutzen. Wir schauen uns hier noch einmal den stochastischen Hintergrund an. Im ersten Fall ist der Grundraum gleich $\Omega_1 \times \Omega_2$ mit $\Omega_1 = \Omega_2 = \Omega^\infty$ und (das Zufallsband) r die Projektion vom Grundraum auf Ω_1 sowie (das Zufallsband) s die Projektion auf Ω_2. Im zweiten Fall ist der Grundraum nur Ω^∞ und r die Projektion des Grundraums auf sich selbst, also einfach die Identität.

Um die volle Mächtigkeit der Notation auszunutzen noch ein etwas komplexerer Sachverhalt:

$$P_{r,s,t}(x \cdot y + 1 = A_t(x, y) : x = A_r(3, 2), y = A_s(2, 1)) = \frac{3}{8}.$$

Hierbei sind x und y Hilfsvariablen, ohne die der gleiche Ausdruck sehr unübersichtlich geworden wäre. Wenn Algorithmen benutzt werden, deren Ausgabe nicht nur ein einzelner Wert ist, sondern aus mehreren Werten besteht, wird die Verwendung von Hilfsvariablen sogar noch nützlicher.

2.4 Eigenschaften von Zufallsvariablen

Wir haben es häufig mit Zufallsvariablen zu tun, die von PPT-Algorithmen verarbeitet werden. Hierbei ist es meistens nicht vonnöten, dass verschiedene Zufallsvariablen wirklich identisch verteilt oder stochastisch unabhängig sind. Es reicht, dass die PPT-Algorithmen keinen Unterschied zu Zufallsvariablen feststellen können, die wirklich diese Eigenschaften haben. Man nennt diese dann *rechnerisch ununterscheidbar* bzw. *rechnerisch unabhängig*.

Definition 2.4.1. *Zwei Folgen von Zufallsvariablen $(X_n)_{n\in\mathbb{N}}$ und $(Y_n)_{n\in\mathbb{N}}$ sind genau dann* rechnerisch ununterscheidbar*, wenn für jeden PPT-Algorithmus D gilt, dass*

$$|P(D(1^n, X_n) = 1) - P(D(1^n, Y_n) = 1)|$$

vernachlässigbar in n ist. Wir schreiben auch kurz $(X_n) \sim_c (Y_n)$.

Auch die rechnerische Ununterscheidbarkeit ist (wie die identische Verteiltheit) eine Äquivalenzrelation, wobei die Transitivität hierbei aus Lemma 2.2.9 folgt. Außerdem folgt aus derselben Verteilung zweier Folgen von Zufallsvariablen auch deren rechnerische Ununterscheidbarkeit. Eine Frage, die sich vielleicht stellt, ist, ob sich an der rechnerischen Ununterscheidbarkeit etwas ändert, wenn man dem Unterscheider mehrere stochastisch unabhängige Werte/Kopien der Zufallsvariablen zu Verfügung stellt bzw. ihm erlaubt die Zufallsvariablen öfter auszuwerten/abzufragen. Diese Frage kann mit „nein“ beantwortet werden (siehe Theorem 3.2.6 in [Gol03]), sofern es sich um *PTC Folgen* handelt. Dies sind Folgen von Zufallsvariablen, die auch von einem PPT-Algorithmus erzeugt werden könnten.

Definition 2.4.2. *Eine Folge $(X_n)_{n\in\mathbb{N}}$ von Zufallsvariablen heißt genau dann* in polynomialer Zeit konstruierbar *oder kurz* PTC *(engl. polynomial-time-constructible), wenn es einen PPT-Algorithmus S gibt mit $S(1^n) \sim X_n$ für alle $n \in \mathbb{N}$.*

Dies ist in unserem Fall allerdings keine wirkliche Einschränkung, da ohnehin alle auftauchenden Variablen entweder uniform verteilt sind oder von PPT-Algorithmen berechnet werden.

Definition 2.4.3. *Zwei Folgen von Zufallsvariablen $(X_n)_{n\in\mathbb{N}}$ und $(Y_n)_{n\in\mathbb{N}}$ sind genau dann* rechnerisch unabhängig*, wenn eine Folge $(\tilde{X}_n, \tilde{Y}_n)_{n\in\mathbb{N}}$ von Paaren aus Zufallsvariablen existiert, wobei $\tilde{X}_n$ und $\tilde{Y}_n$ für alle $n \in \mathbb{N}$ stochastisch unabhängig sind und die Folgen $(X_n, Y_n)_{n\in\mathbb{N}}, (\tilde{X}_n, \tilde{Y}_n)_{n\in\mathbb{N}}$ rechnerisch ununterscheidbar.*

Aus der stochastischen Unabhängigkeit zweier Folgen von Zufallsvariablen folgt auch deren rechnerische Unabhängigkeit. Die beiden hier definierten Begriffe verhalten sich im Prinzip genauso wie ihre entsprechenden Gegenstücke aus der Stochastik (identisch verteilt, stochastisch unabhängig). Da wir diese Eigenschaften später für einige Begründungen heranziehen werden, insbesondere im Beweis zu Satz 6.3.3, und der Begriff „rechnerisch unabhängig“ in dieser Arbeit neu eingeführt wurde, folgen hier ein paar Sätze, die diese Eigenschaften genauer beleuchten:

Satz 2.4.4. *Es seien X, Y, X', Y' Zufallsvariablen, wobei X und Y sowie X' und Y' stochastisch unabhängig sind. Wenn $X \sim X'$ und $Y \sim Y'$ gilt, dann auch $(X, Y) \sim (X', Y')$.*

Beweis. Für alle x und y gilt

$$\begin{aligned} P((X,Y) = (x,y)) &= P(X = x \wedge Y = y) \\ &= P(X = x) \cdot P(Y = y) \\ &= P(X' = x) \cdot P(Y' = y) \\ &= P(X' = x \wedge Y' = y) \\ &= P((X',Y') = (x,y)). \end{aligned}$$

□

Satz 2.4.5. *Es seien $(X_n)_{n\in\mathbb{N}}, (Y_n)_{n\in\mathbb{N}}, (X'_n)_{n\in\mathbb{N}}, (Y'_n)_{n\in\mathbb{N}}$ PTC Folgen von Zufallsvariablen, wobei jeweils X_n und Y_n sowie X'_n und Y'_n stochastisch unabhängig sind für alle $n \in \mathbb{N}$. Wenn $(X_n) \sim_c (X'_n)$ und $(Y_n) \sim_c (Y'_n)$ gilt, dann auch $(X_n, Y_n) \sim_c (X'_n, Y'_n)$.*

Beweis. Wir können für den Beweis davon ausgehen, dass X_n, X'_n, Y_n, Y'_n paarweise stochastisch unabhängig sind für alle $n \in \mathbb{N}$. Ansonsten führen wir noch zwei PTC Folgen $(\tilde{X}_n)_{n\in\mathbb{N}}, (\tilde{Y}_n)_{n\in\mathbb{N}}$ aus stochastisch unabhängigen (voneinander und den restlichen Variablen) Zufallsvariablen ein mit $X_n \sim \tilde{X}_n, Y_n \sim \tilde{Y}_n$ und zeigen dann $(X_n, Y_n) \sim_c (\tilde{X}_n, \tilde{Y}_n) \sim_c (X'_n, Y'_n)$.

Wir zeigen zunächst, dass $(X_n, Y_n) \sim_c (X'_n, Y_n)$ gilt. Die eigentliche Aussage folgt dann aus Symmetriegründen und der Transitivität von $\sim_c$. Angenommen die Aussage wäre falsch, dann gäbe es einen PPT-Algorithmus D, so dass

$$|P(D(1^n, X_n, Y_n) = 1) - P(D(1^n, X'_n, Y_n) = 1)|$$

nicht vernachlässigbar in n wäre. Es sei nun S ein PPT-Algorithmus mit $S(1^n) \sim Y_n$ und D' der Algorithmus mit $D'(1^n, x) = D(1^n, x, S(1^n))$. Dieser ist wieder ein PPT-Algorithmus und wenn M_n die Menge aller Bilder von Y_n bezeichnet (die nach Voraussetzung abzählbar ist), dann ist

$$\begin{aligned}
&|P(D'(1^n, X_n) = 1) - P(D'(1^n, X'_n) = 1)| \\
&= |P(D(1^n, X_n, S(1^n)) = 1) - P(D(1^n, X'_n, S(1^n)) = 1)| \\
&= \Bigg| \sum_{y \in M_n} P(D(1^n, X_n, y) = 1) \cdot P(S(1^n) = y) \\
&\quad - \sum_{y \in M_n} P(D(1^n, X'_n, y) = 1) \cdot P(S(1^n) = y) \Bigg| \\
&= \left| \sum_{y \in M_n} P(D(1^n, X_n, y) = 1) \cdot P(Y_n = y) - \sum_{y \in M_n} P(D(1^n, X'_n, y) = 1) \cdot P(Y_n = y) \right| \\
&= |P(D(1^n, X_n, Y_n) = 1) - P(D(1^n, X'_n, Y_n) = 1)|,
\end{aligned}$$

was aber wegen $(X_n) \sim_c (X'_n)$ vernachlässigbar in n ist und somit einen Widerspruch darstellen würde. □

Satz 2.4.6. *Es seien $(X_n)_{n\in\mathbb{N}}, (Y_n)_{n\in\mathbb{N}}, (X'_n)_{n\in\mathbb{N}}, (Y'_n)_{n\in\mathbb{N}}$ PTC Folgen von Zufallsvariablen, wobei jeweils X_n und Y_n sowie X'_n und Y'_n rechnerisch unabhängig sind für alle $n \in \mathbb{N}$. Wenn $(X_n) \sim_c (X'_n)$ und $(Y_n) \sim_c (Y'_n)$ gilt, dann auch $(X_n, Y_n) \sim_c (X'_n, Y'_n)$.*

Beweis. Per Definition gibt es Folgen $(\tilde{X}_n)_{n\in\mathbb{N}}, (\tilde{Y}_n)_{n\in\mathbb{N}}, (\tilde{X}'_n)_{n\in\mathbb{N}}, (\tilde{Y}'_n)_{n\in\mathbb{N}}$ von Zufallsvariablen, wobei jeweils $\tilde{X}_n$ und $\tilde{Y}_n$ sowie $\tilde{X}'_n$ und $\tilde{Y}'_n$ stochastisch unabhängig sind und außerdem $(\tilde{X}_n, \tilde{Y}_n) \sim_c (X_n, Y_n)$ sowie $(\tilde{X}'_n, \tilde{Y}'_n) \sim_c (X'_n, Y'_n)$ gilt. Wir können außerdem annehmen, dass $(\tilde{X}_n)_{n\in\mathbb{N}}, (\tilde{Y}_n)_{n\in\mathbb{N}}, (\tilde{X}'_n)_{n\in\mathbb{N}}, (\tilde{Y}'_n)_{n\in\mathbb{N}}$ PTC sind, denn es gibt z. B. PPT-Algorithmen S, T mit $S(1^n) \sim X_n \sim_c \tilde{X}_n$ und $T(1^n) \sim Y_n \sim_c \tilde{Y}_n$. Deren Ausgaben (von $S(1^n)$ und $T(1^n)$) sind stochastisch unabhängig, womit nach Satz 2.4.5 folgt, dass $(S(1^n), T(1^n)) \sim_c (\tilde{X}_n, \tilde{Y}_n) \sim_c (X_n, Y_n)$ gilt.

Es gilt also $\tilde{X}_n \sim_c X_n \sim_c X'_n \sim_c \tilde{X}'_n$ und $\tilde{Y}_n \sim_c Y_n \sim_c Y'_n \sim_c \tilde{Y}'_n$. Nach Satz 2.4.5 folgt $(\tilde{X}_n, \tilde{Y}_n) \sim_c (\tilde{X}'_n, \tilde{Y}'_n)$ und damit letztlich

$$(X_n, Y_n) \sim_c (\tilde{X}_n, \tilde{Y}_n) \sim_c (\tilde{X}'_n, \tilde{Y}'_n) \sim_c (X'_n, Y'_n).$$

□

Die folgenden Sätze werden später (Beweis zu Satz 6.3.3 und 6.3.4) noch benötigt und beschreiben, dass sich die Verteilung einer in $\{0,1\}^n$ uniform verteilten Zufallsvariablen U durch eine XOR-Verknüpfung mit einer unabhängigen Zufallsvariable X im Wesentlichen nicht verändert, selbst dann nicht, wenn man die gemeinsame Verteilung mit der Zufallsvariable X betrachtet.

Satz 2.4.7. *Es sei $X\colon \Omega \to M$ eine Zufallsvariable und $U\colon \Omega \to \{0,1\}^n$ eine uniform verteilte und von X stochastisch unabhängige Zufallsvariable mit $n \in \mathbb{N}$. Ferner sei $f\colon M \to \{0,1\}^n$ eine beliebige Funktion. Dann gilt*

$$(X, f(X) \oplus U) \sim (X, U).$$

Beweis. Für $x \in M$ und $u \in \{0,1\}^n$ gilt

$$\begin{aligned} P((X, f(X) \oplus U) = (x,u)) &= P(X = x \wedge f(X) \oplus U = u) \\ &= P(X = x \wedge f(x) \oplus U = u) \\ &= P(X = x \wedge U = f(x) \oplus u) \\ &= P(X = x) \cdot P(U = f(x) \oplus u) \\ &= P(X = x) \cdot 2^{-n} \\ &= P(X = x) \cdot P(U = u) \\ &= P(X = x \wedge U = u) \\ &= P((X,U) = (x,u)), \end{aligned}$$

womit dieselbe Verteilung von $(X, f(X) \oplus U)$ und (X,U) gezeigt wäre. □

Satz 2.4.8. *Es sei $(X_n)_{n\in\mathbb{N}}$ eine PTC Folge von M_n-wertigen Zufallsvariablen und $(U_n)_{n\in\mathbb{N}}$ eine von (X_n) rechnerisch unabhängige Folge von $\{0,1\}^{\lambda(n)}$-wertigen Zufallsvariablen, so dass (U_n) rechnerisch ununterscheidbar von einer Folge uniform verteilter $\{0,1\}^{\lambda(n)}$-wertiger Zufallsvariablen ist, wobei $\lambda \in \mathsf{poly}$ ist mit Werten in $\mathbb{N}$. Ferner seien $f_n\colon M_n \to \{0,1\}^{\lambda(n)}$ Funktionen, so dass es einen PT-Algorithmus F gibt mit $f_n(x) = F(1^n, x)$. Dann gilt*

$$(X_n, f_n(X_n) \oplus U_n) \sim_c (X_n, U_n).$$

Beweis. Nach Definition gibt es eine Folge $(\tilde{X}_n, \tilde{U}_n)_{n\in\mathbb{N}}$ von stochastisch unabhängigen Paaren aus Zufallsvariablen mit $(\tilde{X}_n, \tilde{U}_n) \sim_c (X_n, U_n)$. Wir können annehmen, dass diese Zufallsvariablen PTC sind, wie im Beweis zu Satz 2.4.6.

Es sei $(\tilde{U}'_n)_{n\in\mathbb{N}}$ eine Folge uniform verteilter $\{0,1\}^{\lambda(n)}$-wertiger Zufallsvariablen, die jeweils stochastisch unabhängig von allen anderen Variablen sind. Diese Folge ist ebenfalls PTC und es gilt nach Satz 2.4.5, dass $(\tilde{X}_n, \tilde{U}_n) \sim_c (\tilde{X}_n, \tilde{U}'_n)$ gilt, also auch $(X_n, U_n) \sim_c (\tilde{X}_n, \tilde{U}'_n)$.

Es ist dann ebenfalls $(X_n, f_n(X_n) \oplus U_n) \sim_c (\tilde{X}_n, f_n(\tilde{X}_n) \oplus \tilde{U}'_n)$, da sich aus einem Unterscheider D für den einen Fall ein Unterscheider D' für den anderen Fall konstruieren lässt: Man wählt D' so, dass $D'(1^n, x, y) = D(1^n, x, F(1^n, x) \oplus y)$ ist.

Nach Satz 2.4.7 gilt dann $(\tilde{X}_n, f_n(\tilde{X}_n) \oplus \tilde{U}'_n) \sim_c (\tilde{X}_n, \tilde{U}'_n)$ und damit insgesamt

$$(X_n, f_n(X_n) \oplus U_n) \sim_c (\tilde{X}_n, f_n(\tilde{X}_n) \oplus \tilde{U}'_n) \sim_c (\tilde{X}_n, \tilde{U}'_n) \sim_c (X_n, U_n).$$

□

3 Random-Oracle

In diesem Kapitel wird der Begriff des Random-Oracles erklärt, sowie einige Varianten davon und der Zusammenhang zwischen ihnen.

3.1 Definition

Bei vielen kryptographischen Protokollen ist es hilfreich, einem oder mehreren (auch längeren) Werten eine Zufallszahl zuzuordnen, und zwar in einer Weise, dass diese Zuordnung für verschiedene Werte stochastisch unabhängig ist und andere Teilnehmer diese Zuordnung ebenfalls vornehmen bzw. überprüfen können. Damit kann man z. B. interaktive Protokolle zu nicht-interaktiven machen oder auch sonst recht einfache Protokolle aufstellen. Die erste Arbeit in der solch ein Mechanismus explizit benutzt wurde ist [FS87]. Später wurde dieses Vorgehen in [BR93] genau formalisiert. Das Konstrukt, das so etwas ermöglicht, nennt man *Random-Oracle*. Dabei handelt es sich um eine Zufallsfunktion, die beliebige Eingaben akzeptiert und für jede Eingabe zufällige Ausgaben in einem bestimmten Wertebereich liefert. Diese Ausgaben sind für unterschiedliche Eingaben uniform und unabhängig im Wertebereich verteilt, sind für dieselbe Eingabe aber immer gleich. Etwas genauer wählt ein Random-Oracle (für eine endliche Menge M) $\mathcal{R}^M$ zufällig eine Funktion $f\colon \{0,1\}^* \to M$ aus, so dass für eine solche Auswahl $f = \mathcal{R}_r^M$ der Wert $\mathcal{R}_r^M(w) = f(w) \in M$ ist, also der Wert einer zufällig gewählten Funktion. Der Index r wird hierbei analog zu der Notation von probabilistischen Algorithmen benutzt. Man kann $\mathcal{R}^M(w)$ auch als M-wertige Zufallsvariable ansehen, wobei man sich auf die Auswertung der noch auszuwählenden Funktion an einer Stelle beschränkt. Die Wahl der Funktion f soll dabei gemäß einer „uniformen“ Verteilung stattfinden. Allerdings ist die Menge $\{f\colon \{0,1\}^* \to M\}$ der Funktionen von $\{0,1\}^*$ nach M überabzählbar und somit eine uniforme Verteilung nicht ohne weiteres definierbar. Man kann das Problem aber umgehen, indem man fordert, dass die einzelnen Bilder $\mathcal{R}^M(w)$ für verschiedene w jeweils uniform verteilt (und stochastisch unabhängig) sind.

Definition 3.1.1. *Es sei M eine endliche Menge, sowie*

$$\mathcal{R}^M\colon \Omega^\infty \to \{f\colon \{0,1\}^* \to M\}, r \mapsto \mathcal{R}_r^M \qquad \textit{und} \qquad \mathcal{R}^M(w)\colon \Omega^\infty \to M, r \mapsto \mathcal{R}_r^M(w).$$

Die Abbildung $\mathcal{R}^M$ ist genau dann ein Random-Oracle *(für M), wenn $\mathcal{R}^M(w), w \in \{0,1\}^*$ stochastisch unabhängige und uniform verteilte Zufallsvariablen sind.*

Es ist hierbei zu beachten, dass das Argument r für die Abbildung $\mathcal{R}^M$ im Index steht und damit $\mathcal{R}_r^M$ das Bild von r ist. Für $\mathcal{R}_r^M$, was wiederum eine Funktion ist, wird das Argument dann normal in Klammern angegeben, also $\mathcal{R}_r^M(w)$ für das Bild von w (siehe auch Abbildung 3.1). Im Folgenden sei $\mathcal{R}^M$ immer ein Random-Oracle für M. Bei

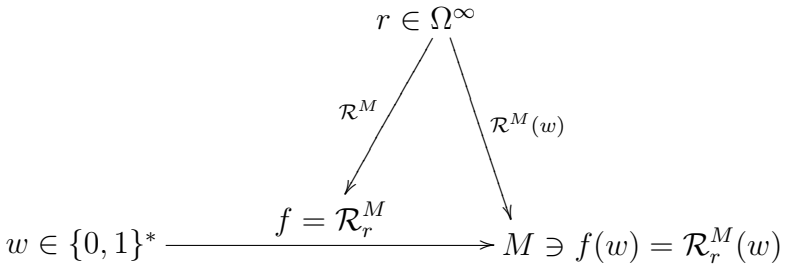

Abbildung 3.1: Random-Oracle

gegebenem $r \in \Omega^\infty$ heißt $\mathcal{R}_r^M$ eine (feste/gegebene) Instanz von $\mathcal{R}^M$. Des Weiteren wird die Abkürzung $\mathcal{R}^l := \mathcal{R}^{\{0,1\}^l}$ benutzt, wobei $l \in \mathbb{N}$ ist.

Wenn wir bei der Betrachtung einer Wahrscheinlichkeit die Notation mit den Indizes nicht benutzen und ein Random-Oracle mehrfach in einem Ausdruck auftaucht, dann ist damit immer die gleiche Instanz eines Random-Oracles gemeint, also z. B.

$$P(\mathcal{R}^l(1) = \mathcal{R}^l(1)) = P_r(\mathcal{R}_r^l(1) = \mathcal{R}_r^l(1)) = 1.$$

Dass Random-Oracles als theoretisches Konstrukt wirklich existieren können und wie man sich diese vorstellen kann, zeigt folgendes Beispiel:

Beispiel 3.1.2. *Es sei $l \in \mathbb{N}, r \in \Omega^\infty$ mit $\Omega = \{0,1\}$ und $w \in \{0,1\}^*$. Wenn man das binäre Wort $1w$ als natürliche Zahl n interpretiert und $\mathcal{R}_r^l(w) := y = y_1 \cdots y_l$ setzt mit $y_i = r_{2^{n-1}-1+i2^n}$, dann ist die dadurch definierte Funktion $\mathcal{R}^l$ ein Random-Oracle.*

Beweis. Angenommen für $i, j, n, m \in \mathbb{N}$ ist

$$\begin{aligned} & 2^{n-1} - 1 + i2^n = 2^{m-1} - 1 + j2^m \\ \Leftrightarrow \qquad & (2i+1)2^{n-1} = (2j+1)2^{m-1}, \end{aligned}$$

dann folgt aufgrund der Eindeutigkeit der Primfaktorzerlegung, dass $i = j$ und $n = m$ sein muss. Für unterschiedliche w gibt $\mathcal{R}^l$ also immer unterschiedliche Folgenglieder von r zurück. Diese sind aber per Definition unabhängig und uniform verteilt. □

3.2 Effiziente Oracle-Transformation

In den meisten Fällen werden Random-Oracles für $\{0,1\}^l$ oder $\mathbb{Z}_p^*$ verwendet, insbesondere auch mit $l = 1$. Es stellt sich die Frage, ob die verschiedenen Arten der Random-Oracles alle gleichwertig sind, oder ob vielleicht manche zu „mächtigeren“ Algorithmen

führen. Es wird zunächst gezeigt, dass es für Bildmengen vom Typ $\{0,1\}^l$ innerhalb gewisser Grenzen im Prinzip keine Rolle spielt wie groß l ist. Allerdings stellt man bei anderen Bildbereichen fest, dass die Random-Oracles dafür nicht ganz gleichwertig sind, aber für die allermeisten Fälle doch zu annähernd „gleichmächtigen“ Algorithmen führen, so dass man sich in der Praxis nicht um die genaue Gestalt des Random-Oracles kümmern muss.

Satz 3.2.1. *Es seien $f, g \in \mathsf{poly}$ PT-Funktionen mit Werten in $\mathbb{N}$, dann existiert ein PT-Orakel-Algorithmus A, so dass zu jedem Random-Oracle $\mathcal{R}^{f(n)}$ und $M = \{0,1\}^{g(n)}$ die Abbildung*

$$B\colon \Omega^\infty \to \{h\colon \{0,1\}^* \to M\}, r \mapsto B_r \quad \textit{mit} \quad B_r(w) = A^{\mathcal{R}_r^{f(n)}}(1^n, w)$$

ein Random-Oracle für M ist.

Beweis. Der Algorithmus $A^{\mathcal{R}_r^{f(n)}}$ verhält sich bei Eingabe von $(1^n, w)$ wie folgt (siehe auch Abbildung 3.2):

1. Setze Zähler $z = 1$.
2. Für $z = \sum_{i=0}^{k} z_i 2^i$ mit $k = \lfloor \log_2(z) \rfloor$ und $z_i \in \{0,1\}$ setze $w_z = z_0 0 z_1 0 \cdots z_k 011 w$.
3. Setze $x_z = \mathcal{R}_r^{f(n)}(w_z)$.
4. Wenn $zf(n) < g(n)$, dann zähle z um eins hoch und springe zu Schritt 2.
5. Setze $x = x_1 \cdots x_z$.
6. Gib die ersten $g(n)$ Bits von x zurück.

Alle Schritte lassen sich effizient (bzgl. $(1^n, w)$) berechnen und der Zähler z wird maximal $g(n)$ mal erhöht, womit auch der gesamte Algorithmus effizient ist. Die Kodierung der w_j ist so gewählt, dass für verschiedene w oder j auch w_j verschieden ist. Damit sind die Rückgaben für verschiedene w als Zufallsvariablen bzgl. r unabhängig und uniform verteilt. □

Für effiziente Orakel-Algorithmen die Eingaben der Länge n verarbeiten und Zugriff auf ein Random-Oracle für $\{0,1\}^{f(n)}$ haben, kann man das Random-Oracle also auch durch eines für $\{0,1\}^{g(n)}$ ersetzen und der Algorithmus bleibt nach entsprechender Abänderung (interner Aufruf des obigen Algorithmus) immer noch effizient. Dies funktioniert insbesondere für konstante Abbildungen f oder g.

Betrachten wir nun eine möglichst allgemeine Transformation. Wir versuchen also mit Hilfe eines PT-Orakel-Algorithmus ein Random-Oracle für N in eines für M zu transformieren, wobei dies nur unter bestimmten Bedingungen möglich sein wird oder mit gewissen Einschränkungen bzgl. der Laufzeit oder der uniformen Verteilung. Am

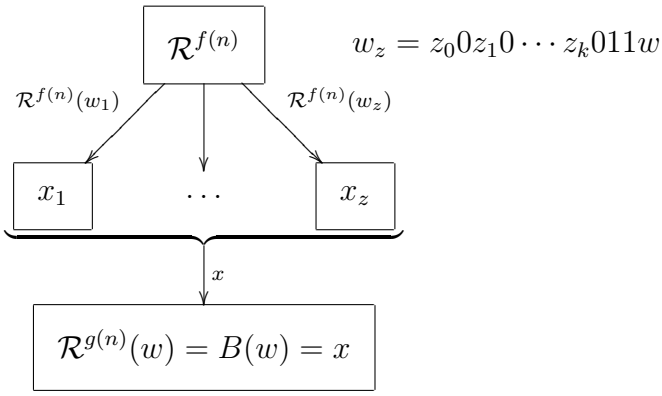

Abbildung 3.2: Einfache Random-Oracle-Transformation

Ende dieses Abschnittes finden sich dazu noch einige Beispiele, die den Satz und seine Folgerungen auf ein paar Random-Oracles für Mengen anwenden, die in der Literatur häufig benutzt werden.

