Diplomarbeit

„Effiziente Pseudozufallsgeneratoren: RSA- und x^2-mod-N-Generator“

Roger Fischlin

email: fischlin@mi.informatik.uni-frankfurt.de
URL: http://www.mi.informatik.uni-frankfurt.de

eingereicht im Juni 1997 am

Fachbereich Mathematik (12)
Johann-Wolfgang-Goethe-Universität
Frankfurt am Main

25. Oktober 1997

In der vorliegenden Fassung sind die im Rahmen der Bewertung aufgefallenen, kleine Fehler und Unstimmigkeiten behoben sowie aktuelle Ergänzungen hinzugefügt.

Zusammenfassung

In der vorliegenden Diplomarbeit beschäftigen wir uns mit kryptographisch sicheren Pseudozufallsgeneratoren. Diese effizienten Algorithmen erzeugen zu zufälliger Eingabe deterministisch eine längere Bitfolge, die praktisch von einer Folge zufälliger Münzwürfe nicht unterscheidbar ist. Wir geben die Definitionen von A. Yao sowie M. Blum und S. Micali, beweisen die Äquivalenz und charakterisieren den Unterschied zur klassischen Sichtweise von Zufallsgeneratoren.

Mit der Blum-Micali-Konstruktion zeigen wir, wie man aus einer Oneway-Permutation und zugehörigem Hardcore-Prädikat einen kryptographisch sicheren Pseudozufallsgenerator konstruiert: Man wendet auf einen zufälligen Startwert iterativ die Oneway-Funktion an und gibt jeweils das Hardcore-Prädikat des Urbilds aus. Wir stellen das allgemeine Hardcore-Prädikat „inneres Produkt modulo 2“ von L.A. Levin und O. Goldreich vor und beweisen mit Hilfe des XOR-Lemmas von U.V. Vazirani und V.V. Vazirani die Verallgemeinerung zu einer Hardcore-Funktion, die statt eines Prädikats mehrere Bits ausgibt.

Man geht davon aus, daß die Verschlüsselungsfunktionen des RSA- und des Rabin-Public-Key-Kryptosystems Oneway-Permutationen sind. Basierend auf dem Rabin-System haben L. Blum, M. Blum und M. Shub den x^2-mod-N-Generator aufgebaut, W. Alexi, B. Chor, O. Goldreich und C.P. Schnorr haben den RSA-Generator konstruiert und den Sicherheitsbeweis zum x^2-mod-N-Generator verbessert. Diese Generatoren basieren auf der Blum-Micali-Konstruktion mit dem Hardcore-Prädikat des untersten Bits. Durch neue Ideen können wir die beweisbare Sicherheit der Generatoren deutlich erhöhen, so daß in der Praxis kleinere Schlüssellängen genügen. Bisher war zum Beispiel für den x^2-mod-N-Generator bekannt, daß man mit einem Algorithmus A, der das unterste Bit der Wurzel modulo einer n-Bit-Blumzahl mit Wahrscheinlichkeit $\frac{1}{2} + \epsilon$ in Zeit $|A| = \Omega\left(n^3\right)$ berechnet, den Modul in Zeit $\mathcal{O}\left(n^3\epsilon^{-9}|A|\right)$ mit Wahrscheinlichkeit $\frac{\epsilon^2}{64}$ faktorisieren kann. Wir verbessern die Laufzeit zu $\mathcal{O}\left(n\epsilon^{-4}\log_2(n\epsilon^{-1})|A|\right)$ und Wahrscheinlichkeit $\frac{1}{9}$. Diese neuen Resultate wurden auf der Eurocrypt-Konferenz im Mai 1997 in Konstanz vorgestellt, D.E. Knuth hat sie bereits in die neue Auflage seines Standardwerks „The Art of Computer Programming“ aufgenommen.

Inhaltsverzeichnis

Einleitung

Man verwendet Pseudozufallsgeneratoren, um aus einem zufälligen Startwert effizient eine als Zufallszahlen verwendbare Ausgabefolge zu erzeugen. Da Pseudozufallsgeneratoren deterministische Algorithmen sind, ist der Vorteil gegenüber dem direkten Verwenden zufälliger Werte, daß die Ausgabe des Generators eindeutig durch den Startwert festgelegt ist: Beim Debuggen oder Nachvollziehen eines Programms erhalten wir durch den gleichen Startwert auch die gleiche Folge von Zufallszahlen.

Die ersten Pseudozufallsgeneratoren stammen von D.H. Lehmer und J. von Neumann Ende der vierziger Jahre. D.E. Knuths Standardwerk [Knuth81] gibt eine Übersicht über die zahlreichen, zu Beginn der achtziger Jahre bekannten Generatoren. Die klassische Anforderung an einen Pseudozufallsgenerator ist, daß die Ausgabe bekannte statistische Tests besteht. Dieser Ansatz ist aber nicht systematisch, da man nicht ausschließen kann, daß die Generatorausgabe einen neuen Test nicht besteht. Nach Vorarbeit von A. Shamir [Shamir81, Shamir83] haben 1982 A. Yao [Yao82] sowie M. Blum und S. Micali [BM84] perfekte bzw. kryptographisch sichere Pseudozufallsgeneratoren definiert. Statt Zahlenfolgen betrachten sie allgemein Bitfolgen und statt spezieller statistischer Tests alle probabilistischen Polynomialzeit-Algorithmen, d.h. nicht nur ausgewählte Tests wie in der klassischen Sichtweise, sondern man faßt alle probabilistischen Polynomialzeit-Algorithmen als statistische Tests auf. Praktisch soll kein effizienter Algorithmus entscheiden können, ob eine Bitfolge zufällig ist oder es sich um die Ausgabe eines kryptographisch sicheren Generators handelt. Diese Eigenschaft ist äquivalent zur Nicht-Vorhersagbarkeit: Man kann praktisch effizient aus einem Präfix der Generatorausgabe das nächste Bit der Ausgabe nicht besser als Raten, d.h. mit Wahrscheinlichkeit $\frac{1}{2}$, bestimmen. Im Gegensatz zur klassischen Sichtweise erhalten wir aus diesem systematischen Ansatz eine „beweisbare“ Sicherheit, die Voraussetzung für Pseudozufallsgeneratoren in der Kryptographie.

Es liegt nah, Pseudozufallsgeneratoren basierend auf schwierigen Problemen aus der Kryptographie zu konstruieren, um die Sicherheit auf bekannte Probleme wie zum Beispiel Faktorisieren oder Diskreten Logarithmus zurückzuführen. Von M. Blum und S. Micali [BM84] stammt in Verbindung mit einem Resultat von O. Goldreich und L.A. Levin [GL89, Levin93] eine praktikable Konstruktion, um aus jeder Oneway-Permutation (eine Permutation, die effizient nicht invertierbar ist) einen Pseudozofallsgenerator zu konstruieren.

