

Pseudozufallszahlengeneratoren in der Kryptographie

Diplomarbeit

Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik

eingereicht von: Heiko Brandenburg
Betreuer: Prof. Dr. Köbler
Berlin, den 21.12.2006

Inhaltsverzeichnis

1	Einleitung	4
1.1	Einsatzgebiete von Pseudozufallszahlengeneratoren	5
1.2	Anforderungen an Pseudozufallszahlengeneratoren	6
1.3	Das El-Gamal-Signatur-Verfahren	7
1.4	Erzeugung echter Zufallszahlen	8
1.5	Schwierige Probleme für P	9
2	Grundlagen	11
2.1	Einwegfunktionen und Hard-core-Prädikate	11
2.2	Berechnungsmodelle	12
2.3	Definition von Einwegfunktionen	14
2.4	Hard-core-Prädikate	16
3	BMY-Typ Pseudozufallszahlengeneratoren	19
3.1	Definition	19
3.2	Konstruktionen für beliebig lange Pseudozufallszahlen	24
4	Pseudozufallszahlengeneratoren für beliebige Komplexitätsklassen	28
4.1	Grundlegende Definition	29
4.2	Nisans Haupttheorem	32
4.3	Vergleich: BMY-Typ und NW-Typ-Generator	34
5	Anwendung von NW-Typ Generatoren in der Kryptographie	35
5.1	Zero-knowledge-Beweise und weitere Definitionen	35
5.2	Hitting-set-Generatoren	40
5.3	Anwendung	42
6	Praktische Aspekte	45
6.1	Pseudozufallszahlengeneratoren mit öffentlich bekannten Eingaben	45
6.2	Situation in Deutschland	47
6.3	Beispiele für Generatoren	48
6.4	Statistische Tests	50

7 Anhang	51
7.1 Schlusswort	51

Kapitel 1

Einleitung

In dieser Arbeit wird es darum gehen "effiziente" Algorithmen zu untersuchen, welche es zum Ziel haben, aus einer "möglichst kleinen" Menge echter Zufallszahlen eine "möglichst große" Menge an Zahlen zu erzeugen, welche für einen "effizienten" Beobachter von einer gleich langen Sequenz echter Zufallszahlen "ununterscheidbar" sind. Solche Algorithmen werden wir Pseudozufallszahlengeneratoren nennen. Alle in Anführungszeichen gesetzten Begriffe werden wir dabei im Verlaufe der Arbeit klären. Wir wollen uns hier auf solche Generatoren beschränken, die in der Kryptographie genutzt werden können und zunächst dieses Einsatzgebiet von anderen möglichen Szenarien abgrenzen.

Danach betrachten wir einige der Anforderungen an solche Generatoren anhand des El-Gamal-Algorithmuses, um abschließend für das erste Kapitel Beispiele für Funktionen anzugeben, welche als praktische Grundlage für Pseudozufallszahlengeneratoren dienen können.

Im zweiten Kapitel werden wir einige grundlegende Definitionen und Funktionen betrachten welche wir im weiteren Verlauf benötigen. Zunächst klären wir dabei die berechnungstheoretischen Grundlagen und damit einhergehend die Begriffe "Effizienz" und "Ununterscheidbarkeit". Dies führt dann zu einer speziellen Klasse von Funktionen, den Einwegfunktionen. In diesem Zusammenhang werden wir auch einige verwandte Arten wie Einwegpermutationen und Hard-core-Prädikate betrachten.

Diese Funktionen nutzen wir dann im dritten Kapitel um erste Pseudozufallszahlengeneratoren zu konstruieren. Diese Art von Generatoren beruhen auf den Arbeiten von M. Blum, S. Micali und A. C. Yao (siehe u.a. [BM84]). Eine wichtige Rolle wird dabei die enge Verknüpfung zwischen der Existenz von Einwegfunktionen und der Existenz von Pseudozufallszahlengeneratoren einnehmen. Wir werden mit einem Generator beginnen, welcher aus einer n -Bit langen Eingabe eine $(n+1)$ -Bit lange Ausgabe erzeugt, und uns danach anschauen, wie wir Generatoren mit einer größeren Expansion erstellen können.

Das vierte Kapitel wird sich einem anderen Typ von Generatoren widmen, welcher auf der Arbeit von N. Nisan und A. Wigderson [NW94] aufbaut. Dazu beginnen wir mit einer Erweiterung der berechnungstheoretischen Grundlagen durch den Begriff Schaltkreis. Danach können wir einen Existenzbeweis für diese Generatoren formulieren. Abschließend nehmen wir einen kurzen Vergleich zwischen den Generatoren aus Kapitel 3, und denen aus Kapitel 4 vor. Dabei werden wir feststellen, dass letztere zwar schwächere Anforderungen haben, damit aber auch weniger Sicherheit bieten.

Wie wir solche Generatoren aus Kapitel 4 trotzdem in der Kryptographie verwenden können, wird das fünfte Kapitel klären. Dazu betrachten wir so genannte Beweissysteme. Wir werden diesen Begriff klären, und die Anwendung von Pseudozufallszahlengeneratoren in diesem Zusammenhang zeigen.

Nach den bis dahin eher theoretischen Überlegungen, werden wir im sechsten Kapitel einige praxisnähere Aspekte betrachten. Eine wichtige Rolle wird dabei der verbindlicher Anwendungshinweis AIS20 [Sch99] des Bundesamtes für Sicherheit in der Informationstechnologie (BSI) spielen.

1.1 Einsatzgebiete von Pseudozufallszahlengeneratoren

Neben der Kryptographie zählen zu den wichtigsten Einsatzgebieten die Bereiche Prozesssimulation, künstlichen Intelligenz, Komplexitätstheorie und die Derandomisierung probabilistischer Algorithmen. Bei letzterem wird versucht mit Hilfe der Generatoren, welche selbst deterministisch sind, probabilistische, also nicht deterministische Algorithmen, mit deterministischen Algorithmen zu simulieren.

In der Kryptographie werden Zufallszahlen sowohl benutzt um Schlüssel zu erzeugen, als auch bei einigen Verfahren, zum Beispiel beim DSA-Signaturverfahren,

direkt zur Verschlüsselung. Ein weiteres Einsatzgebiet sind Beweissysteme (vergleiche Kapitel 5). Damit ist klar, dass häufig große Mengen an Zufallszahlen benötigt werden. Die Erzeugung echter Zufallszahlen in diesem Umfang wäre mit erheblichen Problemen, sowohl was die Zeit für die Erzeugung, als auch was den technischen Aufwand betrifft, verbunden (vergleiche Abschnitt 1.4).

Einen Ausweg bietet da in einigen Bereichen möglicherweise die Verwendung von Quantenrechnern, mit denen echte Zufallszahlen mit Hilfe der Hadamard-Gatter erzeugt werden können. Bei der Anwendung eines Hadamard-Gatters auf ein Quantenbit welches den Wert 0 oder 1 repräsentiert, wird dabei ein Zustand erzeugt, in dem dieses Bit mit der Wahrscheinlichkeit $\frac{1}{2}$ eine 0, und mit gleicher Wahrscheinlichkeit eine 1 ist.

Leider sind sie aber auf absehbare Zeit noch nicht einsetzbar, so dass wir uns auf Pseudozufallszahlengeneratoren, die sich auf "herkömmlichen" Rechnern implementieren lassen, beschränken werden.

1.2 Anforderungen an Pseudozufallszahlengeneratoren

Die Anforderungen an Pseudozufallszahlengeneratoren sind vielfältig und richten sich in erster Linie nach dem Einsatzzweck. So ist es im Bereich der Simulation von physikalischen, chemischen oder anderen Prozessen ausreichend, dass der Generator, abhängig vom Startwert und den Parametern, eine möglichst lange Folge $\{x_i\}$ von Pseudozufallszahlen mit einer großen Periodenlänge erzeugt, d.h. $x_i = x_{i+n}$ sollte für ein möglichst großes n gelten, welche einer entsprechenden Verteilungsfunktion entspricht.

Um dies zu überprüfen, wurden eine Reihe von Testverfahren entwickelt, welche insbesondere verhindern sollen, dass ein allzu gleichmäßiges Muster in dieser Folge entsteht.

Eine wichtige Methode dazu ist der Spektraltest. Dabei werden je zwei aufeinander folgende generierte Zahlen als Paar $y_i := (x_i, x_{i-1})$ betrachtet. Anhand einer derart erzeugten Folge lassen sich bei zu geringer Güte des verwendeten Verfahrens sehr leicht Muster erkennen, welche bei einer echten Zufallsfolge nicht auftreten würden.

Das folgende Beispiel (Abbildung 1) eines Spektraltestes eines linearen Kongruenzgenerators mit $x_{i+1} := 3x_i \text{ mod } 7$ und $x_0 = 1$ illustriert dieses Verhalten deutlich.

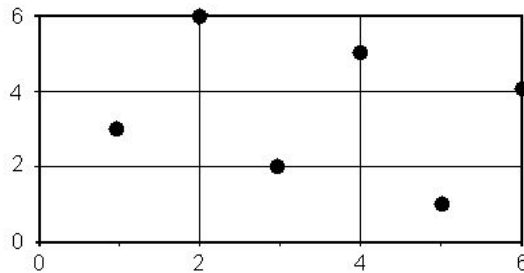


Abbildung 1

In Kapitel 6 werden wir uns sowohl diesen Typ Generator noch einmal etwas genauer ansehen, als auch weitere statistische Testverfahren beschreiben.

Es ist bei den in der Simulation eingesetzten Generatoren unerheblich, ob die Möglichkeit besteht, aus einer Reihe von Pseudozufallszahlen $x_{i-j}, \dots, x_{i-1}, x_i$ zu berechnen.

In der Kryptographie hingegen wäre dies ein ernsthaftes Problem, da viele Algorithmen von der Sicherheit der verwendeten Pseudozufallszahlen abhängig sind und also eine Berechnung der Zufallszahlen gegebenenfalls die Sicherheit des ganzen Verfahrens in Frage stellt. Um diese erhöhten Anforderungen in der Kryptographie zu gewährleisten, können Einwegfunktionen, welche wiederum auf schwierigen Problemen beruhen, benutzt werden.

Dabei ist die Existenz von Einwegfunktionen mit der Existenz von sicheren Pseudozufallszahlengeneratoren eng verbunden. Mit anderen Worten, wir werden sehen, dass genau dann solche Generatoren existieren, wenn es Einwegfunktionen gibt (siehe Kapitel 3). Deren Existenz ist aber derzeit nicht beweisbar, da dies sich auf die Frage $P = NP$ zurückführen lässt.

Versucht man Generatoren zu konstruieren, die ohne die Annahme der Existenz von Einwegfunktionen auskommen, so führt die zu grundsätzlich gesehen schwächeren Generatoren, jedoch können auch diese für die Kryptographie nutzbar gemacht werden.

Formulieren wir also die Anforderungen an Pseudozufallszahlengeneratoren zunächst informal: Einem potentiellen Angreifer darf es unter keine Umständen möglich sein, eine Folge von durch den Generator erzeugten Zahlen oder Bits von einer Folge von echten Zufallszahlen resp. Zufallsbits zu unterscheiden. Insbesondere darf er also auch nicht in der Lage sein aus einer Anzahl vorangegangener erzeugter Pseudozufallszahlen die nächste zu ermitteln, oder eine Periodizität bei den erzeugten Zahlen zu erkennen.

Dabei genügt es im Allgemeinen, dass es dem Angreifer nicht möglich sein darf diese Unterscheidung in Polynomialzeit zu treffen.

1.3 Das El-Gamal-Signatur-Verfahren

Um uns über die grundlegenden Anforderungen an sicheren Pseudozufallszahlengeneratoren klar zu werden betrachten wir zunächst das Beispiel des El-Gamal-Signatur-Verfahrens. Wir werden sehen, dass zum Beispiel eine zu kurze Periodenlänge der erzeugten Zahlen die Sicherheit des Verfahrens gefährdet.

Das El-Gamal-Signatur-Verfahren beruht auf dem Problem des diskreten Logarithmus (siehe Kapitel 1.5). Sei p prim (1024 Bit um die Härte des diskreten Logarithmus zu gewährleisten). Wir wählen nun ein $\alpha \in Z_p^*$ mit $ord(\alpha) = p - 1$ und berechnen für ein $a \in Z_p^* : \beta = \alpha^a \text{ mod } p$. Der private Signierschlüssel k' ist nun das Tupel (p, q, α, a) , der öffentliche Verifikationsschlüssel k ist (p, q, α, β) .

Für das Signieren wird des weiteren eine geheime zufällige Zahl $r \in Z_p^*$ benötigt. Für eine Nachricht $x \in Z_p^*$ ist nun die Signatur $sig(x, r, k') := (y, z)$; mit $y = a^r \text{ mod } p$ und $z = (x - ay)^{r-1} \text{ mod } p - 1$.

Bei der Verifikation wird dann überprüft, ob folgende Beziehung für ein Paar $(x, (y, z))$ gilt: $\alpha^y y^z \bmod p = \alpha^x$.

Nehmen wir nun an, dass ein potentieller Angreifer die Möglichkeit hat r zu berechnen, oder zumindest mit ausreichend hoher Wahrscheinlichkeit r zu erraten. Damit ist er dann auch in der Lage, zusammen mit einer bekannten Signatur $(x, (y, z))$, den privaten Schlüssel a zu berechnen.

Wegen $z = (x - ay)r^{-1} \bmod p - 1$ gilt: $a = (-rz + x)y^{-1} \bmod p - 1$
Somit ist der Angreifer in der Lage beliebige Signaturen zu fälschen.

Ähnliche Betrachtungen ergeben sich, falls der verwendete Generator eine zu kurze Periode hat, also $r_i = r_{i+n}$ für ein bekanntes und hinreichend kleines n ist. In diesem Falle könnte der Angreifer an zwei Paare von Nachrichten und Signaturen gelangen, welche mit derselben Zufallszahl r erstellt wurden. Seien $(x, (y, z))$ und $(x', (y', z'))$ zwei solche Signaturen.

Dann gilt $y = y'$.
 $\Rightarrow \beta^y y^z \equiv \alpha^x \bmod p$ sowie $\beta^{y'} y'^{z'} \equiv \alpha^{x'} \bmod p$
 $\Rightarrow r(z - z') \equiv x - x' \bmod p - 1$

Aus dieser Gleichung lassen sich wiederum eine Anzahl von Kandidaten für r ermitteln, welche sich durch probieren verifizieren lassen. Somit ist man also in der Lage, die Zufallszahl zu bestimmen und kann wie oben fortfahren.

1.4 Erzeugung echter Zufallszahlen

Um einen Startwert (engl. "seed") für die Generatoren zu erhalten, müssen echte Zufallszahlen erzeugt werden. Ein in [Sch99] vorgeschlagener Weg benutzt zuvor eingegebene Zeichenketten. Dabei wird eine Anzahl von Tastatureingaben aufgezeichnet und danach ein Hashverfahren wie zum Beispiel RIPEMD darauf angewendet. Dieses Verfahren führt jedoch zu einer Anzahl von Problemen. Zum einen lässt es sich nur schwer gewährleisten dass die eingegebenen Zeichen wirklich zufällig gewählt sind und zweitens muss sichergestellt werden, dass ein potentieller Angreifer nicht ebenfalls Zugriff auf die Tastatureingaben hat. Während sich der zweite Punkt durch eine Abschottung des Gerätes umgehen lässt, ist insbesondere das erste Problem schwerwiegend.

Ein weiteres beliebtes Mittel zur Erzeugung echter Zufallszahlen ist die Erfassung der Anzahl der Millisekunden seit dem Systemstart oder eines vorher festgelegten Zeitpunktes oder die Messung der Wegstrecke, die ein Mauszeiger seit in einer bestimmten Zeit zurückgelegt hat. Auf diesen Wert kann wiederum eine Hashfunktion angewendet werden um eventuell die Verteilung zu glätten.

Ein sehr sicherer, obschon aufwändiger, Weg, ist die Messung eines radioaktiven Zerfalls, oder die Auswertung einer Rauschquelle.

Abhängig von der verwendeten Quelle können dabei jedoch zwei grundsätzliche Probleme auftreten: zum einen wird häufig keine Gleichverteilung zwischen den Nullen und Einsen erreicht, zum anderen können Korrelationen zwischen den einzelnen Bits auftreten. Das erste Problem lässt sich durch folgende Methode beheben. Dazu werden je zwei aufeinander folgende Bits zusammengefasst. 01 wird zu einer 0, 10 zu einer 1 und die Paare 00 und 11 werden verworfen. Da auch bei einem nicht gleichverteilten Bitstring die Paare 01 und 10 gleich häufig auftreten, ist das Ergebnis also wieder gleichverteilt. Aber auch für das zweite Problem gibt es Lösungsmöglichkeiten (vergleiche u.a. [GB01]).

1.5 Schwierige Probleme für P

Mit "schwierig" für eine Komplexitätsklasse bezeichnen wir Probleme, für die es keinen Algorithmus in dieser Klasse gibt, beziehungsweise für die man keinen Algorithmus kennt, der dieses Problem lösen kann. Für P wären es damit alle Probleme, für die kein polynominell in der Eingabelänge zeitbeschränkter Algorithmus bekannt ist.

Dabei muss man zunächst klarstellen, das bei den folgenden Problemen es derzeit nicht beweisbar ist, ob sie wirklich schwierig sind, da sich dies, wie schon eingangs erwähnt, unmittelbar auf die Frage $P = NP$ zurückführen lässt. Mit anderen Worten: falls $P = NP$ gilt, so sind die im Folgenden angegebenen Probleme effizient, also mit einem polynominell zeitbeschränkter Algorithmus, lösbar. Dies hat zur Konsequenz, dass man sich auf Probleme zurückziehen muss, von denen man annimmt, dass sie schwierig sind.

Im folgenden seien für die Komplexitätsklasse P einige Beispiele genannt.

1. Das Faktorisierungsproblem $f(p, q) = pq$

Die Abbildung $(p, q) \rightarrow f(p, q)$ zu berechnen ist leicht, das heißt, sie ist in $O(1)$ möglich. Hingegen ist für die Abbildung $f(p, q) \rightarrow (p, q) = f^{-1}(p, q)$ kein polynominell beschränkter, wohl aber ein Quanten-Algorithmus bekannt¹. Zu den bekanntesten Anwendungsfällen dieses Problems zählt wahrscheinlich der RSA - Algorithmus.