Satz 3.2.2. *Es seien $f, g \in \mathsf{poly}$ PT-Funktionen mit Werten in $\mathbb{N}$ und $n_i = f(i)$ und $m_i = g(i)$ für $i \in I \subseteq \{0,1\}^*$. Ferner seien N_i, M_i nichtleere Mengen mit $d_i = n_i - |N_i| \in \mathbb{N}$ und $e_i = m_i - |M_i| \in \mathbb{N}$. Außerdem seien $\psi_i \colon N_i \to [0, n_i - 1]_{\mathbb{N}}$ injektive Funktionen und $\varphi_i \colon [0, m_i - 1]_{\mathbb{N}} \to M_i \dot\cup \{\varepsilon\}$ Funktionen mit $\varphi_i|_{\varphi_i^{-1}(M_i)} \colon \varphi_i^{-1}(M_i) \to M_i$ bijektiv, so dass es PT-Algorithmen C und D gibt mit $C(i,x) = \psi_i(x)$ und $D(i,x) = \varphi_i(x)$. Dann existiert ein PT-Orakel-Algorithmus A, so dass für jedes Random-Oracle $\mathcal{R}^{N_i}$ die Abbildung $B \colon \Omega^\infty \to \{h \colon \{0,1\}^* \to M_i\}, r \mapsto B_r$ mit $B_r(w) = A^{\mathcal{R}_r^{N_i}}(i,w)$ und A die folgenden Eigenschaften haben, wobei zur Abkürzung $\hat{z}_i := \lceil \log_{n_i}(m_i^2) \rceil$ sei:*

1. *Die Wahrscheinlichkeit, dass der Algorithmus A den Wert ε zurückgibt ist*

$$P_r\left(A^{\mathcal{R}_r^{N_i}}(i,w) = \varepsilon\right) \leq \frac{(e_i+1)}{m_i}\left(\frac{n_i}{n_i - d_i}\right)^{\hat{z}_i} \leq \frac{(e_i+1)}{m_i} \cdot \frac{n_i}{n_i - d_i\hat{z}_i},$$

   wobei die zweite Ungleichung nur für $n_i - d_i\hat{z}_i > 0$ gilt.

2. *Die Rückgabewerte von A sind relativ gleichmäßig verteilt.*

   Für $a \in M_i$ gilt:

$$P_r\left(A^{\mathcal{R}_r^{N_i}}(i,w) = a\right) \leq \left\lfloor \frac{n_i^{\hat{z}_i}}{m_i} \right\rfloor \cdot \frac{1}{|N_i|^{\hat{z}_i}} \leq \frac{1}{m_i}\left(\frac{n_i}{n_i - d_i}\right)^{\hat{z}_i} \leq \frac{1}{m_i} \cdot \frac{n_i}{n_i - d_i\hat{z}_i},$$

   wobei die dritte Ungleichung nur für $n_i - d_i\hat{z}_i > 0$ gilt.

   Für

$$F_{i,k} := \left|\left\{ b \in M_i \;\middle|\; P_r\left(A^{\mathcal{R}_r^{N_i}}(i,w) = b\right) \leq \left\lfloor \frac{n_i^{\hat{z}_i}}{m_i} - k \right\rfloor \cdot \frac{1}{|N_i|^{\hat{z}_i}} \right\}\right|$$

gilt

$$F_{i,k} \leq \frac{\hat{z}_i d_i n_i^{\hat{z}_i - 1}}{k}$$

und insbesondere

$$\frac{F_{i,\overline{k}}}{|M_i|} \leq \frac{2\hat{z}_i d_i}{n_i} \cdot \frac{m_i}{m_i - e_i}$$

für $\overline{k} = \left\lceil \frac{n_i^{\hat{z}_i}}{2m_i} \right\rceil$.

Beweis. Der Algorithmus $A^{\mathcal{R}_r^{N_i}}$ verhält sich bei Eingabe von (i, w) wie folgt (siehe auch Abbildung 3.3):

1. Setze Zähler $z = 1$.
2. Für $z = \sum_{j=0}^{k} z_j 2^j$ mit $k = \lfloor \log_2(z) \rfloor$ und $z_j \in \{0, 1\}$ setze $w_z = z_0 0 z_1 0 \cdots z_k 011 w$.
3. Setze $x'_z = \mathcal{R}_r^{N_i}(w_z)$.
4. Berechne $x_z = \psi_i(x'_z)$.
5. Wenn $n_i^z < m_i^2$, dann zähle z um eins hoch und springe zu Schritt 2.
6. Berechne $x = \prod_{j=1}^{z} x_j n_i^{j-1}$.
7. Wenn $x \geq m_i \lfloor n_i^z / m_i \rfloor$ ist, gib ε zurück und beende den Algorithmus.
8. Berechne $y' = x \bmod m_i$.
9. Berechne $y = \varphi_i(y')$.
10. Gib y zurück.

Alle Schritte lassen sich effizient (bzgl. (i, w)) berechnen und der Zähler z wird $\hat{z}_i \leq 2 \log_2(m_i)$ mal erhöht, womit auch der gesamte Algorithmus effizient ist. Die Kodierung der w_j ist so gewählt, dass für verschiedene w oder j auch w_j verschieden ist, womit die x'_j für $j = 1, \ldots, k$ unabhängig und uniform verteilt in N_i sind.

Im Folgenden sei $z := \hat{z}_i$. Für $a \in [0, n_i - 1]_{\mathbb{N}}$, $1 \leq j \leq z$ und $b \in [0, n_i^z - 1]_{\mathbb{N}}$ ergibt sich damit

$$P_r(x_j = a) = \begin{cases} 0 & \Leftrightarrow a \notin \psi_i(N_i) \\ \frac{1}{|N_i|} & \Leftrightarrow a \in \psi_i(N_i) \end{cases}$$

und

$$P_r(x = b) \in \left\{ 0, \frac{1}{|N_i|^z} \right\}. \qquad (*)$$

Für das weitere Vorgehen ist es hilfreich sich zunächst klar zu machen, dass gemäß Konstruktion von A die Ungleichungen $n_i^z \geq m_i^2$ und $n_i^z \leq n_i m_i^2$ erfüllt sind, also auch

$$m_i \leq \left\lfloor \frac{n_i^z}{m_i} \right\rfloor \leq \frac{n_i^z}{m_i} \leq n_i m_i.$$

Wir fragen uns nun, wann A den Wert ε ausgibt. Dies geschieht in zwei Fällen, zum einen wenn $x \geq m_i \left\lfloor \frac{n_i^z}{m_i} \right\rfloor$ und zum anderen wenn $\varphi_i(y') = \varepsilon$. Wir betrachten zunächst den ersten Fall:

$$P_r\left(x \geq m_i \left\lfloor \frac{n_i^z}{m_i} \right\rfloor\right) \leq \left(n_i^z - m_i \left\lfloor \frac{n_i^z}{m_i} \right\rfloor\right) \cdot \frac{1}{|N_i|^z} \leq \frac{m_i}{|N_i|^z}.$$

Für den zweiten Fall betrachten wir ein $c \in [0, m_i - 1]$:

$$P_r(y' = c) \leq \left\lfloor \frac{n_i^z}{m_i} \right\rfloor \cdot \frac{1}{|N_i|^z} \leq \frac{n_i^z}{m_i} \cdot \frac{1}{|N_i|^z},$$

wobei bemerkt sei, dass wegen $(*)$ die Wahrscheinlichkeit $P_r(y' = c)$ nur ein ganzzahliges Vielfaches von $\frac{1}{|N_i|^z}$ sein kann. Es gilt dann $P_r(\varphi_i(y') = \varepsilon) = e_i \cdot \max\{P_r(y' = c) \mid c \in [0, m_i - 1]\}$. Somit ergibt sich zusammen

$$\begin{aligned} P_r\left(A^{\mathcal{R}_r^{N_i}}(i, w) = \varepsilon\right) &\leq \frac{\frac{n_i^z}{m_i} e_i + m_i}{|N_i|^z} \leq \frac{(e_i + 1) n_i^z}{m_i |N_i|^z} \leq \frac{(e_i + 1) n_i^z}{m_i (n_i - d_i)^z} \\ &\leq \frac{(e_i + 1)}{m_i} \cdot \frac{n_i}{n_i - z d_i}, \end{aligned}$$

wobei die letzte Ungleichung aus der Taylor-Approximation (siehe [Kö95]) folgt und nur für $n_i - z d_i > 0$ gilt.

Es gilt nun weiter

$$P_r\left(A^{\mathcal{R}_r^{N_i}}(i, w) = a\right) \leq \max\{P_r(y' = c) \mid c \in [0, m_i - 1]\} \leq \left\lfloor \frac{n_i^z}{m_i} \right\rfloor \cdot \frac{1}{|N_i|^z},$$

woraus der Rest der Ungleichungen analog zu oben folgt.

Um $F_{i,k}$ zu bestimmen, stellen wir zunächst fest, dass

$$|\{b \in [0, n_i^z - 1] \mid P_r(x = b) = 0\}| \leq z d_i n_i^{z-1}$$

ist. Daraus folgen dann die Ungleichungen für $F_{i,k}$ und $F_{i,\overline{k}}$. □

Wir sehen also, dass man, sofern $d_i, e_i, \hat{z}_i$ nicht zu groß sind, durch die eben beschriebene Transformation aus Random-Oracles für M_i effizient Zufallsfunktionen für N_i berechnen kann, die relativ selten einen Fehler ε ausgeben und deren Werte außerdem relativ gleichmäßig in N_i verteilt sind. Da man solche Fehlerausgaben prinzipiell nicht möchte, kann man aus dem effizienten Algorithmus mit Fehlerausgabe einen Algorithmus machen, der zwar keine Fehler mehr produziert, dafür aber nur noch eine erwartete polynomiale Laufzeit besitzt.

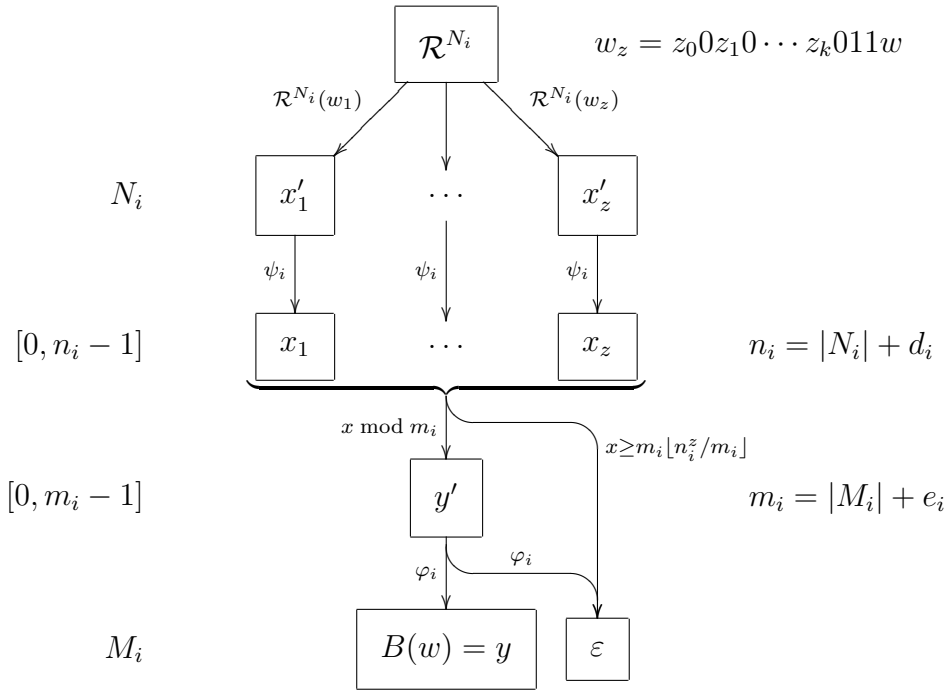

Abbildung 3.3: Random-Oracle-Transformation

Korollar 3.2.3. *Es seien die Bedingungen wie in Satz 3.2.2 gegeben. Dann existiert ein Orakel-Algorithmus A', so dass für jedes Random-Oracle $\mathcal{R}^{N_i}$ die Abbildung $B\colon \Omega^\infty \to \{h\colon \{0,1\}^* \to M_i\}, r \mapsto B_r$ mit $B_r(w) = {A'}^{\mathcal{R}_r^{N_i}}(i, w)$ und A' die folgenden Eigenschaften haben, wobei zur Abkürzung $p_i := P_r\left(A^{\mathcal{R}_r^{N_i}}(i, w) = \varepsilon\right)$ und $\hat{z}_i := \lceil \log_{n_i}(m_i^2) \rceil$ sei:*

1. *Wenn ein $q \in \mathsf{poly}$ existiert mit $\frac{1}{1-p_i} = q(|i|)$, dann ist der Algorithmus A' ein Las-Vegas-Algorithmus, d. h. er gibt niemals ε zurück und hat eine erwartete polynomiale Laufzeit (in etwa $\frac{1}{1-p_i}$ mal die Laufzeit von A).*

2. *Die Rückgabewerte von A' sind relativ gleichmäßig verteilt. Die Wahrscheinlichkeiten sind jeweils $\frac{1}{1-p_i}$ mal größer als in Satz 3.2.2.*

   Für $a \in M_i$ gilt:

$$\begin{aligned}(1 - p_i) \cdot P_r\left({A'}^{\mathcal{R}_r^{N_i}}(i, w) = a\right) &\leq \left\lfloor \frac{n_i^{\hat{z}_i}}{m_i} \right\rfloor \cdot \frac{1}{|N_i|^{\hat{z}_i}} \leq \frac{1}{m_i} \left(\frac{n_i}{n_i - d_i}\right)^{\hat{z}_i} \\ &\leq \frac{1}{m_i} \cdot \frac{n_i}{n_i - d_i \hat{z}_i},\end{aligned}$$

   wobei die dritte Ungleichung nur für $n_i - d_i \hat{z}_i > 0$ gilt.

Für

$$F_{i,k} := \left| \left\{ b \in M_i \;\middle|\; (1-p_i) \cdot P_r\left(A'^{\mathcal{R}_r^{N_i}}(i,w) = b \right) \leq \left\lfloor \frac{n_i^{\hat{z}_i}}{m_i} - k \right\rfloor \cdot \frac{1}{|N_i|^{\hat{z}_i}} \right\} \right|$$

gilt

$$F_{i,k} \leq \frac{\hat{z}_i d_i n_i^{\hat{z}_i - 1}}{k}$$

und insbesondere

$$\frac{F_{i,\overline{k}}}{|M_i|} \leq \frac{2\hat{z}_i d_i}{n_i} \cdot \frac{m_i}{m_i - e_i}$$

für $\overline{k} = \left\lceil \frac{n_i^{\hat{z}_i}}{2m_i} \right\rceil$.

Beweis. Der Algorithmus $A'^{\mathcal{R}_r^{N_i}}$ verhält sich bei Eingabe von (i, w) wie folgt:

1. Setze Zähler $z' = 1$.
2. Für $z' = \sum_{j=0}^{k} z'_j 2^j$ mit $k = \lfloor \log_2(z') \rfloor$ und $z'_j \in \{0,1\}$ setze $w_{z'} = z'_0 0 z'_1 0 \cdots z'_k 011 w$.
3. Berechne $y = A^{\mathcal{R}_r^{N_i}}(i, w_{z'})$.
4. Wenn $y = \varepsilon$, dann zähle z' um eins hoch und springe zu Schritt 2.
5. Gib y zurück.

Für den Erwartungswert von z' bei der Ausgabe also die Anzahl der Aufrufe von A gilt

$$\begin{aligned} E_r(z') &= \sum_{j=1}^{\infty} j p_i^{j-1}(1-p_i) \\ &= \sum_{j=1}^{\infty} j p_i^{j-1} - \sum_{j=1}^{\infty} j p_i^j \\ &= \sum_{j=0}^{\infty} (j+1) p_i^j - \sum_{j=0}^{\infty} j p_i^j \\ &= \sum_{j=0}^{\infty} p_i^j \\ &= \frac{1}{1-p_i}, \end{aligned}$$

wobei die letzte Gleichung gemäß der geometrischen Reihe (siehe [Kö95]) gilt. Da A effizient ist und im Mittel nur $\frac{1}{1-p_i}$ mal aufgerufen wird, ergibt sich im Mittel auch wieder eine polynomiale Laufzeit.

Wir betrachten nun die Ausgabe-Verteilungen der Algorithmen A und A' bei Eingabe von (i, w), die beide nicht von w abhängen, da für verschiedene w verschiedene Anfragen an das Random-Oracle $\mathcal{R}^{N_i}$ gestellt werden und dieses wiederum für verschiedene Anfragen unabhängige und in N_i uniform verteilte Werte liefert. Wir schreiben zur Abkürzung $A'(w) := A'^{\mathcal{R}_r^{N_i}}(i, w)$ und $A(w) := A^{\mathcal{R}_r^{N_i}}(i, w)$. Es ist dann für jedes i

$$\begin{aligned}
P_r(A'(w) = b) &= \sum_{z'=1}^{\infty} P_r\left(A(w_1) = \varepsilon \wedge \cdots \wedge A(w_{z'-1}) = \varepsilon \wedge A(w_{z'}) = b\right) \\
&= \sum_{z'=1}^{\infty} P_r\left(A(w_1) = \varepsilon\right) \cdots P_r\left(A(w_{z'-1}) = \varepsilon\right) \cdot P_r\left(A(w_{z'}) = b\right) \\
&= \sum_{z'=1}^{\infty} p_i^{z'-1} \cdot P_r\left(A(w) = b\right) \\
&= \sum_{z'=0}^{\infty} p_i^{z'} \cdot P_r\left(A(w) = b\right) \\
&= \frac{1}{1-p_i} \cdot P_r\left(A(w) = b\right).
\end{aligned}$$

Das bedeutet, dass die Ausgabe-Wahrscheinlichkeiten von A' um das $\frac{1}{1-p_i}$-fache größer sind als die von A, woraus die restlichen Behauptungen des Korollars folgen. □

Sobald man die Mengen N_i ohne Lücken auf Intervalle abbilden kann, also $d_i = 0$ gilt, erhält man durch die obige Transformation eine Zufallsvariable die uniform verteilt auf M_i ist, also ein Random-Oracle.

Korollar 3.2.4. *Unter den Bedingungen von Korollar 3.2.3 und wenn $d_i = 0$ gilt, ist die dortige Abbildung B ein Random-Oracle für M_i.*

Beweis. Nach Eigenschaft 2 ist zum einen

$$(1 - p_i) \cdot P_r\left(A'^{\mathcal{R}_r^{N_i}}(i, w) = a\right) \leq \left\lfloor \frac{n_i^{\hat{z}_i}}{m_i} \right\rfloor \cdot \frac{1}{|N_i|^{\hat{z}_i}}$$

und zum anderen

$$F_{i,1} \leq \frac{\hat{z}_i d_i n_i^{\hat{z}_i - 1}}{k} = 0,$$

woraus folgt, dass für alle $b \in M_i$ gilt:

$$(1 - p_i) \cdot P_r\left(A'^{\mathcal{R}_r^{N_i}}(i, w) = b\right) > \left\lfloor \frac{n_i^{\hat{z}_i}}{m_i} - 1 \right\rfloor \cdot \frac{1}{|N_i|^{\hat{z}_i}}.$$

Da $(1-p_i) \cdot P_r\left(A'^{\mathcal{R}_r^{N_i}}(i, w) = b\right) = P_r\left(A^{\mathcal{R}_r^{N_i}}(i, w) = b\right)$ gemäß dem Beweis zu Satz 3.2.2 und Korollar 3.2.3 nur ein ganzzahliges Vielfaches von $\frac{1}{|N_i|^{\hat{z}_i}}$ sein kann, folgt

$$(1 - p_i) \cdot P_r\left(A'^{\mathcal{R}_r^{N_i}}(i, w) = b\right) = \left\lfloor \frac{n_i^{\hat{z}_i}}{m_i} \right\rfloor \cdot \frac{1}{|N_i|^{\hat{z}_i}},$$

womit die Ausgabe von A' uniform verteilt ist. □

Korollar 3.2.5. *Unter den Bedingungen von Korollar 3.2.4 und wenn $e_i \leq cm_i - 1$ ist für $0 \leq c < 1$, ist der Algorithmus A' ein Las-Vegas-Algorithmus.*

Beweis. Nach Eigenschaft 1 aus Satz 3.2.2 ist

$$p_i \leq \frac{e_i + 1}{m_i} \left(\frac{n_i}{n_i - d_i}\right)^{\hat{z}_i} \leq \frac{cm_i - 1 + 1}{m_i} \left(\frac{n_i}{n_i}\right)^{\hat{z}_i} \leq \frac{cm_i}{m_i} = c,$$

womit die Aussage nach Eigenschaft 1 aus Korollar 3.2.3 folgt. □

Nachdem die Frage der uniformen Verteilung beantwortet ist, bleibt noch zu klären, wann diese Transformation effizient durchführbar ist und nicht nur in erwarteter polynomialer Zeit.

Korollar 3.2.6. *Es seien die Bedingungen von Satz 3.2.2 gegeben. Außerdem gelte $d_i = e_i = 0$ und m_i habe nur Primteiler, die auch n_i teilen, d. h. es gibt $t_i \in \mathbb{N}$ mit $m_i \mid n_i^{t_i}$. Dann lässt sich der Algorithmus A so abändern, dass er weiterhin effizient ist und die Abbildung B ein Random-Oracle für M_i ist.*

Beweis. Den Algorithmus A kann man wie folgt abändern: Schritt 5: Wenn $m_i \nmid n^z$, dann zähle z um eins hoch und springe zu Schritt 2.

Damit ist x uniform verteilt in $[0, n_i^z - 1]$, also auch y' in $[0, m_i - 1]$ und damit y in M_i.

Die Anzahl der Wiederholungen z ist hierbei durch $\log_2(m_i)$ beschränkt, da die größtmögliche Potenz einer Primzahl in m_i die der 2 ist. □

Der Satz 3.2.1 ist ein Spezialfall hiervon. Die Aussage lässt sich auch umkehren, also wenn $|M_i|$ einen Primteiler hat, der nicht $|N_i|$ teilt, kann es keine effiziente Random-Oracle-Transformation von N_i nach M_i geben.

Korollar 3.2.7. *Es seien die Bedingungen von Satz 3.2.2 gegeben. Außerdem habe $|M_i|$ einen Primteiler p, der nicht $|N_i|$ teilt. Dann gibt es keinen effizienten Algorithmus A, so dass die Abbildung B ein Random-Oracle für M_i ist.*

Beweis. Angenommen es gäbe einen solchen Algorithmus A, dann könnte man also mit endlich vielen (nämlich z) Aufrufen von $\mathcal{R}^{N_i}$ ein Random-Oracle B für M_i produzieren. Wenn man nun die Elemente von M_i durchnummeriert also $M_i = \{a_1, \ldots, a_{|M_i|}\}$ und die Teilmenge $U = \{a_1, \ldots, a_t\}$ mit $t = |M_i|/p$ betrachtet, dann ergibt sich $P_r(B_r(w) \in U) = 1/p$. Dieses Ereignis muss sich also irgendwie als Vereinigung von Elementarereignissen mit der Wahrscheinlichkeit $1/|N_i|^z$ schreiben lassen, da man nur z unabhängige uniform verteilte Zufallsvariablen über N_i zur Verfügung hat und die Wahrscheinlichkeit der daraus resultierenden Elementarereignisse gerade $(1/|N_i|)^z$ ist. Damit muss das gesuchte Ereignis ein Vielfaches von $1/|N_i|^z$ sein, was wegen $p \nmid |N_i|$ aber nicht geht. □

Es wurden in diesem Abschnitt also Kriterien dargestellt, wann eine Transformation zu einem Random-Oracle führt und wann diese Transformation effizient durchführbar ist. Zur Verdeutlichung hier noch einige Beispiele mit ein paar Random-Oracles für Mengen, die in der Literatur häufig benutzt werden.

Beispiel 3.2.8. *Im Folgenden werden einige Transformationen betrachtet und welche Aussagen die obigen Sätze dazu treffen. Es seien hierbei* $f, g \in \mathsf{poly}$ *Funktionen mit Werten in* $\mathbb{N}$.

- *Für* $\mathcal{R}^{\{0,1\}^{f(i)}} \to \mathcal{R}^{\{0,1\}^{g(i)}}$ *ist nach Satz 3.2.1 bzw. Korollar 3.2.6 wegen* $|N_i| = n_i = 2^{f(i)}, |M_i| = m_i = 2^{g(i)}$ *eine effiziente Random-Oracle-Transformation möglich.*
- *Insbesondere gilt dies für* $f = 1$ *oder* $g = 1$.
- *Für* $\mathcal{R}^{\{0,1\}^{f(i)}} \to \mathcal{R}^{\mathbb{Z}_{q_i}}$ *mit Primzahlen* $q_i \neq 2$ *ist nach Korollar 3.2.4 zwar eine Random-Oracle-Transformation möglich, aber wegen* $|N_i| = n_i = 2^{f(i)}, |M_i| = m_i = q_i \neq 2$ *und Korollar 3.2.7 keine effiziente. Die Transformation läuft gemäß Korollar 3.2.5 allerdings mit erwarteter polynomialer Laufzeit (Las-Vegas-Algorithmus).*
- *Für* $\mathcal{R}^{\{0,1\}^{f(i)}} \to \mathcal{R}^{\mathbb{Z}^*_{q_i}}$ *mit Primzahlen* $q_i \neq 2$ *gilt das Gleiche, sofern* $q_i - 1$ *mindestens einen Primteiler besitzt, der ungleich* 2 *ist, ansonsten ist die Transformation sogar effizient durchführbar.*
- *Für* $\mathcal{R}^{\{0,1\}^{f(i)}} \to \mathcal{R}^{\mathbb{Z}_{m_i}}$ *mit einem RSA-Modul* $m_i = q_i q_i'$, *verschiedenen Primzahlen* q_i, q_i' *und bekannter oder unbekannter Faktorisierung ist nach Korollar 3.2.4 zwar eine Random-Oracle-Transformation möglich, aber wegen* $|N_i| = n_i = 2^{f(i)}, 2 \nmid |M_i| = m_i = q_i q_i'$ *und Korollar 3.2.7 keine effiziente. Die Transformation läuft gemäß Korollar 3.2.5 allerdings mit erwarteter polynomialer Laufzeit (Las-Vegas-Algorithmus).*
- *Für* $\mathcal{R}^{\{0,1\}^{f(i)}} \to \mathcal{R}^{\mathbb{Z}^*_{m_i}}$ *mit einem RSA-Modul* $m_i = q_i q_i'$, *verschiedenen Primzahlen* q_i, q_i' *und bekannter oder unbekannter Faktorisierung gilt das Gleiche, sofern* $q_i - 1$ *oder* $q_i' - 1$ *mindestens einen Primteiler besitzt, der ungleich* 2 *ist, ansonsten ist die Transformation (bei dann bekannter bzw. berechenbarer Faktorisierung) sogar effizient durchführbar. Hierbei ist* $|M_i| = (q_i - 1)(q_i' - 1) \neq m_i$, *also* $e_i = q_i + q_i' - 1 \leq \frac{5}{6} m_i - 1$, *womit Korollar 3.2.5 auch weiterhin anwendbar ist.*
- *Für* $\mathcal{R}^{\mathbb{Z}_{q_i}} \to \mathcal{R}^{\{0,1\}^{g(i)}}$, $\mathcal{R}^{\mathbb{Z}^*_{q_i}} \to \mathcal{R}^{\{0,1\}^{g(i)}}$ *und* $\mathcal{R}^{\mathbb{Z}_{n_i}} \to \mathcal{R}^{\{0,1\}^{g(i)}}$ *mit einem RSA-Modul* $n_i = q_i q_i'$, *verschiedenen Primzahlen* q_i, q_i' *und bekannter oder unbekannter Faktorisierung gilt das Gleiche wie für die umgekehrten Richtungen.*
- *Für* $\mathcal{R}^{\mathbb{Z}^*_{n_i'}} \to \mathcal{R}^{\{0,1\}^{g(i)}}$ *mit einem RSA-Modul* $n_i' = q_i q_i'$, *verschiedenen Primzahlen* q_i, q_i' *und bekannter Faktorisierung ist nach Korollar 3.2.4 zwar eine Random-Oracle-Transformation möglich, aber wegen* $2 \nmid |N_i| = n_i = (q_i - 1)(q_i' - 1), |M_i| =$

$m_i = 2^{g(i)}$ und Korollar 3.2.7 keine effiziente, sofern nicht $q_i - 1$ und $q_i' - 1$ Potenzen von 2 sind. Die Transformation läuft gemäß Korollar 3.2.5 allerdings mit erwarteter polynomialer Laufzeit (Las-Vegas-Algorithmus).