Pseudozufallsgeneratoren haben ihrerseits zahlreiche Anwendungen in der Kryptographie. Sie ermöglichen, n Bits auszutauschen bzw. zu hinterlegen, um anschließend bezüglich n polynomiell viele pseudozufällige Bits zu erzeugen. Auf Pseudozufallsgeneratoren basiert zum Beispiel eine bekannte Konstruktion [GGM86] für pseudozufällige Funktionen: Durch $2n$ zufällige Bits definieren wir mit Hilfe eines Pseudozufallsgenerators einen Funktion, die man in Polyno-

mialzeit praktisch nicht von einer zufällig aus der Menge der Abbildungen $\{0,1\}^n \to \{0,1\}^n$ gewählten Funktion unterscheiden kann. Zum Vergleich: Um eine Funktion aus der Menge aller Abbildungen $\{0,1\}^n \to \{0,1\}^n$ als Vektor der Funktionswerte zu spezifizieren, sind $n2^n$ Bits nötig.

Ein weitere Anwendung von Pseudozufallsgeneratoren ist Bit-Commitment: Ein Teilnehmer fixiert ein Bit, hinterlegt es und gibt es erst später bekannt. Aus der Hinterlegung soll man praktisch nicht auf das Bit schließen können und umgekehrt soll der Teilnehmer nicht in der Lage sein, später einen anderen Wert bekanntzugeben. Ein solches Schema hat M. Naor [Naor91] auf Basis eines Pseudozufallsgenerators angeben. Der Teilnehmer wählt einen zufälligen Startwert, kodiert das fixierte Bit in die Ausgabe des Pseudozufallsgenerators und hinterlegt diesen Bitstring. Ohne Kenntnis des Startwerts ist die Ausgabe des Generators praktisch nicht von zufälligen Bits zu unterscheiden, durch Offenlegen des Startwerts erhält man aus dem hinterlegten Bitstring durch Abgleich das vom Teilnehmer zu Beginn fixierte Bit.

Die Existenz von Pseudozufallsgeneratoren ist wie die von anderen kryptographischen Primitiven ein offenes Problem der Kryptographie bzw. Komplexitätstheorie, wenngleich Existenz der kryptographischen Primitiven allgemein angenommen wird. Basierend auf den Verschlüsselungsverfahren der bekannten RSA- und Rabin-Kryptosysteme [RSA78, Rabin79] wurden der RSA- und der x^2-mod-N-Pseudozufallsgenerator [ACGS88, BBS86] entworfen: Wer die Ausgabe des Generators von echten Zufallsbits unterscheiden kann, ist in der Lage, in Polynomialzeit diese Kryptosysteme zu brechen, d.h. verschlüsselte Nachrichten ohne Kenntnis des geheimen Schlüssels zu dechiffrieren.

Für die Praxis ist man an exakter Sicherheit interessiert, d.h. welche Laufzeit hat ein statistischer Test, der zum Beispiel die Ausgabe des RSA- oder x^2-mod-N-Pseudozufallsgenerator für einen gegebenen Public-Key von echten Zufallsbits mit Toleranz 1% unterscheidet. Zwar ist dies kein Polynomialzeit-Algorithmus, aber in der Praxis ist die Bitlänge und die Laufzeitschranke explizit gegeben. Wir nehmen heuristisch untere Schranken für die Laufzeit zum Brechen des Kryptosystems an und vergleichen diese mit der Laufzeit, wenn wir das System mit Hilfe eines statistischen Tests brechen. Da die untere Schranke auch für das Verfahren mit Hilfe des statistischen Tests gilt, erhalten wir für feste Länge des Public-Keys eine obere Schranke für die Laufzeit des statistischen Tests oder für eine feste Laufzeitschranke des Tests eine untere Grenze für die Länge des Public-Keys, so daß die Generatorausgabe „sicher“ gegen diese statistischen Tests ist. Diese Analyse setzt effiziente Verfahren zum Invertieren des Kryptosystems mit Hilfe eines statistischen Tests voraus. Die neuen Resultate [FSch97], die wir in den Kapiteln 2 und 3 vorstellen, verbessern die bekannten Konstruktionen [ACGS88, VV84] für den Sicherheitsbeweis zum RSA- oder x^2-mod-N-Pseudozufallsgenerator. Falls die Generatorausgabe des x^2-mod-N-Pseudozufallsgenerators voraussagbar ist, so daß ein Algorithmus A existiert, der das unterste Bit einer der Wurzel modulo einer n-Bit-Blumzahl mit Wahrscheinlichkeit $\frac{1}{2}+\epsilon$ in Zeit $|A| = \Omega\left(n^3\right)$ berechnet, können wir den Modul in $\mathcal{O}\left(n\epsilon^{-4}\log_2(n\epsilon^{-1})|A|\right)$ Schritten mit Wahrscheinlichkeit $\frac{1}{9}$ faktorisieren und damit das Rabin-System brechen. Durch die bekannten Konstruktionen aus [ACGS88, VV84] kann man den Modul in Zeit $\mathcal{O}\left(n^3\epsilon^{-9}|A|\right)$ mit Wahrscheinlichkeit $\frac{\epsilon^2}{64}$ faktorisieren.

Mein Dank gilt insbesondere Prof. Dr. C.P. Schnorr für die Zusammenarbeit an der Arbeit [FSch97] und Prof. Dr. G. Kersting für die Hinweise auf Unstimmigkeiten und die Sicht des Themas aus der Blickrichtung eines Stochastikers. Ich bedanke mich bei M. Fischlin, M. Nälsund (KTH Stockholm) und J.-P. Seifert für Diskussionen.

Kapitel 1

Pseudozufallsgeneratoren

Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.

— John von Neumann (1951)

1.1 Klassische Sichtweise

In der klassischen Sichtweise ist ein Pseudozufallsgenerator ein Algorithmus, der deterministisch zu einem gegebenen Startwert (Saat oder *Seed* genannt) eine Folge von Zahlen ausgibt. Die Folge wird zumeist durch eine einfache Rekursion beschrieben und die Werte liegen in einem endlichen Bereich (zum Beispiel $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$ für ein $m \in \mathbb{N}$). Der bekannte Lineare-Kongruenz-Generator von D.H. Lehmer (1949) beginnt mit dem Seed $x_0 \in \mathbb{Z}_m$ und erzeugt die Folge $(x_n)_{n\in\mathbb{N}}$ gemäß

$$x_{n+1} := a \cdot x_n + c \bmod m,$$

wobei $m \in \mathbb{N}$ und $a, c \in \mathbb{Z}_m$ zuvor fest gewählt wurden.