Ein wichtiger Sicherheitsfaktor ist dabei die Länge der verwendeten Zahlen. So sind derzeit für den RSA-Algorithmus eine Schlüssellänge von wenigstens 2048 Bit zu empfehlen.

2. Der diskrete Logarithmus

Sei g ein Erzeuger der Gruppe Z_m , d.h. $Z_m = \{g^0, g^1, g^2, \dots, g^{m-1}\}$ und $e \in Z_m$. Die bijektive Abbildung $\varphi : Z_m \rightarrow Z_m$ mit $\varphi(e) = g^e$ lässt sich durch wiederholtes Quadrieren und Multiplizieren effizient berechnen, wohingegen für die Umkehrabbildung kein effizienter Algorithmus bekannt ist.

¹siehe auch [Sh97]

Ein weiteres Beispiele für schwierige Probleme ist das Teilmengensummenproblem (siehe Kapitel 6.1). In den folgenden Kapiteln werden wir voraussetzen, dass solche schwierigen Probleme existieren, und die oben genannten Probleme als Kandidaten für die in Kapitel 2 zu definierenden Einwegfunktionen identifizieren.

Kapitel 2

Grundlagen

In diesem Kapitel wollen wir einige theoretische Grundlagen legen, auf denen wir in den folgenden Kapiteln aufbauen werden.

2.1 Einwegfunktionen und Hard-core-Prädikate

Das Ziel in diesem Kapitel ist es Funktionen zu konstruieren, welche unumkehrbar sind; resp. bei denen die Umkehrfunktion nur unter sehr hohem Aufwand berechenbar ist.

Mit anderen Worten, eine Funktion $f : A \rightarrow B$ ist eine Einwegfunktion, wenn, bei gegebenen $y \in B$ ein Wert $x \in A$ mit $f(x) = y$ nicht mit einer Wahrscheinlichkeit wesentlich größer als $\frac{1}{|A|}$ und mit polynominalen Zeitaufwand für die meisten x berechnet werden kann.

Im folgenden sei o.B.d.A. $A = \{0, 1\}^n$ und $B = \{0, 1\}^{l(n)}$; wobei $l(n)$ ein Polynom ist. Des weiteren verstehen wir unter einer effizient berechenbaren Funktion eine Funktion, für die es einen polynominal in der Laufzeit beschränkten Algorithmus gibt, welcher f berechnet. Im Abschnitt 2.2 werden wir dies noch genauer betrachten. Wir sprechen nun von $f : A \rightarrow B$ als einer Einwegfunktion, falls f eine effizient berechenbare Funktion ist und für zufällige Werte Y aus B gilt, dass für jede andere effizient berechenbare Funktion $g : B \rightarrow A$ die Wahrscheinlichkeit dass $g(Y) \in f^{-1}(Y)$ ist, sehr klein ist. Wir werden dies in Kapitel 2.3 noch genauer formalisieren. Falls $n = l(n)$ ist, A und B also gleich mächtig sind, und überdies f bijektiv ist, so sprechen wir auch von einer Einwegpermutation.

Für den weiteren Konstruktionsprozess wichtige Funktionen sind die sogenannten Einweg- oder Hard-core-Prädikate. Prädikate sind Funktionen, die jedes Element einer Menge auf wahr oder falsch, repräsentiert durch 1 oder 0, abbilden. Ein mögliches Beispiel wäre bei einem Bitstring die Parität. Die Hard-core-Prädikate lassen sich nun für beliebige Einwegfunktionen definieren, wobei, mit

den obigen Bezeichnungen, die Prädikate von A auf $\{wahr, falsch\}$ abbilden. Die Schwierigkeit besteht nun darin, aus der Ausgabe von f auch das Ergebnis des Hard-core-Prädikats zu bestimmen, wobei vorausgesetzt sei, dass die Funktion und das Prädikat die gleiche Eingabe haben. Es lässt sich auch hier nicht, bzw. nur unter einem Aufwand der dem der Approximation von f nahe kommt, vorhersagen, ob das Ergebnis des Prädikats auf *wahr* oder auf *falsch* abgebildet wird.

In einigen Quellen findet sich auch der Begriff "hidden bits". Bei unseren Betrachtungen ist für Einwegfunktionen von Bedeutung, dass kein Urbild effizient berechnet werden kann. Jedoch schließt dies nicht aus, dass zumindest einige Bits davon berechenbar sind. Die Bits hingegen, welche für einen effizienten, hier also polynominell zeitbeschränkten, Algorithmus nicht ermittelbar sind, heißen "hidden bits". Man kann insofern Hard-core-Prädikate auch als eine Möglichkeit auffassen diese "hidden bits" einer Einwegfunktion zu extrahieren.

2.2 Berechnungsmodelle

Wenn wir im Folgenden von polynominell zeitbeschränkten Algorithmen sprechen, so meint dies, dass eine Turingmaschine existiert, welche das Ergebnis in einer polynominell von der Länge der Eingabe abhängenden Zeit berechnet.

Eine wichtige Rolle in den weiteren Betrachtungen werden probabilistische polynominell zeitbeschränkte Algorithmen, oder etwas kürzer probabilistische Polynomialzeitalgorithmen, einnehmen. Es sind zwei äquivalente Wege zur Beschreibung solcher Algorithmen zu finden. (vergleiche u.a. [Gol91])

Der erste Weg besteht darin, dass wir es dem Algorithmus ermöglichen zufällige Schritte zu machen. Diese zufälligen Schritte werden auch als "interne Münzwürfe" bezeichnet. Wir können dies wieder mit Hilfe einer Turingmaschine verdeutlichen. Bei einer Turingmaschine besteht die Übergangsfunktion aus Abbildungen der Form $(Zustand, Zeichen) \rightarrow (Zustand, Zeichen, Kopfbewegung)$. Den rechten Teil dieser Abbildung nennen wir Folgezustand. Für eine probabilistische Turingmaschine kommt ein zusätzlicher möglicher Folgezustand hinzu. Welchen der beiden möglichen Schritte die Turingmaschine macht, hängt vom Ausgang eines Münzwurfes ab. Beide Möglichkeiten haben dabei die gleiche Wahrscheinlichkeit. Damit ist die Ausgabe einer solchen Maschine bei Eingabe eines Wertes x aber nicht mehr eindeutig bestimmt. Wenn wir im Folgenden von der Wahrscheinlichkeit sprechen, dass eine Maschine M bei Eingabe von einem x einen Wert y ausgibt, also $Pr(M(x) = y)$, so wird diese Wahrscheinlichkeit über alle möglichen Ausgänge der internen Münzwürfe genommen.

Der zweite Weg beruht darauf, dass die internen Münzwürfe durch eine zusätzliche Eingabe eines zufälligen gleichverteilten Strings ersetzt werden. Statt also eine Münze zu werfen, wird ein Bit der Zusatzeingabe gelesen. Der Zufall wird dadurch aus dem eigentlichen Algorithmus nach außen verlagert.

Für beide Konzepte gilt dabei, dass wir fordern, dass die Laufzeit des Algorithmus polynominell von der Länge der Eingabe abhängt. Dadurch ist auch die Anzahl der Münzwürfe, beziehungsweise die Länge der zusätzlichen Eingabe polynominell beschränkt. Des Weiteren muss die Wahrscheinlichkeit, dass der Algorithmus zu einer Funktion f den korrekten Wert berechnet nicht vernachlässigbar sein, es muss also ein Polynom $p(\cdot)$ geben, so dass gilt: $Pr(M(x) = f(x)) \geq \frac{1}{p(|x|)}$. Wir nennen $\frac{1}{p(|x|)}$ dann Vorteil. Wir betrachten solche Algorithmen im Folgenden als effizient.

Die folgenden Bezeichnungen dienen lediglich zur Abkürzung der bereits besprochenen Begriffe.

Bezeichnung 2.1: [P-Funktionen]

Sei $l(\cdot)$ ein Polynom. Eine Familie von Funktion $f := \{f_n : \{0, 1\}^n \rightarrow \{0, 1\}^{l(n)}\}$ heiße Familie von P-Funktion, falls es einen polynominell in der Laufzeit beschränkten Algorithmus A gibt, welcher $A(x) = f_n(x)$ für alle n und alle $x \in \{0, 1\}^n$ berechnet.

Entsprechend heißt f von einem probabilistische P-Funktion, falls A ein probabilistischer Polynomialzeitalgorithmus ist. Dies führt zu einer neuen Komplexitätsklasse welche wir nun definieren können.

Definition 2.2: BPP ist die Klasse von Sprachen welche von einem probabilistischen Polynomialzeitalgorithmus akzeptiert werden. Eine Sprache L wird von einem probabilistischen Polynomialzeitalgorithmus A akzeptiert falls gilt:

$$\text{für jedes } x \in L \text{ gilt: } Pr(A(x) = 1) \geq \frac{2}{3}$$

$$\text{für jedes } x \notin L \text{ gilt: } Pr(A(x) = 1) \leq \frac{1}{3}$$

Die Grenze $\frac{2}{3}$ ist dabei willkürlich gewählt. Jede andere Konstante größer als $\frac{1}{2}$ führt zur gleichen Klasse. Wir können also auch fordern, dass ein Polynom $p(\cdot)$ existieren muss, so das die oben angegebenen Wahrscheinlichkeiten größer gleich $\frac{1}{2} + \frac{1}{p(|x|)}$ resp. kleiner gleich $\frac{1}{2} - \frac{1}{p(|x|)}$ sind.

An dieser Stelle sei auf den Unterschied zwischen nichtdeterministischen und probabilistischen Algorithmen hingewiesen. Auch nichtdeterministische Algorithmen können interne Münzwürfe nutzen. Wir sagen ein nichtdeterministischer Algorithmus A berechnet einen Wert $f(x)$, falls es einen Berechnungspfad, also eine Abfolge von Zustandsübergängen, gibt, für den gilt: $A(x) = f(x)$. Dieser Berechnungspfad wird auch Zeuge für x genannt. Mit anderen Worten, ein nichtdeterministischer Algorithmus kann für eine Eingabe x den richtigen Wert einfach raten, die Wahrscheinlichkeit das A $f(x)$ berechnet muss also lediglich größer Null sein, ein probabilistischer Algorithmus hingegen muss, wie bereits besprochen, einen Vorteil von mindestens $\frac{1}{p(|x|)}$, wobei $p(\cdot)$ ein Polynom ist, haben.

Wir haben also einen Rahmen für den Begriff Effizienz definiert. Damit können wir nun die Frage stellen, wann wir zwei Objekte als equivalent bezeichnen, um diese Beziehung später zum Vergleich von Pseudo- und echten Zufallszahlen zu nutzen. Als Objekte bezeichnen wir in diesem Zusammenhang sowohl Familien von Zufallsvariablen, als auch Familien von Funktionen.

Zwei Objekte gelten dabei als ununterscheidbar, falls es keinen effizienten Algorithmus gibt, welcher unendlich viele Werte des einen Objekts akzeptiert und gleichzeitig unendlich viele Werte des anderen zurück weist.

Definition 2.3: [P-Ununterscheidbarkeit]

Zwei Familien von Zufallsvariablen $X := \{X_n\}_{n \in \mathbb{N}}$ und $Y := \{Y_n\}_{n \in \mathbb{N}}$ heißen P-ununterscheidbar, falls für alle probabilistischen Polynomialzeitalgorithmen A , alle Polynome $s(\cdot)$ und alle hinreichend großen n gilt:

$$|Pr(A(X_n) = 1) - Pr(A(Y_n) = 1)| < \frac{1}{s(n)}$$

Das heißt insbesondere, dass diese Ungleichung für endlich viele n nicht erfüllt sein kann, sie aber für unendlich viele n erfüllt sein muss. Wir können also analog fordern, dass sie für fast alle n gelten muss.

Ein weiteres Konzept welches später von Bedeutung sein wird, sind nichtuniforme Algorithmen. Im Gegensatz zum uniformen Modell bei dem ein einziger Algorithmus Eingaben von verschiedenen Längen erhält (siehe Bezeichnung 2.1), besteht ein nichtuniformer Algorithmus A aus einer Menge von Algorithmen $\{A_i\}_{i \in \mathbb{N}}$, wobei jeder einzelne nur Eingaben einer bestimmten Länge bearbeitet. Die jeweiligen Algorithmen können dabei völlig unterschiedlich sein.

Vergleichen wir dies mit der Bezeichnung 2.1, so gibt es im uniformen Modell also einen Algorithmus welcher die Funktionswerte aller f_n berechnen kann, beim nichtuniformen Modell wird dagegen für jedes n ein jeweils anderer Algorithmus A_n verwendet.

2.3 Definition von Einwegfunktionen

In diesem Abschnitt wollen wir einige Definitionen angeben, die im weiteren Verlauf die Grundlage für Pseudozufallszahlengeneratoren bilden werden. Dazu zunächst eine formale Definition von gleichverteilten Zufallsvariablen.

Definition 2.4: [Gleichverteilung; $U \sim \{0, 1\}_n$]

Eine Zufallsvariable X heißt auf $\{0, 1\}$ gleichverteilt, falls gilt:

$Pr(X = 0) = Pr(X = 1) = \frac{1}{2}$. Wir schreiben dann kurz $X \sim U_1$.

Ist X ein n Bit langer Vektor mit $X = (X^{(1)}, X^{(2)}, \dots, X^{(n)})$, $X^{(i)} \sim U_1$

$\forall 0 < i \leq n$, sowie alle $X^{(i)}$ stochastisch unabhängig, so schreiben wir $X \sim U_n$.

Nun können wir auch den Begriff Einwegfunktion definieren. Diese Funktionen werden in Kapitel 3 die Basis der dort vorgestellten Generatoren sein, und wir werden sehen, dass die Existenz solcher Funktionen mit der Existenz von Pseudozufallszahlengeneratoren eng verknüpft ist.

Definition 2.5: [Einwegfunktion]

Eine Familie von P-Funktionen $f := \{f_n : \{0, 1\}^n \rightarrow \{0, 1\}^{l(n)}\}_{n \in \mathbb{N}}$ heißt Einwegfunktion, falls für jeden probabilistischen Polynomialzeitalgorithmus $A : \{0, 1\}^{l(n)} \rightarrow \{0, 1\}^n$, alle Polynome $s(\cdot)$ und eine Zufallsvariablen X mit $X \sim U_n$ gilt:

$$Pr(f(A(f(X))) = f(X)) \leq \frac{1}{s(n)}$$

für fast alle n .

Nicht immer ist es in der Praxis notwendig, dies tatsächlich für alle Polynome $s(\cdot)$ zu fordern. Oft genügt es, dass die Ungleichung für ein bestimmtes, für den entsprechenden Zweck hinreichend großes, Polynom erfüllt ist.

Definition 2.6: [$s(n)$ -sichere Einwegfunktion]

Sei $s(\cdot)$ ein Polynom. Dann heißt eine Familie von P-Funktion

$f := \{f_n : \{0, 1\}^n \rightarrow \{0, 1\}^{l(n)}\}_{n \in \mathbb{N}}$ $s(n)$ -sichere Einwegfunktion, falls für jeden probabilistische Polynomialzeitalgorithmus $A : \{0, 1\}^{l(n)} \rightarrow \{0, 1\}^n$ und eine Zufallsvariable X mit $X \sim U_n$ gilt:

$$Pr(f(A(f(X))) = f(X)) \leq \frac{1}{s(n)}$$

für fast alle n .

Das Polynom $s(\cdot)$ wird dann Sicherheitsfunktion und n Sicherheitsparameter genannt.

Solche $s(n)$ -sichere Einwegfunktionen werden auch als schwache Einwegfunktionen bezeichnet. Jedoch lässt sich zeigen (vergleiche u.a. [Lub96]) dass die Existenz von schwachen Einwegfunktionen die Existenz von Einwegfunktion wie in Definition 2.5 impliziert.

Eine Spezialform der Einwegfunktionen stellen die Einwegpermutationen dar. Sie besitzen dabei die gleichen Sicherheitseigenschaften wie Einwegfunktionen.

Definition 2.7: [Einwegpermutation]

Eine Familie von Einwegfunktionen $f := \{f_n : \{0, 1\}^n \rightarrow \{0, 1\}^n\}_{n \in \mathbb{N}}$ heißt Familie von Einwegpermutationen, falls f bijektiv ist.

Diese Einwegpermutationen werden dann im drittem Kapitel die Grundlage für unsere erste Konstruktion von Pseudozufallszahlengeneratoren bilden.

2.4 Hard-core-Prädikate

Wir können nun auch die für die Konstruktion von Pseudozufallszahlengeneratoren für uns zunächst wichtigen Hard-core-Prädikate definieren. Wir werden später noch weitere Wege ohne diese Prädikate sehen, um solche Generatoren zu konstruieren.

Definition 2.8: Ein Prädikat $Q : \{0, 1\}^* \rightarrow \{0, 1\}$ heißt Hard-core-Prädikat für eine Funktion f , falls für jeden probabilistischen Polynomalzeitalgorithmus A , jedes Polynom $s(\cdot)$ und fast alle n gilt:

$$\Pr(A(f(U_n)) = Q(U_n)) < \frac{1}{2} + \frac{1}{s(n)}$$

Für den in Kapitel 3 folgenden Beweis von Satz 3.1 wird die Existenz solcher Hard-core-Prädikate vorausgesetzt. Dass solche Prädikate existieren, unter der Voraussetzung der Existenz von Einwegfunktionen, zeigt folgender Satz.

Satz 2.1: [GL89] Sei f eine Einwegfunktion. Wir definieren nun $g(x, r) := (f(x), r)$, wobei $|x| = |r|$. Es sei für die beiden binären Vektoren x und r das Prädikat $Q(x, r) := (x^T r) \bmod 2$ definiert als das innere Produkt modulo 2. Dann ist Q ein Hard-core-Prädikat für g .

Beweis:

(Vergleiche dazu [Gol91]; der ursprüngliche Beweis ist in [GL89] zu finden.) Nehmen wir an, es gibt eine probabilistische P-Funktion G , welche bei Eingabe von $g(x, r) = (f(x), r)$ das innere Produkt von x und r mit einer Wahrscheinlichkeit wesentlich größer als $1/2$ berechnen kann. Wir können dann daraus eine Funktion konstruieren, welche bei Eingabe von $f(x)$ ein Urbild x mit Wahrscheinlichkeit $\frac{1}{n^{O(1)}}$ berechnet. Dies wäre dann ein Widerspruch zur Einweg-eigenschaft von f .