- *Für $\mathcal{R}^{\mathbb{Z}^*_{n_i'}} \to \mathcal{R}^{\{0,1\}^{g(i)}}$ mit einem RSA-Modul $n_i' = q_i q_i'$, verschiedenen Primzahlen q_i, q_i' und unbekannter Faktorisierung ist $n_i \neq |N_i| = |\mathbb{Z}^*_{n_i'}|$ bzw. $d_i \neq 0$, da sonst die Faktorisierung bekannt wäre. Über eine Random-Oracle-Transformation lassen sich damit leider nur die Ergebnisse aus Satz 3.2.2 und Korollar 3.2.7 verwenden. Sofern also nicht $q_i - 1$ und $q_i' - 1$ Potenzen von 2 sind, ist keine effiziente Transformation möglich. Es lässt sich vermutlich auch keine exakte uniforme Verteilung erreichen, sofern man zumindest auf eine erwartete polynomiale Laufzeit Wert legt.*

4 Hashfunktionen

Hashfunktionen spielen in der modernen Kryptographie eine große Rolle, da sie in vielen Protokollen benutzt werden. Die berühmtesten Vertreter sind wohl digitale Signaturen und Zertifikate. Hierbei wird eine Hashfunktion benutzt, um zu einem (langen) Datensatz einen (kurzen) „Fingerabdruck“ zu erzeugen. Dieser soll einerseits möglichst zufällig wirken, so dass man ihn nicht „beeinflussen“ kann, andererseits soll er aber eindeutig aus dem Datensatz berechnet werden können.

Im ersten Abschnitt werden bereits bekannte Varianten von Hashfunktionen dargestellt, die im zweiten Abschnitt durch zwei neue Varianten ergänzt werden. Sowohl die bereits bekannten als auch die neuen Hashfunktionen werden schließlich im dritten Abschnitt miteinander verglichen und der „Stärke“ nach angeordnet, soweit dies möglich ist.

4.1 Definitionen

Wir schauen uns in diesem Abschnitt die bereits bekannten Varianten von Hashfunktionen an, beschränken uns dabei allerdings auf ein paar grundlegende Varianten, da die anderen für diese Arbeit nicht weiter von Interesse sind.

Zunächst betrachten wir eine allgemeine Definition von Hashfunktionen, die durch die verschiedenen Varianten dann weiter spezifiziert wird.

Definition 4.1.1. *Es sei* $\ell\colon \mathbb{N} \to \mathbb{N}$*, dann ist* $\{h_i\colon \{0,1\}^* \to \{0,1\}^{\ell(|i|)} \mid i \in \{0,1\}^*\}$ *oder kurz* $\{h_i\}$ *genau dann eine* Familie von Hashfunktionen*, wenn es einen PPT-Algorithmus* I *gibt, so dass folgende Aussagen gelten:*

Indizierung (technisch): *Die Ausgabelänge* $|I(1^n)|$ *von* I *ist eine Funktion* $\iota\colon \mathbb{N} \to \mathbb{N}, n \mapsto |I(1^n)|$ *von* n *und es kann* 1^n *in polynomialer Zeit aus* $I(1^n)$ *berechnet werden.*

Effiziente Berechnung: *Es existiert ein PT-Algorithmus* H *mit* $H(i,x) = h_i(x)$ *für jedes* i *im Bildbereich von* I *und* $x \in \{0,1\}^*$*.*

Die Funktion I *wird* Index-Algorithmus *genannt. Außerdem definieren wir die Hashlänge* $\lambda\colon \mathbb{N} \to \mathbb{N}, n \mapsto \ell(\iota(n))$*.*

Wir werden als Abkürzung für den Begriff „Familie von Hashfunktionen“ auch den Ausdruck *Hashfamilie* benutzen. Die Hashfunktionen, wie sie hier definiert sind, bilden

beliebig lange Zeichenketten auf Zeichenketten einer festen Länge ab. Die Urbilder nennt man auch oft Nachrichten. Allerdings fehlt bei dieser Definition jegliche Forderung nach Ähnlichkeit der Hashwerte mit zufälligen Werten. Solche allgemeinen Hashfunktionen finden trotzdem in der Informatik Anwendung, sofern sie die Bilder einigermaßen gleichmäßig über den Bildbereich verteilen. Die Forderung nach Ähnlichkeit mit Zufallswerten bzw. der nicht Beeinflussbarkeit findet sich in den nachfolgenden Definitionen sowie in Abschnitt 4.2.

Zunächst die *Einweg-Hashfunktionen*. Bei diesen fordert man, dass zu einem gegeben Bild nicht effizient das zugehörige Urbild bestimmt werden kann.

Definition 4.1.2. *Eine Familie von Hashfunktionen* $\{h_i\}$ *besitzt genau dann die* Einweg-Eigenschaft *bzgl.* S*, wenn gilt:*

Sampling: *Es ist* S *ein PPT-Algorithmus, der bei Eingabe von* i *einen Wert* $x \in \{0,1\}^*$ *ausgibt.*

Einweg: *Für jeden PPT-Algorithmus* A *gilt:*

$$P(h_i(x) = h_i(y) : i = I(1^n), x = S(i), y = A(i, h_i(x)))$$

ist vernachlässigbar in n*.*

Die Einweg-Eigenschaft hängt hier stark vom Sampling-Algorithmus S ab. Eine Hashfamilie kann die Einweg-Eigenschaft bzgl. einem Sampling-Algorithmus S besitzen, aber für einen anderen Sampling-Algorithmus S' nicht. Wenn S z. B. immer nur die Zeichenkette 0 ausgibt, dann kann A mit der Ausgabe 0 immer die Einweg-Eigenschaft brechen.

Die nächste Art von Hashfunktionen nennt sich *kollisionsresistent*, was bedeutet, dass es nicht effizient möglich ist, zwei verschiedene Urbilder zu finden, die den gleichen Hashwert erzeugen.

Definition 4.1.3. *Eine Familie von Hashfunktionen* $\{h_i\}$ *ist genau dann* kollisionsresistent*, wenn für jeden PPT-Algorithmus* A *gilt:*

$$P(h_i(x) = h_i(y) \wedge x \neq y : i = I(1^n), (x, y) = A(i))$$

ist vernachlässigbar in n*.*

Es gibt noch eine dritte Variante von Hashfunktionen, die häufiger benutzt wird, nämlich die sogenannten *universellen Einweg-Hashfunktionen*. Diese sind eine „schwächere“ Variante der kollisionsresistenten Hashfunktionen, d. h. jede kollisionsresistente Familie von Hashfunktionen besitzt auch die universelle Einweg-Eigenschaft (siehe Abschnitt 6.4.3 in [Gol04]). Sie sind in dieser Arbeit aber nicht von weiterem Interesse.

Einen etwas anderen Ansatz haben die sogenannten *perfekten Einweg-Hashfunktionen* aus [CMR98]. Diese wurden zunächst in [Can97] unter dem Begriff *Oracle-Hashing* eingeführt. Bei der Definition dieser Hashfunktionen geht es im Wesentlichen darum, dass

die Hashwerte keine Informationen über die Urbilder beinhalten bzw. preisgeben. Dabei haben die Autoren verschiedene Varianten vorgestellt, die verschiedene Sicherheitsniveaus haben. Wir werden in dieser Arbeit nur eine (zum besseren Vergleich, an diese Arbeit angepasste) Variante dieser Art von Hashfunktionen betrachten, da die anderen zu ähnlichen Ergebnissen führen, die aber hier nicht von weiterem Interesse sind. Diese Hashfunktionen werden etwas anders benutzt als die bisherigen. Bei jeder Berechnung eines Hashwertes wird zunächst eine Zufallszahl r ausgewählt und diese dann zusammen mit der Nachricht bzw. dem Urbild x verrechnet und mit ausgegeben. Man erhält damit bei jedem Aufruf der Hashfunktion ein anderes Wertepaar $(r, h(xr))$. Um nun zu überprüfen, ob dieser Hashwert auch zu dem Urbild passt, muss man nur wieder $h(xr)$ berechnen. Diese Technik wird unter anderem häufig benutzt, um Vorberechnungen eines Angreifers zu verhindern, z. B. beim Berechnen von Hashwerten von Passwörtern. Den zufälligen Wert r nennt man in diesem Fall Salt.

Definition 4.1.4. *Eine Familie von Hashfunktionen $\{h_i\}$ besitzt genau dann die d-wertige* perfekte Einweg-Eigenschaft *bzgl. S, wenn gilt:*

Zufallsbereich: *Es existiert ein PPT-Algorithmus R, der bei Eingabe von 1^n die Elemente in $R(1^n) \subset \{0,1\}^*$ alle mit gleicher Wahrscheinlichkeit zurückgibt.*

Kollisionsresistenz: *Für jeden PPT-Algorithmus A gilt, dass*

$$P(h_i(xr) = h_i(yr) \wedge x \neq y : i = I(1^n), (x, y, r) = A(i))$$

vernachlässigbar in n ist.

Sampling: *Es ist S ein PPT-Algorithmus, der bei Eingabe von 1^n einen Wert $x \in \{0,1\}^*$ ausgibt.*

Perfekt Einweg: *Für jeden PPT-Algorithmus D ist*

$$\max \Big\{ |P_{s,t,u_j}(D_s(h_i(xr_1), \ldots, h_i(xr_d)) = 1 : x = S_t(1^n), r_j = R_{u_j}(1^n)) \\ - P_{s,t_j,u_j}(D_s(h_i(x_1 r_1), \ldots, h_i(x_d r_d)) = 1 : x_j = S_{t_j}(1^n), r_j = R_{u_j}(1^n))| \\ \Big| \; i \in I(1^n) \Big\}$$

vernachlässigbar in n.

Es darf hierbei d auch eine Funktion von n sein.

Die perfekte Einweg-Eigenschaft besagt also, dass man nicht unterscheiden kann, ob verschiedene Hashwerte von einer einzigen Nachricht berechnet wurden oder von verschiedenen. Diese Definition ist offensichtlich noch stärker als die der kollisionsresistenten Hashfunktionen, da die Kollisionsresistenz explizit gefordert wird.

4.2 Neue Hashfunktionen

In dieser Arbeit werden zwei neue Arten von Hashfunktionen betrachtet, die sich in gewisser Weise ähnlich verhalten sollen, wie ein Random-Oracle. Daher werden die Hashfunktionen in den Definitionen mit Random-Oracles verglichen, was sich von den bisherigen Definitionen deutlich unterscheidet. Zunächst führen wir die *unvorhersehbaren Hashfunktionen* ein, wobei dies allerdings keinerlei Ähnlichkeit zur Unvorhersehbarkeit von Pseudozufallsgeneratoren besitzt (vgl. [Gol03]). Bei diesen neuen Hashfunktionen, hat ein Angreifer keine Chance, den Hashwert oder Teile davon in irgendeiner Weise zu beeinflussen oder vorherzusagen, zumindest nicht mehr als ein anderer Angreifer bei einem Random-Oracle.

Definition 4.2.1. *Eine Familie von Hashfunktionen* $\{h_i\}$ *ist genau dann* unvorhersehbar vom Grad d*, wenn sie kollisionsresistent ist und für alle PPT-Algorithmen* K, D, A *gilt: Es existiert ein PPT-Algorithmus* B*, so dass*

$$\Big|P\Big(D(1^n, k, h_i(x_1), \ldots, h_i(x_d), z) = 1 \wedge |z| \leq |\iota(n)|/2 : i = I(1^n), k = K(1^n), (x_1, \ldots, x_d, z) = A(i, k)\Big) - P\Big(D(1^n, k, \mathcal{R}^l(x_1), \ldots, \mathcal{R}^l(x_d), z) = 1 \wedge |z| \leq |\iota(n)|/2 : l = \lambda(n), k = K(1^n), (x_1, \ldots, x_d, z) = B^{\mathcal{R}^l}(1^n, k)\Big)\Big|$$

vernachlässigbar in n *ist.*

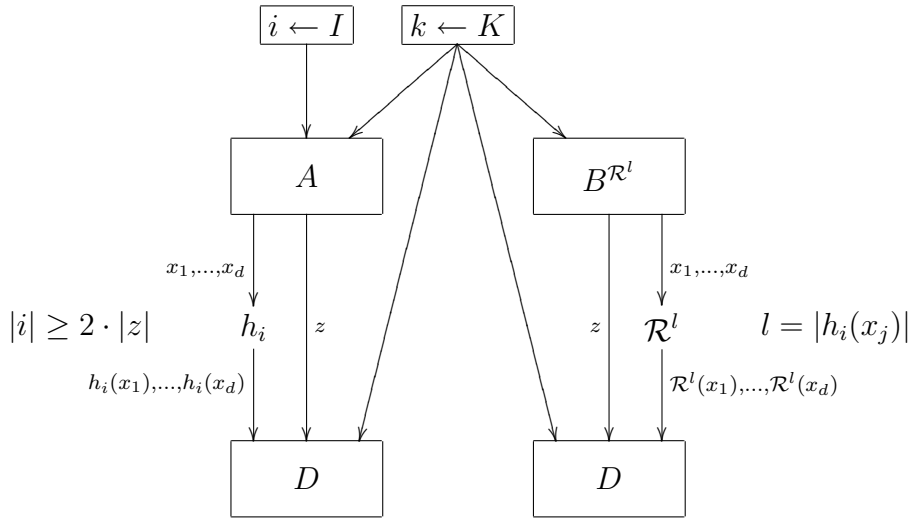

Abbildung 4.1: Unvorhersehbare Hashfunktionen

Bei der nächsten Art von Hashfunktionen handelt es sich um *zufallerhaltende Hashfunktionen*. Hierbei soll gewährleistet werden, dass für zufällige Eingaben auch zufällige

Hashwerte entstehen, selbst wenn die Eingaben vorher noch eine injektive Funktion durchlaufen und der Unterscheider die Hashfunktion kennt.

Definition 4.2.2. *Eine Familie von Hashfunktionen $\{h_i\}$ ist genau dann* zufallerhaltend*, wenn für jede Familie $\left\{ f_n \colon \{0,1\}^{\lambda(n)} \to \{0,1\}^* \;\middle|\; n \in \mathbb{N} \right\}$ aus injektiven Funktionen, für die es einen PT-Algorithmus F mit $F(x, 1^n) = f_n(x)$ gibt, und jeden PPT-Algorithmus D gilt, dass*

$$\Big| P\left(D(i, h_i(f_n(x))) = 1 : i = I(1^n), x = \mathcal{R}^{\lambda(n)}(1)\right) - P\left(D(i, x) = 1 : i = I(1^n), x = \mathcal{R}^{\lambda(n)}(1)\right) \Big|$$

vernachlässigbar in n ist.

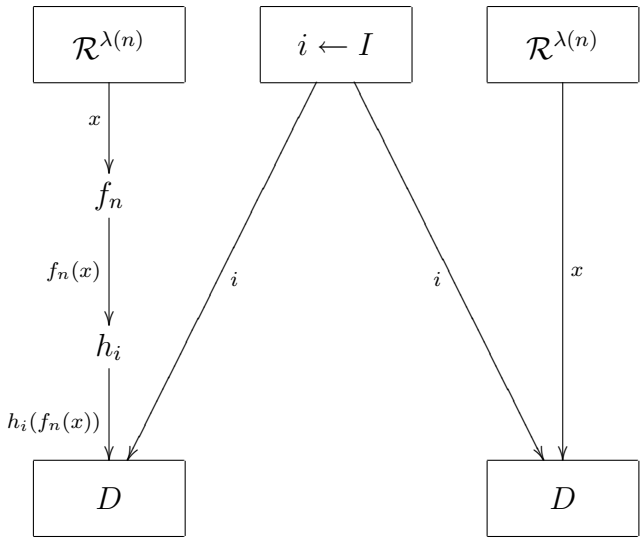

Abbildung 4.2: Zufallerhaltende Hashfunktionen

Für diese Arbeit hätte auch eine einfachere Definition ausgereicht, nämlich die Eigenschaft von zufallerhaltenden Hashfunktionen aus folgendem Satz:

Satz 4.2.3. *Wenn eine Familie von Hashfunktionen $\{h_i\}$ zufallerhaltend ist, dann gilt für jeden PPT-Algorithmus D und jedes Wort $w \in \{0,1\}^*$, dass*

$$\Big| P\left(D(i, h_i(wx)) = 1 : i = I(1^n), x = \mathcal{R}^{\lambda(n)}(1)\right) - P\left(D(i, x) = 1 : i = I(1^n), x = \mathcal{R}^{\lambda(n)}(1)\right) \Big|$$

vernachlässigbar in n ist.

Beweis. Folgt direkt aus der Definition 4.2.2 für zufallerhaltende Hashfamilien, wenn man die Funktionen f_n so wählt, dass $f_n(x) = wx$ ist. □

Diese Eigenschaft ist aber sehr speziell, weswegen in dieser Arbeit die allgemeinere Definition benutzt wurde.

Es ist bis jetzt leider nicht ganz klar, ob die beiden Arten von Hashfunktionen überhaupt existieren bzw. sich mit bisherigen Standardannahmen konstruieren lassen. Daher müssen wir für den Rest der Arbeit annehmen, dass diese Hashfunktionen existieren.

Annahme 4.2.4. *Es existieren unvorhersehbare Familien von Hashfunktionen und zufallerhaltende Familien von Hashfunktionen, sowie unvorhersehbare zufallerhaltende Familien von Hashfunktionen.*

Die Eigenschaft aus Satz 4.2.3 würde bereits von einer einfachen Hashfunktion erfüllt werden, die einfach immer nur die letzten $\lambda(n)$ Bits der Eingabe zurückliefert. Dies kann als Indiz dafür gewertet werden, dass die Annahme richtig ist (zumindest bzgl. der unvorhersehbaren Hashfamilien).

4.3 Vergleich und Eigenschaften von Hashfunktionen

In diesem Abschnitt werden einige Eigenschaften der vorher definierten Hashfunktionen betrachtet und sofern möglich Vergleiche zwischen ihnen angestellt.

Fangen wir zunächst mit den beiden grundlegenden Hashfunktionen an, den Einweg-Hashfunktionen und den kollisionsresistenten Hashfunktionen. Es stellt sich die Frage, welche der beiden Definitionen die stärkere ist. Dies kann im Allgemeinen nicht beantwortet werden, da dies stark vom Sampling-Algorithmus S abhängt. Wenn man allerdings fordert, dass sehr viele Kollisionen existieren und dass S einigermaßen gleichmäßig verteilt ist, erhält man das Ergebnis, dass kollisionsresistente Hashfunktionen auch die Einweg-Eigenschaft besitzen.

Satz 4.3.1. *Eine Familie von Hashfunktionen $\{h_i\}$, die kollisionsresistent ist, besitzt die Einweg-Eigenschaft bzgl. S, wenn gilt:*

- *Die Wahrscheinlichkeit*

$$P(x \notin K_i : i = I(1^n), x = S(i))$$

  ist vernachlässigbar in n, wobei $K_i = \{x \in S(i) \mid |h_i^{-1}(h_i(x))| \geq 2\}$ die Menge aller Bilder von $S(i)$ ist, die eine Kollision unter h_i erzeugen können.

- *Es existieren $c, d \in \mathbb{N}$, so dass für alle $n \in \mathbb{N}$, $i \in I(1^n)$ und $x \in K_i$ gilt*

$$\frac{1}{c \cdot |S(i)|} \leq P(S(i) = x) \leq \frac{d}{|S(i)|}.$$

Beweis. Angenommen, die Einweg-Eigenschaft wäre nicht erfüllt, dann gäbe es einen PPT-Algorithmus A, der diese mit der Wahrscheinlichkeit

$$p(n) := P(h_i(x) = h_i(y) : i = I(1^n), x = S(i), y = A(i, h_i(x))),$$

brechen könnte, welche in n nicht vernachlässigbar ist.

Damit gilt für den PPT-Algorithmus A' mit $A'_{r,s}(i) = (S_s(i), A_r(i, h_i(S_s(i))))$, der versucht Kollisionen zu finden, dass

$$\begin{aligned} p(n) &\leq \sum_{i \in I(1^n)} \sum_{x \in S(i)} P(i = I(1^n)) \cdot P(x = S(i)) \cdot P(h_i(x) = h_i(y) : y = A(i, h_i(x))) \\ &\leq q(n) + q'(n) \end{aligned}$$

ist mit

$$\begin{aligned} q(n) &:= \sum_{i \in I(1^n)} \sum_{x \in S(i) \setminus K_i} P(i = I(1^n)) \cdot P(x = S(i)) \cdot P(h_i(x) = h_i(y) : y = A(i, h_i(x))) \\ &\leq \sum_{i \in I(1^n)} \sum_{x \in S(i) \setminus K_i} P(i = I(1^n)) \cdot P(x = S(i)) \\ &\leq \sum_{i \in I(1^n)} P(i = I(1^n)) \cdot P(x \notin K_i : x = S(i)) \\ &\leq P(x \notin K_i : i = I(1^n), x = S(i)) \end{aligned}$$

und

$$\begin{aligned} q'(n) &:= \sum_{i \in I(1^n)} \sum_{x \in K_i} P(i = I(1^n)) \cdot P(x = S(i)) \cdot P(h_i(x) = h_i(y) : y = A(i, h_i(x))) \\ &\leq \sum_{i \in I(1^n)} \sum_{x \in K_i} P(i = I(1^n)) \cdot \frac{d}{|S(i)|} \cdot P(h_i(x) = h_i(y) : y = A(i, h_i(x))) \\ &\overset{(*)}{\leq} \sum_{i \in I(1^n)} \sum_{x \in K_i} P(i = I(1^n)) \cdot \frac{d}{|S(i)|} \cdot 2 \cdot P(h_i(x) = h_i(y) \wedge x \neq y : y = A(i, h_i(x))) \\ &\leq \sum_{i \in I(1^n)} \sum_{x \in K_i} P(i = I(1^n)) \cdot ed \cdot P(x = S(i)) \\ &\qquad \cdot 2 \cdot P(h_i(x) = h_i(y) \wedge x \neq y : y = A(i, h_i(x))) \\ &\leq 2ed \cdot \sum_{i \in I(1^n)} \sum_{x \in S(i)} P(i = I(1^n)) \cdot P(x = S(i)) \\ &\qquad \cdot P(h_i(x) = h_i(y) \wedge x \neq y : y = A(i, h_i(x))) \\ &\leq 2ed \cdot P(h_i(x) = h_i(y) \wedge x \neq y : i = I(1^n), x = S(i), y = A(i, h_i(x))) \\ &\leq 2ed \cdot P(h_i(x) = h_i(y) \wedge x \neq y : i = I(1^n), (x, y) = A'(i)). \end{aligned}$$

Da sowohl q als auch q' vernachlässigbar sind, gilt dies gemäß Lemma 2.2.9 auch für p, was ein Widerspruch zur Annahme wäre. Es bleibt noch die Ungleichung $(*)$ zu erläutern. Vorher bemerken wir noch, dass $|h_i^{-1}(w)| \geq 2$ für alle $w \in h_i(K_i)$ gilt. Für alle $n \in \mathbb{N}$ und $i \in I(1^n)$ ist

$$
\begin{aligned}
&\sum_{x \in K_i} P(h_i(x) = h_i(y) : y = A(i, h_i(x))) \\
&= \sum_{w \in h_i(K_i)} \sum_{x \in h_i^{-1}(w)} P(h_i(x) = h_i(y) : y = A(i, h_i(x))) \\
&= \sum_{w \in h_i(K_i)} \sum_{x \in h_i^{-1}(w)} P(w = h_i(y) : y = A(i, w)) \\
&= \sum_{w \in h_i(K_i)} \sum_{x \in h_i^{-1}(w)} \sum_{y \in A(i,w)} P(y = A(i, w)) \cdot P(w = h_i(y)) \\
&= \sum_{w \in h_i(K_i)} |h_i^{-1}(w)| \sum_{y \in A(i,w)} P(y = A(i, w)) \cdot P(w = h_i(y)) \\
&= \sum_{w \in h_i(K_i)} \frac{|h_i^{-1}(w)|}{|h_i^{-1}(w)| - 1} (|h_i^{-1}(w)| - 1) \sum_{y \in A(i,w)} P(y = A(i, w)) \cdot P(w = h_i(y)) \\
&\leq \sum_{w \in h_i(K_i)} \frac{|h_i^{-1}(w)|}{|h_i^{-1}(w)| - 1} \sum_{x \in h_i^{-1}(w)} \sum_{y \in A(i,w) \setminus \{x\}} P(y = A(i, w)) \cdot P(w = h_i(y)) \\
&\leq 2 \cdot \sum_{w \in h_i(K_i)} \sum_{x \in h_i^{-1}(w)} \sum_{y \in A(i,w) \setminus \{x\}} P(y = A(i, w)) \cdot P(w = h_i(y)) \cdot P(x \neq y) \\
&= 2 \cdot \sum_{w \in h_i(K_i)} \sum_{x \in h_i^{-1}(w)} \sum_{y \in A(i,w)} P(y = A(i, w)) \cdot P(w = h_i(y)) \cdot P(x \neq y) \\
&= 2 \cdot \sum_{x \in K_i} P(h_i(x) = h_i(y) \wedge x \neq y : y = A(i, h_i(x))).
\end{aligned}
$$

□

Eine Einweg-Hashfunktion ist aber nicht unbedingt kollisionsresistent, selbst wenn der Sampling-Algorithmus S die beiden Bedingungen aus Satz 4.3.1 erfüllt.