Die Ausgabe des Generators soll sich ähnlich einer Folge unabhängig und zufällig aus $\mathbb{Z}_m$ gezogener Werte verhalten. Um diese Eigenschaft zu untersuchen, entstanden im Laufe der Zeit zahlreiche statistische Tests (vergleiche Kapitel 3.3 in D.E. Knuths Standardwerk [Knuth81]):

- Empirische Tests untersuchen die Folge für feste Startwerte und liefern statistische Aussagen über die Folge (zum Beispiel die Verteilung der Folgenglieder).
- Theoretische Tests untersuchen charakteristische Eigenschaften (zum Beispiel die Periodenlänge) mit Hilfe zahlentheoretischer Sätze.

Ein Generator, dessen Ausgabe die bekannten Standardtests besteht, gilt nach klassischer Sichtweise als Pseudozufallsgenerator.

1.2 Kryptographische Sichtweise

Das Vorgehen in der klassischen Theorie ist nicht systematisch: Zwar kann ein Generator alle bekannten statistischen Tests bestehen, dennoch ist nicht auszuschließen, daß die Ausgabefolge einen anderen Test nicht besteht. Für kryptographische Anwendungen von Pseudozufallsgeneratoren ist (zumindest unter allgemein anerkannten Annahmen) ein Sicherheitsbeweis Voraussetzung. Man ist an Aussagen interessiert, daß die Generatorausgabe allgemein jeden Test (mit vertretbarer Laufzeit) besteht und dies nicht für bestimmte Startwerte, sondern im Durchschnitt für alle Startwerte gilt.

Ein erster Ansatz stammt von A. Shamir [Shamir81, Shamir83] aus dem Jahr 1981 und basiert auf dem RSA-Kryptosystem (siehe Kapitel 2). Der Startwert $x_0 \in_{\mathrm{R}} \mathbb{Z}_N^*$ ist ein zufälliges Element aus der Nachrichtenmenge $\mathbb{Z}_N^*$ für einen zufälligen RSA-Modul N. Der Generator gibt die RSA-Verschlüsselung $x \mapsto x^e \bmod N$ für geeignete, paarweise verschiedene Kodierexponenten $e_1, e_2, \ldots$ aus:

$$x_i = x_0^{e_i} \bmod N \qquad \text{für } i = 1, 2, \ldots \tag{1.1}$$

A. Shamir zeigt, daß ein Polynomialzeit-Algorithmus, der zu gegebenen Kodierexponenten $e_1, e_2, \ldots$ und $x_1, x_2, \ldots, x_i$ den Wert x_{i+1} berechnet (d.h. die Folge vorhersagt), effizient auch kodierte Nachrichten ohne Kenntnis des geheimen Schlüssels dechiffrieren kann. Dies ist ein Widerspruch zur akzeptierten Hypothese, daß die RSA-Kodierung effizient nicht gebrochen werden kann. Da die RSA-Kodierung $x \mapsto x^e \bmod N$ eine Permutation auf $\mathbb{Z}_N^*$ ist, folgt aus (1.1), daß für zufälligen Startwert $x \in_{\mathrm{R}} \mathbb{Z}_N^*$ jedes x_i gleichverteilt in $\mathbb{Z}_N^*$ ist. Die x_i sind aber nicht unabhängig, wir können die Ausgabefolge effizient von einer Folge zufällig und unabhängig aus $\mathbb{Z}_N^*$ gezogener Werte unterscheiden, denn die Hälfte der Zahlen aus $\mathbb{Z}_N^*$ hat das Jacobi-Symbol $+1$, die andere Hälfte den Wert -1, wenngleich die Elemente der Folge (1.1) jedoch dasselbe Jacobi-Symbol wie der Startwert x_0 haben.

A. Yao [Yao82] sowie M. Blum und S. Micali [BM84] haben 1982 die Grundlagen für perfekte bzw. kryptographisch sichere Pseudozufallsgeneratoren gelegt. Da in der Kryptographie nur sichere Pseudozufallsgeneratoren betrachtet werden, spricht man kurz von Pseudozufalls- oder Pseudorandom-Generatoren. Die Theorie der sicheren Pseudozufallsgeneratoren basiert allgemein auf Bitfolgen statt auf Folgen von Zahlen aus einem bestimmten Zahlbereich. Yaos Überlegung ist, allgemein jeden (probabilistischen) Algorithmus, der als Eingabe eine Bitfolge erhält und eine $\{0,1\}$-Ausgabe hat, als statistischen Test anzusehen. Wir interpretieren die Ausgabe 1, daß der Algorithmus die Bitfolge als Ausgabe eines Generators und nicht als echte Zufallsbits ansieht. Diese Definition ist willkürlich: Da wir über alle Algorithmen argumentieren, betrachten wir auch zu einem statistischen Test, dessen Ausgabe umgekehrt zu interpretieren ist, den Algorithmus mit negierter Ausgabe. Wir vergleichen die Wahrscheinlichkeiten, daß der Algorithmus bei Generatorausgabe und bei einer Folge zufälliger Bits den gleichen Wert (zum Beispiel 1) ausgibt. Je näher diese Wahrscheinlichkeiten beieinanderliegen, desto schwieriger bzw. praktisch nicht möglich ist es, daß man durch diesen Algorithmus (statistischen Test) die Generatorausgabe von echten Zufallsbits unterscheiden kann. Wenn der Absolutbetrag der Differenz hinreichend klein ist, nennen wir den Generator „perfekt“ (Yao) oder „kryptographisch sicher“ (Blum, Micali).

Wir fordern vom statistischen Test, daß er effizient das Ergebnis liefert, denn Verfahren, deren Laufzeit (Anzahl Operationen) zum Beispiel exponentiell in der Eingabelänge (Länge

der Bitfolge) ist, können bereits für kurze Eingaben in der Praxis nicht mehr durchgeführt werden. Wie in der Kryptographie üblich, beschränken wir uns daher auf Polynomialzeit-Algorithmen, d.h. die Laufzeit ist durch ein Polynom in der Eingabelänge beschränkt.