Sei dazu $\varepsilon_G(n)$ der durchschnittliche Vorteil, dass G bei Eingabe von $f(x)$ und r , mit $x \in_R \{0, 1\}^n$ und $r \in_R \{0, 1\}^n$ $Q(x, r)$ vorhersagen kann.

Also

$$\varepsilon_G(n) := \Pr(G(f(X_n), R_n) = Q(X_n, R_n)) - \frac{1}{2}$$

mit $X_n, R_n \sim U_n$. Wäre Q kein Hard-core-Prädikat, müsste damit eine unendliche Menge N sowie ein Polynom $p(\cdot)$ existieren, so dass für alle $n \in N$ gilt: $\varepsilon_G(n) > \frac{1}{p(n)}$.

Sei von nun an G eine solche Funktion und $\varepsilon := \varepsilon_G$. Wir beschränken uns des weiteren auf n 's aus N .

Zunächst müssen wir zeigen, dass eine Menge $S_n \subseteq \{0, 1\}^n$ existiert, mit mindestens $\frac{\varepsilon(n)}{2} \cdot 2^n$ Elementen, so dass für alle $x \in S_n$ gilt:

$$s(x) := Pr(G(f(x), R_n) = Q(x, R_n)) \geq \frac{1}{2} + \frac{\varepsilon(n)}{2}$$

Die Wahrscheinlichkeit wird dabei über alle möglichen R_n genommen. Dies folgt aus der Markov-Ungleichung. Daher können wir im Folgenden uns auf Werte x aus S_n konzentrieren.

Wir werden nun einen Invertierungsalgorithmus A für f konstruieren. Zur Vereinfachung nehmen wir an, dass f eine längenerhaltende Einwegfunktion ist, also die Länge der Bilder von f denen der jeweiligen Urbilder entspricht.

(i) Bei Eingabe y aus dem Bildraum von f , setzt A nun $n := |y|$ und $l := \lceil \log_2(2n \cdot p(n)^2 + 1) \rceil$, wobei $p(\cdot)$ dasjenige Polynom ist, für welches gilt: $\varepsilon(n) > \frac{1}{p(n)}$ für unendlich viele $n \in N$.

(ii) A wählt gleichverteilt und unabhängig $s^1, \dots, s^l \in \{0, 1\}^n$, sowie $\sigma^1, \dots, \sigma^l \in \{0, 1\}$.

(iii) Dann wird für jede nicht leere Teilmenge $J \subseteq \{1, 2, \dots, l\}$ ein String $r^J \leftarrow \bigoplus_{j \in J} s^j$ und ein Bit $p^J \leftarrow \bigoplus_{j \in J} \sigma^j$ berechnet. Für jedes $i \in \{1, \dots, n\}$ und jedes nicht leere $J \subseteq \{1, 2, \dots, l\}$ berechnet A nun $z_i^J \leftarrow p^J \oplus G(y, r^J \oplus e^i)$, wobei e^i der i 'te Einheitsvektor ist.

(iv) A setzt $z_i = 1$ falls die Mehrheit der z_i^J gleich 1 ist und 0 sonst, und gibt $z = z_1 \dots z_n$ aus.

Die Überlegungen welche zu diesem Algorithmus führen, sind in [Gol91] zu finden.

Wir wollen nun die Erfolgswahrscheinlichkeit von A bei Eingabe von $f(x)$, für $x \in S_n$ mit $n \in N$ analysieren. Zunächst werden wir zeigen, dass, falls die σ^j korrekt sind, mit konstanter Wahrscheinlichkeit gilt: $z_i = x_i$ für alle $i \in \{1, \dots, n\}$. Dazu bestimmen wir eine untere Schranke für die Wahrscheinlichkeit, dass die Mehrheit der z_i^J gleich x_i sind.

Behauptung: Für jedes $x \in S_n$ und alle i ; $1 \leq i \leq n$ gilt:

$$Pr(|\{J : Q(x, r^J) \oplus G(f(x), r^J \oplus e^i) = x_i\}| > \frac{1}{2} \cdot (2^l - 1)) > 1 - \frac{1}{2n} \quad (2.1)$$

mit $r^J := \bigoplus_{j \in J} s^j$ und $s^j \in \{0, 1\}^n$ unabhängig und gleichverteilt gewählt.

Für jedes J definieren wir eine Zufallsvariable $\chi^J \in \{0, 1\}$ welche genau dann 1 ist, wenn

$$Q(x, r^J) \oplus G(f(x), r^J \oplus e^i) = x_i$$

gilt.

Da alle s^j unabhängig und gleichverteilt gewählt wurden, folgt auch, dass jedes r^J gleichverteilt über $\{0, 1\}^n$ ist. Damit sind auch alle χ^J mit einer Wahrscheinlichkeit $s(x) \geq \frac{1}{2}$, für die $x \in S_n$ also mindestens $\frac{1}{2} + \frac{1}{2p(n)}$.

Wir zeigen nun, dass die χ^J paarweise unabhängig sind, indem wir die paarweise Unabhängigkeit der r^J zeigen. Sei o.B.d.A. $J \neq K$, sowie $j \in J$ und $k \in K - J$. Dann gilt für alle $\alpha, \beta \in \{0, 1\}^n$:

$$\begin{aligned} Pr(r^K = \beta | r^J = \alpha) &= Pr(s^k = \beta | s^j = \alpha) \\ &= Pr(s^k = \beta) \\ &= Pr(r^K = \beta) \end{aligned}$$

Mit anderen Worten, die r^J sind paarweise unabhängig. Sei $m := 2^l - 1$. Dann folgt mit Hilfe der Tschebyschow-Ungleichung:

$$\begin{aligned} Pr\left(\sum_J \chi^J \leq \frac{1}{2} \cdot m\right) &\leq Pr\left(\left|\sum_J \chi^J - \left(\frac{1}{2} + \frac{1}{2p(n)}\right) \cdot m\right| \geq \frac{1}{2p(n)} \cdot m\right) \\ &< \frac{Var(\chi^{\{1\}})}{\left(\frac{1}{2p(n)}\right)^2 \cdot m^2} \\ &< \frac{Var(\chi^{\{1\}})}{\left(\frac{1}{2p(n)}\right)^2 \cdot (2n \cdot p(n)^2)} \\ &< \frac{\frac{1}{4}}{\left(\frac{1}{2p(n)}\right)^2 \cdot (2n \cdot p(n)^2)} \\ &= \frac{1}{2n} \end{aligned}$$

Daraus folgt die Behauptung (2.1).

Ist nun $\sigma^j = Q(x, s^j)$ für alle j , so gilt $p^J = Q(x, r^J)$ für alle nicht leeren Mengen J . In diesem Falle würde die Ausgabe z von A mit einer Wahrscheinlichkeit mindestens $1/2$ mit x übereinstimmen. Dies geschieht nun aber, unabhängig von Behauptung (2.1), mit einer Wahrscheinlichkeit von $2^{-l} = \frac{1}{2n \cdot p(n)^2}$. Falls also $x \in S_n$ ist, so invertiert A die Funktion f bei Eingabe von $f(x)$ mit einer Wahrscheinlichkeit von mindestens $\frac{1}{2p(|x|)}$. Da aber $|S_n| > \frac{1}{2p(n)} \cdot 2^n$ ist, folgt für jedes $n \in N$, dass A f bei Eingabe $f(U_n)$ mit einer Wahrscheinlichkeit von mindestens $\frac{1}{8p(n)^2}$ invertiert.

Da A von G lediglich $2n \cdot p(n)^2$ mal benutzt, und zusätzlich dazu nur polynominell viele Schritte macht, ist die Laufzeit von A also auch durch ein Polynom beschränkt. Daraus ergibt sich ein Widerspruch zur Einwegigkeit von f .

◇

Kapitel 3

BMY-Typ Pseudozufallszahlengeneratoren

Wir wollen zunächst den von Blum, Micali und Yao [BM84, Yao82] entwickelten Typ von Generatoren betrachten, welcher unter Hinblick auf deren kryptographische Verwendung konstruiert wurde.

Nachdem wir in Kapitel 2 die grundlegenden Mechanismen zur Konstruktion von Pseudozufallszahlengeneratoren definiert haben, können nun die eigentlichen Generatoren konstruiert werden. Zunächst werden wir den Begriff Pseudozufallszahlengenerator definieren und uns einige wichtige Aussagen dazu anschauen. Danach betrachten wir einen Generator der aus einem Startwert der Länge n eine Pseudozufallszahlen der Länge $n + 1$ generiert. Diesen kann man dann nutzen, um Pseudozufallszahlen beliebiger Länge zu erhalten.

3.1 Definition

Das Ziel der späteren Konstruktionen ist es, eine Familie von Funktionen zu definieren, welche Zahlen erzeugen, die von einer gleichverteilten Zufallsvariablen nicht zu unterscheiden sind. Dazu verwenden wir folgende Definition:

Definition 3.1:

Eine Familie von P-Funktionen $f := \{f_n : \{0, 1\}^n \rightarrow \{0, 1\}^{l(n)}\}_{n \in \mathbb{N}}$ heißt BMY-Typ-Pseudozufallszahlengenerator, falls $l(n) > n$ ist und für $Y \sim \{U_{l(n)}\}_{n \in \mathbb{N}}$ sowie für fast alle $x \in \{0, 1\}^*$ gilt: $f(x)$ ist P-ununterscheidbar von Y .

Das Polynom $l(n)$ heißt dann Expansionsfaktor.

Bis zum Ende dieses Kapitels sind alle Pseudozufallszahlengeneratoren vom BMY-Typ.

Die erste Möglichkeit der Konstruktion welche wir betrachten, nutzt die in Kapitel 2 definierten Hard-core-Prädikate.

Satz 3.1: [Gol91] Sei $f := \{f_n : \{0, 1\}^n \rightarrow \{0, 1\}^n\}_{n \in \mathbb{N}}$ eine Familie von Einwegpermutationen und $Q := \{Q_n : \{0, 1\}^n \rightarrow \{0, 1\}\}_{n \in \mathbb{N}}$ ein Hard-core-Prädikat für f . Es bezeichne weiterhin $x \frown y$ das anhängen eines Bitstrings y an x . Dann ist die Familie $G := \{G_n : \{0, 1\}^n \rightarrow \{0, 1\}^{n+1}\}_{n \in \mathbb{N}}$ mit $G(X) := f(X) \frown Q(X)$; $X \in_U \{0, 1\}^*$ ein Pseudozufallszahlengenerator.

Ausgehend von einem Zufallswert wird also eine um ein Bit längere Bitfolge erzeugt, indem die Ausgabe des Hard-core-Prädikats an die Ausgabe der Einwegpermutation gehängt wird, welche für jeden polynominell beschränkten Beobachter nicht von einer echten Zufallszahl zu unterscheiden ist.

Beweis Um Satz 3.1 nachzuweisen, kann folgende Beweisidee verwendet werden: wenn es einen effizienten Algorithmus A gäbe, welcher bei Eingabe eines $y \in \{0, 1\}^{n+1}$ entscheiden kann, ob dieses y eine gleichverteilte Zufallsvariable ist oder von G erzeugt wurde so kann A die Entscheidung nur aufgrund des letzten Bits treffen, da f eine Einwegpermutation ist. Dies führt zu einem Widerspruch zur Hard-core-Eigenschaft von Q .

Nehmen wir an, es gibt einen probabilistischen polynominell zeitbeschränkten Algorithmus A und ein Polynom $p(\cdot)$, so dass gilt:

$$|Pr_{y \in G(U_n)}(A(y) = 1) - Pr(A(U_{n+1}) = 1)| > \frac{1}{p(n)}.$$

Wenn A also als Ergebnis eine 0 zurück gibt, so war die Eingabe wahrscheinlicher eine gleichverteilte Zufallsvariable, liefert A hingegen eine 1, so ist es wahrscheinlicher, dass es sich um die Ausgabe unseres Generators handelt. O.B.d.A. können wir diese Ungleichung auch ohne Betrag schreiben.

Wenden wir nun A auf $G(x) = f(x) \frown b$ an, so gibt A wahrscheinlicher eine 1 zurück, falls $b = Q(x)$ ist.

$$\begin{aligned} Pr_{x \in U_n, b \in U_1}(A(f(x) \frown b) = 1) &= Pr(A(f(x) \frown b) = 1 | b = Q(x)) \cdot Pr(b = Q(x)) \\ &\quad + Pr(A(f(x) \frown b) = 1 | b = \overline{Q(x)}) \cdot Pr(b = \overline{Q(x)}) \\ &= \frac{1}{2}(\alpha + \beta) \end{aligned}$$

mit

$$\alpha = Pr(A(f(x) \frown b) = 1 | b = Q(x))$$

und

$$\beta = Pr(A(f(x) \frown b) = 1 | b = \overline{Q(x)}).$$

Also gilt:

$$\begin{aligned}
 \Pr_{x \in U_n}(A(f(x) \frown Q(x)) = 1) - \Pr_{x \in U_n}(A(f(x) \frown b) = 1) &= \alpha - \frac{1}{2}(\alpha + \beta) \\
 &= \frac{1}{2}(\alpha - \beta) \\
 &> \frac{1}{p(n)}.
 \end{aligned}$$

Wir können nun einen Algorithmus A' konstruieren, welcher bei Eingabe von $f(x)$ mit einer Wahrscheinlichkeit deutlich größer als $\frac{1}{2}$ das Hard-core-Prädikat $Q(x)$ berechnet.

Alg. A' : (Eingabe: $f(x)$)

- wähle $b \in \{0, 1\}$
- berechne $\gamma := A(f(x) \frown b)$
- falls $\gamma = 1$ Ausgabe: b
- sonst Ausgabe: \bar{b}

Es bleibt noch zu zeigen, dass die Erfolgswahrscheinlichkeit hinreichend groß ist.

$$\begin{aligned}
 \Pr(A'(f(x)) = Q(x)) &= \Pr(A(f(x) \frown b) = 1 | b = Q(x)) \cdot \Pr(b = Q(x)) \\
 &\quad + \Pr(A(f(x) \frown b) = 0 | b = \overline{Q(x)}) \cdot \Pr(b = \overline{Q(x)}) \\
 &= \frac{1}{2}\alpha + \frac{1}{2}(1 - \beta) \\
 &= \frac{1}{2} + \frac{1}{2}(\alpha - \beta) \\
 &> \frac{1}{2} + \frac{1}{p(n)}
 \end{aligned}$$

Dies widerspricht jedoch der Hard-core-Eigenschaft von Q , also ist G ein Pseudozufallszahlengenerator.

◇

Satz 3.2: *Pseudozufallszahlengeneratoren existieren genau dann, wenn Einwegfunktionen existieren.*

Diese Beziehung wurde 1999 von J. Håstad R. Impagliazzo, L. Levin und M. Luby [HILL99] vollständig bewiesen. Aus Satz 3.1 und unter zu Hilfenahme von Satz 2.1 wissen wir bereits, dass die Existenz von Einwegpermutationen die Existenz von Pseudozufallszahlengeneratoren impliziert. Wir werden zunächst die einfachere Richtung von Satz 3.2 zeigen.

Lemma 3.3: *Wenn Pseudozufallszahlengeneratoren existieren, so existieren auch Einwegfunktionen.*

Beweis (Lemma 3.3): (siehe u.a. [Lub96]) Sei G ein Pseudozufallszahlengenerator mit $l(n) = 2n$. Wir werden zeigen, dass $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ mit $f(x, y) := G(x) \forall |x| = |y|$ eine Einwegfunktion ist. Da G ein Pseudozufallszahlengenerator ist, kann f also effizient berechnet werden, ist also eine P-Funktion. Sei $s(\cdot)$ nun ein Polynom und A ein probabilistischer Polynomialzeitalgorithmus für den gilt:

$$\Pr(f(A(f(x))) = f(x)) > \frac{1}{s(n)}. (*)$$

Mit anderen Worten: A kann zu einer beliebigen Funktionswert $f(x)$ mit einer hinreichend großen Wahrscheinlichkeit ein Urbild berechnen. Wir werden mit Hilfe von A einen Algorithmus D konstruieren, welcher $G(U_n)$ von U_{2n} unterscheiden kann.

Alg. D : (Eingabe: $\alpha \in \{0, 1\}^*$)

- berechne $\beta := A(\alpha)$
- falls $\alpha = f(\beta)$ Ausgabe: 1
- sonst Ausgabe: 0

Wegen (*) gilt also

$$\Pr(f(A(f(U_{2n}))) = f(U_{2n})) > \frac{1}{s(n)}$$

und damit:

$$\Pr(D(G(U_n)) = 1) > \frac{1}{s(n)}.$$

Nach der Konstruktion von f haben aber nur 2^n verschiedene $2n$ -Bit lange Strings ein Urbild. Folglich gilt

$$\Pr(f(A(U_{2n})) = U_{2n}) \leq 2^{-n}$$

, es folgt also

$$|\Pr(D(G(U_n)) = 1) - \Pr(D(U_{2n}) = 1)| > \frac{1}{s(n)} - \frac{1}{2^n} > \frac{1}{2s(n)}.$$

Dies widerspricht aber der Pseudozufälligkeit von G .

◇

Die Gegenrichtung werden wir lediglich skizzieren. Der Beweis verläuft in mehreren Schritten, um, ausgehend von einer beliebigen Einwegfunktion, zu einem Pseudozufallszahlengenerator zu gelangen. Wir werden dabei jedoch nur den ersten Schritt der Konstruktion betrachten. Zunächst müssen wir dazu eine Anzahl von Bezeichnungen und Definitionen festlegen. (vergleiche [HILL99])

Ein wichtiges Hilfsmittel in diesem Beweis sind die universellen Hashfunktionen. Unter einer universellen Hashfunktion (siehe auch [CW79]) verstehen

wir eine P-Funktion $h : \{0, 1\}^{l_n} \times \{0, 1\}^n \rightarrow \{0, 1\}^{m_n}$ wenn für alle $x \in \{0, 1\}^n$, $x' \in \{0, 1\}^n \setminus \{x\}$ und alle $a, a' \in \{0, 1\}^{m_n}$ gilt:

$Pr[(h_R(x) = a) \text{ und } (h_R(x') = a')] = \frac{1}{2}^{2m_n}$ mit $R \in_U \{0, 1\}^{l_n}$. R ist dabei eine Beschreibung für die Funktion h_R welche n Bits auf m_n Bits abbildet. Diese Funktionen haben bei unseren Konstruktionen die Aufgabe die Verteilung zu glätten, um nahezu eine Gleichverteilung der Ausgabe zu erreichen.