Satz 4.3.2. *Es sei $\{h_i\}$ eine Familie von Hashfunktionen, die die Einweg-Eigenschaft bzgl. S besitzt. Dann gibt es einen Sampling-Algorithmus $\tilde{S}$ und eine Familie von Hashfunktionen $\{\tilde{h}_i\}$, die nicht kollisionsresistent ist, aber die Einweg-Eigenschaft bzgl. $\tilde{S}$ besitzt.*

Beweis. Es sei $\tilde{S}_{rs}(i) := rS_s(i)$, wobei $r \in \{0, 1\}$ der Beginn und s den Rest des Zufallsbandes von $\tilde{S}$ darstellt. Weiter sei $\tilde{h}_i(0x) := \tilde{h}_i(1x) := h_i(x)$ und $\tilde{h}_i(\varepsilon) := h_i(\varepsilon)$. Dann ist $\tilde{h}_i(\tilde{S}_{rs}(i)) = h_i(S_s(i))$, womit $\{\tilde{h}_i\}$ die Einweg-Eigenschaft bzgl. $\tilde{S}$ besitzt, da aus einem Urbild von y bzgl. $\tilde{h}_i$ leicht ein Urbild bzgl. h_i bestimmt werden kann (was falls möglich dann einen Widerspruch zur Einweg-Eigenschaft von h_i darstellen würde).

Ferner lassen sich leicht Kollisionen für $\tilde{h}_i$ konstruieren, nämlich Paare der Form $(0x, 1x)$ mit $x \in \{0,1\}^*$. □

Eine Frage, die sich bei kollisionsresistenten Hashfunktionen stellt, ist, wie groß die Hashlänge λ mindestens sein muss. Wenn sie nämlich zu klein ist, kann man leicht Kollisionen finden, wie der folgende Satz zeigt (vgl. [Gol04] S. 513).

Satz 4.3.3. *Es sei $\{h_i\}$ eine kollisionsresistente Familie von Hashfunktionen. Dann gilt*

$$\lambda \in \omega(\log_2(\mathcal{N})).$$

Beweis. Angenommen der Satz wäre falsch, also $\lambda \notin \omega(\log_2(\mathcal{N}))$, dann würde aus der Definition von ω folgen, dass

$$\exists k_0 \in \mathbb{N}^+ \colon \forall n_0 \in \mathbb{N} \colon \exists n \geq n_0 \colon\ \lambda(n) < k_0 \log_2(n)$$

gilt. Für solche n wäre dann

$$2^{\lambda(n)} < 2^{k_0 \log_2(n)} = n^{k_0},$$

so dass in polynomialer Laufzeit (in n) $2^{\lambda(n)} + 1$ Hashwerte berechnet werden können, worunter mindestens eine Kollision ist, da es nur $2^{\lambda(n)}$ verschiedene Hashwerte gibt. Dies wäre aber ein Widerspruch zur Kollisionsresistenz. □

Perfekte Einweg-Hashfunktionen sind per Definition kollisionsresistent. Wenn allerdings der Sampling-Algorithmus S beispielsweise immer nur einen Wert ausgibt, dann besitzt auch jede kollisionsresistente Familie von Hashfunktionen die perfekte Einweg-Eigenschaft bzgl. S. Es lassen sich aber auch leicht Fälle konstruieren, in denen aus der Kollisionsresistenz nicht die perfekte Einweg-Eigenschaft folgt.

Satz 4.3.4. *Es sei $\{h_i\}$ eine Familie von Hashfunktionen, die die d-wertige perfekte Einweg-Eigenschaft bzgl. S besitzt, mit $d > 1$ und S so, dass die beiden Wahrscheinlichkeiten*

$$P(S(i) \in 0\{0,1\}^* : i = I(1^n)) \quad \textit{und} \quad P(S(i) \in 1\{0,1\}^* : i = I(1^n))$$

größer als n^{-k} sind für genügend große n und ein $k \in \mathbb{N}$. Dann ist $\{\tilde{h}_i\}$ mit

$$\tilde{h}_i(ax) := ah_i(ax) \quad \textit{für} \quad a \in \{0,1\}, x \in \{0,1\}^*$$

und $\tilde{h}_i(\varepsilon) := 0h_i(\varepsilon)$ kollisionsresistent, aber besitzt keine 2-wertige perfekte Einweg-Eigenschaft bzgl. S.

Beweis. Die Kollisionsresistenz ergibt sich direkt aus der Konstruktion, da Kollisionen für $\tilde{h}_i$ auch Kollisionen für h_i sind. Es gibt aber einen Unterscheider D, der die perfekte Einweg-Eigenschaft brechen kann. Dieser muss nur vergleichen, ob die (ersten) beiden Hashwerte mit dem gleichen Zeichen beginnen und ggf. eine 0 ausgeben, ansonsten eine 1. Der Unterscheider, der die Hashwerte für nur ein x erhält, gibt immer 0 aus. Der andere gibt mit einer in n nicht vernachlässigbaren Wahrscheinlichkeit $u(n)$ eine 1 aus, da für

$$p(n) := P(S(i) \in 0\{0,1\}^* : i = I(1^n)) \quad \text{und}$$
$$q(n) := P(S(i) \in 1\{0,1\}^* : i = I(1^n))$$

gilt, dass diese Wahrscheinlichkeit

$$u(n) \geq 2p(n)q(n) \geq 2n^{-2k}$$

ist für genügend große n, womit $u(n)$ nicht vernachlässigbar in n ist. □

Damit haben wir bereits einen Vergleich der drei bereits bekannten Arten von Hashfunktionen. Kommen wir nun zu den neuen Hashfunktionen. Die unvorhersehbaren Familien von Hashfunktionen sind per Definition auch kollisionsresistent. Allerdings gibt es kollisionsresistente Hashfunktionen, die nicht unvorhersehbar sind.

Satz 4.3.5. *Es sei $\{h_i\}$ eine kollisionsresistente Familie von Hashfunktionen. Dann ist die wie folgt definierte Familie von Hashfunktionen $\{\tilde{h}_i\}$ ebenfalls kollisionsresistent, aber nicht unvorhersehbar vom Grad 1: Wir definieren*

$$\tilde{h}_i(x) := h_i(x)h_i(x).$$

Beweis. Die so definierte Familie von Hashfunktionen $\{\tilde{h}_i\}$ ist kollisionsresistent, da Kollisionen von $\tilde{h}_i$ ebenfalls Kollisionen von h_i sind. Allerdings ist $\{\tilde{h}_i\}$ nicht unvorhersehbar vom Grad 1. Es sei D der Algorithmus, der genau dann 1 ausgibt, wenn sein drittes Argument von der Form yy ist mit $y \in \{0,1\}^*$, also

$$D(1^n, k, x, z) = 1 \quad \Leftrightarrow \quad \exists y \in \{0,1\}^* \colon x = yy.$$

Für den PPT-Algorithmus A aus Definition 4.2.1 für $\{\tilde{h}_i\}$, der immer $(0, \varepsilon)$ zurückgibt, gibt D mit einer Wahrscheinlichkeit von 1 den Wert 1 zurück. Für jeden Algorithmus B ist es genauso schwer solche Ausgaben eines Random-Oracles zu produzieren, wie echte Kollisionen für ein Random-Oracle halber Länge zu finden, was ihm aber nach Satz 2.2.11 und 4.3.3 nur mit einer in n vernachlässigbaren Wahrscheinlichkeit gelingen kann. Damit ist $\{\tilde{h}_i\}$ nicht unvorhersehbar vom Grad 1. □

In dieser Arbeit werden zufallerhaltende Hashfunktionen nur in Verbindung mit unvorhersehbaren Hashfunktionen benutzt, also zufallerhaltende unvorhersehbare Hashfunktionen. Es ist bisher leider unbekannt, ob dies eine echte weitere Einschränkung ist oder bereits jede unvorhersehbare Familie von Hashfunktionen auch zufallerhaltend ist. Vermutlich ist aber ersteres der Fall, da der Unterscheider im Fall von zufallerhaltenden Hashfunktionen auch den Index der Hashfunktion kennt. Damit ergibt sich als vergleichender Überblick Abbildung 4.3.

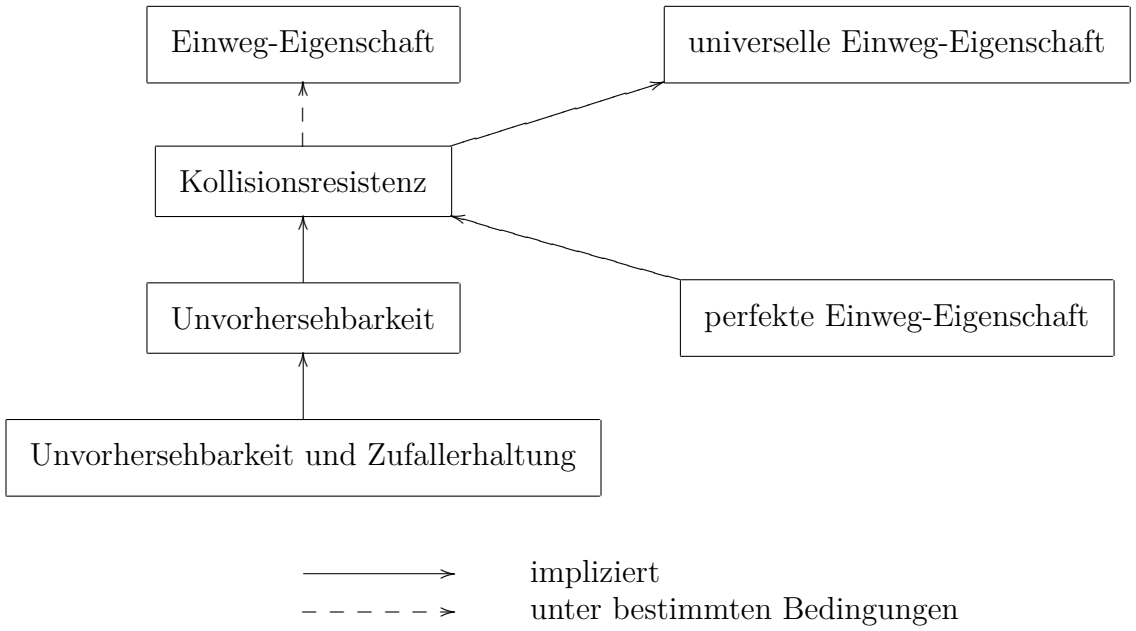

Abbildung 4.3: Vergleich von verschiedenen Hashfunktionen

5 Protokolle, Angriffe und Erfolge

In der Kryptographie betrachtet man häufig Protokolle und deren Sicherheit, wobei man verschiedene Angriffe und Erfolge unterscheidet. Dieses Kapitel ist in zwei Abschnitte aufgeteilt. Im ersten Abschnitt wird dargestellt, welche Art von Protokollen in dieser Arbeit betrachtet werden und wie diese genau definiert sind. Im zweiten Abschnitt wird dann definiert, welche Angriffe es gibt bzw. betrachtet werden und welche Erfolge diese haben können.

5.1 Protokolle

Da der Begriff des Random-Oracles bzw. die Random-Oracle-Methodik zuerst im Umfeld von Signaturen und Zero-Knowledge-Beweisen aufgetaucht ist und dort auch immer noch sehr oft verwendet wird, werden wir in dieser Arbeit nur Protokolle betrachten, die eine gewisse Ähnlichkeit zu Signaturen haben. Darunter werden aber noch mehr Protokolle fallen, die Hashfunktionen benutzen, wie z. B. nicht-interaktive Zero-Knowledge-Beweise (NIZK). Dabei betrachten wir im Wesentlichen zwei Varianten, die wir Protokolle vom Typ I bzw. Protokolle vom Typ II nennen werden. Die erste Variante ist so gewählt, dass die meisten relevanten Protokolle aus der Literatur direkt in das Schema passen. Bei der zweiten Variante handelt es sich im Prinzip um die gleichen Protokolle, die allerdings leicht abgeändert sind, so dass sie sich theoretisch besser behandeln lassen. Im Wesentlichen wird dabei nur die Berechnung des Hashwertes durch Benutzung eines Zufallswertes leicht abgeändert, ähnlich wie bei der perfekten Einweg-Eigenschaft (vgl. Definition 4.1.4). Aufgrund der Struktur der Protokolle vom Typ I und Typ II lassen sie zusammen mit unvorhersehbaren und zufallerhaltenden Hashfamilien eine weitergehende und genauere Betrachtung der Sicherheit zu als die bisherige einfache Anwendung der Random-Oracle-Methodik. Dies gilt insbesondere für die Protokolle vom Typ II. Es wird sich aber herausstellen, dass dies kein besonders großer Nachteil für die Protokolle vom Typ I ist, da sich diese ohne Probleme in Protokolle vom Typ II transformieren lassen, wie wir im nächsten Kapitel sehen werden.

Die hier betrachteten Protokolle teilen sich jeweils in drei Teile auf, den Generator-Algorithmus, den eigentlichen Protokoll-Algorithmus, den wir der Einfachheit halber hier Signatur-Algorithmus nennen wollen (auch wenn es sich nicht um ein Signatur-Protokoll handeln sollte), sowie den Verifikations-Algorithmus. Der Generator-Algorithmus erzeugt nach Eingabe des Sicherheitsparameters 1^n den öffentlichen Schlüssel pk, den geheimen Schlüssel sk und den Index i für die zu verwendende Hashfunktion. Der Signatur-

Algorithmus erhält als Eingabe den Sicherheitsparameter, alle Schlüssel sowie eine Eingabe m variabler Länge, die im Allgemeinen Nachricht genannt wird, und erzeugt daraus eine Ausgabe σ, die wir Signatur nennen. Mit dem Verifikations-Algorithmus wird überprüft, ob ein Datensatz von dem Signatur-Algorithmus erzeugt wurde oder nicht. Die Eingabe umfasst hier den Sicherheitsparameter, den öffentlichen Schlüssel und eine Eingabe variabler Länge (die Nachricht). Die Ausgabe besagt dann, ob die Eingaben zueinander passen oder nicht.

Der Generator-Algorithmus Gen ist bei Protokollen vom Typ I und Typ II gleich und erzeugt zunächst ein Schlüsselpaar $(\mathsf{pk}, \mathsf{sk}) \in K(1^n)$ mittels eines zum Protokoll passenden PPT-Algorithmus K. Hierbei soll der Einfachheit halber 1^n ein Teil von pk sein, so dass wir den Sicherheitsparameter 1^n nicht immer noch zusätzlich als Eingabe angeben müssen. Als zweites wird noch der Index $i \in I(1^n)$ erzeugt, wobei der Index-Algorithmus I der Familie von Hashfunktionen benutzt wird, die in dem Protokoll zum Einsatz kommen soll. Die von K und I benutzten Zufallswerte sollen hierbei stochastisch unabhängig voneinander sein. Die Ausgabe von Gen ist dann das Tripel $(\mathsf{pk}, \mathsf{sk}, i)$.

Die Signatur- und Verifikations-Algorithmen der Protokolle vom Typ I und Typ II sind allerdings unterschiedlich. Aber bevor wir uns diese anschauen, definieren wir zunächst, was wir unter einem Signatur-Protokoll verstehen, wobei hierunter wie bereits gesagt auch noch andere Protokolle fallen, die aber eine ähnliche Struktur haben.

Definition 5.1.1. *Ein Tupel* $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ *ist ein* Signatur-Protokoll*, wenn folgende Bedingungen erfüllt sind:*

- $\{h_i\}$ *ist eine Familie von Hashfunktionen mit dem Index-Algorithmus I und der Hashlänge λ.*
- Gen *ist ein PPT-Algorithmus, der bei Eingabe von 1^n ein Schlüsselpaar pk, sk ausgibt, sowie einen Index $i = I(1^n)$ für die Hashfunktion, der (stochastisch) unabhängig von dem Schlüsselpaar berechnet wird.*
- $\mathrm{Sig}^{\mathcal{H}}$ *ist ein PPT-Orakel-Algorithmus, der bei Eingabe von $(\mathsf{pk}, \mathsf{sk}, m)$ die Signatur σ zurückgibt, wobei $m \in \{0,1\}^*$ ist und das Orakel $\mathcal{H}$ entweder die Hashfunktion h_i oder ein Random-Oracle $\mathcal{R}^{\lambda(n)}$ ist.*
- $\mathrm{Ver}^{\mathcal{H}}$ *ist ein PPT-Orakel-Algorithmus, der entweder 0 oder 1 ausgibt. Bei Eingabe von $(\mathsf{pk}, m, \mathrm{Sig}^{\mathcal{H}}(\mathsf{pk}, \mathsf{sk}, m))$ gibt er allerdings immer 1 aus (Durchführbarkeit). Hierbei ist das Orakel $\mathcal{H}$ auch entweder die Hashfunktion h_i oder ein Random-Oracle $\mathcal{R}^{\lambda(n)}$.*

Über die Korrektheit (keine Fälschungen möglich) wird hier noch keine Aussage getroffen, da es hier wie bereits oben erwähnt verschiedene Varianten gibt, die wir im nächsten Abschnitt betrachten werden.

Kommen wir nun zu den beiden speziellen Varianten der Signatur- und Verifikations-Algorithmen.

Typ I Bei Protokollen vom Typ I läuft der Signatur-Algorithmus $\mathrm{Sig}^{\mathcal{H}}$ bei Eingabe von $(\mathsf{pk}, \mathsf{sk}, m)$ wie folgt ab:

$$\begin{aligned}(z, z') &= \mathrm{Sub}_1(\mathsf{pk})\\ y &= \mathcal{H}(z'm)\\ \sigma &= (\mathrm{Sub}_2(\mathsf{sk}, z, y), z')\\ \text{Ausgabe: } \sigma &= (\sigma_1, \sigma_2).\end{aligned}$$

Mit Sub_j sind PPT-Algorithmen gemeint, die bei Eingabe von x die Ausgabe $\mathrm{Sub}_j(x)$ liefern, wobei für Sub_1 die Einschränkung gilt, dass $|z'|$ eine Funktion von n ist, die wir im Weiteren mit ζ bezeichnen, also $|z'| = \zeta(n)$ gilt. Außerdem kann die Nachricht m auch leer sein bzw. nicht als Eingabe zugelassen sein, so dass der Hashwert nur von z' abhängt. Ferner ist zu bemerken, dass in z auch der Wert pk enthalten sein kann, so dass dieser nicht wieder explizit als Eingabe für Sub_2 aufgeführt wird.

Der Verifikations-Algorithmus $\mathrm{Ver}^{\mathcal{H}}$ läuft bei Eingabe von (pk, m, σ) wie folgt ab:

$$\begin{aligned}y &= \mathcal{H}(\sigma_2 m)\\ \nu &= \mathrm{Sub}'(\mathsf{pk}, y, \sigma_1, \sigma_2)\\ \text{Ausgabe: } \nu &\in \{0, 1\}.\end{aligned}$$

Hierbei gibt die Ausgabe 1 an, dass die Verifikation positiv verlaufen ist, also die Eingabe vermutlich eine Ausgabe des Signatur-Algorithmus ist. Eine 0 bedeutet, dass die Eingabewerte nicht zueinander passen bzw. nicht vom Signatur-Algorithmus ausgegeben wurden.

Typ II Bei Protokollen vom Typ II läuft der Signatur-Algorithmus $\mathrm{Sig}^{\mathcal{H}}$, wenn er $(\mathsf{pk}, \mathsf{sk}, m)$ als Eingabe erhält, wie folgt ab:

$$\begin{aligned}(z, z') &= \mathrm{Sub}_1(\mathsf{pk})\\ r &= R(1^n)\\ y &= \mathcal{H}(0z'm) \oplus \mathcal{H}(1r)\\ \sigma &= (\mathrm{Sub}_2(\mathsf{sk}, z, y), z', r)\\ \text{Ausgabe: } \sigma &= (\sigma_1, \sigma_2, \sigma_3).\end{aligned}$$

Dabei ist R ein PPT-Algorithmus, der von $\mathsf{sk}, \mathsf{pk}, i, m, z, z'$ unabhängige und in $\{0, 1\}^{\lambda(n)}$ uniform verteilte Werte liefert, wobei λ die zur gewählten Familie von Hashfunktionen gehörende Hashlänge ist und der von R erzeugte Wert bei jedem Aufruf jeweils eine neue Zufallszahl liefert. Der Rest ist wie bei den Protokollen vom Typ I.

Der Verifikations-Algorithmus $\mathrm{Ver}^{\mathcal{H}}$ läuft bei Eingabe von (pk, m, σ) wie folgt ab:

$$\begin{aligned}y &= \mathcal{H}(0\sigma_2 m) \oplus \mathcal{H}(1\sigma_3)\\ \nu &= \mathrm{Sub}'(\mathsf{pk}, y, \sigma_1, \sigma_2)\\ \text{Ausgabe: } \nu &\in \{0, 1\}.\end{aligned}$$

Zu bemerken ist hierbei, dass sich die Algorithmen nur wenig verändert haben. Der Hashwert wird etwas komplizierter berechnet, und die Signatur σ hat eine Komponente mehr, aber die eigentlichen Bestandteile der Algorithmen, nämlich $\mathrm{Sub}_1, \mathrm{Sub}_2$ und Sub' bleiben gleich.

Die drei Bestandteile sowie die zu verwendende Familie von Hashfunktionen werden zu einem Protokoll $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ zusammengefasst, was die folgende Definition noch einmal verdeutlichen soll.

Definition 5.1.2. *Signatur-Protokolle* $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ *der ersten Variante heißen* Protokolle vom Typ I. *Die der zweiten Variante heißen* Protokolle vom Typ II.

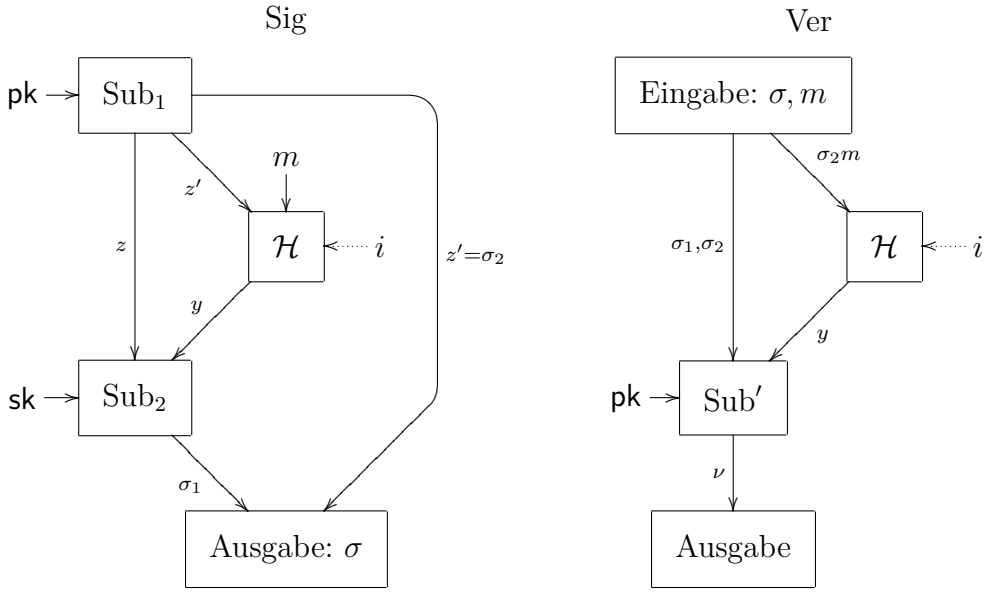

Abbildung 5.1: Protokolle vom Typ I

Da sich Protokolle vom Typ I und Typ II (vgl. auch Abbildungen 5.1 und 5.2) nur wenig voneinander unterscheiden und insbesondere die gleichen Algorithmen benutzen, können wir aus einen Protokoll vom Typ I eindeutig ein Protokoll vom Typ II erstellen. Wir sagen, dass solch ein Protokoll vom Typ II aus dem Protokoll vom Typ I erzeugt wird. Damit ist geklärt, welche Art von Protokollen überhaupt für eine weitere Betrachtung in Frage kommt.

5.2 Angriffe und Erfolge

In dieser Arbeit werden zwei Arten von Angriffen betrachtet. Das ist zum einen der Angriff ohne bekannte Signaturen, wobei der Angreifer nur die öffentlichen Parameter bzw.

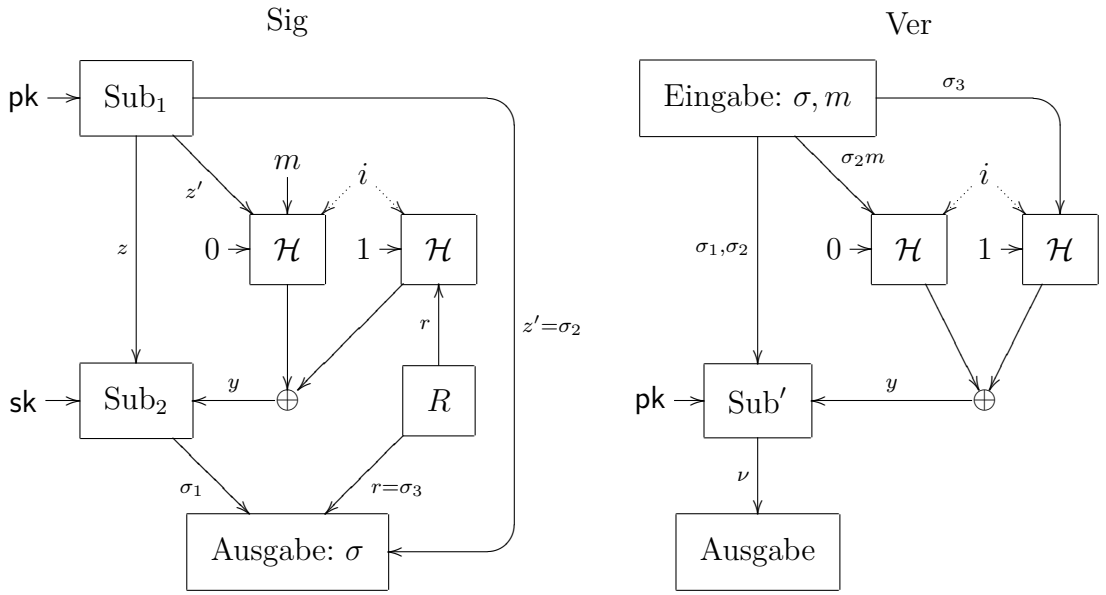

Abbildung 5.2: Protokolle vom Typ II

öffentlichen Schlüssel kennt und damit eine vom Verifikations-Algorithmus akzeptierte Eingabe produzieren muss. Zum anderen wird der adaptive Angriff mit gewählten Nachrichten betrachtet, bei dem der Angreifer Zugriff auf den Signatur-Algorithmus hat und anschließend ebenfalls eine vom Verifikations-Algorithmus akzeptierte Eingabe produzieren muss. Allerdings darf er hier natürlich nicht direkt die vom Signatur-Algorithmus erhaltenen Ausgaben produzieren. Die genauen Details regeln die folgenden Definitionen.