1.3 Definition

Wir definieren Pseudozufallsgeneratoren. Um asymptotische Aussagen machen zu können, fassen wir Verteilungen auf Bitstrings verschiedener Länge zu einem Ensemble zusammen:

Definition 1.3.1 (Ensemble)
Sei $\ell(n) \geq 1$ durch ein Polynom beschränkt. Eine Folge $(B_n)_{n\in\mathbb{N}}$ von Wahrscheinlicheitsverteilungen B_n auf $\{0,1\}^{\ell(n)}$ nennen wir ein Ensemble vom Typ $\ell(n)$.

Yao bezeichnet den Typ $\ell(n)$ als Längenfunktion des Ensembles. Wir formalisieren den Begriff des „statistischen Tests“:

Bezeichnung 1.3.2 (Statistischer Test)
Jeder probabilistischer Polynomialzeit-Algorithmus mit $\{0,1\}$-Ausgabe ist ein statistischer Test.

Sei U_n die Gleichverteilung auf $\{0,1\}^n$. Im weiteren verwenden wir die gleiche Notation sowohl für die Wahrscheinlichkeitsverteilung als auch für gemäß dieser Verteilung verteilten Zufallsvariablen. Falls mehrere, unabhängige Zufallsvariablen der gleichen Verteilung verwendet werden, unterscheiden wir diese durch zusätzliche Striche (zum Beispiel B'_n und B''_n).

Definition 1.3.3 (Statistischen Test bestehen)
Ein Ensemble $(B_n)_{n\in\mathbb{N}}$ vom Typ $\ell(n)$ besteht den statistischen Test T mit Toleranz $\delta(n) \geq 0$, falls ein $n_0 \in \mathbb{N}$ existiert, so daß für alle $n \geq n_0$ gilt

$$\left|\mathrm{Ws}[T(B_n) = 1] - \mathrm{Ws}\left[T\left(U_{\ell(n)}\right) = 1\right]\right| < \delta(n),$$

wobei die Wahrscheinlichkeit über die Eingabe und die internen Münzwürfe des Algorithmus' T gebildet wird.

Die Laufzeit des statistischen Tests ist polynomiell in n, da die Eingabelänge $\ell(n)$ nach Definition durch ein Polynom in n beschränkt ist.

Ein Ensemble $(B_n)_{n\in\mathbb{N}}$ besteht einen statistischen Test T mit $\delta(n)$-Toleranz, wenn für hinreichend große n die Differenz der Wahrscheinlichkeiten, daß der Test bei gemäß B_n verteilter oder zufälliger Eingabe den gleichen Wert ausgibt, kleiner als $\delta(n)$ ist. Wir setzen:

$$\Delta_T(B_n) := \mathrm{Ws}[T(B_n) = 1] - \mathrm{Ws}\left[T\left(U_{\ell(n)}\right) = 1\right]$$

Wenn ein Ensemble $(B_n)_{n\in\mathbb{N}}$ vom Typ $\ell(n)$ einen statistischen Test T mit Toleranz $\delta(n)$ nicht besteht, gilt für unendlich viele n, daß $|\Delta_T(B_n)| \geq \delta(n)$ ist. Wir können O.B.d.A. annehmen,

daß für unendlich viele n gilt $\Delta_T(B_n) \geq \delta(n)$, denn andernfalls invertiere man die Ausgabe des statistischen Tests T. Folgende Definition eines pseudozufälligen Ensembles geht auf A. Yao [Yao82] zurück:

Definition 1.3.4 (Pseudozufälliges Ensemble)
Ein Ensemble $(B_n)_{n\in\mathbb{N}}$ ist pseudozufällig, wenn es jeden statistischen Test mit Toleranz n^{-t} für alle $t \in \mathbb{N}$ besteht.

Sei $(B_n)_{n\in\mathbb{N}}$ ein pseudozufälliges Ensemble. Für jeden statistischen Test T und jedes Polynom $p : \mathbb{N} \to \mathbb{N}$ gilt für hinreichend große n:

$$|\Delta_T(B_n)| < \frac{1}{p(n)}$$

In Worten: $|\Delta_T(B_n)|$ verschwindet schneller als jeder polynomieller Anteil. A. Yao verwendet die Notation

$$|\Delta_T(B_n)| = \mathcal{O}(\nu(n)),$$

wobei ν an das englische „to vanish“ (verschwinden) angelehnt ist. Allgemeiner heißt in der Kryptographie eine Funktion $f : \mathbb{N} \to \mathbb{R}$ mit $f(n) = \mathcal{O}(\nu(n))$ *vernachlässigbar* (negligible). Diese Forderung an $|\Delta_T(B_n)|$ ist eng verbunden mit der polynomiellen Laufzeit des statistischen Tests, wie wir in den Beweisen dieses und der folgenden Kapitel sehen werden.

Sei G ein deterministischer Algorithmus, der also insbesondere keine internen Münzwürfe verwendet, und $(B_n)_{n\in\mathbb{N}}$ ein Ensemble. Wir bezeichnen mit $G(B_n)$ die Ausgabeverteilung von G zu gemäß B_n verteilter Eingabe.

Definition 1.3.5 (Pseudozufallsgenerator)
Ein deterministischer Algorithmus $G : \{0,1\}^* \to \{0,1\}^*$ heißt (krypographisch sicherer) Pseudozufallsgenerator vom Typ $\ell(n)$, wenn:

1. *G ist ein Polynomialzeit-Algorithmus.*
2. *Für alle $n \in \mathbb{N}$ gilt $G(\{0,1\}^n) \subseteq \{0,1\}^{\ell(n)}$.*
3. *Für alle $n \in \mathbb{N}$ gilt $\ell(n) \geq n + 1$.*
4. *Das Ensemble $(G(U_n))_{n\in\mathbb{N}}$ vom Typ $\ell(n)$ ist pseudozufällig.*

Die Forderungen 2 und 3 sichern, daß die Eingabe um mindestens ein Bit expandiert wird. Da Algorithmus G polynomielle Laufzeit hat, ist $\ell : \mathbb{N} \to \mathbb{N}$ durch ein Polynom beschränkt. Wir werden in Satz 1.7.2 auf Seite 50 zeigen, daß wir zu jedem durch ein Polynom beschränkten $m(n) \geq n+1$ aus einen Pseudozufallsgenerator des Typs $\ell(n)$ einen kryptographisch sicheren Generator des Typs $m(n)$ konstruieren können. Für Eigenschaft 4 nehmen wir an, daß der Algorithmus des Generators G öffentlich ist, d.h. es ist bekannt, wie das Ensemble erzeugt wird und ein statistischer Test kann insbesondere selbst den Generator anwenden. Wir lernen im folgenden Kapitel eine äquivalente Charakterisierung pseudozufälliger Ensembles kennen,

mit deren Hilfe man nachweist, daß unter plausiblen Hypothesen bestimmte Ensembles pseudozufällig sind.