Gegeben sei nun eine Einwegfunktion $f : \{0, 1\}^n \rightarrow \{0, 1\}^{l(n)}$. Dazu konstruieren wir uns eine Familie von P-Funktion f' für $x \in \{0, 1\}^n$, $i \in \{0, \dots, n-1\}$ und $r \in \{0, 1\}^{p_n}$ wie folgt: $f'(x, i, r) = \langle f(x), h_r(x)_{\{1, \dots, i + \lceil \log(2n) \rceil\}}, i, r \rangle$, wobei h eine universelle Hashfunktion mit $h : \{0, 1\}^{p_n} \times \{0, 1\}^n \rightarrow \{0, 1\}^{\lceil \log(2n) \rceil}$ ist. Dabei bezeichnet p_n die Wahrscheinlichkeit, dass für $\tilde{D}_f(z) = \lceil \log(\#\{x \in \{0, 1\}^n : f(x) = z\}) \rceil$ für $z \in Im(f)$ gilt, dass $I \leq \tilde{D}_f(f(X))$ mit $I \in_U \{0, \dots, n-1\}$ ist. $\tilde{D}_f(z)$ wird als approximative Degeneration bezeichnet, und ist ein Mass dafür, wie groß die Abnahme der Entropie durch Anwendung einer Funktion f auf einen Wert z ist, welcher gleichverteilt gewählt wurde (vergleiche u.a. [HILL99]).

Die oben definierte Funktion f' ist wiederum eine Einwegfunktion, mit deren Hilfe wir einen Pseudoentropiegenerator konstruieren können. Der Unterschied zwischen einem Pseudozufallszahlengenerator und einem Pseudoentropiegenerator besteht darin, dass die Ausgabe beim zweiten nicht notwendigerweise P-ununterscheidbar von einer Gleichverteilung sein muss, sondern nur P-ununterscheidbar von einer Familie von Zufallsvariablen, welche eine höhere Entropie hat als die Eingabe des Pseudoentropiegenerators. Er stellt somit einen Zwischenschritt von der Einwegfunktion zum Pseudozufallszahlengenerator dar.

Sei $\Xi = \langle X, I, R \rangle$ die Eingabe von f' , c_n die Länge von Ξ und c'_n die Länge von $f'(\Xi)$. Wir setzen $e_n := H(f'(\Xi))$, $Q(\Xi, y) := x \odot y$ und $k_n = 2000n^6$.

Die pseudozufälligen Bits werden nun wie folgt gebildet: Sei $\Xi' = \Xi^{k_n}$ und $Y' = Y^{k_n}$. Wir berechnen zuerst $f'^{k_n}(\Xi')$ und $Q^{k_n}(\Xi', Y')$. Wir sollten nun $k_n c_n - H\langle f'^{k_n}(\Xi'), Q^{k_n}(\Xi', Y') \rangle$ Bits ausgehend von Ξ' erhalten, da dies die bedingte Entropie ist, welche nach Berechnung von f'^{k_n} und Q^{k_n} bleibt. Mittels einiger weiterer Überlegungen können wir feststellen, dass wir $\frac{k_n}{2n}$ Bits mehr erhalten, als die Eingabe von f'^{k_n} lang ist. Da diese Methode aber nicht perfekt ist, müssen wir $2nk_n^{\frac{2}{3}}$ Bits wieder "opfern", weswegen wir k_n wie oben wählten, um zu gewährleisten, dass $\frac{k_n}{2n} > 2nk_n^{\frac{2}{3}}$ ist. Für die vollständige Argumentation dazu siehe [HILL99].

Sei $m_n := k_n(c_n - e_n - p_n + \frac{1}{2n}) - 2nk_n^{\frac{2}{3}}$, $m'_n = k_n p_n - 2nk_n^{\frac{2}{3}}$ und $m''_n = k_n e_n - 2nk_n^{\frac{2}{3}}$. Seien weiterhin R_1, R_2 und R_3 Indizes von Hashfunktionen derart, dass h_{R_1} $k_n c_n$ Bits auf m_n Bits, h_{R_2} k_n Bits auf m'_n Bits und h_{R_3} $k_n c'_n$ Bits auf m''_n Bits abbildet.

Nun können wir eine Funktion g wie folgt definieren:
 $g(X', Y', R_1, R_2, R_3) = \langle h_{R_1}(X'), h_{R_2}(b^{k_n}(X', Y')), h_{R_3}(f'^{k_n}(X')), Y', R_1, R_2, R_3 \rangle$.

Lemma 3.4: Falls f eine Einwegfunktion ist, dann ist g ein schwach nicht-uniformer Pseudozufallszahlengenerator.

(Beweis siehe [HILL99])

Im letzten Schritt lassen sich dann daraus Pseudozufallszahlengeneratoren konstruieren.

Lemma 3.5: Sei für jedes $a_n \in \{0, \dots, k_n\}$ die Familie von P -Funktionen $g : \{0, 1\}^{\lceil \log(k_n) \rceil} \times \{0, 1\}^n \rightarrow \{0, 1\}^{l_n}$ mit $l_n > nk_n$ ein schwach nicht-uniformer Pseudozufallszahlengenerator falls der erste Teil der Eingabe a_n ist. Sei weiterhin $x' \in \{0, 1\}^{k_n \times n}$. Dann ist die Familie von P -Funktionen $g'(x') := \bigoplus_{i=1}^{k_n} g(i, x'_i)$ ein Pseudozufallszahlengenerator

(Beweis siehe [HILL99])

Wir haben also, ausgehend von einer Einwegfunktion, einen Pseudozufallszahlengenerator konstruiert.

3.2 Konstruktionen für beliebig lange Pseudozufallszahlen

Da in den meisten Fällen die Anzahl der echten Zufallsbits stark beschränkt ist, konstruieren wir nun Generatoren mit möglichst großem Expansionsfaktor. Naturgemäß können wir hier nicht alle, von verschiedenen Autoren, vorgeschlagenen Konstruktionen betrachten und werden uns daher auf eine kleine Auswahl beschränken.

Algorithmus 3.1:

Sei G_1 ein Pseudozufallszahlengenerator wie in Satz 3.1 wobei f eine Einwegpermutation ist. G_1 erzeugt damit, ausgehend von Startwerten s mit $|s| = n$, Pseudozufallszahlen der Länge $n + 1$. Sei $l(\cdot)$ ein Polynom. Dann definieren wir $G(s) := \sigma_1 \dots \sigma_{l(n)}$ mit $s_0 := s$; σ_i das erste Bit von $G_1(s_{i-1})$ und s_i die letzten n Bits von $G_1(s_{i-1})$ für alle $0 < i \leq l(n)$.

Satz 3.6: [Gol91] G wie in Algorithmus 3.1 ist ein Pseudozufallszahlengenerator.

Beweis:

Wie man leicht sieht, folgt mit dieser Konstruktion aus der effizienten Berechenbarkeit von G_1 auch die von G . Der Expansionsfaktor des Generators ist $l(n)$.

Die Pseudozufälligkeit von Algorithmus 3.3 folgt aus der Pseudozufälligkeit von G_1 mittels des sogenannten Hybridarguments. Dabei versuchen wir, zwischen $\{G(U_n)\}$ wobei U_n eine gleichverteilte Zufallsvariable der Länge n ist, und $U_{l(n)}$, ebenfalls gleichverteilt und $l(n)$ Bit lang, $l(n)$ Zwischenwerte, die Hybride H_k mit $0 \leq k \leq l(n)$ zu finden. Zwei Nachbarhybride unterscheiden sich bei dieser Konstruktion lediglich in einem Bit.

Dazu sei $H_0 = \{G(U_n)\}$ und $H_{l(n)} = U_{l(n)}$. Falls $G(U_n)$ und $U_{l(n)}$ unterschieden werden können, dann muss dieser Unterschied auch zwischen zwei benachbarten Hybriden H_k und H_{k+1} auftauchen. Gelingt dies nicht, so können auch H_0 und $H_{l(n)}$ nicht unterschieden werden.

Nehmen wir nun also an, dass G kein Pseudozufallszahlengenerator ist. Wir konstruieren die Hybride dann wie folgt:

$H_k := U'_k g_{l(n)-k}(U''_n)$ mit U'_k und U''_n zwei unabhängige gleichverteilte Zufallsvariablen über $\{0, 1\}^k$ resp. $\{0, 1\}^n$ sowie $g_k : \{0, 1\}^n \rightarrow \{0, 1\}^k$ mit $g_0 := \varepsilon$ und $g_{k+1} := \sigma g_k(y)$ mit σ das erste Bit und y der n Bit lange Rest von $G_1(x)$. Damit besteht H_k also aus einem k Bit langen Anfang von U_n , welcher mit dem $(l(n) - k)$ Bit langen Ende von $G(U_n)$ ergänzt wird.

Betrachten wir nun zwei benachbarte Hybride H_k und H_{k+1} . Da nach Definition von g gilt $g_m(x) = f_m(G_1(x))$, folgt:

$$H_k := U'_k g_{l(n)-k}(U''_n) = U'_k f_{l(n)-k}(G_1(U''_n)) ;$$

und wegen $f_{m+1}(\sigma y) = \sigma g_m(y)$ folgt

$$H_{k+1} := U'_{k+1} g_{l(n)-k-1}(U''_n) = U'_k f_{l(n)-k}(U''_{n+1}) .$$

Wir haben also die Unterscheidbarkeit von $\{G(U_n)\}$ und $U_{l(n)}$ auf die Unterscheidbarkeit von zwei Nachbarhybriden zurückgeführt.

Sei nun D ein probabilistischer Polynomialzeitalgorithmus, welcher die von G erzeugten Zahlen von einer gleichverteilten Zufallsvariable mit einer Wahrscheinlichkeit $> \varepsilon$ unterscheiden kann. Dann können wir daraus einen Algorithmus \hat{D} konstruieren, welcher auch $G_1(U_n)$ von U_{n+1} unterscheiden kann.

- Alg. \hat{D} : (Eingabe: $\alpha \in \{0, 1\}^{n+1}$)
- wähle (gleichverteilt) ein k aus $\{0, 1, \dots, l(n) - 1\}$
 - wähle (gleichverteilt) ein β aus $0, 1^k$
 - Ausgabe: $D(\beta f^{l(n)-k}(\alpha))$

Wie man leicht sieht, ist \hat{D} in Polynomialzeit berechenbar.

Aus der Annahme, dass G kein Pseudozufallszahlengenerator ist, folgt damit, dass auch G_1 keiner sein kann, was wiederum ein Widerspruch zur Voraussetzung ist.

◇

Die im Satz 3.1 vorgestellte Konstruktion mittels Hard-core-Prädikaten lässt sich nun auch wie folgt abändern. Sei $l(\cdot)$ ein Polynom. Wir wenden dann auf einen Startwert x_0 mit $|x| = n$ eine Einwegpermutation an, erhalten dadurch einen Wert x_1 und berechnen ein zur Permutation gehöriges Hard-core-Prädikat welches wir ausgeben. Diesen Schritt wiederholen wir nun $l(n)$ mal mit x_i als Startwert für $0 < i < l(n)$ und erhalten so einen $l(n)$ -Bit langen String.

Algorithmus 3.2:

Sei dazu zunächst f eine Familie von Einwegpermutationen, Q ein Hard-core-Prädikat für f und $l(\cdot)$ ein Polynom. Dann ist die $l(|x|)$ Bit lange Sequenz $Q(x), Q(f(x)), \dots, Q(f(f(\dots(f(x))\dots)))$ eine $l(|x|)$ -Bit lange Pseudozufallszahl.

Um zu zeigen, dass diese Konstruktion tatsächlich ein Pseudozufallszahlengenerator ist, zeigt man zunächst, dass, falls die ersten i Bits der Ausgabe gegeben sind, es nicht möglich ist, das $i + 1$ Bit mit einer Wahrscheinlichkeit wesentlich größer als $\frac{1}{2}$ zu bestimmen. Diese Eigenschaft wird Unvorhersagbarkeit des nächsten Bits, bzw. "next bit unpredictability" genannt.

Definition 3.2

Eine Familie von P-Funktion $f := \{f_n : \{0, 1\}^n \rightarrow \{0, 1\}^{l(n)}\}_{n \in \mathbb{N}}$ wird P-unvorhersagbar bzgl. des nächsten Bits genannt, falls für jeden polynominal beschränkten Algorithmus A , alle hinreichend großen n , alle Polynome $s(\cdot)$ sowie alle $x \in \{0, 1\}^n$ gilt: $Pr(A(f(x)) = next_A(f(x))) < \frac{1}{2} + \frac{1}{s(n)}$.

Dabei liefert $next_A$ angewandt auf ein y das $(i + 1)$ te Bit von y , während A aus i gelesenen Bits von y das $(i + 1)$ te berechnet.

Es lässt sich aus der Definition der Hard-core-Prädikaten folgern, dass die vom Algorithmus 3.2 erzeugte Funktion P-unvorhersagbar bzgl. des nächsten Bits ist. Folgender Satz 3.7 liefert uns nun einen Zusammenhang zwischen Pseudozufälligkeit und P-Unvorhersagbarkeit. Damit ergibt sich, zusammen mit obiger Beobachtung, dass Algorithmus 3.2 in der Tat ein Pseudozufallszahlengenerator ist.

Satz 3.7: [Lub96] Eine Familie von P-Funktion $f := \{f_n : \{0, 1\}^n \rightarrow \{0, 1\}^{l(n)}\}_{n \in \mathbb{N}}$ ist genau dann ein Pseudozufallszahlengenerator, wenn sie P-unvorhersagbar bezüglich des nächsten Bits und $l(n) > n$ ist.

Beweisidee: Satz 3.7 lässt sich indirekt zeigen und es ergibt sich dann folgende Beweisidee:

(\Rightarrow) Aus der Annahme f sei nicht P-unvorhersagbar bzgl. des nächsten Bits, lässt sich leicht ein Algorithmus konstruieren, welcher zwischen der Ausgabe von f und $U_{l(n)}$ unterscheiden kann.

(\Leftarrow) Aus der Annahme f sei nicht pseudozufällig, es also eine probabilistische P-Funktion gibt welche zwischen der Ausgabe von f und $U_{l(n)}$ unterscheiden kann, lässt sich mit Hilfe des Hybridarguments ein Algorithmus konstruieren, welcher das nächste Bit vorhersagt, was ein Widerspruch zur Voraussetzung wäre (siehe [Gol91]).

Eine weitere von Goldreich vorgeschlagene Möglichkeit Pseudozufallszahlengeneratoren zu konstruieren, besteht in der Verwendung mehrerer Einwegfunktionen. Die Reihenfolge und die Auswahl der Funktionen steht dabei in Abhängigkeit vom Startwert. Auf diese Art und Weise erreicht man eine weitere Verlängerung der Pseudozufallszahlen.

Algorithmus 3.3: [Gol91]

Seien G_1, \dots, G_m Pseudozufallszahlengeneratoren mit $G_i : 0, 1^n \rightarrow 0, 1^{l(n)}$. Sei weiterhin π eine Einwegfunktion mit $\pi : \{0, 1\}^n \times \{0, 1, \dots, m\} \rightarrow \{0, 1, \dots, m\}$. Dann ist $G : \{0, 1\}^n \rightarrow \{0, 1\}^{ml(n)}$ mit $G := (G_{\pi(x,1)}(x), \dots, G_{\pi(x,m)}(x))$ ein Pseudozufallszahlengenerator.

Beweisidee: Die P-Ununterscheidbarkeit von G zu $U_{ml(n)}$ lässt sich wiederum durch das Hybridargument zeigen. Aus der Negation dieser Annahme lässt sich dann folgern, dass mindestens einer der Generatoren G_i kein Pseudozufallszahlengenerator sein kann.

Wir sind also nun mit Hilfe der BMY-Typ-Generatoren in der Lage, die Existenz von Einwegfunktionen vorausgesetzt, Pseudozufallszahlen der Länge m aus einem Startwert der Länge m^δ für alle $\delta > 0$ zu generieren (vergleiche Satz 3.2), und können dabei gewährleisten, dass die derart erzeugten Zahlen für kryptographische Zwecke sicher zu gebrauchen sind. In den folgenden Kapiteln werden wir uns einen weiteren, nicht ursprünglich für die Kryptographie entwickelten Ansatz betrachten.

Kapitel 4

Pseudozufallszahlengeneratoren für beliebige Komplexitätsklassen

Eines der größten Probleme der bisher betrachteten Konstruktionen ist, dass sie lediglich für die Komplexitätsklasse P definiert sind, und sich nicht ohne weiteres auf beliebige Klassen übertragen lassen. Des weiteren beruhen sie auf der bislang unbewiesenen Annahme, dass $P \neq NP$ gilt. Wir wollen nun eine von N. Nisan [Nis90, NW94] vorgeschlagene Konstruktion betrachten, welche beide Probleme vermeidet, und für jede Komplexitätsklasse C definiert ist. Dabei bleibt zu beachten, dass diese Art von Generatoren nicht die gleichen Sicherheitsanforderungen wie die bisher betrachteten Generatoren erfüllen.

Wenn wir nun für beliebige Klassen Generatoren konstruieren, wird es notwendig eine neue Betrachtungsweise einzuführen. Die bisherigen Betrachtungen stützten sich lediglich auf polynominell in der Zeit beschränkte Algorithmen; im Folgenden werden wir auch den Begriff des Schaltkreises hinzunehmen. Unter einem Schaltkreis versteht man eine Kombination aus "and", "or" und "not" Gattern. Die Größe eines solchen Schaltkreises beschreibt dabei die Anzahl der Gatter. Wenn wir im Zusammenhang von Schaltkreisen dann von zeitlich beschränkten Kreisen reden, so enthält diese zeitliche Beschränkung nur die Laufzeit, nicht aber die Zeit, welche nötig wäre um eine Beschreibung dieses Schaltkreises zu finden. Diese Zeit wiederum kann exponentiell von der Größe abhängen. Da im Allgemeinen für jede Eingabelänge ein anderer Schaltkreis benötigt wird, repräsentieren sie also das nicht uniforme Modell (vergleiche Kapitel 2.2).