Ein Angriff kann verschiedene Erfolge haben. Es kann sein, dass der Angreifer zwar irgendeine (neue) Signatur erzeugt, aber keinen weiteren Einfluss auf die Nachricht hat, so dass diese höchstwahrscheinlich keinen Sinn ergibt. Dies nennt man existentielle Fälschung. Es kann aber auch sein, dass der Angreifer zu jeder Nachricht eine gültige Signatur produzieren kann, was man universelle Fälschung nennt. Falls der Angreifer sogar den geheimen Schlüssel bestimmen kann, ist das Protokoll total gebrochen. Diese Unterscheidung verdeutlicht, dass der geheime Schlüssel nicht unbedingt Ziel eines Angriffs sein muss, selbst wenn man beliebige Nachrichten fälschen will.

Es folgen nun die formalen Definitionen. Die Standard-Definition für Signaturen ist, dass ein Protokoll gegen den stärksten Angriff sicher sein muss und dabei nicht einmal der kleinste Erfolg erreichbar ist. Das ist die stärkste der hier definierten Sicherheitseigenschaften.

Definition 5.2.1. *Ein Signatur-Protokoll* $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ *ist genau dann sicher gegen existentielle Fälschung unter einem adaptiven Angriff mit gewählten Nachrichten (im Standard-Modell), wenn für jeden PPT-Algorithmus A die Wahrscheinlichkeit*

$$P\left(\mathrm{Ver}^{h_i}(\mathsf{pk}, m, \sigma) = 1 : (\mathsf{pk}, \mathsf{sk}, i) = \mathrm{Gen}(1^n), (m, \sigma) = A^{\mathrm{Sig}^{h_i}(\mathsf{pk},\mathsf{sk})}(i, \mathsf{pk})\right)$$

vernachlässigbar in n ist, wobei $\mathrm{Sig}^{h_i}(\mathsf{pk}, \mathsf{sk})$ *ein Orakel bezeichnet, das bei Eingabe von* m' *die Ausgabe* $\mathrm{Sig}^{h_i}(\mathsf{pk}, \mathsf{sk}, m')$ *liefert und dabei jedesmal neue Zufallszahlen benutzt. Außerdem darf* m *nicht als Anfrage von A an das Orakel gestellt worden sein.*

Da wir die Sicherheit von Protokollen auch im Random-Oracle-Modell betrachten wollen und nicht nur im Standard-Modell bei Benutzung der Hashfamilie, brauchen wir dafür ebenfalls eine Definition, die sich aber nur leicht von der obigen unterscheidet.

Definition 5.2.2. *Ein Signatur-Protokoll* $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ *ist genau dann sicher gegen existentielle Fälschung unter einem adaptiven Angriff mit gewählten Nachrichten im Random-Oracle-Modell, wenn für jeden PPT-Algorithmus A die Wahrscheinlichkeit*

$$P\Big(\mathrm{Ver}^{\mathcal{R}^{\lambda(n)}}(\mathsf{pk}, m, \sigma) = 1 : (\mathsf{pk}, \mathsf{sk}, i) = \mathrm{Gen}(1^n), (m, \sigma) = A^{\mathrm{Sig}^{\mathcal{R}^{\lambda(n)}}(\mathsf{pk},\mathsf{sk}), \mathcal{R}^{\lambda(n)}}(i, \mathsf{pk}) \Big)$$

vernachlässigbar in n ist, wobei $\mathrm{Sig}^{\mathcal{R}^{\lambda(n)}}(\mathsf{pk}, \mathsf{sk})$ *ein Orakel ist, das bei Eingabe von* m' *die Ausgabe* $\mathrm{Sig}^{\mathcal{R}^{\lambda(n)}}(\mathsf{pk}, \mathsf{sk}, m')$ *liefert und dabei jedesmal neue Zufallszahlen (aber dieselbe Instanz des Random-Oracles) benutzt. Außerdem darf* m *nicht als Anfrage von A an das Orakel gestellt worden sein.*

Im Folgenden werden nur noch die Definitionen für das Standard-Modell angegeben. Die Versionen für das Random-Oracle-Modell ergeben sich genau wie oben, nämlich durch Ersetzen von h_i durch eine Instanz von $\mathcal{R}^{\lambda(n)}$ und Hinzufügen des Zugriffs von A auf das Orakel $\mathcal{R}^{\lambda(n)}$.

Die nächste Definition hat einen schwächeren Angriff, aber den gleichen Erfolg.

Definition 5.2.3. *Ein Signatur-Protokoll* $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ *ist genau dann sicher gegen existentielle Fälschung unter einem Angriff ohne bekannte Signaturen, wenn für jeden PPT-Algorithmus A die Wahrscheinlichkeit*

$$P\left(\mathrm{Ver}^{h_i}(\mathsf{pk}, m, \sigma) = 1 : (\mathsf{pk}, \mathsf{sk}, i) = \mathrm{Gen}(1^n), (m, \sigma) = A(i, \mathsf{pk})\right)$$

vernachlässigbar in n ist.

Zu bemerken ist hierbei, dass der Signatur-Algorithmus bei dieser Definition gar keine Rolle spielt.

Wir kommen nun zu den Varianten, bei denen ein erfolgreicher Angriff darin besteht, beliebige Nachrichten fälschen zu können. Zunächst die Variante mit dem stärkeren Angriff.

Definition 5.2.4. *Ein Signatur-Protokoll* $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ *ist genau dann sicher gegen universelle Fälschung unter einem adaptiven Angriff mit gewählten Nachrichten, wenn für alle PPT-Algorithmen S und A die Wahrscheinlichkeit*

$$P\left(\mathrm{Ver}^{h_i}(\mathsf{pk}, m, \sigma) = 1 : m = S(1^n), (\mathsf{pk}, \mathsf{sk}, i) = \mathrm{Gen}(1^n), \sigma = A^{\mathrm{Sig}^{h_i}(\mathsf{pk},\mathsf{sk})}(i, \mathsf{pk}, m)\right)$$

vernachlässigbar in n ist, wobei $\mathrm{Sig}^{h_i}(\mathsf{pk}, \mathsf{sk})$ *ein Orakel bezeichnet, das bei Eingabe von* m' *die Ausgabe* $\mathrm{Sig}^{h_i}(\mathsf{pk}, \mathsf{sk}, m')$ *liefert und dabei jedesmal neue Zufallszahlen benutzt. Außerdem darf* m *nicht als Anfrage von A an das Orakel gestellt worden sein.*

Und nun noch die Variante mit dem schwachen Angriff.

Definition 5.2.5. *Ein Signatur-Protokoll* $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ *ist genau dann sicher gegen universelle Fälschung unter einem Angriff ohne bekannte Signaturen, wenn für alle PPT-Algorithmen S und A die Wahrscheinlichkeit*

$$P\left(\mathrm{Ver}^{h_i}(\mathsf{pk}, m, \sigma) = 1 : m = S(1^n), (\mathsf{pk}, \mathsf{sk}, i) = \mathrm{Gen}(1^n), \sigma = A(i, \mathsf{pk}, m)\right)$$

vernachlässigbar in n ist.

Auch hier spielt der Signatur-Algorithmus wieder keine Rolle.

Und zu guter Letzt die Fälle, in denen der Angreifer das Protokoll total brechen kann. Zunächst die Variante mit dem stärkeren Angriff.

Definition 5.2.6. *Ein Signatur-Protokoll* $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ *ist genau dann sicher gegen totales Brechen unter einem adaptiven Angriff mit gewählten Nachrichten, wenn für jeden PPT-Algorithmus A die Wahrscheinlichkeit*

$$P\left(A^{\mathrm{Sig}^{h_i}(\mathsf{pk},\mathsf{sk})}(i, \mathsf{pk}) = \mathsf{sk} : (\mathsf{pk}, \mathsf{sk}, i) = \mathrm{Gen}(1^n)\right)$$

vernachlässigbar in n ist, wobei $\mathrm{Sig}^{h_i}(\mathsf{pk}, \mathsf{sk})$ *ein Orakel bezeichnet, das bei Eingabe von m die Ausgabe* $\mathrm{Sig}^{h_i}(\mathsf{pk}, \mathsf{sk}, m)$ *liefert und dabei jedesmal neue Zufallszahlen benutzt.*

Bei dieser Variante spielt der Verifikations-Algorithmus keine Rolle. Der Vollständigkeit halber noch die Variante mit dem schwachen Angriff.

Definition 5.2.7. *Ein Signatur-Protokoll* $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ *ist genau dann sicher gegen totales Brechen unter einem Angriff ohne bekannte Signaturen, wenn für jeden PPT-Algorithmus A die Wahrscheinlichkeit*

$$P\left(A(i, \mathsf{pk}) = \mathsf{sk} : (\mathsf{pk}, \mathsf{sk}, i) = \mathrm{Gen}(1^n)\right)$$

vernachlässigbar in n ist.

Hier spielt weder der Signatur- noch der Verifikations-Algorithmus eine Rolle. Wie bereits oben erwähnt, wird meistens nur die allererste Variante dieser Sicherheitsbegriffe benutzt, da diese die stärkste ist.

6 Die Random-Oracle-Methodik

Die Random-Oracle-Methodik ist ein sehr beliebtes Hilfsmittel, um kurze einfache Protokolle zu erstellen. Allerdings gibt es genügend Beispiele, die zeigen, dass die Methodik im Allgemeinen nicht sicher ist. In diesem Kapitel werden wir dies etwas genauer betrachten und zeigen, wann die Methodik (vermutlich) doch sicher ist. Aber zunächst verschaffen wir uns einen Überblick, was die Random-Oracle-Methodik ist und wie sie angewendet wird.

6.1 Einordnung und Anwendung der Random-Oracle-Methodik

Davon auszugehen, dass sich eine Hashfunktion so verhält wie ein Random-Oracle, macht es meistens erst möglich oder zumindest deutlich einfacher, die Sicherheit von Protokollen zu beweisen, die Hashfunktionen benutzen. Man nennt dieses Vorgehen Random-Oracle-Methodik. Es wird also zunächst die Hashfunktion gegen ein Random-Oracle ausgetauscht, dann die Sicherheit des Protokolls im Random-Oracle-Modell bewiesen und anschließend das Protokoll mit der Hashfunktion benutzt, davon ausgehend, dass die Sicherheit des Protokolls weiterhin gegeben ist.

In [FS87] wurde dies benutzt, um aus einem interaktiven Authentifikations-Protokoll einen Signatur-Algorithmus zu konstruieren. Dieser war deutlich effizienter als die bis dahin bekannten Signatur-Algorithmen. Hierbei gibt es zwar kein explizites Random-Oracle, allerdings wählt bei dem interaktiven Authentifikations-Protokoll der Verifizierer eine zufällige Nachricht, was sozusagen einer Anfrage an ein Random-Oracle gleich kommt und im Signatur-Algorithmus durch den Hashwert der zu signierenden Nachricht ersetzt wird. Dieses spezielle Vorgehen wird seitdem als Fiat-Shamir-Heuristik bezeichnet.

Eine Verallgemeinerung und Formalisierung dieser Idee wurde dann in [BR93] dargestellt. Dabei wurde der Begriff des Random-Oracles geprägt und gezeigt, wie sich damit effiziente Protokolle erstellen lassen. Dies wurde anhand von Verschlüsselungen, Signaturen und Zero-Knowledge-Beweisen gezeigt. Hierzu musste allerdings auch noch definiert werden, was Zero-Knowledge im Random-Oracle-Modell bedeutet.

Es gibt aber auch Alternativen zum Random-Oracle, sofern es darum geht, dass zwei Parteien Zufallswerte erhalten sollen, die sie nicht beeinflussen können. Zunächst gibt es die einfache Möglichkeit, dass zwei Parteien sich interaktiv auf einen Zufallswert einigen, was z. B. durch [Blu83] erledigt werden kann unter der Standardannahme, dass

die Primfaktorzerlegung von großen Zahlen schwer ist, oder unter der etwas stärkeren RSA-Annahme. Hierbei ist aber zu bemerken, dass zum einen eine Interaktion der beiden Parteien nötig ist und zum anderen, dass auch nur diese beide Parteien von der Zufälligkeit des Zufallswertes überzeugt sind, alle anderen Parteien können das nicht nachprüfen. Das Ganze ließe sich zwar auf mehrere Parteien ausdehnen, aber da in vielen Fällen eine Interaktion nicht möglich ist oder die Anzahl von Parteien nicht vorher bekannt ist, wird für diese Fälle eine andere Möglichkeit benötigt.

Wenn nur eine begrenzte Anzahl von Zufallsbits benötigt wird, kann man ein „Common Random String“ [BFM88] benutzen, welcher eine Folge vorher festgelegter und öffentlicher Zufallsbits ist. Mit Hilfe solcher Folgen lassen sich z. B. aus interaktiven Zero-Knowledge-Protokollen nicht-interaktive Protokolle erzeugen. Weitere Namen für dieses und ähnliche Modelle sind „Common Reference String“ [Pas03, CF01] und „Auxiliary String“ [Dam00] bzw. „Public Parameter“ [FF00].

Ein weiteres Modell liefert die Benutzung von „Beacons“, das zwischen dem Random-Oracle-Modell und dem Common-Random-String-Modell angesiedelt ist. Es wurde von Rabin [Rab83] eingeführt und in [MRH04] in Beziehung gesetzt zu Random-Oracles und Common Random Strings. Bei dieser Betrachtung wird das Beacon, hier dann „asynchronous beacon“ genannt, als eine Zufallsfunktion behandelt, die sich im Prinzip genauso verhält wie ein Random-Oracle, allerdings für eine Eingabe x, wenn man diese als Zahl interpretiert, x Schritte bis zur Ausgabe benötigt, wohingegen bei einem Random-Oracle diese Zeit für alle Eingaben konstant ist. Die Autoren zeigen auch, dass sich diese drei Modelle grundlegend voneinander unterscheiden, indem sie beweisen, dass ein Random-Oracle nicht durch ein Beacon ersetzt werden kann und ein Beacon nicht durch ein Common Random String. Mit Ersetzen ist hier jeweils gemeint, dass es keinen Algorithmus bzw. PT-Algorithmus gibt, so dass dieser z. B. mit Hilfe eines Beacons ein Random-Oracle simulieren kann und die jeweiligen Ausgaben ununterscheidbar sind. Man nennt dies auch Reduktion, in diesem Beispiel von einem Random-Oracle zu einem Beacon.

Diese Reduktionen sind auch ein beliebtes Beweismittel, wobei das Vorgehen meistens so ist, dass man die Sicherheit eines Protokolls zeigt, indem man annimmt man hätte einen Angreifer, der erfolgreich das Protokoll brechen könnte, und dann einen Algorithmus konstruiert, der mit Hilfe des Angreifers ein grundlegendes Problem lösen könnte, wie z. B. das RSA-Problem. Speziell im Random-Oracle-Modell kann man diese Reduktionsbeweise in zwei Arten unterteilen. Die einen gestatten dem Algorithmus des Angreifers auch weiterhin normalen Zugriff auf das Random-Oracle (wie z. B. im Beispiel von [MRH04]), die anderen simulieren die Antworten des Random-Oracles für den angreifenden Algorithmus, so dass dieser keinen Unterschied bemerkt (wie z. B. im Beweis zu Satz 6.3.7), aber die Werte eine bestimmte Eigenschaft haben, mit deren Hilfe das grundlegende Problem gelöst werden kann. Das erste Modell wird dabei als Nicht-programmierbares-Random-Oracle-Modell bezeichnet und das zweite als Programmierbares-Random-Oracle-Modell. Auch diese beiden Modelle sind wesentlich verschieden voneinander [Nie02], da es in dem letzten der beiden Modelle möglich ist so-

genannte „non-interactive non-committing encryption“ Protokolle zu konstruieren, was in dem ersten Modell nicht der Fall ist.

Unter diesen vielen Möglichkeiten spielt die Benutzung des Random-Oracle-Modells (wobei in der Regel die programmierbare Version gemeint ist), also die Random-Oracle-Methodik eine herausragende Rolle, da sie zum einen häufig benutzt wird und zum anderen aber nicht klar ist, inwieweit die Sicherheit dabei gewährleistet bleibt (siehe auch Abschnitt 6.2). Bevor wir zur Sicherheitsbetrachtung kommen, schauen wir uns zunächst noch einige ausgewählte Anwendungen an, die die Random-Oracle-Methodik für den Sicherheitsbeweis benutzen. Zunächst wäre da die RSA-OAEP-Verschlüsselung [BR94], in der sogar zwei Hashfunktionen zum Einsatz kommen und welche die deterministische RSA-Verschlüsselung zu einer zufälligen Verschlüsselung macht. Die Sicherheit von RSA-OAEP konnte zunächst nur im Random-Oracle-Modell bewiesen werden [FOPS01, Sho01], später aber sogar ohne Benutzung eines Random-Oracles [BF06]. Ein weiteres sehr häufig benutztes Verfahren ist das sogenannte Hash-and-Sign, bei dem zuerst ein Hashwert einer Nachricht gebildet wird und anschließend dieser Hashwert mit einem Signatur-Verfahren signiert wird. Es kann gezeigt werden, dass dieses Vorgehen sicher ist, wenn die Hashfunktion kollisionsresistent ist und das Signatur-Verfahren (für kurze Nachrichten) sicher ist gegen existentielle Fälschung unter einem adaptiven Angriff mit gewählten Nachrichten [Gol04]. Sowohl für die RSA- als auch für die ElGamal-Signatur trifft das letztere ohne Benutzung einer Hashfunktion nicht zu [MOV96]. Es gibt aber auch Hash-and-Sign Verfahren, deren Sicherheit im Standard-Modell bewiesen wurde, wie z. B. [GHR99]. Hierbei werden allerdings starke Bedingungen an die Hashfunktion gestellt, die aber erfüllt werden können, wie von den Autoren gezeigt wird. Ein ähnlicher Ansatz ist das sogenannte „Full-Domain-Hash“, wobei auch zunächst von der Nachricht ein Hashwert gebildet wird und dann eine Trapdoor-Einweg-Permutation darauf angewendet wird, so wie es z. B. in [DH76] schon vorgeschlagen wurde. Sofern nicht noch weitere Eigenschaften der Trapdoor-Einweg-Permutation gefordert werden, wie z. B. die Homomorphie bei RSA, kann es laut [DOP05] keinen Blackbox-Reduktionsbeweis für die Sicherheit von „Full-Domain-Hash“ geben. Wobei hierbei aber nur eine Aussage über „fully-BB reductions“ und „relativizing reductions“ gemacht wird, wie sie in [RTV04] definiert werden. Weitere Anwendungen für das Random-Oracle-Modell sind z. B. Bezahlsysteme [Bra93], die Schnorr-Signatur [Sch89] oder generell Zero-Knowledge-Beweise [CS97].

6.2 Unsicherheit der Random-Oracle-Methodik

So schön und plausibel die Random-Oracle-Methodik auch ist, es lassen sich trotzdem relativ leicht Beispiele konstruieren, die zeigen, dass die Methodik im Allgemeinen unsicher ist. Der wesentliche Unterschied zwischen dem Random-Oracle-Modell und der Implementierung mit Hashfunktionen ist, dass im Random-Oracle-Modell zwar jeder Zugriff auf das Orakel hat, aber keine weiteren Informationen darüber besitzt, wie z. B.

den internen Aufbau oder Ähnliches. Wenn man hingegen ein Protokoll mit Hashfunktionen betrachtet, kennt man die Hashfunktion sehr gut und kann sich evtl. interne Strukturen zu Nutze machen, um einen Angriffspunkt auf das Protokoll zu finden. Dies wird im Folgenden vorgeführt.

Zu Beginn wollen wir uns ein ganz einfaches, eher informelles Beispiel anschauen, wie man einen sicheren Signatur-Algorithmus so ändern kann, dass er im Random-Oracle-Modell sicher ist, aber mit einer speziellen Hashfunktion unsicher wird.

Beispiel 6.2.1. *Es sei* Sig *ein Signatur-Algorithmus eines sicheren Protokolls und* $\{h_i\}$ *eine Familie von Hashfunktionen, wobei* $2^{-\lambda(n)}$ *vernachlässigbar in* n *ist. Ferner sei* $\mathcal{H}$ *entweder eine Hashfunktion oder ein Random-Oracle, je nachdem in welchem Modell man sich gerade befindet. Wir betrachten nun folgendes Protokoll* Sig′*:*

Wenn $\mathcal{H}(1) = h_i(1)$*,*
dann gib den geheimen Schlüssel aus,
ansonsten führe Sig *aus und gib dessen Ausgabe zurück.*

Im Random-Oracle-Modell ist $\mathcal{H}(1)$ *ein zufälliger Wert, der nur mit der in* n *vernachlässigbaren Wahrscheinlichkeit* $2^{-\lambda(n)}$ *gleich* $h_i(1)$ *ist. Wenn man* $\mathcal{H}$ *aber durch* h_i *ersetzt, ist die obige Bedingung immer erfüllt und es wird der geheime Schlüssel ausgegeben, was das Protokoll absolut unsicher macht.*

Dies ist ein sehr künstliches Protokoll, das wohl niemand benutzen würde, aber es verdeutlicht das prinzipielle Vorgehen der allgemeineren Beispiele.

Das erste Gegenbeispiel für die Sicherheit der Random-Oracle-Methodik war [CGH98]. Es wurde darin gezeigt, dass es Signaturen und Verschlüsselungen gibt, die im Random-Oracle-Modell sicher sind, aber absolut unsicher werden sobald man sie mit einer Hashfunktion implementiert. Dabei spielt es keine Rolle welche Hashfunktion man benutzt. In der neueren Version [CGH04b] findet sich auch ein allgemeiner Überblick über die Random-Oracle-Methodik und weitere Arbeiten in diesem Gebiet. Auch der etwas speziellere Fall der Fiat-Shamir-Heuristik ist im Allgemeinen nicht sicher, was in [GK03] gezeigt wird. Weitere Arbeiten, in denen die Unsicherheit der Random-Oracle-Methodik dargestellt wird, sind z. B.: [Nie02], die den Unterschied zwischen dem Programmierbares- und Nicht-programmierbares-Random-Oracle-Modell zeigt (vgl. Abschnitt 6.1); [CGH04a], die beweist, dass das Fehlverhalten der Random-Oracle-Methodik auch bei Signaturen mit Nachrichten kurzer, vorher festgelegter Länge eintreten kann; [BBP04], die ein natürlich aussehendes Protokoll für eine Hybrid-Verschlüsselung behandelt, das im Random-Oracle-Modell sicher ist, aber trotzdem im Standard-Modell mit einer speziellen symmetrischen Verschlüsselung unsicher wird, was man dem Protokoll auf den ersten Blick nicht ansieht, wie z. B. bei dem in [MRH04] verwendeten Protokoll (siehe auch Abschnitt 6.1). Die Idee dabei ist in den meisten Fällen im Prinzip immer ähnlich, man nutzt aus, dass sich eine Hashfunktion berechnen bzw. beschreiben lässt, was bei einem Random-Oracle nicht der Fall ist, da diese im Allgemeinen eine unendlich große

Beschreibung (als Turing-Maschine) haben. Auch wenn es sehr viele Beispiele gibt, die zeigen, dass es Protokolle gibt, die sicher im Random-Oracle-Modell, aber unsicher im Standard-Modell sind, gibt es anscheinend doch mehr Protokolle, die im Random-Oracle-Modell unsicher sind, für die es aber eine sichere Implementierung im Standard-Modell gibt (siehe [NK08]).

Wir schauen uns das Beispiel von [MRH04] genauer an, da es relativ einfach ist, im Vergleich zu den anderen. Der grobe Aufbau ist der Gleiche wie in Beispiel 6.2.1. Allerdings wird sich hier nicht auf eine feste Hashfunktion festgelegt, sondern es wird eine Funktion benutzt, die durch die Nachricht beschrieben wird und somit ein beliebiger Algorithmus sein kann, der sich zwar wie jede Hashfunktion verhalten kann, aber nicht wie ein Random-Oracle.

Satz 6.2.2. *Es sei* $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ *ein Signatur-Protokoll, dass sicher ist gegen existentielle Fälschung unter einem adaptiven Angriff mit gewählten Nachrichten im Random-Oracle-Modell. Dann gibt es ein Signatur-Protokoll* $(\mathrm{Gen}, \mathrm{Sig}', \mathrm{Ver}, \{h_i\})$*, das im Random-Oracle-Modell die gleiche Sicherheit bietet, aber im Standard-Modell unter dem gleichen Angriff unsicher wird und das unabhängig von der gewählten Familie von Hashfunktionen* $\{h_i\}$*.*

Beweis. Der Beweis teilt sich in zwei Teile. Zuerst wird die Konstruktion von Sig' beschrieben und anschließend wird analysiert, wie sich das neue Signatur-Protokoll verhält.

Konstruktion: Wir konstruieren Sig' wie folgt (vgl. Abbildung 6.1): Zunächst wird ein PT-Orakel-Algorithmus $D^{\mathcal{H}}$ mit der Nachricht m als Eingabe aufgerufen, der zwischen dem Standard-Modell und dem Random-Oracle-Modell unterscheiden soll. Das Orakel von D ist entweder ein Random-Oracle $\mathcal{R}^{\lambda(n)}$ oder die Hashfunktion h_i. Wenn D den Wert 1 ausgibt, dann gibt Sig' das Schlüsselpaar pk, sk aus, was zu einem totalen Brechen der Signatur führt. Wenn D den Wert 0 zurückgibt, dann ruft Sig' den ursprünglichen Algorithmus Sig auf und gibt dessen Ausgabe zurück.

Es bleibt zu klären wie der Algorithmus D vorgeht. Zunächst interpretiert D die Eingabe m als $\pi 01^t$, wobei π ein Programm ist, das z. B. als Eingabe für eine universelle Turingmaschine dient oder auch ein „normales“ Programm in einer gängigen Programmiersprache für einen „normalen“ Computer sein kann. Es sei $q = 2|\pi| + n$, wobei n der Sicherheitsparameter für das Protokoll ist. Algorithmus D simuliert nun für $x = 1, \ldots, q$ jeweils maximal t Schritte des Programms π mit der Eingabe x und merkt sich die Ausgaben $\pi(1), \ldots, \pi(q)$. Falls das Programm vorzeitig abgebrochen wird, ist die Ausgabe leer, also $\pi(x) = \varepsilon$. Ebenso fragt D sein Orakel $\mathcal{H}$ nach den gleichen Werten und merkt sich die Antworten $\mathcal{H}(1), \ldots, \mathcal{H}(q)$. Wenn für alle $x \in \{1, \ldots, q\}$ die Werte $\pi(x)$ und $\mathcal{H}(x)$ übereinstimmen, dann gibt D den Wert 1 zurück, ansonsten den Wert 0. Falls es zu Beginn bei der Interpretation von m zu Problemen kommt, weil die Nachricht keine 0 enthält, gibt D ebenfalls den Wert 0 zurück.