Sei $(B_n)_{n\in\mathbb{N}}$ ein pseudozufälliges Ensemble vom Typ $\ell(n)$. Ein statistischer Test erhält nur eine Eingabe und soll entscheiden, ob diese gemäß B_n verteilt oder eine zufällige Bitfolge ist. Wir zeigen (unter einer Voraussetzung), daß $(B_n)_{n\in\mathbb{N}}$ auch jeden statistischen Test, der beliebig viele Beispiele anfordern darf, besteht. Wir setzen voraus, daß das Ensemble polynomiell erzeugbar ist:

Definition 1.3.6 (Polynomiell erzeugbares Ensemble)
Ein Ensemble $(B_n)_{n\in\mathbb{N}}$ vom Typ $\ell(n)$ ist polynomiell erzeugbar, wenn es einen probabilistischen Polynomialzeit-Algorithmus gibt, der zu Eingabe 1^n (n unär) ein Element $\{0,1\}^{\ell(n)}$ gemäß Verteilung B_n ausgibt.

Insbesondere ist für jeden Pseudozufallsgenerator G das Ensemble $(G(U_n))_{n\in\mathbb{N}}$ polynomiell erzeugbar: Wir wählen einen zufälligen Startwert aus $\{0,1\}^n$ und wenden den Generator G an.

Da der statistische Test ein Polynomialzeit-Algorithmus ist, kann der Test nur polynomiell viele Beispiele anfordern. Wir modellieren diese Situation durch Produktensembles:

Definition 1.3.7 (Produktensemble)
Sei $(B_n)_{n\in\mathbb{N}}$ ein Ensemble vom Typ $\ell(n)$ und $k(n) \geq 1$ durch ein Polynom beschränkt. Das $k(n)$-fache Produktensemble $\left(B_n^{\times k(n)}\right)_{n\in\mathbb{N}}$ zu $(B_n)_{n\in\mathbb{N}}$ besteht aus dem $k(n)$-fachen Kreuzprodukt der Verteilungen:

$$B_n^{\times k(n)} = \underbrace{B'_n \times B''_n \times \cdots \times B_n^{(k)}}_{k(n)\text{-mal}}$$

Die Zufallsvariable $B_n^{\times k(n)}$ umfaßt $k(n)$ unabhängig und gemäß B_n verteilten Zufallsvariablen.

Wir erhalten als erstes, offensichtliches Ergebnis, daß mit dem Produktensemble auch das eigentliche Ensemble pseudozufällig ist:

Lemma 1.3.8
Sei $(B_n)_{n\in\mathbb{N}}$ ein Ensemble vom Typ $\ell(n)$ und $k(n) \geq 1$ durch ein Polynom beschränkt. Sei T ein statistischer Test, den das Ensemble $(B_n)_{n\in\mathbb{N}}$ mit Toleranz $\delta(n)$ nicht besteht. Den statistischen Test für das $k(n)$-fache Produktensemble zu $(B_n)_{n\in\mathbb{N}}$, der T auf die erste Eingabe anwendet, besteht das $k(n)$-fache Produktensemble zu $(B_n)_{n\in\mathbb{N}}$ mit Toleranz $\delta(n)$ nicht.

Für die umgekehrte Richtung setzen wir voraus, daß das Ensemble polynomiell erzeugbar ist. Für Algorithmus 1.3.1 gilt:

Lemma 1.3.9
Sei $(B_n)_{n\in\mathbb{N}}$ ein polynomiell erzeugbares Ensemble vom Typ $\ell(n)$ und $k(n) \geq 1$ durch ein Polynom beschränkt. Sei T ein statistischer Test, den das $k(n)$-fache Produktensemble zu

$(B_n)_{n\in\mathbb{N}}$ mit Toleranz $\delta(n)$ nicht besteht. Dann besteht das Ensemble $(B_n)_{n\in\mathbb{N}}$ den statistischen Test T' (Algorithmus 1.3.1) mit Toleranz $\delta'(n) := \frac{\delta(n)}{k(n)}$ nicht. Der statistische Test T' hat die Laufzeit $|T'| = |T| + k(n)\cdot|B_n| + \mathcal{O}(1)$, wobei $|B_n|$ die Laufzeit zum Erzeugen eines gemäß B_n verteilten Elements sei.

Algorithmus 1.3.1 Statistischer Test T' für B_n aus Test T für Produktensemble

---

EINGABE: $\left(b_1, b_2, \ldots, b_{k(n)}\right) \in \left(\{0,1\}^{\ell(n)}\right)^{k(n)}$

/∗ Sei T ein statistischer Test, den das $k(n)$-fache Produktensemble zu $(B_n)_{n\in\mathbb{N}}$ mit Toleranz $\delta(n)$ nicht besteht. ∗/

1. Wähle $i \in_{\mathrm{R}} \{0, 1, \ldots, k(n)-1\}$ zufällig.

2. FOR $j := 1$ TO i DO $h_j := b_j$

3. FOR $j := i+1$ TO $k(n)$ DO wähle $h_j \in_{\mathrm{R}} \{0,1\}^{\ell(n)}$ zufällig.

4. Gib $T\left(h_1, h_2, \ldots, h_{\ell(n)}\right)$ aus.

Beweis. Wir können o.B.d.A. annehmen, daß $\Delta_T(B_n) \geq 0$ für unendlich viele n gilt (sonst invertiere die Ausgabe des statistischen Tests T). Wir betrachten Folgen *hybrider* Verteilungen (lat. hybrid: von zweierlei Herkunft) auf $\left(\{0,1\}^{\ell(n)}\right)^{k(n)}$ speziell für diese unendlich vielen n:

$$H_n^{(j)} = \underbrace{B_n \times B_n \times \cdots \times B_n}_{j\text{-mal}} \times \underbrace{U_{\ell(n)} \times U_{\ell(n)} \times \cdots \times U_{\ell(n)}}_{(k(n)-j)\text{-mal}} \qquad \text{für } j := j(n) = 0, 1, \ldots, k(n)$$

Die ersten j Komponenten der Verteilung $H_n^{(j)}$ stammen von $B_n^{\times k(n)}$, die restlichen $k(n) - j$ von $U_{\ell(n)}^{\times k(n)}$. Insbesondere gilt:

- $U_{\ell(n)}^{\times k(n)} = H_n^{(0)}$
- $B_n^{\times k(n)} = H_n^{(k(n))}$

Bei der Hybrid-Methode folgert man aus der Möglichkeit, die beiden äußeren Verteilungen, nämlich $H_n^{(0)}$ und $H_n^{(k(n))}$, zu unterscheiden, dies auch für zwei benachbarte, hybride Verteilungen zu können. Wir schätzen zunächst folgende Teleskopsumme nach unten ab:

$$\begin{aligned} &\sum_{j=0}^{k(n)-1} \mathrm{Ws}\left[T\left(H_n^{(j)}\right) = 1\right] - \sum_{j=0}^{k(n)-1} \mathrm{Ws}\left[T\left(H_n^{(j+1)}\right) = 1\right] \\ &= \mathrm{Ws}\left[T\left(H_n^{(0)}\right) = 1\right] - \mathrm{Ws}\left[T\left(H_n^{(k(n))}\right) = 1\right] \\ &\geq \delta(n) \end{aligned} \tag{1.2}$$

Da wir in Schritt 1 von T' (Algorithmus 1.3.1) die Position $i \in_R \{0, 1, \ldots, k(n)-1\}$ zufällig wählen, gilt:

$$\mathrm{Ws}\left[T'(B_n) = 1\right] = \frac{1}{k(n)} \sum_{j=0}^{k(n)-1} \mathrm{Ws}\left[T\left(H_n^{(j)}\right) = 1\right]$$

$$\mathrm{Ws}\left[T'(U_{\ell(n)}) = 1\right] = \frac{1}{k(n)} \sum_{j=0}^{k(n)-1} \mathrm{Ws}\left[T\left(H_n^{(j+1)}\right) = 1\right]$$

Wir erhalten mit Abschätzung (1.2), daß für unendlich viele n gilt:

$$\Delta_{T'}(B_n) = \left|\mathrm{Ws}\left[T'(B_n) = 1\right] - \mathrm{Ws}\left[T'(U_{\ell(n)}) = 1\right]\right| \geq \frac{\delta(n)}{k(n)}$$

Das Ensemble $(B_n)_{n\in\mathbb{N}}$ besteht den statistischen Test T' mit Toleranz $\delta'(n) := \frac{\delta(n)}{k(n)}$ nicht. ■

Ohne die Voraussetzung, daß das Ensemble $(B_n)_{n\in\mathbb{N}}$ polynomiell erzeugbar ist, gilt die Aussage von Lemma 1.3.9 im allgemeinen nicht: O. Goldreich und B. Meyer [GM96] konstruieren ein pseudozufälliges Ensemble $(B_n)_{n\in\mathbb{N}}$, dessen 2-faches Produktensemble aber nicht pseudozufällig ist.

Die in Lemma 1.3.9 verwendete *Hybrid-Methode* ist eine zentrale Beweisidee im Bereich der Pseudozufallsgeneratoren und -funktionen (vergleiche [GGM86]). Wir werden die Hybrid-Methode, die von M. Blum und S. Micali [BM84] stammt, im weiteren noch mehrmals anwenden. Die Bezeichnung „Hybrid-Methode“ geht laut [Goldreich95a] auf L.A. Levin zurück.

Es folgt, daß es in der Definition von (polynomiell erzeugbaren) pseudozufälligen Ensembles genügt, sich auf statistische Tests zu beschränken, die nur ein Beispiel als Eingabe erhalten.

Satz 1.3.10
Sei $(B_n)_{n\in\mathbb{N}}$ ein polynomiell erzeugbares Ensemble vom Typ $\ell(n)$ und $k(n) \geq 1$ durch ein Polynom beschränkt. Das Ensemble $(B_n)_{n\in\mathbb{N}}$ ist genau dann pseudozufällig, wenn das $k(n)$-fache Produktensemble zu $(B_n)_{n\in\mathbb{N}}$ pseudozufällig ist.

Beweis. Wir zeigen, daß $(B_n)_{n\in\mathbb{N}}$ genau dann nicht pseudozufällig ist, wenn das $k(n)$-fache Produktensemble zu $(B_n)_{n\in\mathbb{N}}$ nicht pseudozufällig ist.

„⇒“ Sei das $k(n)$-fache Produktensemble zu $(B_n)_{n\in\mathbb{N}}$ nicht pseudozufällig, d.h. es existiert ein statistischer Test, den das Produktensemble mit n^{-c}-Toleranz für ein festes $c \in \mathbb{N}$ nicht besteht. Nach Lemma 1.3.9 gibt es einen statistischen Test T, den das Ensemble $(B_n)_{n\in\mathbb{N}}$ mit Toleranz $\frac{n^{-c}}{k(n)}$ nicht besteht. Da $k(n)$ durch ein Polynom beschränkt ist, existiert eine Konstante $d \in \mathbb{N}$ mit $k(n) \leq n^d$ für hinreichend große n. Wir erhalten einen statistischen Test, den das Ensemble $(B_n)_{n\in\mathbb{N}}$ mit Toleranz $n^{-(c+d)}$ für feste $c, d \in \mathbb{N}$ nicht besteht.

„⇐“ Sei $(B_n)_{n\in\mathbb{N}}$ nicht pseudozufällig, d.h. es existiert ein statistischer Test, den das Ensemble mit n^{-c}-Toleranz für ein festes $c \in \mathbb{N}$ nicht besteht. Nach Lemma 1.3.8 gibt es einen statistischen Test T, den das Produktensemble zu $(B_n)_{n\in\mathbb{N}}$ mit Toleranz n^{-c} nicht besteht.

Dies war zu zeigen. ■

M. Blum und S. Micali [BM84] lassen statt Polynomialzeit-Algorithmen Schaltkreisfamilien $(S_n)_{n\in\mathbb{N}}$ polynomieller Größe als statistische Tests zu. Bei der Schaltkreisfamilie handelt es sich um ein nicht-uniformes Rechenmodell: Zu jeder Eingabelänge existiert ein Schaltkreis, während wir nur einen uniformen Algorithmus für alle Eingabelängen zulassen. Zum Beispiel haben wir in Algorithmus 1.3.1 die Bitposition $i := i(n)$ zufällig gewählt, während wir diese im Schaltkreis S_n fest vorgeben können. O. Goldreich und B. Meyer [GM96] zeigen, daß es ein (nicht-polynomiell erzeugbares) Ensemble gibt, das einen statistischen Test in Form einer Schaltkreisfamilie polynomieller Größe nicht besteht, aber jeden probabilistischen Polynomialzeit-Algorithmus. Da unser Hauptaugenmerk auf polynomiell erzeugbaren Ensembles liegt, argumentieren wir im weiteren stets über probabilistische Polynomialzeit-Algorithmen, zumal diese effektive Konstruktionen liefern, während wir bei nicht-uniformen Schaltkreisfamilien nur die Existenz entsprechender Familien voraussetzen können. Die Aussagen der folgenden Kapitel sind auf das Rechenmodell der Schaltkreisfamilien übertragbar.