Eine Erweiterung dieser Schaltkreise sind probabilistische Schaltkreise. Sie enthalten zusätzliche Auswahl-, resp. "choice"-Gatter. Ein Auswahlgatter hat keinen Eingang und liefert mit der Wahrscheinlichkeit $\frac{1}{2}$ eine 1 oder 0. Wir sagen ein probabilistische Schaltkreis C akzeptiert ein Wort x , falls ein Polynom $p(\cdot)$

existiert, so dass die Wahrscheinlichkeit, genommen über alle Ausgaben der Auswahlgatter, dass C bei Eingabe von x eine 1 ausgibt größer als $\frac{1}{2} + \frac{1}{p(|x|)}$ ist. Sie bilden somit ein Pendant zu probabilistischen Algorithmen.

Im weiteren Verlauf werden wir neben deterministischen Schaltkreisen auch nichtdeterministische Schaltkreise nutzen. Ein nichtdeterministischer Schaltkreis $C(x, y)$ ist ein Schaltkreis mit x als primärer Eingabe und y als Zeuge. Das heißt, für jedes $x \in \{0, 1\}^*$ definieren wir $C(x) = 1$ falls ein y existiert, so dass $C(x, y) = 1$ ist. Analog dazu definieren wir einen co-nichtdeterministischen Schaltkreis, jedoch ist hier $C(x) = 0$ falls ein y existiert, so dass $C(x, y) = 0$ ist.

Zwei weitere Begriffe sollen an dieser Stelle zumindest informell geklärt werden. Eine Funktion f heißt in $DTIME(p(n))$ liegend, falls ein deterministischer Algorithmus F , dessen Laufzeit von der Eingabelänge n abhängt und durch $p(n)$ beschränkt ist, f berechnen kann. Eine Funktion f heißt in $EXPTIME$ oder EXP liegend, falls ein deterministischer Algorithmus F , dessen Laufzeit exponentiell von der Eingabelänge abhängt, f berechnen kann, d.h. $EXPTIME = \bigcup_{p(n)} DTIME(2^{p(n)})$, wobei $p(\cdot)$ Polynome sind.

4.1 Grundlegende Definition

Zunächst wollen wir uns klar machen, was wir für beliebige Klassen unter dem Begriff "Härte" verstehen, da wir die Existenz harter Funktionen im weiteren Verlauf voraussetzen. Damit eine Funktion als hart bezeichnet werden kann, benötigen wir nicht nur, dass diese Funktion nicht von kleinen Schaltkreisen, also Schaltkreisen bis zu einer bestimmten Größe, berechnet werden, sondern darüberhinaus auch, dass sie von solchen Schaltkreisen nicht approximiert werden kann.

Definition 4.1: Sei $f : \{0, 1\}^n \rightarrow \{0, 1\}$ eine boolesche Funktion. Wir sagen f kann nicht von Schaltkreisen der Größe $s(n)$ approximiert werden, falls für einige Konstanten k , hinreichend große n und alle Schaltkreise C_n mit einer Größe von höchstens $s(n)$ gilt:

$$Pr(C_n(x) \neq f(x)) > n^{-k}$$

, wobei x zufällig gleichverteilt aus $\{0, 1\}^n$ gewählt wird.

Mit anderen Worten, der Anteil an fehlerhaften Werten, welche C ausgibt darf nicht vernachlässigbar sein. Dies ist jedoch eine relativ schwache Forderung. Unser Ziel ist es jedoch Funktionen zu erhalten, die von einem kleineren Schaltkreis lediglich mit einem vernachlässigbaren Vorteil berechnet werden können.

Definition 4.2: Sei $f : \{0, 1\}^n \rightarrow \{0, 1\}$ eine boolesche Funktion. Dann heißt f (ε, s) -hart, falls für jeden Schaltkreis C mit einer Größe von maximal s gilt:

$$|Pr(C(x) = f(x)) - \frac{1}{2}| < \frac{\varepsilon}{2}$$

, wobei x zufällig gleichverteilt aus $\{0, 1\}^n$ gewählt wird.

Sei $f = \{f_n : \{0, 1\}^n \rightarrow \{0, 1\}\}_{n \in \mathbb{N}}$ eine Familie von Prädikaten.

Dann bezeichnet $H_f(n) := \max_{h_n \in \mathbb{Z}} \{f_n \text{ ist } (\frac{1}{h_n}, h_n) - \text{hart}\}$ die Härte von f für n .

Damit ist die Härte einer Funktion f für n also der größte ganzzahlige Wert h_n , für den gilt: f_n ist $(\frac{1}{h_n}, h_n)$ -hart.

Lemma 4.1: [Nis90] Für jede in der Größe beschränkte Funktion $s(m)$ mit $m \leq s(m) \leq 2^m$ gilt: falls eine Funktion f in EXPTIME existiert, welche nicht von Schaltkreisen der Größe $s(m)$ approximiert werden kann, dann gibt es ein $c > 0$, so dass eine Funktion f' in EXPTIME existiert, welche die Härte $s(mc) \leq H_{f'}(m)$ hat.

Ohne Beweis. Lemma 4.1 ist eine Folgerung aus Yao's Lemma [Yao82] welches aussagt, dass für jedes $\delta > 0$, falls alle Funktionen f_i (ε, s) -hart sind, die Funktion $f(x_1, \dots, x_k) := \sum_{i=1}^k f_i(x_i) \bmod 2$ eine $(\varepsilon^k + \delta, \delta^2(1 - \varepsilon)^2 s)$ -harte Funktion ist.

Wir können damit auch den Begriff Pseudozufallszahlengenerator neu fassen.

Definition 4.3:

Eine Familie G von Funktionen, mit $G = \{G_n : \{0, 1\}^{l(n)} \rightarrow \{0, 1\}^n\}_{n \in \mathbb{N}}$ heißt NW-Typ-Pseudozufallszahlengenerator, falls für alle Schaltkreise C mit einer Größe von maximal n gilt:

$$|Pr(C(U_n) = 1) - Pr(C(G(x)) = 1)| < \frac{1}{n}$$

; mit $x \in_U \{0, 1\}^{l(n)}$ und $l(n) < n$.

G heißt schneller NW-Typ-Pseudozufallszahlengenerator, falls G deterministisch ist und die Laufzeit von G exponentiell von der Eingabelänge abhängt; $G \in DTIME(2^{O(l(n))})$.

An dieser Stelle sei darauf hingewiesen, dass die Laufzeit der NW-Typ-Generatoren deutlich größer sein darf, als die der BMY-Typ-Generatoren, welche polynominell zeitbeschränkt sind.

Um möglichst viele Pseudozufallsbits zu erhalten, werden wir die Funktion auf viele verschiedene, fast disjunkte, Teilmengen anwenden. Dazu benötigen wir folgende Definition.

Definition 4.4:

Eine Menge $S = \{S_1, \dots, S_n\}$ mit $S_i \subset \{1, \dots, l\}$ heißt (k, m) -Design, falls

- (i) $\forall i$ gilt $|S_i| = m$
- (ii) $\forall i \neq j$ gilt $|S_i \cap S_j| \leq k$

Eine $n \times l$ Matrix A über $\{0, 1\}$ heißt (k, m) -Design, falls die Zeilen $a_i, 0 < i \leq n$, interpretiert als Teilmengen von $\{1, \dots, l\}$ ein (k, m) -Design sind.

Definition 4.5:

Sei A eine $n \times l$ Matrix über $\{0, 1\}$ und f ein Prädikat. Sei weiterhin $x = (x_1, \dots, x_l)$ ein String von $x_i \in \{0, 1\}$. Dann bezeichnet $f_A(x)$ den n -Bit-Vektor, welcher entsteht, indem man das Prädikat f auf die Teilmengen der Bits von x anwendet, welche durch die n verschiedenen Spalten von A angegeben sind.

Beispiel: Sei $x = (1, 0, 1)$ und

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

sowie

$$f(\nu) := \begin{cases} 1, & \text{falls die Anzahl der 1 in } \nu \text{ gerade ist} \\ 0, & \text{sonst} \end{cases}$$

für alle Vektoren ν über $\{0, 1\}$.

Dann ist $f_A(x) = (0, 1, 1, 0)$.

Mit Hilfe dieser Designs können wir nun NW-Typ-Pseudozufallszahlengeneratoren konstruieren.

Lemma 4.2: [Nis90] Sei $f : \{0, 1\}^m \rightarrow \{0, 1\}$ ein Prädikat mit $H_f(m) \geq n^2$. Sei weiterhin A eine $n \times l$ Matrix über $\{0, 1\}$, wobei A ein $(\log n, m)$ -Design ist, dann ist $G : \{0, 1\}^l \rightarrow \{0, 1\}^n$ mit $G(x) := f_A(x)$ ein NW-Typ-Pseudozufallszahlengenerator.

Beweis:

Die Aussage von Lemma 4.2 kann bewiesen werden, indem die Annahme, G sei kein NW-Typ-Pseudozufallszahlengenerator, zum Widerspruch geführt wird. (Siehe [Nis90]).

Falls G kein Pseudozufallszahlengenerator ist, so muss es einen Schaltkreis C der Größe n geben, für den gilt:

$$|Pr(C(y) = 1) - Pr(C(G(x)) = 1)| > \frac{1}{n} \text{ mit } x \in_U \{0, 1\}^l \text{ und } y \in_U \{0, 1\}^n.$$

Wir zeigen nun, dass daraus folgt, dass ein Bit von $f_A(x)$ aus den vorhergehenden ermittelt werden kann, also $f_A(x)$ nicht unvorhersagbar bezüglich des nächsten Bits ist.

Für alle $i, 0 \leq i \leq n$ definieren wir Hybride H_i wie folgt: die ersten i Bits von H_i werden durch die ersten i Bits von $f_A(x)$ gesetzt, die restlichen $n - i$ durch U_{n-i} . Wie definieren nun $p_i := Pr(C(H_i) = 1)$. Da $p_0 - p_n > \frac{1}{n}$ ist, muss also

für ein i gelten: $p_{i-1} - p_i > \frac{1}{n^2}$. Darauf aufbauend können wir einen Schaltkreis konstruieren, welcher das i te Bit voraussagen kann.

Wir definieren einen probabilistischen Schaltkreis D , welcher als Eingabe die ersten $i-1$ Bits von $f_A(x)$, wir bezeichnen sie als y_1, \dots, y_{i-1} , erhält und daraus y_i vorhersagt. D generiert zuerst $n-i+1$ zufällige Bits r_i, \dots, r_n und berechnet dann $C(y_1, \dots, y_{i-1}, r_i, \dots, r_n)$. Falls C eine 1 zurück gibt, so liefert D r_i als Ergebnis, sonst das Komplement.

Der Vorteil ist dann $|Pr(D_n(y_1, \dots, y_{i-1}) = y_i) - \frac{1}{2}| > \frac{1}{n^2}$. [Yao82]

Wir können daraus mit Hilfe einer Mittelwertbildung einen deterministischen Schaltkreis D' konstruieren, welcher die Zufallsbits r_j durch Konstanten ersetzt, ohne dadurch den Vorteil zu verringern. Um nun einen Widerspruch zur Annahme über die Härte zu erhalten, verändern wir diesen Schaltkreis derart, dass er y_i bei Eingabe von x_1, \dots, x_l vorhersagt. O.B.d.A. nehmen wir an, dass y_i von den ersten m Bits abhängt d.h. $y_i = f(x_1, \dots, x_m)$.

Da y_i also nicht von den restlichen Bits von x abhängig ist, ist es möglich diese Bits als konstant zu setzen. Wenn wir nun einen Mittelwert bilden, so erhalten wir Konstanten c_{m+1}, \dots, c_l derart, dass wir $x_j = c_j$ für alle $m < j \leq l$ setzen können, ohne die Erfolgswahrscheinlichkeit zu mindern. Jedes der Bits y_1, \dots, y_{i-1} hängt wiederum von höchstens $\log n$ der Bits von x_1, \dots, x_m ab, da der Durchschnitt der Mengen der x'_k definiert durch die y_i dadurch von oben durch $\log n$ für alle $i \neq j$ beschränkt ist.

Wir können nun jedes y_i als eine CNF-Formel, also eine Formel in konjunktiver Normalform, berechnen. Dies führt wiederum zu einem Schaltkreis $D''(x_1, \dots, x_m)$ welcher y_i vorhersagt. Da die Größe von D'' höchstens n^2 und der Vorteil größer als $\frac{1}{n^2}$ ist, folgt daraus ein Widerspruch zur Voraussetzung $H_f(m) > n^2$

◇

4.2 Nisans Haupttheorem

Zunächst definieren wir eine Spezialform des Pseudozufallszahlengenerators mit einer Expansion von einem Bit.

Definition 4.6: Eine Funktion $G := \{G_n : \{0, 1\}^l \rightarrow \{0, 1\}^{l+1}\}$ heißt $n(l)$ -Extender, falls für alle alle Schaltkreise C mit einer maximalen Größe von $n := n(l)$ gilt: $|Pr(C(U_{l+1}) = 1) - Pr(C(G(x)) = 1)| \leq \frac{1}{n}$; mit $x \in \{0, 1\}^l$. G heißt schneller Extender, falls die Laufzeit von G exponentiell von der Eingabelänge abhängt; $G \in DTIME(2^{O(l)})$.

Der nun folgende Satz liefert uns eine hinreichende und notwendige Bedingung für die Existenz schneller NW-Typ-Pseudozufallszahlengeneratoren.

Satz 4.3: [Nis90, NW94] Für jede größenbeschränkte Funktion $l \leq s(l) \leq 2^l$ sind folgende Aussagen äquivalent:

(i) Für einige $c > 0$ können einige Funktionen in $EXPTIME$ nicht von Schaltkreisen der Größe $s(l^c)$ approximiert werden.

(ii) Für einige $c > 0$ existiert eine Funktion mit der Härte $s(l^c)$.

(iii) Für einige $c > 0$ existiert ein schneller $s(l^c)$ -Extender

$G : \{0, 1\}^l \rightarrow \{0, 1\}^{l+1}$

(iv) Für einige $c > 0$ existiert ein schneller NW-Typ-Pseudozufallszahlengenerator

$G : \{0, 1\}^l \rightarrow \{0, 1\}^{s(l)}$

Beweis:

Für den Äquivalenzbeweis zeigen wir folgenden Ringschluss $(i) \Rightarrow (ii) \Rightarrow (iv) \Rightarrow (iii) \Rightarrow (i)$

Bei dieser Argumentation gehen wir davon aus, dass $(s(l))^c \leq s(l^c)$ ist, andernfalls müsste $s(l^c)$ durch $s(l^c)^c$ ersetzt werden. (vgl. [NW94])

$(i) \Rightarrow (ii)$ Lemma 4.1

$(iv) \Rightarrow (iii)$ nach Definition 4.6

$(iii) \Rightarrow (i)$ Sei $G = \{G_1\}$ ein Extender wie in (iii). Betrachten wir das Problem "Ist y durch G berechenbar?". Dieses Problem kann durch Ausprobieren in Exponentialzeit gelöst werden, aber kein Schaltkreis der Größe $s(l^c)$ kann es approximieren, da dieser Schaltkreis sonst die Ausgabe von G von einer gleichverteilten Zufallsvariable unterscheiden könnte.

Falls G bijektiv ist, so ist klar, dass kein Schaltkreis der Größe $s(l^c)$ diese Sprache approximieren kann. Ist G nicht bijektiv, so nutzen wir folgende Aussage aus [BFNW91]: falls jede Funktion in $EXPTIME$ von Schaltkreisen der Größe $s(n)$ in $\frac{1}{n^2}$ approximiert werden kann, so kann auch jede Funktion in $EXPTIME$ von Schaltkreisen der Größe $s(n)p(n)$ berechnet werden, wobei $p(\cdot)$ ein Polynom ist.

$(ii) \Rightarrow (iv)$ Sei f eine Funktion in $EXPTIME$ mit der Härte $s(l^c)$. Wir konstruieren daraus nun einen schnelle NW-Typ-Pseudozufallszahlengenerator $G : \{0, 1\}^l \rightarrow \{0, 1\}^n$ mit $n = s(m^{\frac{c}{4}})$. Für jedes n sei A_n eine $n \times l$ Matrix über $\{0, 1\}$, wobei A ein $(\log n, \sqrt{l})$ -Design ist. Da $H_f(m) > n^2$ ist, folgt mit Lemma 4.1, dass $G_n(x) := f_{A_n}(x)$ ein NW-Typ-Pseudozufallszahlengenerator ist. Weiterhin ist aber f in $EXPTIME$, also ist $G = \{G_n\}$ ein schneller NW-Typ-Pseudozufallszahlengenerator.

◇

Wenn wir diese Aussage mit Satz 3.2 vergleichen, so stellen wir fest, dass Satz 4.3 zwar eine schwächere Formulierung ist, aber auch schwächere Voraussetzungen braucht. Einen weiteren Vergleich zwischen BWY- und NW-Typ-Generator wollen wir im folgendem Abschnitt vornehmen.

4.3 Vergleich: BMY-Typ und NW-Typ-Generator

Um beide Typen von Generatoren besser vergleichen zu können, wollen wir eine allgemeinere Definition einführen. (vergleiche u.a. [BOV05])

Definition 4.6: Eine Funktion $G : \{0, 1\}^l \rightarrow \{0, 1\}^m$ heißt (s, ε) -Pseudozufallszahlengenerator gegen Schaltkreise, falls $l < m$ ist, sowie für alle Schaltkreise $C : \{0, 1\}^m \rightarrow \{0, 1\}$ mit einer Größe von höchstens s gilt:
 $|Pr(C(G(U_l)) = 1) - Pr(C(U_m) = 1)| < \varepsilon$.

Mit dieser Definition können wir nun sowohl BMY-Typ, als auch NW-Typ-Generatoren neu fassen.

Definition 4.7: Eine Familie von Funktionen $G : \{G_m : \{0, 1\}^l \rightarrow \{0, 1\}^m\}_{m \in \mathbb{N}}$ heißt BMY-Typ-Generator mit Eingabelänge $l = l(m)$, falls G eine Familie von P-Funktionen ist, und für jede Konstante c sowie alle hinreichend große m , G_m ein $(m^c, \frac{1}{m^c})$ -Pseudozufallszahlengenerator ist.

Analog dazu definieren wir NW-Typ-Generatoren:

Definition 4.8: Eine Familie von Funktionen $G : \{G_m : \{0, 1\}^l \rightarrow \{0, 1\}^m\}_{m \in \mathbb{N}}$ heißt NW-Typ-Generator mit Eingabelänge $l = l(m)$, falls G in einer Zeit von $2^{O(l)}$ berechenbar, und G_m ein $(m^2, \frac{1}{m^2})$ -Pseudozufallszahlengenerator ist.