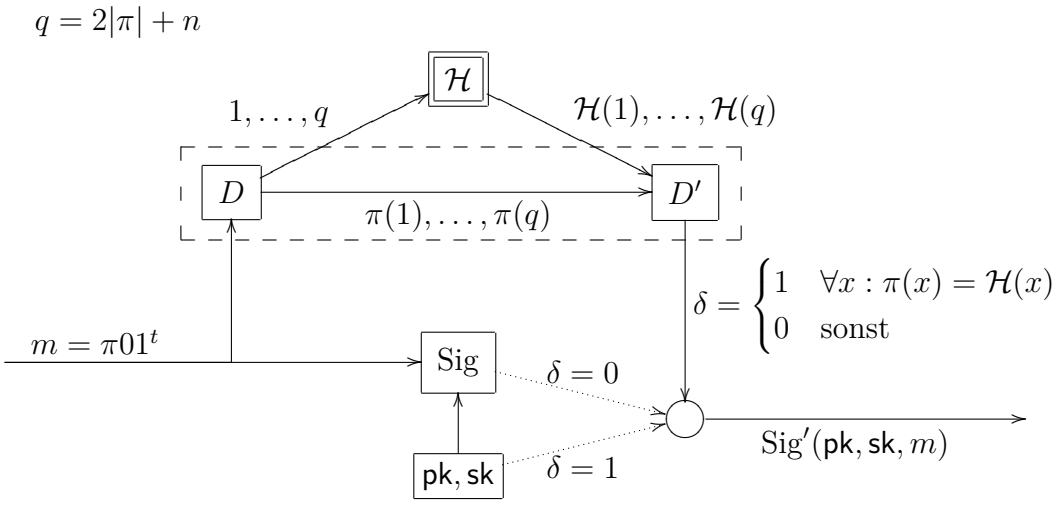

Abbildung 6.1: Unsichere Anwendung der Random-Oracle-Methodik

Analyse: Der Algorithmus Sig' ist weiterhin effizient, da sich die Laufzeit von Sig' wie folgt zusammensetzt: Zunächst wird die Eingabe interpretiert, was in $O(|m|)$ Schritten abgearbeitet werden kann. Die anschließende Berechnung von $\pi(1), \ldots, \pi(q)$ kostet $O(tq)$ Schritte. Da $t \leq |m|$ und $q \leq 2|m| + n$ ist, ist das Produkt tq durch ein Polynom (in der Eingabelänge) beschränkt. Die Orakelanfragen und Vergleiche kosten weitere $O(q\,\lambda(n))$ Schritte und der Aufruf von Sig ist auch weiterhin effizient.

Im Random-Oracle-Modell ist das neue Signatur-Protokoll auch weiterhin sicher, da D nur mit einer in n vernachlässigbaren Wahrscheinlichkeit den Wert 1 zurückgibt, wie wir gleich sehen werden. Es sei $p_{\pi,x}$ die Wahrscheinlichkeit, dass $\pi(x) = \mathcal{R}^{\lambda(n)}(x)$ ist, ohne dass das Programm vorzeitig beendet wird, also

$$p_{\pi,x} := P\left(\pi(x) = \mathcal{R}^{\lambda(n)}(x)\right),$$

welche höchstens $\frac{1}{2}$ sein kann (sehr grob abgeschätzt). Die Wahrscheinlichkeit p_π, dass die Ausgaben von π und $\mathcal{R}^{\lambda(n)}$ für alle Eingaben $x = 1, \ldots, q$ übereinstimmen, ist demnach

$$p_\pi = \prod_{x=1}^{q} p_{\pi,x} \leq \left(\frac{1}{2}\right)^q = 2^{-q}.$$

Die Wahrscheinlichkeit p_l, dass die Ausgaben für irgendein Programm π der Länge $|\pi| = l$ übereinstimmen, lässt sich dann durch

$$p_l \leq \sum_{\pi \in \{0,1\}^l} p_\pi \leq 2^l \cdot 2^{-q} = 2^l \cdot 2^{-2l-n} = 2^{-l-n}$$

abschätzen. Und die Wahrscheinlichkeit p, dass es überhaupt ein Programm beliebiger Länge gibt, so dass die Ausgaben mit denen des Random-Oracles übereinstimmen, lässt

sich wiederum durch

$$p \leq \sum_{l=1}^{\infty} p_l \leq \sum_{l=1}^{\infty} 2^{-l-n} \leq 2^{-n} \sum_{l=1}^{\infty} 2^{-l} = 2^{-n}$$

abschätzen, was vernachlässigbar in n ist. Damit verhält sich Sig$'$ im Random-Oracle-Modell fast immer genauso wie Sig, außer mit einer in n vernachlässigbaren Wahrscheinlichkeit.

Im Standardmodell ist es einfach ein Programm zu schreiben, dass die Hashwerte $h_i(x)$ berechnet, man muss nur aufpassen, dass man t groß genug wählt. Mit dem so aus der ersten Anfrage erhaltenen Schlüssel ist es dann leicht eine Signatur zu fälschen, sogar beliebige. □

Rein formal müsste man auch noch den Verifikations-Algorithmus anpassen, da ansonsten die Durchführbarkeit nicht mehr gewährleistet ist (in dem Fall wo das Schlüsselpaar ausgegeben wird). Das wurde hier aber vernachlässigt, da es den Sachverhalt nur unnötig verkomplizieren würde. Alternativ hätte man auch den Verifikations-Algorithmus entsprechend anpassen können. Dann wäre das Signatur-Protokoll existentiell fälschbar unter einem Angriff ohne bekannte Signaturen.

Mit ähnlichen Mitteln wie bei den obigen Beispielen lässt sich auch zeigen, dass das generische Gruppen-Modell (Generic Group model) die gleichen Schwachpunkte besitzt wie das Random-Oracle-Modell (siehe [Den02]). Im generischen Gruppen-Modell hat man nur Zugriff auf ein Orakel, das die Gruppenoperationen ausführt, aber ansonsten eine rein zufällige Kodierung benutzt. Damit kann man nur die Gruppeneigenschaften ausnutzen, aber keine weiteren speziellen Eigenschaften der Gruppe. Bei der Implementierung legt man sich dann auf eine Gruppe fest und hofft wie bei der Random-Oracle-Methodik, dass die Sicherheit erhalten bleibt. Es gibt aber kryptographische Protokolle, die im generischen Gruppen-Modell sicher sind, jedoch unsicher werden sobald man das Protokoll mit einer beliebigen Gruppe implementiert. Ein weiteres Modell mit der gleichen Schwäche ist das Ideal-Cipher-Model (siehe [Bla05]). Dieses Verhalten scheint also ein allgemeines Problem der Generalisierung von Funktionen und Algorithmen zu sein.

6.3 Sicherheit mit neuen Hashfunktionen

Obwohl das Beispiel im Satz 6.2.2 zeigt, dass die Random-Oracle-Methodik im Allgemeinen unsicher ist, erfreut sie sich aufgrund ihrer Einfachheit großer Beliebtheit. Es stellt sich also die Frage, ob es nicht möglich ist, dieses Verfahren doch irgendwie auf sicherere Beine zu stellen. Am schönsten wäre es natürlich, wenn man einfach nur ein paar zusätzliche Bedingungen an die benutzte Hashfunktion stellen müsste und damit die Sicherheit auch im Standard-Modell erhielte. Wie das Beispiel aber gezeigt hat, ist dies nicht möglich, da es für alle Hashfunktionen unsicher im Standard-Modell wird. Man kommt also nicht umhin auch an das Protokoll Bedingungen zu stellen. Wie sich

herausstellen wird, ist die Forderung, dass das Protokoll vom Typ I oder II sein muss, eine solche geeignete Bedingung.

6.3.1 Protokolle vom Typ I unter schwachem Angriff

Für Protokolle von Typ I lässt sich bei Verwendung von unvorhersehbaren Hashfunktionen die Sicherheit im Standard-Modell aus der Sicherheit im Random-Oracle-Modell folgern, allerdings nur für einfache Angriffe, nämlich Angriffe ohne bekannte Signaturen.

Satz 6.3.1. *Es sei* $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ *ein Protokoll vom Typ I und* $\{h_i\}$ *eine unvorhersehbare Familie von Hashfunktionen vom Grad 1. Ferner sei* $|\sigma| \leq \iota(n)/2$*, wobei* n *der vom Protokoll und der Hashfamilie gemeinsam benutzte Sicherheitsparameter ist. Wenn das Protokoll im Random-Oracle-Modell sicher ist gegen existentielle Fälschung unter einem Angriff ohne bekannte Signaturen, dann ist es auch im Standard-Modell sicher gegen existentielle Fälschung unter einem Angriff ohne bekannte Signaturen.*

Beweis. Es sei $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ ein Protokoll vom Typ I und $\{h_i\}$ eine unvorhersehbare Familie von Hashfunktionen vom Grad 1. Das Protokoll sei im Random-Oracle-Modell sicher gegen existentielle Fälschung unter einem Angriff ohne bekannte Signaturen.

Angenommen das Protokoll ist im Standard-Modell nicht sicher gegen existentielle Fälschung unter einem Angriff ohne bekannte Signaturen, dann gibt es also einen PPT-Algorithmus A (der Angreifer), so dass die Wahrscheinlichkeit

$$P\left(\mathrm{Ver}^{h_i}(\mathsf{pk}, m, \sigma) = 1 : (\mathsf{pk}, \mathsf{sk}, i) = \mathrm{Gen}(1^n), (m, \sigma) = A(i, \mathsf{pk})\right)$$

nicht vernachlässigbar in n ist. Wir setzen nun $k = \mathsf{pk}$ und interpretieren die Ausgabe (m, σ) von A als (x, z) mit $x = m$ und $z = \sigma$. Weiterhin setzen wir noch $a = h_i(x)$. Es sei nun D der Algorithmus, der bei Eingabe von $(1^n, k, a, z)$ den Algorithmus Sub' von Ver mit der Eingabe (k, z_1, a, z_2) aufruft und dessen Ergebnis zurückgibt.

Da die Hashfamilie $\{h_i\}$ unvorhersehbar ist, gibt es also einen PPT-Algorithmus B, für den im Random-Oracle-Modell bei analoger Konstruktion die entsprechende Wahrscheinlichkeit ebenfalls nicht vernachlässigbar ist. Dies ist aber ein Widerspruch zur Sicherheit des Protokolls im Random-Oracle-Modell. □

6.3.2 Protokolle vom Typ II unter starkem Angriff

Für Protokolle vom Typ II gilt vermutlich auch, dass sich die Sicherheit im Random-Oracle-Modell bei einem Angriff mit gewählten Nachrichten ins Standard-Modell überträgt. Die Sachlage ist allerdings ungleich komplizierter, da sich bis jetzt noch nicht genau modellieren lässt, was ein Angreifer bei einem erfolgreichen Angriff an Informationen benötigt bzw. erhält (oder auch nicht). Wir schauen uns zunächst die Verteilungen der Variablen im Protokoll bei einem Angriff mit gewählten Nachrichten an. Hierfür bietet

unser Modell genügend formale Grundlagen, um eine präzise Aussage treffen zu können, nämlich dass sich fast alle Variablen in beiden Modellen rechnerisch gleich verhalten. Zunächst aber noch ein kleiner technischer Hilfssatz.

Satz 6.3.2. *Es sei $\lambda \in \mathsf{poly} \cap \omega(\log_2(\mathcal{N}))$ mit Werten in $\mathbb{N}$ und U_n uniform verteilte $\{0,1\}^{\lambda(n)}$-wertige Zufallsvariablen. Des Weiteren sei $\left\{ f_n \colon \{0,1\}^{\lambda(n)} \to \{0,1\}^* \;\middle|\; n \in \mathbb{N} \right\}$ eine Familie aus injektiven Funktionen, für die es einen PT-Algorithmus F gibt mit $F(x, 1^n) = f_n(x)$. Dann gilt für alle PPT-Orakel-Algorithmen A, dass $\left(\mathcal{R}^{\lambda(n)}(f_n(U_n))\right)$ und $\left(A^{\mathcal{R}^{\lambda(n)}}(1^n)\right)$ rechnerisch unabhängig sind.*

Beweis. Wir zeigen dies, indem wir nachweisen, dass für uniform verteilte $\{0,1\}^{\lambda(n)}$-wertige Zufallsvariablen $\tilde{U}_n$, die stochastisch unabhängig von allen anderen Zufallsvariablen sind,

$$\left(\mathcal{R}^{\lambda(n)}(f_n(U_n)), A^{\mathcal{R}^{\lambda(n)}}(1^n)\right) \sim_c \left(\tilde{U}_n, A^{\mathcal{R}^{\lambda(n)}}(1^n)\right)$$

gilt. Es seien zunächst $M_{n,s,u}$ die Mengen der Anfragen, die $A_u^{\mathcal{R}_s^{\lambda(n)}}(1^n)$ an $\mathcal{R}_s^{\lambda(n)}$ stellt. Dann gilt für alle PPT-Algorithmen D:

$$\begin{aligned}
&P_{r,s,t,u}\left(D_r\left(\mathcal{R}_s^{\lambda(n)}(f_n(U_n(t))), A_u^{\mathcal{R}_s^{\lambda(n)}}(1^n)\right) = 1\right) \\
&= P_{r,s,t,u}\left(D_r\left(\mathcal{R}_s^{\lambda(n)}(f_n(U_n(t))), A_u^{\mathcal{R}_s^{\lambda(n)}}(1^n)\right) = 1 \wedge f_n(U_n(t)) \in M_{n,s,u}\right) \qquad (*) \\
&\quad + P_{r,s,t,u}\left(D_r\left(\mathcal{R}_s^{\lambda(n)}(f_n(U_n(t))), A_u^{\mathcal{R}_s^{\lambda(n)}}(1^n)\right) = 1 \wedge f_n(U_n(t)) \notin M_{n,s,u}\right) \\
&= \nu(n) + P_{r,s,t,u}\left(D_r\left(\mathcal{R}_s^{\lambda(n)}(f_n(U_n(t))), A_u^{\mathcal{R}_s^{\lambda(n)}}(1^n)\right) = 1 \wedge f_n(U_n(t)) \notin M_{n,s,u}\right) \\
&= \nu(n) + P_{r,s,t,u,v}\left(D_r\left(\tilde{U}_n(v), A_u^{\mathcal{R}_s^{\lambda(n)}}(1^n)\right) = 1 \wedge f_n(U_n(t)) \notin M_{n,s,u}\right) \qquad (+) \\
&= \nu(n) + P_{r,s,t,u,v}\left(D_r\left(\tilde{U}_n(v), A_u^{\mathcal{R}_s^{\lambda(n)}}(1^n)\right) = 1\right) \\
&\quad - P_{r,s,t,u,v}\left(D_r\left(\tilde{U}_n(v), A_u^{\mathcal{R}_s^{\lambda(n)}}(1^n)\right) = 1 \wedge f_n(U_n(t)) \in M_{n,s,u}\right) \qquad (**) \\
&= \nu(n) + P_{r,s,u,v}\left(D_r\left(\tilde{U}_n(v), A_u^{\mathcal{R}_s^{\lambda(n)}}(1^n)\right) = 1\right) - \nu'(n),
\end{aligned}$$

wobei ν und ν' vernachlässigbar sind, da in $(*)$ und $(**)$ die entsprechende Wahrscheinlichkeit jeweils kleiner ist als $P_{s,t,u}\left(f_n(U_n(t)) \in M_{n,s,u}\right) = \frac{|M_{n,s,u}|}{2^{\lambda(n)}}$ und diese wiederum vernachlässigbar ist wegen $|M_{n,s,u}| \leq g(n)$ für ein $g \in \mathsf{poly}$ und Satz 2.2.11.

Die Gleichung $(+)$ gilt, da wegen der zweiten Bedingung alle Anfragen von A an das Orakel ungleich $f_n(U_n)$ sind und damit per Definition die Antworten des Orakels an A stochastisch unabhängig sind von den Antworten des Orakels auf die Anfragen $f_n(U_n)$.

Da $\tilde{U}_n$ und $A^{\mathcal{R}^{\lambda(n)}}(1^n)$ stochastisch unabhängig sind, ist gemäß Definition 2.4.3 die Behauptung gezeigt. □

Es ist anzumerken, dass hier eine stochastische Unabhängigkeit nicht unbedingt gegeben ist, z. B. im Fall $A^{\mathcal{R}^{\lambda(n)}}(1^n) = \mathcal{R}^{\lambda(n)}(1^{\lambda(n)})$ und $f_n = id$. Kommen wir nun zu der Betrachtung der Verteilungen der beteiligten Variablen.

Satz 6.3.3. *Es sei* $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ *ein Protokoll vom Typ II und* $\{h_i\}$ *eine zufallserhaltende Familie von Hashfunktionen. Die gemeinsame Verteilung der Variablen* $\mathsf{pk}, \mathsf{sk}, z, y, \sigma_1, \sigma_2 = z'$ *im Random-Oracle-Modell ist von der im Standard-Modell rechnerisch ununterscheidbar, unabhängig vom Angreifer, d. h. die Wahl des Angreifers von* m *hat keinen Einfluss auf die rechnerische Ununterscheidbarkeit. Insbesondere ist* y *rechnerisch ununterscheidbar von einer uniform verteilten Zufallsvariable und rechnerisch unabhängig von* $\mathsf{pk}, \mathsf{sk}, z, z' = \sigma_2, m$ *sowie im Standard-Modell auch von* i*. Ebenso ist* σ_1 *im Standard-Modell rechnerisch unabhängig von* i*.*

Beweis. Zunächst haben per Konstruktion $\mathsf{pk}, \mathsf{sk}, z$ und $z' = \sigma_2$ dieselbe gemeinsame Verteilung, da sie in beiden Modellen von denselben Algorithmen mit denselben Eingaben bzw. Eingaben mit derselben Verteilung berechnet werden. Es bleiben also noch y und σ_1 zu betrachten. Wenn y sich ebenfalls in beiden Modellen rechnerisch gleich verhält, auch in Bezug auf sk und z, dann folgt dies auch für σ_1, da der Algorithmus Sub_2 als Eingabe nur sk, z und y erhält, deren gemeinsame Verteilung dann in beiden Modellen rechnerisch nicht zu unterscheiden ist und daher auch alle von diesen Variablen durch einen PPT-Algorithmus berechneten Werte. Es seien zunächst noch U_n uniform verteilte $\{0,1\}^{\lambda(n)}$-wertige Zufallsvariablen, die stochastisch unabhängig von allen anderen Variablen sind.

Wir betrachten uns als erstes die Sachlage im Standard-Modell. Für die zufallerhaltende Hashfamilie $\{h_i\}$ gilt per Definition

$$(i, h_i(1r)) \sim_c (i, \mathcal{R}^{\lambda(n)}(1)) \sim (i, U_n).$$

Dann ist auch

$$((\mathsf{pk}, \mathsf{sk}, z, z', m, i), h_i(1r)) \sim_c ((\mathsf{pk}, \mathsf{sk}, z, z', m, i), U_n),$$

denn angenommen es gäbe einen Angreifer A, so dass $((\mathsf{pk}, \mathsf{sk}, z, z', m, i), h_i(1r))$ rechnerisch unterscheidbar von $((\mathsf{pk}, \mathsf{sk}, z, z', m, i), U_n)$ wäre, es also einen Unterscheider D gäbe, der den Unterschied bemerkt, dann wäre auch $(i, h_i(1r))$ rechnerisch unterscheidbar von (i, U_n). Wir könnten nämlich einen passenden Unterscheider D' konstruieren, indem D' bei Eingabe von i und x zunächst den Algorithmus Gen des Protokolls aufruft, sk und pk übernimmt und i verwirft, da er diesen Wert schon hat (er hat dieselbe Verteilung und ist ebenfalls unabhängig von sk und pk). Dann ruft er den Algorithmus A auf und mit dessen Ergebnissen Sub_1 des Protokolls, womit D' die Werte $\mathsf{pk}, \mathsf{sk}, z, z'$ und m erhält, die er dann zusammen mit i und x an D übergibt, der dann die Entscheidung trifft. Die Konstruktion von D' ist so gewählt, dass die simulierten Werte zusammen mit der Eingabe von D' dieselbe Verteilung haben wie die Eingabe von D bei einem Angriff von A auf das Protokoll.

Damit folgt nach Satz 2.4.8, dass

$$((\mathsf{pk}, \mathsf{sk}, z, z', m, i), h_i(0z'm) \oplus h_i(1r)) \sim_c ((\mathsf{pk}, \mathsf{sk}, z, z', m, i), h_i(1r))$$

gilt und nach Satz 2.4.6, dass

$$((\mathsf{pk}, \mathsf{sk}, z, z', m, i), h_i(0z'm) \oplus h_i(1r)) \sim_c ((\mathsf{pk}, \mathsf{sk}, z, z', m, i), U_n)$$

gilt. Insgesamt gilt dann auch, dass $(\mathsf{pk}, \mathsf{sk}, z, z', m, i)$ und $y = h_i(0z'm) \oplus h_i(1r)$ rechnerisch unabhängig sind. Außerdem ist y rechnerisch ununterscheidbar von U_n.

Im Random-Oracle-Modell besitzen $\mathsf{pk}, \mathsf{sk}, z, z' = \sigma_2$ wie bereits oben gesagt dieselbe gemeinsame Verteilung wie im Standard-Modell. Nach Satz 6.3.2 ist $(\mathsf{pk}, \mathsf{sk}, z, z', m)$ rechnerisch unabhängig von $\mathcal{R}^{\lambda(n)}(1r)$, da hier genauso gut angenommen werden kann, dass $(\mathsf{pk}, \mathsf{sk}, z, z', m)$ von einem einzigen PPT-Orakel-Algorithmus erzeugt wird. Damit folgt nach Satz 2.4.8, dass

$$((\mathsf{pk}, \mathsf{sk}, z, z', m), \mathcal{R}^{\lambda(n)}(0z'm) \oplus \mathcal{R}^{\lambda(n)}(1r)) \sim_c ((\mathsf{pk}, \mathsf{sk}, z, z', m), \mathcal{R}^{\lambda(n)}(1r))$$

gilt und nach Satz 2.4.6, dass

$$((\mathsf{pk}, \mathsf{sk}, z, z', m), \mathcal{R}^{\lambda(n)}(0z'm) \oplus \mathcal{R}^{\lambda(n)}(1r)) \sim_c ((\mathsf{pk}, \mathsf{sk}, z, z', m), U_n)$$

gilt. Insgesamt gilt dann auch, dass $(\mathsf{pk}, \mathsf{sk}, z, z', m)$ und $y = \mathcal{R}^{\lambda(n)}(0z'm) \oplus \mathcal{R}^{\lambda(n)}(1r)$ rechnerisch unabhängig sind. Außerdem ist y rechnerisch ununterscheidbar von U_n.

In beiden Modellen wird σ_1 durch den gleichen Algorithmus aus den Variablen sk, z, y berechnet, deren gemeinsame Verteilungen in den jeweiligen Modellen nach Satz 2.4.6 rechnerisch ununterscheidbar sind. Deshalb ist nicht nur die gemeinsame Verteilung von $\mathsf{pk}, \mathsf{sk}, z, y, z' = \sigma_2$ nach Satz 2.4.6 im Standard-Modell rechnerisch ununterscheidbar von der im Random-Oracle-Modell, sondern auch die von $\mathsf{pk}, \mathsf{sk}, z, y, \sigma_1, z' = \sigma_2$.

Weil y und i im Standard-Modell rechnerisch unabhängig sind, ist auch (sk, z, y) rechnerisch unabhängig von i und damit auch $\sigma_1 = \mathrm{Sub}_2(\mathsf{sk}, z, y)$: Aus einem Unterscheider D, der $((\mathsf{sk}, z, y), i)$ von $((\mathsf{sk}, z, y), I(1^n))$ unterscheiden kann, wobei $I(1^n)$ stochastisch unabhängig von den restlichen Variablen ist, lässt sich ein Unterscheider D' konstruieren, der (y, i) und $(y, I(1^n))$ unterscheiden kann. Bei Eingabe von y und i berechnet D' die Werte $(\mathsf{pk}, \mathsf{sk}, i') = \mathrm{Gen}(1^n)$ und $(z, z') = \mathrm{Sub}_1(\mathsf{pk})$. Er gibt dann den Wert von $D((\mathsf{sk}, z, y), i)$ zurück. Die Konstruktion von D' ist dabei so gewählt, dass die simulierten Werte zusammen mit der Eingabe von D' nach Satz 2.4.6 rechnerisch ununterscheidbar sind von der Eingabe von D bei einem Angriff von A auf das Protokoll. □

Es ist hier noch zu bemerken, dass auch die Variable $r = \sigma_3$ in beiden Modellen dieselbe Verteilung hat, allerdings die gemeinsamen Verteilungen aller Variablen in den beiden Modellen nicht mehr rechnerisch ununterscheidbar sind. Ein möglicher Angriffspunkt kann also, wenn überhaupt, nur mit Hilfe der Variablen r und der Verknüpfung mit anderen Variablen gefunden werden. Wenn man die Sicherheit durch unvorhersehbare Hashfunktionen weiter erhöht, dann lassen sich die benutzten Hashwerte bei einem

Angriff vermutlich so schlecht kontrollieren, dass ein erfolgreicher Angriff reiner Zufall wäre. Wir schauen uns zunächst einmal an, welche Möglichkeiten wir beweisbar ausschließen können.