1.4 Vorhersagbarkeit und Prediktor

Bei A. Shamirs Generator (1.1) auf Seite 10 kann aus dem Präfix $x_1, x_2, \ldots, x_i$ der Ausgabefolge effizient nicht der nachfolgende Wert x_{i+1} bestimmt werden (sofern die RSA-Kodierung sicher ist). Wir übertragen diese Eigenschaft auf Ensembles und zeigen, daß die Nichtvorhersagbarkeit des nächstens Bits genau auf pseudozufällige Ensembles zutrifft. Ein Polynomialzeit-Algorithmus zur Vorhersage eines Bits nennen wir Prediktor. Analog zum statistischen Test definieren wir allgemein:

Definition 1.4.1 (Prediktor zur Vorhersage)
Sei $(B_n)_{n\in\mathbb{N}}$ ein Ensemble des Typs $\ell(n)$. Ein probabilistischer Polynomialzeit-Algorithmus, der zur Eingabe $(i,b) \in \{0,1,\ldots,\ell(n)-1\} \times \{0,1\}^{\ell(n)}$ nur die ersten i Bits der Eingabe b liest und eine $\{0,1\}$-Ausgabe erzeugt, ist ein Prediktor zur Vorhersage (nach rechts) des $(i+1)$-Bits des Einsembles $(B_n)_{n\in\mathbb{N}}$.

Die Laufzeit ist polynomiell in der Länge der gesamten Eingabe, insbesondere in $\ell(n)$ und damit in n. Zu $b = (b_1, b_2, \ldots, b_k) \in \{0,1\}^k$ bezeichne $\mathrm{bit}_i(b)$ das i-te Bit b_i falls $1 \le i \le k$ und sonst sei $\mathrm{bit}_i(b) := 0$. Im weiteren bezeichnen wir mit $I_{\ell(n)}$ die Gleichverteilung auf $\{0,1,\ldots,\ell(n)-1\}$. Wir definieren den Vorteil eines Prediktors zur Vorhersage:

Definition 1.4.2 (Vorteil bei Vorhersage)
Sei $(B_n)_{n\in\mathbb{N}}$ ein Ensemble vom Typ $\ell(n)$ und $i := i(n)$ mit $0 \le i < \ell(n)$.

a) Ein Prediktor P hat einen $\epsilon(n)$-Vorteil (gegenüber Raten) bei der Vorhersage des $(i+1)$-ten Bits des Ensembles $(B_n)_{n\in\mathbb{N}}$, wenn für unendlich viele $n \in \mathbb{N}$ gilt:

$$\mathrm{Ws}[\,P(i, B_n) = \mathrm{bit}_i(B_n)] \ge \tfrac{1}{2} + \epsilon(n)$$

b) Ein Prediktor P hat einen $\epsilon(n)$-Vorteil bei der Vorhersage des Ensembles $(B_n)_{n\in\mathbb{N}}$, wenn für unendlich viele $n \in \mathbb{N}$ gilt:

$$\mathrm{Ws}\Big[\, P\left(I_{\ell(n)}, B_n\right) = \mathrm{bit}_{I_{\ell(n)}}(B_n)\Big] \geq \tfrac{1}{2} + \epsilon(n)$$

Die Wahrscheinlichkeit wird jeweils über die Eingaben und die internen Münzwürfe des Prediktors gebildet.

Falls wir den Prediktor P auf eine Bitposition $i = i(n)$ fixieren, schreiben wir kurz P_i. Der Prediktor zur Vorhersage des $(i+1)$-ten Bits setzt voraus, daß wir die geeignete Bitposition explizit angeben, obwohl wir teilweise nur die Existenz nachweisen können. Falls ein Prediktor zur Vorhersage des $(i+1)$-ten Bits mit Vorteil $\epsilon(n)$ existiert, ist dies auch ein allgemeiner Prediktor mit Vorteil mindestens $\frac{\epsilon(n)}{\ell(n)}$. Umgekehrt folgt aus einem allgemeinen Prediktor, daß eine Bitposition $i := i(n)$ existiert, so daß der Prediktor das $(i+1)$-te Bit mit gleichem Vorteil vorhersagt.

Wir nennen ein Ensemble *vorhersagbar*, wenn es ein festes $c \in \mathbb{N}$ und einen Prediktor mit Vorteil n^{-c} zur Vorhersage gibt. Ein Ensemble heißt unvorhersagbar, wenn jeder Prediktor nur einen vernachlässigbaren Vorteil bei der Vorhersage hat.

Definition 1.4.3 (Unvorhersagbares Ensemble)
Ein Ensemble $(B_n)_{n\in\mathbb{N}}$ ist unvorhersagbar, wenn für jeden Prediktor P und alle $t \in \mathbb{N}$ ein $n_0 \in \mathbb{N}$ existiert, so daß für alle $n \geq n_0$ gilt

$$\left|\mathrm{Ws}\Big[\, P\left(I_{\ell(n)}, B_n\right) = \mathrm{bit}_{I_{\ell(n)}}(B_n)\Big] - \tfrac{1}{2}\right| < n^{-t},$$

wobei die Wahrscheinlichkeit über die Eingaben und die internen Münzwürfe des Prediktors gebildet wird.

Während A. Yao [Yao82] pseudozufällige Ensembles über statistische Tests definiert hat, haben M. Blum und S. Micali [BM84] pseudozufällige Ensembles mit Hilfe von Prediktoren sowie der Unvorhersagbarkeit charakterisiert. Wir zeigen ein Resultat von A. Yao, daß die beiden Eigenschaften, „pseudozufällig“ und „unvorhersagbar“ zu sein, äquivalent sind. Wir beweisen zunächst, daß ein pseudozufälliges Ensemble unvorhersagbar ist:

Lemma 1.4.4
Sei P_i ein Prediktor mit Vorteil $\epsilon(n)$ bei der Vorhersage des $(i+1)$-ten Bits des Ensembles $(B_n)_{n\in\mathbb{N}}$ vom Typ $\ell(n)$. Dann existiert ein statistischer Test T mit Laufzeit $|T| = |P_i| + \mathcal{O}(1)$, den das Ensemble $(B_n)_{n\in\mathbb{N}}$ mit $\epsilon(n)$-Toleranz nicht besteht.