Schauen wir uns beide Typen von Generatoren an, so sieht man also deutlich, dass NW-Typ-Generatoren schwächer gegen Angriffe sind. Während BMY-Typ-Generatoren auch gegen Angreifer standhalten müssen, welche eine höhere Laufzeit haben als der Generator selbst, müssen NW-Typ-Generatoren nur Schaltkreise mit geringerer Laufzeit täuschen. Dies ist darauf zurückzuführen, dass NW-Typ-Generatoren nicht primär für die Kryptographie, als eher zur Derandomisierung probabilistischer Algorithmen gedacht waren.

Ein wichtiger Vorteil der NW-Typ-Generatoren ist es jedoch, dass sie in der Lage sind auch nichtdeterministische Schaltkreise zu täuschen (vergleiche auch [AK01]). Solche Generatoren mit einer Eingabelänge von $l = O(\log m)$ existieren, falls es Funktionen in $E = DTIME(2^{O(n)})$ gibt, welche nichtdeterministische Schaltkreise mit einer Komplexität von $2^{\Omega(n)}$ benötigen (siehe auch [MV99]). Für BMY-Typ-Generatoren lässt sich dies nicht erreichen, da ein nichtdeterministischer Schaltkreis bei Eingabe eines pseudozufälligen Strings ein entsprechendes Urbild raten und anschließend den Generator darauf anwenden kann, um dieses Urbild zu prüfen. Für NW-Typ-Generatoren ist jedoch eine größere Laufzeit erlaubt, als dieser Schaltkreis für die Überprüfung zu Verfügung hat.

Im Folgendem Kapitel werden wir solche Generatoren trotz ihrer vermeintlichen Schwäche (s.o.) für die Kryptographie nutzbar machen.

Kapitel 5

Anwendung von NW-Typ Generatoren in der Kryptographie

In diesem Kapitel werden wir betrachten, wie NW-Typ-Generatoren eingesetzt werden können, um möglichst gute, also sichere und effiziente, Beweissysteme zu erstellen.

Dazu werden wir zunächst den Begriff Beweissystem betrachten und definieren und uns dann einen Spezialfall eines Pseudozufallszahlengenerators ansehen, welchen wir abschließend einsetzen werden um ein weiteres Beweissystem zu konstruieren.

5.1 Zero-knowledge-Beweise und weitere Definitionen

Ein Beweissystem ist ein Protokoll, also eine festgelegte Abfolge von Nachrichten zwischen mehreren Parteien, bei dem eine Partei P (abgeleitet von prover), welche über unbeschränkte Komplexität verfügt und im Allgemeinen nicht vertrauenswürdig ist, eine zweite Partei V (abgeleitet von verifier), die wiederum polynominell zeitbeschränkt ist, von der Gültigkeit einer mathematischen Aussage überzeugen kann. Dieses Beweissystem heißt zero-knowledge-Beweissystem, falls V durch dieses ganze Verfahren über die Aussage selbst keine weiteren Informationen erhält, als dass sie gültig ist. Bei diesem Verfahren senden sich P und V eine Reihe von Nachrichten zu. Das Optimum wäre in diesem Zusammenhang die Verwendung von möglichst wenigen kurzen Nachrichten zwischen den Parteien P und V sowie die Einhaltung des zero-knowledge-Paradigmas. Eine Anwendung finden solche Protokolle zum Beispiel bei der Authentikation resp.

bei Signaturen. So ist es also zu vermeiden, dass der Signierschlüssel öffentlich bekannt wird, trotzdem soll der Signierer die Gültigkeit der Signatur beweisen können.

Einen weiteren Begriff welchen wir im Zusammenhang mit Beweissystemen finden, ist die Interaktivität. Wir wollen dies mit Hilfe von Turingmaschinen verdeutlichen. Eine interaktive Turingmaschine ist eine probabilistische Turingmaschine, welche über zwei zusätzliche Kommunikationsbänder verfügt. Eines dient der ausgehenden Kommunikation und nur Schreiboperationen sind erlaubt, das zweite wird als eingehende Kommunikation genutzt, auf ihm sind nur Leseoperationen gestattet. Ein interaktives Beweissystem ist dann ein Paar solcher interaktiven Turingmaschinen wobei das Band für die eingehende Kommunikation der einen Maschine das Band der ausgehenden Kommunikation der Anderen ist (siehe Abbildung 2).

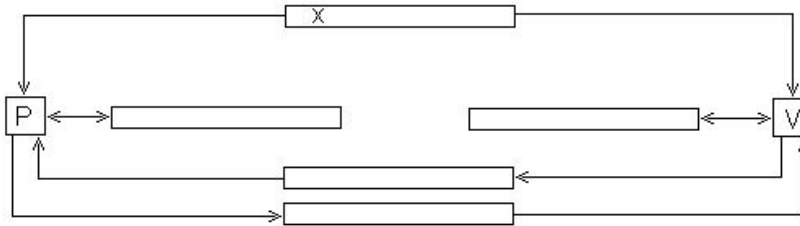


Abbildung 2

Definition 5.1: Ein geordnetes Paar interaktiver Turingmaschinen (P, V) heißt interaktives Beweissystem für eine Sprache $L \in NP$ falls folgende Bedingungen gelten:

- (i) Effizienz: Bei gemeinsamer Eingabe x ist die Anzahl und Länge der Nachrichten, welche zwischen P und V ausgetauscht werden durch ein Polynom $l(|x|)$ beschränkt und die Laufzeit von V ist polynominell beschränkt.
- (ii) Vollständigkeit: Für jedes $x \in L$ gilt:

$$Pr((P, V)(x) = 1) \geq \frac{2}{3}$$

Ist die Wahrscheinlichkeit gleich 1, so sprechen wir von perfekter Vollständigkeit.

- (iii) Korrektheit: Für jedes $x \notin L$ und für jeden Algorithmus P^* gilt:

$$Pr((P^*, V)(x) = 1) \leq \frac{1}{3}$$

Ist die Wahrscheinlichkeit gleich 0, so sprechen wir von perfekter Korrektheit.

Mit anderen Worten fordern wir, dass V für ein $x \in L$ mit hoher Wahrscheinlichkeit akzeptiert, für ein $x \notin L$ hingegen mit hoher Wahrscheinlichkeit und unabhängig vom Verhalten von P ablehnt.

Alternativ zu den Schranken $\frac{2}{3}$ und $\frac{1}{3}$ findet sich auch häufig die Forderung nach einer vernachlässigbaren Wahrscheinlichkeit. Eine Funktion $v(n)$ heißt dabei vernachlässigbar, falls sie langsamer wächst als jede Inverse eines Polynoms, d.h. für alle Polynome $p(\cdot)$ existiert ein n_0 , so dass für alle $n \geq n_0$ gilt: $v(n) < \frac{1}{p(n)}$. Existiert jedoch ein Polynom, so dass diese Ungleichung nicht erfüllt ist, so ist $v(n)$ nicht vernachlässigbar. Die Eigenschaft der Vollständigkeit ist also dann erreicht, falls für alle $x \in L$ und eine nichtvernachlässigbare Funktion $v(\cdot)$ gilt:

$$\Pr((P, V)(x) = 1) \geq \frac{1}{2} + v(|x|)$$

Analog kann für die Korrektheit also auch gefordert werden, dass die Wahrscheinlichkeit, dass es einer unehrlichen Partei P^* , also einer Partei welche sich nicht an das Protokoll hält, gelingt eine falsche Aussage zu beweisen, vernachlässigbar ist.

Betrachten wir nun die Frage der zero-knowledge-Eigenschaft. Es lassen sich grundsätzlich drei Arten unterscheiden. Zum einen die perfekte zero-knowledge Eigenschaft und zum anderen die beiden abgeschwächten Varianten statistische sowie berechenbare zero-knowledge-Eigenschaft.

Unter einer *Sicht* verstehen wir im Folgendem eine Zufallsvariable, welche alles umfasst was die Partei V während des Ablaufes eines Beweissystems an Informationen erhält. Fixieren wir die internen Münzwürfe beider Parteien, so enthält die zugehörige *Sicht* die vollständige Kommunikation zwischen P und V , die gemeinsame Eingabe sowie die internen Münzwürfe von V . Die Verteilung der *Sicht* hängt dabei von den internen Münzwürfen von P und V ab.

Definition 5.2:

Ein interaktives Beweissystem (P, V) für eine Sprache L hat die perfekte Zero-knowledge-Eigenschaft, falls ein probabilistischer Polynomialzeitalgorithmus A existiert, so dass für alle $x \in L$ gilt: die Zufallsvariablen $A(x)$ und *Sicht* sind identisch verteilt. Die Verteilung von $A(x)$ hängt von den internen Münzwürfen von A bei Eingabe von x ab.

(P, V) hat die statistische Zero-knowledge-Eigenschaft falls ein probabilistischer Polynomialzeitalgorithmus A existiert, so dass für alle $c > 0$ und alle hinreichend große $x \in L$ gilt:

$$\sum_{\alpha} |\Pr(A(x) = \alpha) - \Pr(\text{Sicht} = \alpha)| < \frac{1}{|x|^c}$$

(P, V) hat die berechenbare Zero-knowledge-Eigenschaft falls $A(x)$ und *Sicht* P-ununterscheidbar sind.

Betrachten wir uns dazu folgendes aus [GMW91] entnommenes Beispiel. Gegeben sei ein Graph $G := (K, E)$. Die Partei P möchte nun beweisen, dass G

3-färbbar ist und das sie diese Färbung $\phi : V \rightarrow \{1, 2, 3\}$ kennt. Dieses Protokoll betrachten wir dabei in seiner "physikalischen" Variante (siehe [GMW91, S. 713]), das bedeutet wir verwenden abschließbare Boxen statt eines Commitment-Schemas oder probabilistischen Verschlüsselungsfunktionen, da es uns hier lediglich darum geht, die grundlegenden Eigenschaften eines Zero-knowledge-Beweises zu erkennen.

Die folgenden Schritte werden $|E|^2$ -mal wiederholt.

(P1) P wählt zufällig eine Permutation $\pi \in_R S_3$ und gibt für jeden Knoten $i \in K$ $\pi(\phi(i))$ in eine Box B_i , verschließt sie und sendet alle B_j , $j \in K$ an V .

(V1) V wählt nun zufällig eine Kante $e = (u, v) \in_R E$ und sendet sie an P .

(P2) P sendet an V die Schlüssel zu den Boxen B_u und B_v .

(V2) V öffnet die beiden Boxen und überprüft ob sie unterschiedliche Farben enthalten. Falls nicht bricht V das Protokoll ab und akzeptiert nicht, andernfalls beginnt die nächste Runde.

V akzeptiert, falls alle $|E|^2$ Runden durchgeführt wurden.

Zunächst ist klar, dass dieses Protokoll effizient und perfekt vollständig ist. Ist der Graph nicht mit drei Farben färbbar, so gilt für jede Runde, dass V mit einer Wahrscheinlichkeit von mindestens $\frac{1}{|E|}$ stoppt. Da $|E|^2$ Runden absolviert werden, ist die Wahrscheinlichkeit das P V täuschen kann durch $(1 - \frac{1}{|E|})^{|E|^2}$, also rund $e^{-|E|}$, nach oben beschränkt. Damit ist dieses Verfahren also auch korrekt.

Es bleibt noch die Zero-knowledge-Eigenschaft. V erhält in jeder Runde lediglich ein Paar verschiedener zufällig gewählter Elemente aus $\{1, 2, 3\}$. Hier ist es wichtig, dass zu Beginn jeder Runde die Zahlen zufällig permutiert werden. Daher ist die Belegung eines Knotens in einer Runde unabhängig von seiner Belegung in einer anderen. Über die tatsächliche Färbung erhält V also keinerlei Informationen.

Ein Simulator A könnte eine entsprechende Kommunikation wie folgt erstellen. Da A auch V simuliert, weiß A zu Beginn jeder Runde bereits, welche Kante zur Überprüfung im Schritt (V1) gewählt wird. In die entsprechenden Boxen werden zwei unterschiedliche Farben gelegt, die restlichen Boxen werden mit beliebigen, nicht notwendigerweise verschiedenen, Farben gefüllt. Damit ist A einerseits effizient, andererseits ist die so entstandene Kommunikation nicht von einem echten Ablauf des Protokolls unterscheidbar.

Da diese Zero-knowledge-Eigenschaft jedoch eine sehr starke Forderung ist, welche zu unter Umständen erheblichen Aufwand führt, wollen wir eine abgeschwächte Form betrachten. Das dazu benutzte Konzept der Zeugenununterscheidbarkeit beruht auf der Arbeit von Feige und Shamir [FS90].

Zunächst wollen wir dazu den Begriff Zeuge klären.

Definition 5.3: Sei $W \subseteq \{0, 1\}^* \times \{0, 1\}^*$ eine Relation. Dann ist $W(x) := \{w | (x, w) \in W\}$ und $L(W) := \{x | \exists w \text{ mit } (x, w) \in W\}$. Falls gilt

$w \in W(x)$, so heißt w Zeuge für x . Falls $L = L(W)$ in Polynomialzeit bezüglich der ersten Eingabe entscheidbar, also in NP ist, so heißt W die zu L gehörende Zeugenrelation.

Sprachen in NP lassen sich damit also so charakterisieren, dass $L \in NP$ genau dann gilt, wenn es eine Zeugenrelation W_L gibt, so dass für alle $x \in L$ ein polynominal in der Länge beschränktes w mit $(x, w) \in W_L$ existiert, sowie für $x \notin L$ es kein solches w gibt.

Bei den nun folgenden Definitionen werden wir P einschränken. Hatte P bislang unbeschränkte Komplexität, fordern wir nun, dass P ein probabilistische Polynomialzeitalgorithmus ist. Damit P aber weiterhin einen Vorteil gegenüber einem polynominal zeitbeschränkten Beobachter oder Simulator hat, erhält P zur Eingabe x als zusätzliche Eingabe einen Zeugen w . Im obigen Beispiel der 3-Färbung konnte sich P eine Färbung ϕ noch selbst erstellen. Dies ist mit der Beschränkung nicht mehr möglich. Daher muss P nun eine Färbung als Zusatzeingabe erhalten.

Die Kernidee der folgenden Definition der Zeugenununterscheidbarkeit ist, dass es für einen effizienten Beobachter nicht möglich sein darf, einen Unterschied zwischen der Kommunikation zwischen P und V bei der Verwendung verschiedener Zeugen durch P festzustellen.

Definition 5.4: Ein interaktives Beweissystem (P, V) für eine Sprache L heißt zeugenununterscheidbar, falls für jeden nichtuniformen probabilistischen Polynomialzeitalgorithmus V' , jedes $x \in L$, alle $w_1, w_2 \in W(x)$ und alle Zusatzeingaben z an V' die Verteilung der Sichten auf V' nach einer Ausführung von $(P, V')(x, w_1, z)$ P -ununterscheidbar von der Verteilung der Sichten nach einer Ausführung von $(P, V')(x, w_2, z)$ für V' ist, wobei V' eine der beiden obigen Übertragungen, sowie die Werte (x, w_1, w_2, z) erhält.

In [DN04] ist angemerkt, dass die Zusatzeingabe z auch aus den beiden Zeugen w_1 und w_2 bestehen kann. Desweiteren folgt, dass jedes zero-knowledge-Protokoll zeugenununterscheidbar ist sowie die Aussage, dass Zeugenununterscheidbarkeit abgeschlossen bezüglich paralleler Komposition ist. (siehe [FS90])

Eine wichtige Rolle werden in den folgenden Abschnitten Zaps, [DN04] also ein interaktives "public coin" Beweissystem mit zwei Nachrichten welches Zeugenununterscheidbar ist, spielen.

Ein Zap ist nun ein Protokoll um nachzuweisen, dass ein x Element einer Sprache L in NP ist. Sei (P, V) ein perfekt vollständiges interaktives Beweissystem mit zwei Nachrichten. Die erste Nachricht von V nach P sei p , die zweite, von P nach V heiße π . Dabei müssen folgende zusätzliche Bedingungen gelten:

(i) public coins: Es existiert ein Polynom $k(\cdot)$, so dass die Nachrichten der ersten Runde, also von V an P , eine Verteilung von Strings der Länge $k(n)$

bilden, welche nur von der Länge von x , mit $|x| = n$ abhängt. Die Entscheidung, ob V akzeptiert, beruht nur auf einer P-Funktion welche von x , p und π abhängig ist.

(ii) Zeugenununterscheidbarkeit: Das Beweissystem P, V ist zeugenununterscheidbar. (vergleiche Definition 5.4)

Betrachten wir die Bedingung (i). Eine probabilistische Turingmaschine ist eine public coin Maschine, falls die internen Münzwürfe öffentlich bekannt gegeben werden. Da ein Zap nur zwei Nachrichten hat, muss V damit bereits nach dem ersten Schritt alle für seine Entscheidung bezüglich der Akzeptanz von x relevanten Münzwürfe offengelegt haben. Nach der Antwort von P dürfen keine das Ergebnis beeinflussenden Münzwürfe von V mehr vorgenommen werden.

Es lässt sich zeigen, dass für jede Sprache in NP Zaps existieren, falls Falltürpermutationen existieren. Falltürpermutationen, oder besser Hintertürpermutationen, sind bijektive längenerhaltende Einwegfunktionen mit einer Besonderheit. Generell ist es bei Einwegfunktionen schwierig, die Umkehrabbildung zu berechnen, resp. zu einem gegebenen Bild ein Urbild zu bestimmen. Dies ist aber bei Falltürpermutationen unter gewissen Umständen, eben bei Kenntnis der Falltür, doch effizient lösbar, jedoch darf die Veröffentlichung der Funktion keine Informationen über die Falltür, und damit die Möglichkeit zur Invertierung, bieten.

Definition 5.5: Eine bijektive längenerhaltende Einwegfunktion

$f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ heißt Falltürpermutation, falls ein Polynom $p(\cdot)$ und ein probabilistischer Polynomialzeitalgorithmus A existiert, so dass für jedes k es ein $t_k \in \{0, 1\}^*$ gibt mit $|t_k| \leq p(k)$ und für jedes $x \in \{0, 1\}^*$ ist $A(f(x), t_k) = y$ mit $f(y) = f(x)$.