Satz 6.3.4. *Es sei* $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ *ein Protokoll vom Typ II und* $\{h_i\}$ *eine unvorhersehbare Familie von Hashfunktionen vom Grad 4, die auch zufallerhaltend ist. Ferner seien* $F_n = \{f_{n,j} \colon \{0,1\}^{\lambda(n)} \to W_{n,j} \mid j \in J_n\}$ *mit* $W_{n,j} \subseteq \{0,1\}^*$*, so dass gilt:*

(i) J_n *lässt sich von einem PT-Algorithmus bei Eingabe von* 1^n *berechnen, also insbesondere* $|J_{\mathcal{N}}| \in \mathsf{poly}$*,*

(ii) es exisitiert ein PT-Algorithmus F *mit* $F(1^n, j, x) = f_{n,j}(x)$*,*

(iii) $\exists \nu \in \mathsf{negl} \colon \forall n \in \mathbb{N}, j \in J_n, w \in W_{n,j} \colon \frac{|f_{n,j}^{-1}(w)|}{2^{\lambda(n)}} \leq \nu(n)$*.*

Wenn das Protokoll im Random-Oracle-Modell sicher ist gegen existentielle Fälschung unter einem adaptiven Angriff mit gewählten Nachrichten und ein PPT-Algorithmus A *existiert, dem für das Protokoll im Standard-Modell eine existentielle Fälschung* $(\tilde{m}, \tilde{\sigma})$ *unter einem adaptiven Angriff mit gewählten Nachrichten gelingt, dann gilt für jede Anfrage von* A *und die dadurch berechneten Variablen (abgesehen von einer vernachlässigbaren Wahrscheinlichkeit):*

(1) $\forall j, j' \in J_n \colon f_{n,j}(y) \neq f_{n,j'}(\tilde{y})$*,*

(2) $\forall j, j' \in J_n \colon f_{n,j}(h_i(0\sigma_2 m)) \neq f_{n,j'}(h_i(0\tilde{\sigma}_2\tilde{m}))$*,*

(3) $\forall j, j' \in J_n \colon f_{n,j}(h_i(0\sigma_2 m)) \neq f_{n,j'}(h_i(1\tilde{\sigma}_3))$*,*

(4) $\forall j, j' \in J_n \colon f_{n,j}(h_i(1\sigma_3)) \neq f_{n,j'}(h_i(0\tilde{\sigma}_2\tilde{m}))$*,*

wobei $\tilde{y} = h_i(0\tilde{\sigma}_2\tilde{m}) \oplus h_i(1\tilde{\sigma}_3)$ *ist.*

Beweis. Wir zeigen zunächst, dass wegen der Unvorhersehbarkeit von $\{h_i\}$ auch (2) gilt. Da dem Angreifer A die existentielle Fälschung für eine neue Nachricht $\tilde{m}$ gelingen muss, gilt auf jeden Fall $m \neq \tilde{m}$. Wegen der Kollisionsresistenz ist

$$h_i(0\sigma_2 m) \neq h_i(0\tilde{\sigma}_2\tilde{m})$$

(abgesehen von einer vernachlässigbaren Wahrscheinlichkeit), wobei zu beachten ist, dass $|\sigma_2| = |\tilde{\sigma}_2| = \zeta(n)$ ist. Angenommen (2) würde nicht gelten, dann könnten wir einen Unterscheider D konstruieren, der der Unvorhersehbarkeit von $\{h_i\}$ widersprechen würde. Bei Eingabe von $(1^n, y_1, y_2)$ verhält sich D wie folgt (auf die Eingabe von z, die D laut Definition 4.2.1 der Unvorhersehbarkeit noch bekommen müsste, wird hier der Einfachheit halber verzichtet, da sie D ohnehin nicht benutzt):

- Wenn $y_1 = y_2$ ist, gib 0 zurück.

- Für alle $j, j' \in J_n$ überprüfe, ob $f_{n,j}(y_1) = f_{n,j'}(y_2)$ ist. Wenn dies für mindestens ein Paar (j, j') erfüllt ist, gib 1 zurück.

- Ansonsten gib 0 zurück.

Wegen (i) und (ii) ist D ein PT-Algorithmus. Wenn wir uns den Angreifer A anschauen, der das Protokoll angreift, stellen wir fest, dass dieser ein Orakel-Algorithmus ist, der Zugriff auf ein Signatur-Orakel besitzt, aber ein „Angreifer A' auf die Unvorhersehbarkeit“ von $\{h_i\}$ ein reiner PPT-Algorithmus ist. Das ist allerdings kein Problem, denn wir können A' alle Werte außer i simulieren lassen (inkl. pk und sk) und damit für A ein Signatur-Orakel simulieren, womit A' durch Benutzung von A ein einfacher PPT-Algorithmus wird. Dessen Ausgabe (x_1, x_2) bewirkt mit einer nicht vernachlässigbaren Wahrscheinlichkeit, dass $D(1^n, h_i(x_1), h_i(x_2)) = 1$ ist.

Betrachten wir nun einen PPT-Orakel-Algorithmen B (wie in Definition 4.2.1, aber ohne z und k, da sie von D ohnehin nicht verarbeitet werden) und seine Erfolgswahrscheinlichkeit. Zunächst eine einfachere Betrachtung ohne den Einfluss von B. Es ist

$$
\begin{aligned}
&P\left(D\left(1^n, \mathcal{R}^l(0), \mathcal{R}^l(1)\right) = 1 : l = \lambda(n)\right) \\
&= P\Big(\exists j, j' \in J_n \colon f_{n,j}(\mathcal{R}^l(0)) = f_{n,j'}(\mathcal{R}^l(1)) \wedge \mathcal{R}^l(0) \neq \mathcal{R}^l(1) : l = \lambda(n)\Big) \\
&\leq \sum_{j,j' \in J_n} P\Big(f_{n,j}(\mathcal{R}^l(0)) = f_{n,j'}(\mathcal{R}^l(1)) : l = \lambda(n)\Big) \\
&\leq \sum_{j,j' \in J_n} P\Big(\mathcal{R}^l(0) \in f_{n,j}^{-1}\left(f_{n,j'}(\mathcal{R}^l(1))\right) : l = \lambda(n)\Big) \\
&\leq \sum_{j,j' \in J_n} \sum_{x \in \{0,1\}^{\lambda(n)}} \frac{1}{2^{\lambda(n)}} \cdot P\Big(\mathcal{R}^l(0) \in f_{n,j}^{-1}\left(f_{n,j'}(x)\right) : l = \lambda(n)\Big) \\
&\leq \sum_{j,j' \in J_n} \sum_{x \in \{0,1\}^{\lambda(n)}} \frac{1}{2^{\lambda(n)}} \cdot \nu(n) \qquad \text{(wegen (iii))} \\
&\leq |J_n|^2 \cdot \sum_{x \in \{0,1\}^{\lambda(n)}} \frac{1}{2^{\lambda(n)}} \cdot \nu(n) \\
&\leq |J_n|^2 \cdot \nu(n)
\end{aligned}
$$

vernachlässigbar in n, nach Lemma 2.2.10 und 2.2.6.

Da die Werte des Random-Oracles für verschiedene Anfragen alle stochastisch unabhängig sind und keinerlei „Struktur“ besitzen, kann der PPT-Orakel-Algorithmus B nur zufällig (x_1, x_2) wählen und probieren, ob $D(1^n, \mathcal{R}^l(x_1), \mathcal{R}^l(x_2)) = 1$ ergibt (kann er selber berechnen). Selbst wenn er die getesteten Werte irgendwie sortiert oder speichert und damit seine weiteren Anfragen optimiert, ist die Wahrscheinlichkeit, bei einem weiteren Test erfolgreich zu sein, (wegen der stochastisch unabhängigen und uniformen Verteilung der Random-Oracle-Antworten für verschiedene Anfragen) trotzdem maximal

$|J_n|^2 \cdot \nu(n)$. Da B nur eine Laufzeit von $g(n)$ besitzt für ein $g \in \mathsf{poly}$, ist

$$P\left(D\left(1^n, \mathcal{R}^l(x_1), \mathcal{R}^l(x_2)\right) = 1 : l = \lambda(n), (x_1, x_2) = B^{\mathcal{R}^l}(1^n)\right) \leq g(n) \cdot |J_n|^2 \cdot \nu(n)$$

nach Lemma 2.2.10 vernachlässigbar in n. Damit ist gezeigt, dass (2) gilt.

Analog dazu lassen sich auch (3) und (4) zeigen, da hier die Anfragen an die Hashfunktion bzw. das Random-Oracle wegen dem ersten Bit ebenfalls verschieden sind.

Schauen wir uns nun an, warum (1) gelten muss. Wir gehen dabei ähnlich vor wie bei (2). Wegen der Kollisionsresistenz können wir davon ausgehen, dass

$$h_i(1\sigma_3) \neq h_i(0m\sigma_2) \neq h_i(0\tilde{m}\tilde{\sigma}_2) \neq h_i(1\tilde{\sigma}_3)$$

ist. Angenommen (1) würde nicht gelten, dann könnten wir wieder einen Unterscheider D konstruieren, der der Unvorhersehbarkeit von $\{h_i\}$ widersprechen würde. Bei Eingabe von $(1^n, y_1, y_2, y_3, y_4)$ verhält sich D wie folgt:

- Wenn $y_1 = y_2, y_3 = y_4$ oder $y_1 = y_3$ ist, gib 0 zurück.
- Für alle $j, j' \in J_n$ überprüfe, ob $f_{n,j}(y_1 \oplus y_2) = f_{n,j'}(y_3 \oplus y_4)$ ist. Wenn dies für mindestens ein Paar (j, j') erfüllt ist, gib 1 zurück.
- Ansonsten gib 0 zurück.

Wegen (i) und (ii) ist D ein PT-Algorithmus. Wir überlegen uns zunächst, dass nach Satz 2.4.7 für alle $l \in \mathbb{N}$

$$\left(\mathcal{R}^l(00) \oplus \mathcal{R}^l(01), \mathcal{R}^l(10) \oplus \mathcal{R}^l(01), \mathcal{R}^l(01)\right) \sim \left(\mathcal{R}^l(00), \mathcal{R}^l(10), \mathcal{R}^l(01)\right)$$

gilt, da man die ersten beiden Komponenten zu einer Zufallsvariable zusammenfassen könnte und $f(x) = (x, x)$ für diesen Satz nehmen könnte. Damit folgt dann, dass

$$\begin{aligned}
&P\left(D\left(1^n, \mathcal{R}^l(00), \mathcal{R}^l(01), \mathcal{R}^l(10), \mathcal{R}^l(11)\right) = 1 : l = \lambda(n)\right) \\
&= P\Big(\exists j, j' \in J_n \colon f_{n,j}(\mathcal{R}^l(00) \oplus \mathcal{R}^l(01)) = f_{n,j'}(\mathcal{R}^l(10) \oplus \mathcal{R}^l(11)) \\
&\qquad\qquad \wedge \mathcal{R}^l(01) \neq \mathcal{R}^l(00) \neq \mathcal{R}^l(10) \neq \mathcal{R}^l(11) : l = \lambda(n)\Big) \\
&= P\Big(\exists j, j' \in J_n \colon f_{n,j}(\mathcal{R}^l(00)) = f_{n,j'}(\mathcal{R}^l(10)) \\
&\qquad\qquad \wedge \mathcal{R}^l(01) \neq \mathcal{R}^l(00) \neq \mathcal{R}^l(10) \neq \mathcal{R}^l(11) : l = \lambda(n)\Big) \\
&\leq \sum_{j,j' \in J_n} P\Big(f_{n,j}(\mathcal{R}^l(00)) = f_{n,j'}(\mathcal{R}^l(10)) : l = \lambda(n)\Big) \\
&\ldots \\
&\leq |J_n|^2 \cdot \nu(n)
\end{aligned}$$

und

$$
\begin{aligned}
&P\left(D\left(1^n, \mathcal{R}^l(00), \mathcal{R}^l(01), \mathcal{R}^l(10), \mathcal{R}^l(01)\right) = 1 : l = \lambda(n)\right) \\
&= P\Big(\exists j, j' \in J_n \colon f_{n,j}(\mathcal{R}^l(00) \oplus \mathcal{R}^l(01)) = f_{n,j'}(\mathcal{R}^l(10) \oplus \mathcal{R}^l(01)) \\
&\qquad\qquad \wedge \mathcal{R}^l(01) \neq \mathcal{R}^l(00) \neq \mathcal{R}^l(10) \neq \mathcal{R}^l(01) : l = \lambda(n)\Big) \\
&= P\Big(\exists j, j' \in J_n \colon f_{n,j}(\mathcal{R}^l(00)) = f_{n,j'}(\mathcal{R}^l(10)) \\
&\qquad\qquad \wedge \mathcal{R}^l(01) \neq \mathcal{R}^l(00) \neq \mathcal{R}^l(10) \neq \mathcal{R}^l(01) : l = \lambda(n)\Big) \\
&\leq \sum_{j,j' \in J_n} P\Big(f_{n,j}(\mathcal{R}^l(00)) = f_{n,j'}(\mathcal{R}^l(10)) : l = \lambda(n)\Big) \\
&\ldots \\
&\leq |J_n|^2 \cdot \nu(n)
\end{aligned}
$$

vernachlässigbar in n sind.

Analog zu oben kann ein PPT-Orakel-Algorithmus B nur zufällig $x_1, \ldots, x_4$ wählen und probieren, ob $D(1^n, \mathcal{R}^l(x_1), \ldots, \mathcal{R}^l(x_4)) = 1$ ergibt. Die Wahrscheinlichkeit bei einem weiteren Test erfolgreich zu sein ist maximal $|J_n|^2 \cdot \nu(n)$, auch unabhängig davon, ob er $x_2 = x_4$ wählt oder nicht. Da B nur eine Laufzeit von $g(n)$ besitzt für ein $g \in \mathsf{poly}$, ist

$$P\left(D\left(1^n, \mathcal{R}^l(x_1), \ldots, \mathcal{R}^l(x_4)\right) = 1 : l = \lambda(n), (x_1, \ldots, x_4) = B^{\mathcal{R}^l}(1^n)\right) \leq g(n) \cdot |J_n|^2 \cdot \nu(n)$$

vernachlässigbar in n. Damit ist gezeigt, dass auch (1) gilt. □

Beispiele für solche Funktionen $f_{n,j}$ sind die Projektion auf $\omega(\log_2(n))$ Bits, die XOR-Verknüpfung mit einem konstanten Wert, Vertauschen von Bits, die XOR-Verknüpfung der ersten $\frac{n}{2}$ Bits mit den zweiten, Addition, Multiplikation oder Exponentation modulo 2^n mit einem festen Wert und Verknüpfungen dieser Funktionen oder ähnlichen.

Vermutung 6.3.5. *Es sei* $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ *ein Protokoll vom Typ II und* $\{h_i\}$ *eine unvorhersehbare Familie von Hashfunktionen vom Grad 4, die auch zufallerhaltend ist. Ferner sei* $|\sigma_1| + |\sigma_2| \leq \iota(n)/2$*, wobei* n *der vom Protokoll und der Hashfamilie gemeinsam benutzte Sicherheitsparameter ist. Wenn das Protokoll im Random-Oracle-Modell sicher ist gegen existentielle Fälschung unter einem adaptiven Angriff mit gewählten Nachrichten, dann ist es auch im Standard-Modell sicher gegen existentielle Fälschung unter einem adaptiven Angriff mit gewählten Nachrichten.*

Begründung. Die Begründung teilt sich in drei Teile. Im ersten Teil überlegen wir uns, dass es für jeden PPT-Algorithmus A, der das Protokoll im Standard-Modell angreift einen PPT-Algorithmus B gibt, der bei einem Angriff im Random-Oracle-Modell die gleichen Informationen über sk von Sub_2 erhält wie A. Wir werden den Begriff Information

allerdings nicht im streng theoretischen Sinne benutzen, sondern eher umgangssprachlich. Im zweiten Teil werden wir dann diskutieren, welche Möglichkeiten A bleiben, das Protokoll zu brechen, bzw. wie er es nicht machen kann. Im dritten Teil wird unter einer etwas vereinfachenderen Annahme noch ein formaler Beweis für die Vermutung gegeben.

Teil 1. Wir haben folgendes Bild: Die einzige Information des Protokolls, die die Angreifer vorher nicht haben, ist der geheime Schlüssel sk. Dieser wird aber nur von Sub_2 verarbeitet.

Nach Satz 6.3.3 verhalten sich sowohl die Eingabewerte als auch der Ausgabewert von Sub_2 und σ_2 in beiden Modellen rechnerisch gleich. Für Sub_2 ergibt sich also kein merkbarer Unterschied zwischen den beiden Modellen, weshalb er auch in beiden Modellen die gleichen Informationen über sk preisgibt. Der Signaturteil σ_3 ist ohnehin nur eine unabhängige Zufallszahl und enthält daher für sich genommen auch keine Informationen über sk. Die rechnerische Ununterscheidbarkeit gilt nach Satz 6.3.3 selbst dann, wenn wir σ_1 und σ_2 zusammen betrachten oder gar Zugriff auf die internen Variablen y und z hätten, solange $r = \sigma_3$ nicht hinzugenommen wird.

Teil 2. Also haben die beiden Angreifer A und B in beiden Modellen nach der Befragung ihres Signatur-Orakels die gleichen Informationen, zumindest über sk, die sie von Sub_2 erhalten haben. Aber es gibt trotzdem einen kleinen Unterschied zwischen A und B, nämlich A besitzt Signaturen bzgl. der Hashfunktion und B bzgl. des Random-Oracles. Es könnte immerhin möglich sein, dass A zu den erhaltenen Hashwerten neue Hashwerte produzieren kann, zu denen er auch eine gültige Signatur erzeugen kann, auch ohne Informationen über sk. Dies sollte aber wegen der obigen Überlegung und der Unvorhersehbarkeit der Hashfunktion nicht (oder nur sehr schwer) möglich sein, was im Folgenden genauer erläutert wird.

Nach Satz 6.3.4 kann der von A erzeugte neue Wert $\tilde{y}$ keiner der bereits verwendeten Werte für y sein. Er kann noch nicht mal ähnlich sein (die Bilder unter $f_{n,j}$ und $f_{n,j'}$ sind gleich). Das Gleiche gilt auch für die meisten Hashwerte (Aussagen (2),(3) und (4) von Satz 6.3.4). Ob es nicht vielleicht doch für den Algorithmus A einen Weg gibt, erfolgreich zu sein, kann so leider nicht ausgeschlossen werden, aber zumindest dürfte es wohl nicht für beliebige Hashfamilien möglich sein, die die Voraussetzungen des Satzes erfüllen, sondern nur für speziell zu dem Protokoll passende. Falls es einen solchen erfolgreichen Algorithmus A geben sollte, müsste er vermutlich in seine Berechnungen mehrere Antworten des Signatur-Orakels einfließen lassen.

Teil 3. Sofern wir annehmen, dass die zusätzliche Information, die A und B beim Befragen ihrer Signatur-Orakel erhalten haben, nur von Sub_2 stammen kann und daher wie oben gezeigt unabhängig ist von dem Random-Oracle und der Hashfunktion bzw. deren Index, können wir die Sicherheit des Protokolls auch formal beweisen. Diese Annahme ist zwar etwas künstlich, da der Beweis auch ohne die Kollisionsresistenz der Hashfami-

lie funktionieren würde, aber zum einen wurde diese schon im zweiten Teil behandelt und zum anderen kann man so zumindest bestimmte Angriffsarten ausschließen. Wir werden zeigen, dass der Verifikations-Algorithmus, wegen der Unvorhersehbarkeit der Hashfunktion nicht zwischen den beiden Modellen unterscheiden kann, womit Angreifer A im Standard-Modell die gleichen Erfolgschancen hat wie Angreifer B im Random-Oracle-Modell, nämlich nach Voraussetzung nur eine vernachlässigbare.

Angenommen Angreifer A könnte mit einer nicht vernachlässigbaren Wahrscheinlichkeit eine Signatur fälschen, dann wäre also

$$P\left(\mathrm{Ver}^{h_i}(\mathsf{pk}, m, \sigma) = 1 : (\mathsf{pk}, \mathsf{sk}, i) = \mathrm{Gen}(1^n), (m, \sigma) = A^{\mathrm{Sig}^{h_i}(\mathsf{pk},\mathsf{sk})}(i, \mathsf{pk})\right)$$

nicht vernachlässigbar in n. Die zusätzliche Information, die A und B beim Befragen ihrer Signatur-Orakel erhalten haben, können wir mit info bezeichnen. Da sie unabhängig von dem Random-Oracle und der Hashfunktion bzw. deren Index ist, können wir diese ebenso von Gen oder in der Definition der Unvorhersehbarkeit von K erzeugen lassen. Wir können also die obige Wahrscheinlichkeit umschreiben zu

$$P\left(\mathrm{Ver}^{h_i}(\mathsf{pk}, m, \sigma) = 1 : (\mathsf{pk}, \mathsf{sk}, i, \mathsf{info}) = \mathrm{Gen}(1^n), (m, \sigma) = A(i, \mathsf{pk}, \mathsf{info})\right)$$

Wir setzen nun $k = (k_1, k_2) = (\mathsf{pk}, \mathsf{info})$ und interpretieren die Ausgabe (m, σ) von A als $(x_1, \ldots, x_4, z)$ mit $x_1 = 0\sigma_2 m, x_2 = 1\sigma_3, x_3 = x_4 = 0$ und $z = \sigma$. Weiterhin setzen wir noch $a_j = h_i(x_j)$ für $j = 1, \ldots, 4$. Es sei nun D der Algorithmus, der bei Eingabe von $(1^n, k, a_1, \ldots, a_4, z)$ den Algorithmus Sub' von Ver mit der Eingabe $(k_1, a_1 \oplus a_2, z_1, z_2)$ aufruft und dessen Ergebnis zurückgibt. Dieses wäre mit einer nicht vernachlässigbaren Wahrscheinlichkeit gleich 1.

Da die Hashfamilie $\{h_i\}$ unvorhersehbar ist, gibt es einen PPT-Algorithmus B, für den im Random-Oracle-Modell bei analoger Konstruktion die entsprechende Wahrscheinlichkeit ebenfalls nicht vernachlässigbar ist. Dies ist aber ein Widerspruch zur Sicherheit des Protokolls im Random-Oracle-Modell. □

Etwas weniger spekulativ, dafür aber auch weniger stark ist die gleiche Vermutung, nur dass anstatt Sicherheit gegen existentielle Fälschung Sicherheit gegen totales Brechen des Protokolls betrachtet wird.

Vermutung 6.3.6. *Es sei* $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ *ein Protokoll vom Typ II und* $\{h_i\}$ *eine unvorhersehbare Familie von Hashfunktionen vom Grad 4, die auch zufallerhaltend ist. Ferner sei* $|\sigma_1| + |\sigma_2| \leq \iota(n)/2$*, wobei* n *der vom Protokoll und der Hashfamilie gemeinsam benutzte Sicherheitsparameter ist. Wenn das Protokoll im Random-Oracle-Modell sicher ist gegen totales Brechen unter einem adaptiven Angriff mit gewählten Nachrichten, dann ist es auch im Standard-Modell sicher gegen totales Brechen unter einem adaptiven Angriff mit gewählten Nachrichten.*

Begründung. Siehe Teil 1 der Begründung zur vorherigen Vermutung. □

6.3.3 Vergleich von Protokollen vom Typ I und Typ II

Wir haben bisher Protokolle vom Typ I oder vom Typ II nur jeweils für sich betrachtet. In der Praxis hat man aber eher Protokolle vom Typ I mit einem Sicherheitsbeweis im Random-Oracle-Modell und hätte gerne die Sicherheit, die vermutlich die Protokolle vom Typ II bieten. Der folgende Satz löst dieses Problem teilweise in dem er zeigt, dass man Protokolle vom Typ I ohne Sicherheitsverlust in Protokolle vom Typ II umwandeln kann, zumindest im Random-Oracle-Modell, was aber auch ausreichend ist, da diese Protokolle dann vermutlich sicher im Standard-Modell sind.

Satz 6.3.7. *Es sei* $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ *ein Protokoll vom Typ I. Wenn das Protokoll im Random-Oracle-Modell sicher ist gegen existentielle Fälschung unter einem adaptiven Angriff mit gewählten Nachrichten, dann ist das daraus erzeugte Protokoll vom Typ II im Random-Oracle-Modell ebenfalls sicher gegen existentielle Fälschung unter einem adaptiven Angriff mit gewählten Nachrichten.*

Beweis. Es sei $P_{II} := (\mathrm{Gen}, \overline{\mathrm{Sig}}, \overline{\mathrm{Ver}}, \{h_i\})$ das aus $P_I := (\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ erzeugte Protokoll vom Typ II. Die Variablen in P_{II} werden, sofern sie sich von denen in P_I unterscheiden, in diesem Beweis mit einem Querstrich versehen, also $\overline{y}, \overline{\sigma}$ usw.

Angenommen es gibt einen PPT-Algorithmus B, dem für P_{II} im Random-Oracle-Modell eine existentielle Fälschung unter einem adaptiven Angriff mit gewählten Nachrichten gelingt, also

$$P\Big(\overline{\mathrm{Ver}}^{\overline{\mathcal{R}}^{\lambda(n)}}(\mathsf{pk}, \overline{m}, \overline{\sigma}) = 1 : (\mathsf{pk}, \mathsf{sk}, i) = \mathrm{Gen}(1^n), (\overline{m}, \overline{\sigma}) = B^{\overline{\mathrm{Sig}}^{\overline{\mathcal{R}}^{\lambda(n)}}(\mathsf{pk},\mathsf{sk}), \overline{\mathcal{R}}^{\lambda(n)}}(i, \mathsf{pk})\Big)$$

nicht vernachlässigbar in n ist. Unter dieser Annahme könnten wir aus B einen Angreifer A konstruieren, dem für P_I im Random-Oracle-Modell eine existentielle Fälschung unter einem adaptiven Angriff mit gewählten Nachrichten gelingt, also

$$P\Big(\mathrm{Ver}^{\mathcal{R}^{\lambda(n)}}(\mathsf{pk}, m, \sigma) = 1 : (\mathsf{pk}, \mathsf{sk}, i) = \mathrm{Gen}(1^n), (m, \sigma) = A^{\mathrm{Sig}^{\mathcal{R}^{\lambda(n)}}(\mathsf{pk},\mathsf{sk}), \mathcal{R}^{\lambda(n)}}(i, \mathsf{pk})\Big)$$

nicht vernachlässigbar in n ist.

Konstruktion: Der Angreifer A ersetzt die Orakelaufrufe von B durch eigene Berechnungen wie folgt: Wenn B sein Signatur-Orakel nach der Signatur von $\overline{m}$ fragt, im Weiteren kurz $\overline{\mathrm{Sig}}(\overline{m})$, dann denkt sich A zunächst zwei Zufallszahlen r, r' mit $|r| = |r'| = \lambda(n)$ aus. Anschließend fragt er sein Signatur-Orakel nach $m = \overline{m}r$ und merkt sich dessen Antwort, im Weiteren $\mathrm{Sig}(\overline{m}r)$, als (σ_1, σ_2) und gibt (σ_1, σ_2, r) an B zurück. Er merkt sich $(z', \overline{m}, r)$ in einer Liste L und $(z', \overline{m}, r')$ in einer weiteren Liste L', sofern noch kein Eintrag für das Paar $z', \overline{m}$ in L' vorhanden ist. Es sei hier noch kurz bemerkt, dass $\sigma_2 = z'$ ist.

Wenn B sein Random-Oracle nach $\overline{\mathcal{R}}(\widetilde{m})$ befragt, geht A wie folgt vor:

- Wenn $\widetilde{m} = 0m'$ ist und $|m'| \geq \zeta(n)$ ist (vgl. Abschnitt 5.1), dann interpretiere $\widetilde{m}$ als $0z'\overline{m}$ mit $|z'| = \zeta(n)$. Wenn es keinen Eintrag für $z', \overline{m}$ in der Liste L' gibt, denkt A sich zwei Zufallszahlen r, r' mit $|r| = |r'| = \lambda(n)$ aus und merkt sich $(z', \overline{m}, r)$ in der Liste L sowie $(z', \overline{m}, r')$ in der Liste L'.

- Wenn $\widetilde{m} = 0z'\overline{m}$ ist für einen Eintrag $(z', \overline{m}, r')$ aus der Liste L' (auch aus dem vorherigen Schritt), dann gibt A den Wert $\overline{\mathcal{R}}(\widetilde{m}) = r'$ zurück.

- Wenn $\widetilde{m} = 1r$ ist für einen Eintrag $(z', \overline{m}, r)$ aus der Liste L, dann gibt es dazu auch einen Eintrag $(z', \overline{m}, r')$ in der Liste L'. Die Antwort von A ist dann $\overline{\mathcal{R}}(\widetilde{m}) = \mathcal{R}(z'\overline{m}r) \oplus r'$.

- Ansonsten gibt A den Wert $\widetilde{\mathcal{R}}(\widetilde{m})$ zurück, wobei $\widetilde{\mathcal{R}}$ ein von A simuliertes Random-Oracle ist.

Wenn A von B eine (gefälschte) Signatur $(\sigma_1, \sigma_2, \sigma_3)$ für die Nachricht $\overline{m}$ erhält, gibt A die Signatur (σ_1, σ_2) für die Nachricht $\overline{m}\sigma_3$ zurück.

Die Liste L stellt im Prinzip die Signatur-Anfragen dar, bzw. die zusammengehörenden Anfragen an das Random-Oracle, die insbesondere zum Berechnen der Antworten auf Fragen vom Typ $1r$ an das Random-Oracle benötigt werden. In der Liste L' werden die Antworten des Random-Oracles auf Fragen vom Typ $0z'\overline{m}$ gespeichert, wie sie beim Signieren auftreten.

Analyse: Während der Simulation können verschiedene Anfrage-Situationen auftreten.

Wenn B sein Signatur-Orakel nach der Signatur $\overline{\mathrm{Sig}}(\overline{m}) = (\sigma_1, \sigma_2, \sigma_3)$ von $\overline{m}$ fragt und später die Signatur überprüft (oder auch nicht), sind die Werte von A so gewählt, dass B nur mit in n vernachlässigbarer Wahrscheinlichkeit einen Unterschied zu einer Interaktion mit echten Orakeln bemerkt, denn: Es ist $\overline{\mathrm{Sig}}(\overline{m}) = (\sigma_1, \sigma_2, \sigma_3) = (\mathrm{Sub}_2(\mathsf{sk}, z, y), z', r)$ mit $(z, z') = \mathrm{Sub}_1(\mathsf{pk})$ und $y = \mathcal{R}(z'\overline{m}r)$, wobei $(z', \overline{m}, r)$ in L steht und $(z', \overline{m}, r')$ in L' mit Zufallszahlen r, r'. Weiter ist $\overline{\mathcal{R}}(0z'\overline{m}) = r'$ und $\overline{\mathcal{R}}(1r) = \mathcal{R}(z'\overline{m}r) \oplus r'$, also

$$\overline{y} = \overline{\mathcal{R}}(0z'\overline{m}) \oplus \overline{\mathcal{R}}(1r) = r' \oplus \mathcal{R}(z'\overline{m}r) \oplus r' = \mathcal{R}(z'\overline{m}r) = y.$$

Die Variablen haben auch alle dieselbe Verteilung: Die Werte z, z' und r werden in beiden Fällen gleich berechnet bzw. gewählt. Der Wert von y ist in beiden Fällen uniform verteilt in $\{0,1\}^{\lambda(n)}$ und unabhängig von anderen Aufrufen, sofern sich mindestens eine der verwendeten Variablen $z', \overline{m}$ oder r beim zweiten Aufruf von denen beim ersten Aufruf unterscheidet (sonst ist y bei beiden Aufrufen in beiden Fällen gleich). Das einzige Problem könnte auftreten, wenn der Zufallswert r bei zwei verschiedenen Signaturen verwendet würde, denn dann könnte $\overline{\mathcal{R}}(1r)$ nicht mehr eindeutig berechnet werden. Dies tritt aber nach Satz 2.2.11 und 4.3.3 nur mit einer in n vernachlässigbaren Wahrscheinlichkeit auf.

Wenn B erst das Random-Oracle nach $\overline{\mathcal{R}}(0m')$ fragt und anschließend das Signatur-Orakel nach $\overline{\mathrm{Sig}}(\overline{m})$, dann gibt es zwei Möglichkeiten: Entweder ist $m' = z'\overline{m}$ oder nicht. In beiden Fällen führt die Konstruktion aber zu der gleichen Situation wie oben.

Wenn B erst nach $\overline{\mathcal{R}}(1r)$ fragt und anschließend in einer Signatur der Wert $\sigma_3 = r$ auftaucht, führt dies zu einer Inkonsistenz der Orakel-Werte. Dies tritt aber wie oben nach Satz 2.2.11 und 4.3.3 nur mit einer in n vernachlässigbaren Wahrscheinlichkeit ein.

Abgesehen von Fällen, die nur mit einer in n vernachlässigbaren Wahrscheinlichkeit auftreten, verhält sich die Simulation für B wie eine Interaktion mit echten Orakeln. Die Signatur (σ_1, σ_2) von A für die Nachricht $m = \overline{m}\sigma_3$ ist gültig, wenn $(\sigma_1, \sigma_2, \sigma_3)$ von B eine gültige Signatur für die Nachricht $\overline{m}$ ist, denn es ist (wie oben)

$$y = \mathcal{R}(\sigma_2 m) = \mathcal{R}(z'\overline{m}\sigma_3) = \mathcal{R}(z'\overline{m}r) = \cdots = \overline{y}$$

und damit $\mathrm{Sub}'(\mathsf{pk}, y, \sigma_1, \sigma_2) = \mathrm{Sub}'(\mathsf{pk}, \overline{y}, \sigma_1, \sigma_2)$. □

6.3.4 Protokolle vom Typ I unter starkem Angriff

Wir haben also gesehen, dass aus der Sicherheit von Protokollen vom Typ I im Random-Oracle-Modell die Sicherheit von Protokollen vom Typ II im Random-Oracle-Modell folgt und aus dieser wiederum vermutlich die Sicherheit im Standard-Modell. Zusammen ergäbe sich damit ein Sicherheitsbeweis für viele gängige Protokolle (vom Typ I), sofern man aus diesen Protokolle vom Typ II erzeugt.

Korollar 6.3.8. *Es sei* $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ *ein Protokoll vom Typ I und* $\{h_i\}$ *eine unvorhersehbare Familie von Hashfunktionen vom Grad 4, die auch zufallerhaltend ist. Ferner sei* $|\sigma_1| + |\sigma_2| \leq \iota(n)/2$*, wobei* n *der vom Protokoll und der Hashfamilie gemeinsam benutzte Sicherheitsparameter ist. Wenn die Vermutung 6.3.5 zutrifft und das Protokoll im Random-Oracle-Modell sicher ist gegen existentielle Fälschung unter einem adaptiven Angriff mit gewählten Nachrichten, dann ist das daraus erzeugte Protokoll vom Typ II im Standard-Modell ebenfalls sicher gegen existentielle Fälschung unter einem adaptiven Angriff mit gewählten Nachrichten.*

Beweis. Folgt aus Satz 6.3.7. □

Anstatt sich mit Vermutungen zu begnügen und sich dabei im Detail zu verlieren, könnte man das ganze auch aus der anderen Richtung betrachten und sagen, dass wir nur solche Hashfunktionen für Protokolle mit Sicherheitsbeweis im Random-Oracle-Modell zulassen, die einen sicheren Übergang vom Random-Oracle-Modell zum Standard-Modell erlauben, wie oben beschrieben.

Definition 6.3.9. *Eine Familie von Hashfunktionen* $\{h_i\}$ *ist genau dann* signatursicher*, wenn für alle Protokolle* $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver}, \{h_i\})$ *vom Typ I gilt: Wenn das Protokoll im Random-Oracle-Modell sicher ist gegen existentielle Fälschung unter einem adaptiven Angriff mit gewählten Nachrichten, dann ist das draus erzeugte Protokoll vom Typ II*

im Standard-Modell ebenfalls sicher gegen existentielle Fälschung unter einem adaptiven Angriff mit gewählten Nachrichten.

Ob es solche Hashfamilien gibt, ist nicht klar. Wenn allerdings die Vermutung 6.3.5 und die Annahme 4.2.4 zutreffen, dann sind die unvorhersehbaren Hashfamilien vom Grad 4, die auch zufallerhaltend sind, solche signatursicheren.

7 Ergebnisse, Diskussion und offene Fragen

Durch die Entwicklung von neuen Hashfamilien und die Transformation von Protokollen konnte die Problematik der Random-Oracle-Methodik deutlich abgeschwächt werden. Die meisten Arbeiten, von denen viele in Kapitel 6 vorgestellt wurden, haben sich bis jetzt nur mit der Unsicherheit der Random-Oracle-Methodik beschäftigt, wohingegen in dieser Arbeit eine Absicherung dieser gern und viel benutzten Methodik dargestellt wurde. Möglich wurde dies insbesondere durch eine exakte formale Behandlung des Themas, wozu auch die Einführung des neuen Begriffs der „rechnerischen Unabhängigkeit“ beigetragen hat. Dies ist ein deutlicher Fortschritt gegenüber der bloßen Heuristik in [BR93]. Die Absicherung gegen existentielle Fälschung konnte für einen sehr schwachen Angriff (ohne bekannte Signaturen) streng formal bewiesen werden. Für den Standardangriff (adaptiver Angriff mit gewählten Nachrichten) konnte eine sehr starke Begründung gefunden werden, dass ein entsprechend transformiertes Protokoll sicher ist gegen totales Brechen (Verlust des geheimen Schlüssels). Diese Begründung beruht auf der genauen Betrachtung der verwendeten Variablen und deren Verteilung, die sich durch formale Methoden so weit beschreiben lässt, dass ein Vergleich zwischen dem Standard-Modell und dem Random-Oracle-Modell möglich wurde, der zeigt, dass alle Variablen die zusammen mit dem geheimen Schlüssel in irgendeiner Weise verarbeitet werden in beiden Modellen rechnerisch ununterscheidbar sind. Auch für die Sicherheit gegen existentielle Fälschungen konnten bei gleichem Angriff gute Argumente gefunden werden, wobei einige Möglichkeiten für einen erfolgreichen Angriff ausgeschlossen werden konnten und somit nur noch die Möglichkeit für recht komplizierte Angriffe übrigbleibt. Auch wenn die Verteilung der einzelnen Variablen gut beschrieben werden kann, mangelt es doch an einem brauchbaren Informationsbegriff, mit dessen Hilfe das Wissen und damit die Möglichkeiten eines Angreifers genauer beschrieben werden können. Es hat sich zwar in der Informatik die Shannonsche Informationsentropie durchgesetzt, aber das ist nicht die einzige und auch nicht immer beste Möglichkeit, um Information zu beschreiben (siehe [Lyr02]). Es hat sich allerdings bisher kein geeigneter Informationsbegriff finden lassen, mit dem ein streng formaler Beweis (statt der oben genannten Begründungen) möglich gewesen wäre. Es ist auch nicht klar, ob es einen solchen überhaupt gibt bzw. ob die Beweisbarkeit nur von dem benutzten Informationsbegriff abhängt. Diese Fragen zu klären würde aber über den Rahmen dieser Arbeit weit hinausgehen. Aber auch ohne einen solchen Beweis lassen sich einige Punkte erkennen, die wesentlich für die Sicherheit verantwortlich sind:

- Der Index der Hashfunktion (sowie die Beschreibung der Hashfunktion) und die vom Angreifer wählbaren Nachrichten bzw. Protokolleingaben sollten nur für die Berechnung eines Hashwertes benutzt werden und an keiner anderen Stelle im Protokoll.

- Der im weiteren Protokoll benutzte (Hash-)Wert sollte randomisiert sein, also eine zufällige Komponente enthalten, wobei diese Komponente ebenfalls nicht weiter im Protokoll verarbeitet werden sollte.

- Der Index der Hashfunktion sollte deutlich länger sein, als die Eingaben von einem möglichen Angreifer, die nicht von der Hashfunktion weiterverarbeitet werden.

- Es sollte nur eine konstante Anzahl von Hashanfragen pro Protokollaufruf durchgeführt werden.

Der erste Punkt muss bei Verschlüsselungsprotokollen allerdings angepasst werden, da bei diesen die verschlüsselte Nachricht später wieder entschlüsselt werden muss, was bei einem Hashwert aber prinzipiell nicht möglich ist bzw. nicht möglich sein sollte (widerspricht der Einweg-Eigenschaft). Für solche Protokolle müsste also nochmal eine eigene Betrachtung stattfinden, nicht nur weil die Struktur dieser Protokolle anders ist als bei den hier betrachteten, sondern auch weil die Angriffe und (Sicherheits-)Ziele, die damit erreicht werden sollen, andere sind. Ebenfalls lohnenswert könnte es sein, sich Protokolle anzuschauen, dessen Sicherheitsbeweis auf anderen idealisierten Modellen beruht, wie z. B. dem generischen Gruppen-Modell oder dem Ideal-Cipher-Model. Vielleicht lässt sich dort mit ähnlichen Mitteln ebenfalls ein Zugewinn an Sicherheit erreichen.

Eine grundlegende Voraussetzung für die in dieser Arbeit gezeigten Absicherung ist die Existenz von unvorhersehbaren und zufallerhaltenden Hashfunktionen. Ob diese existieren (können), ist eine Frage, die bis jetzt unbeantwortet ist, auch wenn die geforderten Eigenschaften recht natürlich erscheinen. Eine ebenfalls noch ungeklärte Frage, die mit der vorherigen in starkem Zusammenhang steht, ist, ob sich diese Funktionen unter Standardannahmen wie z. B. der RSA-Annahme oder der Existenz von Einweg-Permutationen konstruieren lassen. Im Falle, dass sich diese Frage nicht beantworten lässt und die erste Frage zumindest keine negative Antwort besitzt, lässt es sich in diesem Rahmen nicht vermeiden, die Existenz solcher Funktionen als weitere Annahme zugrunde zu legen. Allerdings sollten hier, wie bei allen Annahmen, erst noch weitere Untersuchungen abgewartet werden, bevor dieser Annahme ein ähnliches Vertrauen geschenkt wird, wie anderen Standardannahmen.

Ein Random-Oracle lässt sich auf viele Weisen definieren. Insbesondere beim Wertebereich gibt es hier zahlreiche Möglichkeiten, angefangen bei Random-Oracles, die immer nur ein Bit ausgeben, über welche, die einen String fester Länge ausgeben, bis hin zu welchen, die Werte in einer Gruppe, wie z. B. $\mathbb{Z}_p^*$ ausgeben. Es konnte in dieser Arbeit gezeigt werden, dass sich diese Möglichkeiten im Wesentlichen nicht unterscheiden und manche sogar gleichwertig sind ohne irgendwelche Kompromisse einzugehen, seien sie

auch noch so klein. Bei diesen Kompromissen handelt es sich z. B. um das Akzeptieren einer erwarteten polynomialen Laufzeit statt einer polynomialen Laufzeit, die in jedem Fall eingehalten wird, oder das Akzeptieren eines Versagens bzw. Abbruchs eines Algorithmus mit vernachlässigbarer Wahrscheinlichkeit.

Auch wenn einige Fragen offen bleiben, konnten in dieser Arbeit neue Lösungsansätze im Bereich der Random-Oracle-Methodik dargestellt werden, die es erlauben, die Sicherheit der Random-Oracle-Methodik genauer als nur mit reiner Heuristik zu betrachten, und die vermutlich auch als Ansätze in ähnlichen Protokollen und Modellen genutzt werden können.

Literaturverzeichnis

[AKS04] Agrawal, Manindra ; Kayal, Neeraj ; Saxena, Nitin: PRIMES is in P. In: *Annals of Mathematics. Second Series* 160 (2004), Nr. 2, S. 781–793

[Bau91] Bauer, Heinz: *Wahrscheinlichkeitstheorie*. 4., völlig übearbeitete und neugestaltete Auflage. Walter de Gruyter, 1991. – ISBN 3–11–012191–3

[BBP04] Bellare, Mihir ; Boldyreva, Alexandra ; Palacio, Adriana: An Uninstantiable Random-Oracle-Model Scheme for a Hybrid-Encryption Problem. In: Cachin, Christian (Hrsg.) ; Camenisch, Jan (Hrsg.): *EUROCRYPT* Bd. 3027, Springer, 2004 (Lecture Notes in Computer Science). – ISBN 3–540–21935–8, S. 171–188

[BF06] Boldyreva, Alexandra ; Fischlin, Marc: On the Security of OAEP. In: Lai, Xuejia (Hrsg.) ; Chen, Kefei (Hrsg.): *ASIACRYPT* Bd. 4284, Springer, 2006 (Lecture Notes in Computer Science). – ISBN 3–540–49475–8, S. 210–225

[BFM88] Blum, Manuel ; Feldman, Paul ; Micali, Silvio: Non-interactive zero-knowledge and its applications. In: *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*. New York, NY, USA : ACM, 1988. – ISBN 0–89791–264–0, S. 103–112

[Bla05] Black, John: *The Ideal-Cipher Model, Revisited: An Uninstantiable Blockcipher-Based Hash Function*. Cryptology ePrint Archive, Report 2005/210, 2005. – `http://eprint.iacr.org/2005/210`

[Blu83] Blum, Manuel: Coin flipping by telephone a protocol for solving impossible problems. In: *SIGACT News* 15 (1983), Nr. 1, S. 23–27. `http://dx.doi.org/http://doi.acm.org/10.1145/1008908.1008911`. – DOI http://doi.acm.org/10.1145/1008908.1008911. – ISSN 0163–5700

[BR93] Bellare, Mihir ; Rogaway, Phillip: Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In: *ACM Conference on Computer and Communications Security*, 1993, S. 62–73

[BR94] Bellare, Mihir ; Rogaway, Phillip: Optimal Asymmetric Encryption. In: Santis, Alfredo D. (Hrsg.): *EUROCRYPT* Bd. 950, Springer, 1994 (Lecture Notes in Computer Science). – ISBN 3–540–60176–7, S. 92–111

[Bra93] Brands, Stefan: Untraceable Off-line Cash in Wallets with Observers (Extended Abstract). In: Stinson, Douglas R. (Hrsg.): *CRYPTO* Bd. 773, Springer, 1993 (Lecture Notes in Computer Science). – ISBN 3–540–57766–1, S. 302–318

[BSW99] Beutelspacher, Albrecht ; Schwenk, Jörg ; Wolfenstetter, Klaus-Dieter: *Moderne Verfahren der Kryptographie*. 3., verbesserte Auflage. Vieweg, 1999. – ISBN 3–528–26590–6

[Can97] Canetti, Ran: Towards Realizing Random Oracles: Hash Functions That Hide All Partial Information. In: Kaliski, Burton S. J. (Hrsg.): *CRYPTO* Bd. 1294, Springer, 1997 (Lecture Notes in Computer Science). – ISBN 3–540–63384–7, S. 455–469

[CF01] Canetti, Ran ; Fischlin, Marc: Universally Composable Commitments. In: [Kil01], S. 19–40

[CGH98] Canetti, Ran ; Goldreich, Oded ; Halevi, Shai: The Random Oracle Methodology, Revisited (Preliminary Version). In: *STOC*, 1998, S. 209–218

[CGH04a] Canetti, Ran ; Goldreich, Oded ; Halevi, Shai: On the Random-Oracle Methodology as Applied to Length-Restricted Signature Schemes. In: [Nao04], S. 40–57

[CGH04b] Canetti, Ran ; Goldreich, Oded ; Halevi, Shai: The random oracle methodology, revisited. In: *J. ACM* 51 (2004), Nr. 4, S. 557–594

[CMR98] Canetti, Ran ; Micciancio, Daniele ; Reingold, Omer: Perfectly one-way probabilistic hash functions (preliminary version). In: *30th STOC*, ACM, 1998, S. 131–140

[Coh00] Cohen, Henri: *A Course in Computational Algebraic Number Theory*. Fourth Printing. Springer, 2000. – ISBN 0–387–55640–0

[CS97] Camenisch, Jan ; Stadler, Markus: Proof Systems for General Statements about Discrete Logarithms / ETH Zürich. Institute of Theoretical Computer Science, 1997 (260). – Technical Report

[Dam00] Damgård, Ivan: Efficient Concurrent Zero-Knowledge in the Auxiliary String Model. In: Preneel, Bart (Hrsg.): *EUROCRYPT* Bd. 1807, Springer, 2000 (Lecture Notes in Computer Science). – ISBN 3–540–67517–5, S. 418–430

[Den02] Dent, Alexander W.: Adapting the Weaknesses of the Random Oracle Model to the Generic Group Model. In: Zheng, Yuliang (Hrsg.): *ASIACRYPT*

Bd. 2501, Springer, 2002 (Lecture Notes in Computer Science). – ISBN 3–540–00171–9, S. 100–109

[DH76] Diffie, Whitfield ; Hellman, Martin E.: New Directions in Cryptography. In: *IEEE Transactions on Information Theory* IT-22 (1976), Nr. 6, 644–654. `citeseer.ist.psu.edu/diffie76new.html`

[DOP05] Dodis, Yevgeniy ; Oliveira, Roberto ; Pietrzak, Krzysztof: On the Generic Insecurity of the Full Domain Hash. In: Shoup, Victor (Hrsg.): *CRYPTO* Bd. 3621, Springer, 2005 (Lecture Notes in Computer Science). – ISBN 3–540–28114–2, S. 449–466

[FF00] Fischlin, Marc ; Fischlin, Roger: Efficient Non-malleable Commitment Schemes. In: Bellare, Mihir (Hrsg.): *CRYPTO* Bd. 1880, Springer, 2000 (Lecture Notes in Computer Science). – ISBN 3–540–67907–3, S. 413–431

[FOPS01] Fujisaki, Eiichiro ; Okamoto, Tatsuaki ; Pointcheval, David ; Stern, Jacques: RSA-OAEP Is Secure under the RSA Assumption. In: [Kil01], S. 260–274

[FS87] Fiat, Amos ; Shamir, Adi: How to prove yourself: Practical solutions to identification and signature problems. In: *Advances in Cryptology — Crypto '86*. New York : Springer, 1987, S. 186–194

[GHR99] Gennaro, Rosario ; Halevi, Shai ; Rabin, Tal: Secure Hash-and-Sign Signatures Without the Random Oracle. In: Stern, Jacques (Hrsg.): *EUROCRYPT* Bd. 1592, Springer, 1999 (Lecture Notes in Computer Science). – ISBN 3–540–65889–0, S. 123–139

[GK03] Goldwasser, Shafi ; Kalai, Yael T.: On the (In)security of the Fiat-Shamir Paradigm. In: *FOCS*, IEEE Computer Society, 2003. – ISBN 0–7695–2040–5, S. 102–113

[Gol03] Goldreich, Oded: *Foundations of Cryptography*. Bd. I, Basic Tools. Reprinted with corrections. Cambridge University Press, 2003. – ISBN 0–521–79172–3

[Gol04] Goldreich, Oded: *Foundations of Cryptography*. Bd. II, Basic Applications. Cambridge University Press, 2004. – ISBN 0–521–83084–2

[Goo97] Goos, Gerhard: *Vorlesung über Informatik*. Bd. 3. Berechenbarkeit, formale Sprachen, Spezifikationen. Springer, 1997. – ISBN 3–540–60655–6

[Kil01] Kilian, Joe (Hrsg.): *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*. Bd. *2139*. Springer, 2001 (Lecture Notes in Computer Science). – ISBN 3–540–42456–3

[Kli06] Klima, Vlastimil: *Tunnels in Hash Functions: MD5 Collisions Within a Minute*. Cryptology ePrint Archive, Report 2006/105, 2006. – `http://eprint.iacr.org/`

[Kö95] Königsberger, Konrad: *Analysis 1*. 3. überarbeitete und erweiterte Auflage. Springer, 1995. – ISBN 3–540–58876–0

[LWW05] Lenstra, Arjen ; Wang, Xiaoyun ; Weger, Benne de: *Colliding X.509 Certificates*. Cryptology ePrint Archive, Report 2005/067, 2005. – `http://eprint.iacr.org/`

[Lyr02] Lyre, Holger: *Informationstheorie*. Wilhelm Fink Verlag, 2002. – ISBN 3–8252–2289–6

[MOV96] Menezes, Alfred ; Oorschot, Paul C. ; Vanstone, Scott A.: *Handbook of Applied Cryptography*. CRC Press, 1996. – ISBN 0–8493–8523–7

[MRH04] Maurer, Ueli M. ; Renner, Renato ; Holenstein, Clemens: Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In: [Nao04], S. 21–39

[Nao04] Naor, Moni (Hrsg.): *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*. Bd. *2951*. Springer, 2004 (Lecture Notes in Computer Science). – ISBN 3–540–21000–8

[Nie02] Nielsen, Jesper B.: Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-committing Encryption Case. In: Yung, Moti (Hrsg.): *CRYPTO* Bd. 2442, Springer, 2002 (Lecture Notes in Computer Science). – ISBN 3–540–44050–X, S. 111–126

[NK08] Nishioka, Mototsugu ; Komatsu, Naohisa: A Note on the Random Oracle Methodology. In: *IEICE Transactions* 91-A (2008), Nr. 2, S. 650–663

[Pas03] Pass, Rafael: On Deniability in the Common Reference String and Random Oracle Model. In: Boneh, Dan (Hrsg.): *CRYPTO* Bd. 2729, Springer, 2003 (Lecture Notes in Computer Science). – ISBN 3–540–40674–3, S. 316–337

[Rab83] Rabin, Michael O.: Transaction Protection by Beacons. In: *J. Comput. Syst. Sci.* 27 (1983), Nr. 2, S. 256–267

[Rei99] Reischuk, K. R.: *Komplexitätstheorie*. Bd. I: Grundlagen. 2., völlig neubearbeitete und erweiterte Auflage. B. G. Teubner, 1999. – ISBN 3–519–12275–8

[RTV04] Reingold, Omer ; Trevisan, Luca ; Vadhan, Salil P.: Notions of Reducibility between Cryptographic Primitives. In: [Nao04], S. 1–20

[Sch89] Schnorr, Claus-Peter: Efficient Identification and Signatures for Smart Cards. In: Brassard, Gilles (Hrsg.): *CRYPTO* Bd. 435, Springer, 1989 (Lecture Notes in Computer Science). – ISBN 3–540–97317–6, S. 239–252

[Sch98] Schürger, Klaus: *Wahrscheinlichkeitstheorie*. Oldenbourg, 1998. – ISBN 3–486–23659–8

[Sho97] Shoup, Victor: Lower Bounds for Discrete Logarithms and Related Problems. In: Fumy, Walter (Hrsg.): *EUROCRYPT* Bd. 1233, Springer, 1997 (Lecture Notes in Computer Science). – ISBN 3–540–62975–0, S. 256–266

[Sho01] Shoup, Victor: OAEP Reconsidered. In: [Kil01], S. 239–259

[Weg03] Wegener, Ingo: *Komplexitätstheorie*. Springer, 2003. – ISBN 3–540–00161–1

[WFLY04] Wang, Xiaoyun ; Feng, Dengguo ; Lai, Xuejia ; Yu, Hongbo: *Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*. Cryptology ePrint Archive, Report 2004/199, 2004. – `http://eprint.iacr.org/`

Index