Ein bekannter Kandidat für eine Falltürpermutation ist das RSA-System. (vergleiche Kapitel 6.3)

Ein weiteres Beispiel ist folgende von Blum und Williams vorgeschlagene und auf der Arbeit von Rabin basierende Funktion:

Sei $J = \{pq | p \neq q \text{ prim}, |p| = |q|, (p \equiv q \equiv 3) \text{ mod } 4\}$. Sei weiterhin $g_n : Z_n^* \rightarrow Z_n^*$ mit $Z_n^* := \{a | 0 \leq a \leq m - 1, ggT(a, m) = 1\}$, definiert als $g_n(x) := (x^2 \text{ mod } n)$. Dann ist die Funktionenfamilie $\{g_n\}_{n \in J}$ eine Falltürpermutation. Die Falltürinformation für $n = pq$ ist dabei das Paar $t_n = (p, q)$. (siehe auch [GB01])

5.2 Hitting-set-Generatoren

Wir wollen nun die Art von Pseudozufallszahlengeneratoren definieren, welche wir in einem Beweissystem verwenden werden.

Unter einem Hitting-Set-Generator [BOV05] verstehen wir einen polynomiell zeitbeschränkten deterministischen Algorithmus, welcher eine Menge von Bitstrings sowie die Größe des zu täuschenden Schaltkreises erhält und wiederum eine Menge von Bitstrings ausgibt. Sie können als eine schwache Variante von NW-Typ-Pseudozufallszahlengeneratoren verstanden werden und wurden wie diese für die Derandomisierung probabilistischer Algorithmen konstruiert.

Zunächst wollen wir den Begriff "hitting set" klären. Ein "hitting set" in $\{0, 1\}^n$ mit einem Schwellenwert $\delta(n)$ für einen deterministischen Schaltkreis C der Größe $s(n)$ ist eine Teilmenge H von $\{0, 1\}^n$ so dass für C bei Eingaben der Länge n und einer Ausgabe der Länge 1 gilt: falls

$$Pr_{x \in \{0,1\}^n}(C(x) = 1) \geq \delta(n)$$

ist, so gilt $\exists x \in H : C(x) = 1$. Mit anderen Worten, falls C hinreichend viele x akzeptiert, so ist mindestens ein entsprechender Wert auch in H .

Nachfolgender Satz zeigt, dass eine Möglichkeit existiert, wie man ausgehend von einer Wahrheitstabelle einer booleschen Funktion eine solche Hitting-set konstruieren kann.

Unter einem single-valued(SV)-nichtdeterministischen Schaltkreis verstehen wir einen nichtdeterministischen Schaltkreis welcher zusätzlich zum eigentlichen Ausgabebit noch ein zweites, flag genanntes, Bit ausgibt. Ist dieses Bit gesetzt, so signalisiert es, dass bei Eingabe eines Wertes x der korrekte Wert $f(x)$ berechnet wurde. Für jedes x muss dieser Schaltkreis dabei einen Berechnungsweg haben, welcher dieses Bit setzt.

Satz 5.1: [MV99] Für jedes $\epsilon > 0$ und $q \geq 1$ existiert ein Polynomialzeitalgorithmus P , so dass gilt:

Sei $f : \{0, 1\}^m \rightarrow \{0, 1\}$ eine Funktion welche für fast alle m nicht von einem SV-nichtdeterministischen Schaltkreis mit einer Größe kleiner als $2^{\epsilon m}$ berechnet werden kann. Dann existieren Konstanten $\delta = \delta(\epsilon) < \epsilon$ und $k > q$ derart, dass bei einer Eingabe der Wahrheitstabelle von $f : \{0, 1\}^{kl} \rightarrow \{0, 1\}$ P ein Hitting-set $H_f \subseteq \{0, 1\}^n$ für co-nichtdeterministische Schaltkreise der Größe n^q mit einem Schwellenwert von $1 - 2^{-n+n^\delta}$, mit $n = (2l)2^{2l}$, ausgibt.

Der vollständige Beweis findet sich in [MV04].

Wir können nun Hitting-set-Generatoren wie folgt definieren.

Definition 5.6: Sei $H(1^m, 1^s)$ ein in m und s polynomiell zeitbeschränkter deterministischer Algorithmus, welcher eine Menge von Bit-strings der Länge m ausgibt. Dann ist H ein ϵ -Hitting-set-Generator gegen Schaltkreise, falls für alle $m, s \in N$ und Schaltkreise $C : \{0, 1\}^m \rightarrow \{0, 1\}$ mit einer Größe von höchstens s gilt:

$$Pr(C(U_m) = 1) > \epsilon \Rightarrow \exists y \in H(1^m, 1^s) \text{ mit } C(y) = 1 .$$

Eine Möglichkeit einen Hitting-set-Generator zu erhalten ist, wie bereits angesprochen, die folgende. Sei $G : \{0, 1\}^l \rightarrow \{0, 1\}^m$ ein Pseudozufallszahlengenerator, welcher Schaltkreise der Größe s täuscht. Des Weiteren sei die Laufzeit von G durch ein Polynom abhängig von s und m beschränkt. Wir können daraus einen Hitting-set-Generator H_G erhalten, indem wir als Ausgabe von H_G die Menge aller Ausgaben von G über alle Eingaben $x \in \{0, 1\}^l$ setzen.

Insbesondere $\frac{1}{2}$ -Hitting-set-Generatoren sind im Folgenden von Bedeutung.

5.3 Anwendung

Wir werden nun Hitting-set-Generatoren nutzen, um sogenannte NP-Beweissysteme zu erzeugen.

Definition 5.7: Ein NP-Beweissystem ist ein interaktives Beweissystem, welches aus nur einer Nachricht von P nach V besteht, wobei V ein Polynomialzeit-Algorithmus ist, und welches die Eigenschaften der perfekten Korrektheit sowie der perfekten Vollständigkeit hat.

Satz 5.2: [BOV05] Falls ein effizienter $\frac{1}{2}$ -Hitting-set-Generator H gegen Co-nichtdeterministische Schaltkreise, sowie Falltürpermutationen existieren, dann hat jede Sprache in NP ein zeugenununterscheidbares NP-Beweissystem

Beweis: (vergleiche [BOV05]) Um diese Aussage zu beweisen, werden wir die Zaps für Sprachen aus NP in zeugenununterscheidbare NP-Beweissysteme umwandeln, indem wir einen Zap mit Hilfe eines Hitting-set-Generators derandomisieren.

Sei L also eine NP-Sprache mit einer Zeugenrelation W_L und sei (P, V) ein Zap für L . Die erste Nachricht von V an P, bestehend aus einer Zufallszahl, heiße p ; die zweite von P an V heiße π . Sei $l(n)$ die Länge von p bei einer zu beweisenden Aussage der Länge n . Sei weiterhin $x \in \{0, 1\}^n \setminus L$. Wir nennen $p \in \{0, 1\}^{l(n)}$ ausgeschlossen in Abhängigkeit von x , falls keine Nachricht π existiert, für die gilt, dass V das Tupel (x, p, π) akzeptiert. Da (P, V) ein Zap ist, gilt dass für jedes $x \in \{0, 1\}^n \setminus L$ die Wahrscheinlichkeit, dass $p \in_U \{0, 1\}^{l(n)}$ in Abhängigkeit von x ausgeschlossen sehr hoch, insbesondere größer als $\frac{1}{2}$ ist.

Sei $q(n)$ ein Polynom, welches die Laufzeit eines ehrlichen Zap-Verifiers, also einer Partei V, welche sich an das vorgegebene Protokoll hält, bei einer Aussage der Länge n nach oben beschränkt. Für jedes $x \in \{0, 1\}^n \setminus L$ existiert ein Co-nichtdeterministischer Schaltkreis C_x mit einer Größe von höchstens $s(n) < q(n)^2$, welcher genau dann 1 ausgibt, falls ein String p in Abhängigkeit von x ausgeschlossen ist. Dazu benutzen wir, dass die Zeit um die Ausschließbarkeit eines Strings p zu verifizieren nur von der Laufzeit des ehrlichen Zap-Verifizierers abhängt.

Bei Eingabe p gibt also C_x 1 aus, falls keine Nachricht π existiert, so dass das Tupel (x, p, π) akzeptiert wird, und sonst 0. Wie bereits angesprochen gilt

$$Pr(C_x(U_{l(n)}) = 1) > \frac{1}{2}$$

Da H ein $\frac{1}{2}$ -Hitting-set-Generator gegen Co-nichtdeterministische Schaltkreise ist, gilt: für jedes $x \in \{0, 1\}^n \setminus L$ existiert ein $p \in H(1^{l(n)}, 1^{p(n)})$, so dass $C_x(p) = 1$ ist. D.h. für jedes $x \in \{0, 1\}^n \setminus L$ gibt es einen String $p \in H(1^{l(n)}, 1^{p(n)})$, so dass p ausgeschlossen bezüglich x ist. Damit ergibt sich folgende Konstruktion:

Eingabe: gemeinsame Eingabe $x \in \{0, 1\}^n$, sowie eine zusätzliche Eingabe w für P , so dass $(x, w) \in W_L$ ist.

Nachricht von P an V :

1. Berechne $(p_1, \dots, p_m) := H(1^{l(n)}, 1^{p(n)})$.
2. Benutze w und den P-Algorithmus des Zaps, um π_i , also die Antwort auf $p_i \forall i \in \{1, \dots, m\}$ zu berechnen.
3. Sende an V das Tupel (π_1, \dots, π_m) .

Verifikationstest von V :

1. Berechne $(p_1, \dots, p_m) := H(1^{l(n)}, 1^{p(n)})$.
2. Wende den Zap-Verifizieralgorithmus auf $(x, p_i, \pi_i) \forall i \in \{1, \dots, m\}$ an.
3. Akzeptiere, falls alle Tests in Schritt 2 erfolgreich waren.

Diese Konstruktion ist perfekt vollständig, da Zaps bereits die Eigenschaft der perfekten Vollständigkeit haben. Um Satz 5.2 zu beweisen, müssen wir noch die Zeugenununterscheidbarkeit, sowie die perfekte Korrektheit zeigen.

Sei $x \in \{0, 1\}^n \setminus L$. Da H ein Hitting-set-Generator ist, existiert ein $p_i \in H(1^{l(n)}, 1^{p(n)})$ welches ausgeschlossen in Abhängigkeit von x ist. Damit kann kein String π_i den Zap-Verifizierer zur Akzeptanz eines Tupels (x, p_i, π_i) bringen, und also kann auch kein String (π_1, \dots, π_m) vom Verifizierer unserer Konstruktion akzeptiert werden. Damit folgt die perfekte Korrektheit.

Da der P-Algorithmus lediglich m -mal für m verschiedene Nachrichten den P-Algorithmus des Zaps benutzt, selbiger aber bezüglich der Zeugenununterscheidbarkeit abgeschlossen gegen jede mögliche Strategie des Verifizierers, sowie der parallelen Komposition ist, folgt auch die Zeugenununterscheidbarkeit unserer Konstruktion.

◇

Wir erhalten damit ein zeugenununterscheidbares NP-Beweissystem mit einer Nachricht und ohne dass ein gemeinsamer zufälliger Wert benutzt wird. Für ein Zero-knowledge Beweissystem ist dies im Allgemeinen nicht möglich- Goldreich und Oren bewiesen [GO94] dass mindestens drei Nachrichten nötig sind, um das Zero-knowledge-Paradigma für Sprachen außerhalb von BPP zu erreichen.

Eine weitere Anwendung der Hitting-set-Generatoren in der Kryptographie lässt sich bei der Konstruktion eines nicht interaktiven Bit-Commitment-Schemas, also eines Protokolls, bei dem eine Partei ein Bit beweisbar festlegt ohne es zu veröffentlichen, finden. Dabei wird lediglich eine beliebige Einwegfunktion benötigt. (vergleiche [BOV05])

Kapitel 6

Praktische Aspekte

In diesem Kapitel wollen wir nach den bisher eher theoretischen Überlegungen einige praktische Aspekte betrachten. Zunächst werden wir uns anschauen ob und wie die Hinzunahme von zumindest teilweise öffentlich bekannten Eingaben Änderungen bei den Generatoren respektive der Sicherheit ebendieser zur Folge haben. Danach werden wir mit AIS20 ([Sch99]) einen für Deutschland verbindlichen Leitfaden für die Entwicklung und Überprüfung von Pseudozufallszahlengeneratoren skizzieren.

6.1 Pseudozufallszahlengeneratoren mit öffentlich bekannten Eingaben

Durch die vielfältige Verwendung von kryptographischen Funktionen, welche zufällige bzw. pseudozufällige Zahlen benötigen, an Stellen an denen eine vollständige Privatsphäre nicht gewährleistet ist, kann es notwendig werden Pseudozufallszahlengeneratoren auch von Eingabebits abhängig zu machen, welche möglicherweise auch von anderen eingesehen werden können. Diese Bits können dabei hilfreich sein Angriffe zu verhindern, bei denen im Voraus versucht wird den Bildraum der verwendeten Funktion vollständig zu berechnen.

Nehmen wir beispielsweise an, wir hätten einen Generator, welcher ausgehend von n -Bit langen Startwerten Pseudozufallszahlen der Länge $4n$ erzeugt. Eine mögliche Strategie eines Angreifers könnte es dann sein, alle 2^n möglichen Bilder im Voraus zu berechnen um auf Grundlage dieses Datenmaterials eine Unterscheidung vornehmen zu können. Wird nun die Funktion zusätzlich noch von einer n -Bit langen öffentlich bekannten Eingabe abhängig, erhöht sich der Aufwand des Angreifers bei dieser Strategie damit um den Faktor 2^n .

Diese Eingabebits werden wir im folgenden auch als "public input" resp. öffentliche Bits bezeichnen. Dem gegenüber steht der "privat input" resp. die

privaten Bits, also diejenigen Bits der Eingabe, welche aus einer gesicherten Quelle stammen und nicht von außen einsehbar sind.

Damit einhergehend verändert sich auch der Sicherheitsparameter der benutzten Einwegfunktionen. War der Sicherheitsparameter bislang die gesamten Eingabelänge und die Sicherheitsfunktion abhängig von der Erfolgswahrscheinlichkeit die ein Algorithmus hat um bei gegebenem Bild ein Urbild der Einwegfunktionen zu berechnen, hängt der Sicherheitsparameter jetzt nur noch von der Anzahl der privaten Bits ab.

Definition 6.1: Sei $f := \{f_n : \{0, 1\}^{p(n)} \times \{0, 1\}^n \rightarrow \{0, 1\}^{l(n)}\}$ eine Familie von P-Funktionen wobei der erste Parameter der $p(n)$ Bit lange öffentliche Anteil der Eingabe, der zweite, n Bit lange, der private Anteil ist. Es gelte weiterhin für alle Polynome $q(\cdot)$ und alle Polynomialzeitalgorithmen $A : \{0, 1\}^{l(n)} \rightarrow \{0, 1\}^{p(n)} \times \{0, 1\}^n$, dass

$$\Pr(f(A(f(X))) = f(X)) \leq \frac{1}{q(n)}$$

ist, für hinreichend große n mit $X \sim U_{p(n) \times n}$. Dann heißt f Einwegfunktion mit public input.

Als Sicherheitsfunktion $s(\cdot)$, in Anlehnung an die $s(n)$ -sicheren Einwegfunktionen, definieren wir dann $s(n) := n$. Analog dazu lassen sich nun auch alle weiteren Definitionen aus den Kapiteln 2 und 3 anpassen.

Um den Unterschied zwischen der Definition 2.5 und 6.1 deutlich zu machen, betrachten wir folgendes Problem:

Teilmengensummenproblem (subset sum problem)

Sei $a \in \{0, 1\}^n$ und $B = (b_1, \dots, b_n)$ eine Matrix mit $B \in \{0, 1\}^{n \times n}$ und $\langle \cdot, \cdot \rangle$ bezeichne das innere Produkt zweier Vektoren. Sei weiterhin

$$f(a, B) := \left\langle \sum_{i=1}^n a_i b_i, B \right\rangle$$

, wobei die binären Vektoren b_i als Integer interpretiert werden. Die Summe lässt sich in $n^{O(1)}$ berechnen, aber es ist kein effizienter Algorithmus bekannt, welcher bei gegebener $\sum_{i=1}^n a_i b_i$ und gegebenem B ein $a' \in \{0, 1\}^n$ berechnen könnte, für das gilt: $\sum_{i=1}^n a_i b_i = \sum_{i=1}^n a'_i b_i$, falls $a \in U_n$ und $B \in U_{n \times n}$ ist.

Somit kann man also, basierend auf dem Teilmengensummenproblem, eine Einwegfunktion definieren. Für die Sicherheitsfunktion $s(\cdot)$, resp. den Sicherheitsparameter wie sie in Definition 2.6 festgelegt sind, ist es dann allerdings unerheblich, ob der Angreifer die Matrix B kennt, oder nicht. In der Praxis ist dieser Unterschied aber sehr wohl von Bedeutung, so dass die Entscheidung, $s(\cdot)$ nur von der Länge der privaten Eingabe abhängig zu machen relevant ist.

Der entscheidende Punkt ist dabei also, dass die Definitionen 2.6, 2.7 sowie 3.1 um eine mögliche Zusatzeingabe ergänzt werden, welche jedoch keinen Einfluss auf die Sicherheit hat.

Aufbauend auf dieser Definition 6.1 können wir dann auch wieder alle in den Kapiteln 2 und 3 verwendeten Definitionen und Konstruktionen übertragen. Da dies aber zu keinen wesentlichen Unterschieden führt, sei darauf an dieser Stelle verzichtet.

Der entscheidende Punkt ist dabei also, dass diese Definitionen um eine mögliche Zusatzeingabe ergänzt werden, welche jedoch keinen Einfluss auf die Sicherheit hat.

6.2 Situation in Deutschland

Vom BSI (Bundesamt für Sicherheit in der Informationstechnik) wurde 1999 ein verbindlicher Anwendungshinweis (AIS20 [Sch99]) herausgegeben, welcher es Entwicklern erlaubt, sich die Güte ihres Pseudozufallszahlengenerators zertifizieren zu lassen. Thema dieser Arbeit sind die deterministischen Generatoren. Eingeteilt wurden sie in dieser Ausarbeitung in vier verschiedenen Funktionalitätsklassen. Entscheidend für diese Einteilung ist dabei nicht die konkrete Umsetzung, sondern lediglich die theoretische Sicherheit sowie der in der Praxis avisierte Anwendungszweck. Innerhalb einer solchen Klasse wird noch einmal nach Mechanismenstärke unterschieden.

Diese Art der Einteilung wurde damit begründet, dass zum einen viele Resultate über Pseudozufallszahlengeneratoren asymptotischer Natur sind, dies betrifft damit auch die in den vorangegangenen Kapiteln getroffenen Aussagen, zum anderen können Generatoren auch an Stellen eingesetzt werden, an denen geringere Anforderungen an die Sicherheit gestellt werden.

Für K1-Generatoren ist es lediglich erforderlich, dass für eine aus den erzeugten Pseudozufallszahlen gebildete Folge von M Vektoren, welche wiederum aus c Pseudozufallszahlen bestehen, mit "hoher Wahrscheinlichkeit", d.h. $1 - \varepsilon$, die Vektoren paarweise voneinander verschieden sind. Es muss also gelten:

$$Pr((r_{ic}, \dots, r_{ic+(c-1)}) = (r_{jc}, \dots, r_{jc+(c-1)})) < \varepsilon$$

für alle $i \neq j$ und $i, j < \frac{M}{c}$. Die Klasse-K1-Generatoren dienen lediglich dazu einfache Angriffe wie zum Beispiel Replay-Attacken zu verhindern, so dass bereits einfachste Generatoren genügen. Gilt dabei $\varepsilon = 0$ so wird der Mechanismus als hoch, bei $\frac{M^2}{c^2\varepsilon} > 2^{32}$ und $\varepsilon < 2^{-12}$ als mittel u.s.w., eingestuft.

Bei der Klasse K2 wird gefordert, dass die vom Generator erzeugten Zahlen eine Reihe von statistischen Tests bestehen, welche in Abschnitt 6.4. aufgeführt

sind. Ziel dieser Tests ist es eine statistische Ununterscheidbarkeit der Pseudozufallszahlen von echten Zufallszahlen zu erreichen. Insbesondere sollen Angriffe, welche auf statistische Unregelmäßigkeiten beruhen, damit ausgeschlossen werden. Bereits ein linearer Kongruenzgenerator kann diese Tests bei geeigneter Wahl der Parameter bestehen.

Die Klasse K3 bilden Generatoren, welche es einem Angreifer praktisch unmöglich machen, ausgehend von einer bekannte Teilfolge (r_i, \dots, r_{i+j}) mit $i+j < M$, die Zahlen r_{i-1} und r_{i+j+1} zu bestimmen. Der Vorteil eines Angreifers welcher die Teilfolge kennt, gegenüber einem dem sie nicht vorliegt, muss vernachlässigbar sein. Dabei wird vorausgesetzt, dass die inneren Zustände des Generators nicht bekannt sind und auch nicht aus obiger Teilfolge berechnet werden können. Des weiteren muss ein solcher Generator auch die Tests für die Klasse K2 bestehen. Als Anwendung wird die Erzeugung von Signaturschlüsselpaaren, Paddingbits oder Schlüssel für symmetrische Verschlüsselungsverfahren angegeben.

Genauere Angaben darüber was in diesem Zusammenhang "praktisch unmöglich" bedeutet, wurden in [Sch99] mit dem Hinweis darauf, dass sich solche Forderungen für einen konkreten Generator in der Praxis nur äußerst schwierig nachweisen lassen, nicht gegeben.

Für K4-Generatoren gelten die gleichen Anforderungen wie für K3 jedoch unter der zusätzlichen Annahme, dass die inneren Zustände des Generators bekannt sein dürfen.

6.3 Beispiele für Generatoren

In diesem Abschnitt wollen wir einige Beispiele für Pseudozufallszahlengeneratoren betrachten. Dabei werden wir auch die mögliche Einteilung in die in Abschnitt 6.2 angesprochenen Klassen berücksichtigen.

1.) Der lineare Kongruenzgenerator:

Seien $\alpha, \beta, m \in \mathbb{Z}$ mit $\alpha, \beta < m$. Die Gleichung $y = \alpha x + \beta \text{ mod } m$ heißt linearer Kongruenzgenerator mit Eingabe x . In den meisten Anwendungsfällen wird dabei, ausgehend von einer echten Zufallszahl $x_0 := x$, eine ganze Folge $\{x_i\}_{i \in \mathbb{N}}$ mittels der Vorschrift $x_i = \alpha x_{i-1} + \beta \text{ mod } m$; $i > 0$ erzeugt. Wie man leicht sieht, ist diese Folge periodisch mit einer maximalen Periode kleiner gleich m . Häufig wird dabei lediglich das höchstwertige Bit der x_i verwendet.

Bei der richtigen Wahl der Parameter α, β und m kann ein solcher Generator zumindest die Anforderungen für die Klasse K2 bestehen, für kryptographisch sensible Zwecke ist ein solcher Generator jedoch nicht geeignet, da schon bei Kenntnis einer relativ kurzen Sequenz der $\{x_i\}$ sich die Parameter α, β und m bestimmen lassen, beziehungsweise, falls nur jeweils ein Bit der $\{x_i\}$ ausgegeben

wird, sich ein Folglied $x_j \in \{x_i\}$ ermitteln lässt. Damit aber kann die gesamte Ausgabe effizient berechnet werden.

2.) Lineares Schieberegister:

Sei $p : \{0, 1\}^n \rightarrow \{0, 1\}$ eine Funktion mit $p(x) := \sum_{j=0}^{n-1} a_j x^j$. Ausgehend von einem Startwert $s \in \{0, 1\}^n \setminus \{0\}^n$ mit $s = (s_0, \dots, s_{n-1})$ setzen wir die Ausgabebits wie folgt: $x_i := s_i$ für $0 \leq i < n$ und $x_j := p(x_{j-n}, \dots, x_{j-1})$ für $j \geq n$.

Ist das Polynom $p(\cdot)$ bekannt, so benötigt man maximal n Bits der Ausgabe um den gesamten Ausgabestring zu berechnen. Aber auch wenn dies nicht der Fall ist, kann nach circa $2n$ Bits mit Hilfe des Berlekamp-Massey-Algorithmus die Sequenz berechnet werden. Somit zählen diese Generatoren nicht zur Klasse K3. Bei hinreichend großem n können jedoch zumindest die K2-Tests bestanden werden.

3.) Rekursiver Aufruf eines symmetrischen Verschlüsselungsverfahrens:

Wir benutzen dazu eine symmetrischen Blockchiffre wie DES, 3DES oder IDEA. Zuerst wird ein Tupel (r_0, k) bestehend aus einer Nachricht r_0 aus dem Klartextraum der entsprechenden Chiffre und einem Schlüssel k , zufällig generiert. Der Schlüssel k bleibt dabei während des gesamten folgenden Verfahrens konstant. Die weiteren Werte r_i berechnen sich dann wie folgt:

$r_i := (ENC(r_{i-1}, k))$, wobei ENC eines der oben genannten Verschlüsselungsverfahren ist.

Die statistischen Eigenschaften der auf diese Weise erzeugten Pseudozufallszahlen sind ausreichend um die K2-Tests zu bestehen. Ein Angriff auf dieses Verfahren ohne die Kenntnis des inneren Zustandes, hier also die Kenntnis von k , ist mit einem Angriff auf das Verschlüsselungsverfahren bei bekanntem Klartext gleichzusetzen. Ist k hingegen bekannt, so ist das Problem trivial. Somit gehört dieses Verfahren zum K3-Typ-Generator. Wird als Verschlüsselungsverfahren DES genutzt, so gilt aufgrund der mit nur 56 Bit relativ kurzen Schlüssellänge, lediglich die Mechanismusstärke "mittel", für Verfahren wie 3DES und IDEA wird sie als "hoch" angegeben.

4.) RSA-Generator:

Seien $p \neq q$ Primzahlen und $N := pq$. Es bezeichne ϕ die Eulerfunktion, also $\phi(n) = \|Z_n^*\|$ und es sei $e \in_U \{0, \dots, \phi(N)\}$ mit $ggT(e, \phi(N)) = 1$. Bei diesem Generator wenden wir, ausgehend von einem $x_0 := s \in_U Z_N$, wiederholt den RSA-Verschlüsselungsalgorithmus an, $x_i := RSA(x_{i-1}) = x_{i-1}^e \bmod N$ für $i > 0$ und geben dann $r := (r_1, \dots, r_M)$ mit $r_i := x_i \bmod 2$ für $i > 0$ aus.

Aufgrund der Eigenschaften des RSA-Algorithmus erfüllt diese Art von Generatoren sowohl die K3-, als auch K4-Eigenschaften.

Insbesondere der RSA-Generator erfüllt höchste Sicherheitsanforderungen und zählt zu den BMY-Typ-Generatoren. (vergleiche Definition 3.1)

6.4 Statistische Tests

Betrachten wir nun einige weitere statistische Testmethoden, um die Güte der erzeugten Zahlen zu überprüfen. Alle diese Tests muss nach [Sch99] ein Generator ab der Klasse K2 bestehen. Begutachtet wird dabei eine Bitfolge von 20000 erzeugten Bits $b = (b_1, \dots, b_{20000})$.

Das Ziel dieser Tests ist es, dass die Generatoren Angriffe abwehren können, welche auf statistischen Analysen der erzeugten Zahlen beruhen.

1.) Monobittest: Hierbei wird die Anzahl der 1 und 0 getestet. Sei $X := \sum_{j=1}^{20000} b_j$. Für den so ermittelte Wert muss gelten: $9654 < X < 10346$.

2.) Pokertest: Sei $c_j := 8b_{4j-3} + 4b_{4j-2} + b_{4j}$ für $j \in \{1, \dots, 5000\}$, sowie $f(i) := |\{j : c_j = i\}|$ und $Y := \frac{16}{5000}(\sum_{i=0}^{15} f(i)^2) - 5000$. Die Folge b besteht den Test falls $1,03 < Y < 57,4$ gilt.

3.) Runttest: Als Run wird eine maximale Teilfolge bezeichnet, welche ausschließlich aus Nullen oder aus Einsen besteht. Null- und Einsruns werden dabei getrennt gewertet. Der String (111001001) hat also 2 Einsruns der Länge 1, einen der Länge 3 und 2 Nullruns der Länge 2. Es dürfen dabei beispielsweise in b Runs der Länge 1 zwischen 2267 und 2733 mal, der Länge 2 1079-1421 mal, u.s.w., vorkommen. Runs mit einer Länge größer als 5 dürfen nur 90 bis 233 mal vorhanden sein, Runs mit Längen von 34 und mehr, sogenannte "Longruns", dürfen hingegen überhaupt nicht vorkommen.

4.) Autokorrelationstest: Sei $Z_i := \sum_{j=1}^{5000} (b_j \oplus b_{j+i})$ für $i \in \{1, \dots, 5000\}$. Die Folge b besteht den Test, falls $2326 < Z_i < 2674$ für alle $i \in \{1, \dots, 5000\}$ gilt.

Eine Sequenz von echten Zufallsbits würde jeden dieser Tests mit einer Wahrscheinlichkeit von mindestens $1 - 10^{-6}$ bestehen (siehe [Sch99]). Nichtsdestoweniger bleibt festzustellen, dass für kryptographisch sensible Zwecke das Bestehen dieser Tests nicht hinreichend ist, so dass die zusätzlichen Anforderungen, welche an K3 und K4 Generatoren gestellt werden, Beachtung finden müssen.

Kapitel 7

Anhang

7.1 Schlusswort

Zunächst noch einige kurze Bemerkungen zur verwendeten Literatur. Insbesondere das zweite und dritte Kapitel baut auf einer Arbeit von O. Goldreich [Gol91] auf. Analoge Definitionen und Aussagen sind auch in anderen Arbeiten wie zum Beispiel der auf dieser Grundlage basierenden Arbeit [Gol05] oder in [Lub96] zu finden. Jedoch waren diese Quellen häufig zu speziell, in anderen Bereichen wiederum zu allgemein formuliert, so dass ich mich für [Gol91], obschon in dieser Form nicht veröffentlicht, als Hauptgrundlage entschieden habe.

Wir haben gesehen, dass es viele Möglichkeiten gibt Pseudozufallszahlengeneratoren zu konstruieren. Welche Art von Generatoren man verwendet, hängt wiederum stark vom anvisierten Verwendungszweck ab. Dort wo höchste Sicherheitsanforderungen gelten ist der BMY-Typ-Generator vorzuziehen. Aber auch NW-Typ-Generatoren haben ihre Anwendbarkeit in der Kryptographie gezeigt (vergleiche Kapitel 5).

Darüberhinaus können aber auch schwächere Generatoren verwendet werden (vergleiche Kapitel 6) um beispielsweise Replay-Angriffe zu verhindern. Der Einsatz solcher Generatoren in sensiblen Bereichen kann hingegen ein ernsthaftes Sicherheitsrisiko darstellen.

Ebenfalls wichtig für die Verwendung von Pseudozufallszahlengeneratoren in der Praxis ist die Effizienz der Algorithmen, da häufig sowohl Zeit- als auch Platzbeschränkungen vorgegeben sind. So sind zum Beispiel Chipkarten auf möglichst effiziente, zugleich aber sichere Verfahren angewiesen.

Betrachten wir die Generorentypen so stellt man aber fest, das sich deren Sicherheit nicht abschließend beweisen lässt. BMY-Typ-Generatoren benötigen die Annahme der Existenz von Einwegfunktionen als Voraussetzung. Aber auch

die Existenz von NW-Typ-Generatoren, gleichwohl sie von schwächeren Voraussetzungen ausgehen, ist nur unter gewissen Annahmen (vergleiche Abschnitt 4.3) bewiesen. In den vergangenen Jahren konnten jedoch die benötigten Voraussetzungen sukzessive verringert werden, und es steht zu erwarten, dass auch in Zukunft weitere Verbesserungen, auch mit Hinblick auf die Effizienz der Generatoren, hinzukommen werden.

Literaturverzeichnis

- [AGS03] AKAVIA, A. ; GOLDWASSER, S. ; SAFRA, S.: Proving Hard-CorePredicates Using List Decoding. In: *Proceedings of the 44th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society Press, 2003, S. 146–157
- [AK01] ARVIND, V. ; KÖBLER, J.: On pseudorandomness and resource-bounded measure. In: *Theoretical Computer Science* 255(1-2) (2001), S. 205–221
- [BFNW91] BABAI, L. ; FORTNOW, L. ; NISAN, N. ; WIGDERSON, A.: BPP has weak subexponential simulations unless EXPTIME has publishable proofs. In: *proceedings of structures in complexity theory* (1991)
- [BM84] BLUM, M. ; MICALI, S.: How to generate cryptographically strong sequences of pseudo-random bits. In: *SIAM Journal on Computing* 13(4) (1984), S. 850–864
- [BOV05] BARAK, B. ; ONG, S. J. ; VADHAN, S.: Derandomization in Cryptography. In: *Electronic Colloquium on Computational Complexity* 114 (2005)
- [Bra99] BRANDT, S.: *Data Analysis*. Springer Verlag, 1999
- [CW79] CARTER, L. ; WEGMAN, M.: Universal classes of hash functions. In: *J. Computer System Sci.* 18 (1979), S. 143–154
- [DN04] DWORK, C. ; NAOR, M.: *Zaps and Their Applications*. 2004. – <http://www.wisdom.weizmann.ac.il/naor/PAPERS/zap.pdf>
- [FS90] FEIGE, U. ; SHAMIR, A.: Witness indistinguishable and witness hiding protocols. In: *Protocols proc. 22nd ACM Symposium on the Theory of Computing*, 1990, S. 415–426
- [GB01] GOLDWASSER, S. ; BELLARE, M.: *Lecture Notes on Cryptography*. (2001)
- [GL89] GOLDREICH, O. ; LEVIN, L.: A Hard-Core Predicate for Every One-Way Function. In: *Symposium on Theory of Computation* (1989)

- [GMR89] GOLDWASSER, S. ; MICALI, S. ; RACKOFF, C.: The knowledge complexity of interactive proof systems. In: *SIAM Journal on Computing* 19(1) (1989), S. 186–208
- [GMW91] GOLDWASSER, S. ; MICALI, S. ; WIGDERSON, A.: Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. In: *Journal of the ACM* 38(1) (1991), S. 691–729
- [GO94] GOLDREICH, O ; OREN, Y.: Definitions and properties of zero-knowledge proof systems. In: *Journal of Cryptology* 7(1) (1994), S. 1–32
- [Gol91] GOLDREICH, O.: *Foundations of Cryptography*. 1991. – extracted from a working draft
- [Gol05] GOLDREICH, O.: *Foundations of Cryptography - A Primer*. Boston-Delft : now Publishers Inc., 2005
- [HILL99] HÅSTAD, J. ; IMPAGLIAZZO, R. ; LEVIN, L. ; LUBY, M.: A pseudorandom generator from any one-way function. In: *SIAM Journal on Computing* 28(4) (1999), S. 1364–1396
- [Köb03] KÖBLER, J.: Vorlesungsskript Kryptologie 1 Wintersemester 03/04 / Humboldt-Universität zu Berlin, Lehrstuhl Komplexität und Kryptografie. 2003.
- [Lub96] LUBY, M.: *Pseudorandomness and Cryptographic Applications*. Princeton University Press, 1996
- [MV99] MILTERSEN, P. B. ; VINODCHANDRAN, N. V.: Derandomizing Arthur-Merlin games using hitting sets. In: *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society Press, 1999, S. 71–80
- [MV04] MILTERSEN, P. B. ; VINODCHANDRAN, N. V. *Derandomizing Arthur-Merlin games using hitting sets*. 2004
- [Nis90] NISAN, N.: *Using Hard Problems to Create Pseudorandom Generators*. The MIT Press, 1990
- [NR98] NAOR, M. ; REINGOLD, O.: *From Unpredictability to Indistinguishability: A Simple Construction of Pseudo-Random Functions from MACs*. 1998. – Preliminary Version
- [NW94] NISAN, N. ; WIGDERSON, A.: Hardness vs. Randomness. In: *Journal of Computer and System Sciences* 49(2) (1994), S. 149–167

- [Sch99] SCHINDLER, W.: AIS 20, Version 1, Funktionalitätsklassen und Evaluationsmethodologie für deterministische Zufallszahlengeneratoren / BSI. 1999. <http://www.bsi.bund.de/zertifiz/zert/interpr/ais20.pdf>
- [Sho97] SHOR, P.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. In: *SIAM Journal on Computing* 26(5) (1997), S. 484–1509
- [Yao82] YAO, A. C.: Theory and applications of trapdoor functions, IEEE Computer Society Press, 1982, S. 80–91

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Berlin, den 21.12.2006