Beweis. Wir definieren den statistischen Test gemäß:

$$T(b) := (P_i(b) + \mathrm{bit}_{i+1}(b) + 1) \bmod 2 = \begin{cases} 1 & \text{falls } P_i(b) = \mathrm{bit}_{i+1}(b) \\ 0 & \text{sonst} \end{cases} \tag{1.3}$$

Der Test gibt genau dann eine 1 aus, wenn der Prediktor das $(i+1)$-te Bit korrekt voraussagt. Da er einen $\epsilon(n)$-Vorteil hat, gilt für unendlich viele $n \in \mathbb{N}$:

$$\mathrm{Ws}[T(B_n) = 1] \geq \tfrac{1}{2} + \epsilon(n)$$

Falls die Eingabe des statistischen Tests eine zufällige Folge und insbesondere das $(i+1)$-te Bit zufällig ist, gilt:

$$\mathrm{Ws}\left[T\left(U_{\ell(n)}\right) = 1\right] = \tfrac{1}{2}$$

Insgesamt erhalten wir, daß für unendlich viele $n \in \mathbb{N}$ gilt:

$$\Delta_T(B_n) = \mathrm{Ws}[T(B_n) = 1] - \mathrm{Ws}\left[T\left(U_{\ell(n)}\right) = 1\right] \geq \epsilon(n)$$

Die Laufzeit $|T| = |P_i| + \mathcal{O}(1)$ des statistischen Tests folgt unmittelbar aus Definition (1.3). ■

Das folgende Lemma liefert die Umkehrung von Lemma 1.4.4. Man beachte, daß sich das Verhältnis von Toleranz und Vorteil im Gegensatz zu Lemma 1.4.4 unterscheidet.

Lemma 1.4.5
Sei $(B_n)_{n\in\mathbb{N}}$ ein Ensemble vom Typ $\ell(n)$, das einen statistischen Test T mit $\delta(n)$-Toleranz nicht besteht. Dann existieren eine Bitposition $i = i(n)$ mit $0 \leq i < \ell(n)$ und ein Prediktor P_i zur Vorhersage des $(i+1)$-ten Bits mit Vorteil $\epsilon(n) := \frac{\delta(n)}{\ell(n)}$ und Laufzeit $|P_i| = |T| + \mathcal{O}(\ell(n))$.

Beweis. Wir können o.B.d.A. annehmen, daß $\Delta_T(B_n) \geq 0$ für unendlich viele n gilt (sonst invertiere die Ausgabe des statistischen Tests T) und betrachten im weiteren nur diese n.

Um die Hybrid-Methode anzuwenden, definieren wir Folgen hybrider Verteilungen. Wir wählen ein zufälliges Element $\left(b_1, b_2, \ldots, b_{\ell(n)}\right) \in \{0,1\}^{\ell(n)}$ gemäß der Verteilung B_n und ersetzen die Bits $b_{j+1}, b_{j+2}, \ldots, b_{\ell(n)}$ durch zufällige Bits für $j := j(n) = 0, 1, \ldots, \ell(n)$. Sei $H_n^{(j)}$ die entstandene Verteilung auf $\{0,1\}^{\ell(n)}$. Wir betrachten die Ensembles $\left(H_n^{(j)}\right)_{n\in\mathbb{N}}$ des Typs $\ell(n)$. Insbesondere gilt:

- $U_{\ell(n)} = H_n^{(0)}$
- $B_n = H_n^{(\ell(n))}$

Wir setzen für $j = 0, 1, \ldots, \ell(n)$:

$$p_j := p_j(n) := \mathrm{Ws}\left[T\left(H_n^{(j)}\right) = 1\right] \geq 0 \tag{1.4}$$

Nach Voraussetzung gilt für folgende Teleskopsumme:

$$\begin{aligned}
\sum_{j=0}^{\ell(n)-1} (p_{j+1} - p_j) &= p_1 - p_0 + p_2 - p_1 + \cdots + p_{\ell(n)} - p_{\ell(n)-1} \\
&= p_{\ell(n)} - p_0 \\
&= \mathrm{Ws}[T(B_n) = 1] - \mathrm{Ws}\left[T\left(U_{\ell(n)}\right) = 1\right] \\
&\geq \delta(n)
\end{aligned} \tag{1.5}$$

Wir erhalten, daß ein $i \in \{0, 1, \ldots, \ell(n) - 1\}$ existiert mit

$$p_{i+1} - p_i \geq \frac{\delta(n)}{\ell(n)}, \tag{1.6}$$

denn sonst wäre die Summe in (1.5) kleiner als $\delta(n)$. Wir konstruieren in Algorithmus 1.4.1 einen Prediktor P_i zur Vorhersage des $(i+1)$-ten Bits.

Algorithmus 1.4.1 Prediktor P_i aus statistischem Test T

EINGABE: $\left(b_1, b_2, \ldots, b_{\ell(n)}\right) \in \{0,1\}^{\ell(n)}$

/* Sei T ein statistischer Test, den das Ensemble $(B_n)_{n\in\mathbb{N}}$ vom Typ $\ell(n)$ mit $\delta(n)$-Toleranz nicht besteht. Sei die Bitposition $i := i(n)$ wie in (1.6). */

1. FOR $j := i+1$ TO $\ell(n)$ DO wähle $h_j \in_{\mathrm{R}} \{0,1\}$ zufällig

2. IF $T\left(b_1, \ldots, b_i, h_{i+1}, \ldots, h_{\ell(n)}\right) = 1$ THEN Ausgabe h_{i+1} ELSE Ausgabe $1 - h_{i+1}$

Sei P_i der Algorithmus 1.4.1. Offenbar ist die Laufzeit des Algorithmus' $|P_i| = |T| + \mathcal{O}(\ell(n))$. Wir analysieren den Vorteil des Prediktors P_i bei der Vorhersage des $(i+1)$-ten Bits des Ensembles $(B_n)_{n\in\mathbb{N}}$. Es gilt für gemäß B_n verteiltes $\left(b_1, b_2, \ldots, b_{\ell(n)}\right) \in \{0,1\}^{\ell(n)}$ und zufällige Bits $h_{i+1}, \ldots, h_{\ell(n)} \in_{\mathrm{R}} \{0,1\}$

$$p_{i+1} = \mathrm{Ws}\left[T\left(b_1, \ldots, b_i, h_{i+1}, \ldots, h_{\ell(n)}\right) = 1 \;\middle|\; b_{i+1} = h_{i+1}\right],$$

da $\left(b_1, \ldots, b_{i+1}, h_{i+2}, \ldots, h_{\ell(n)}\right)$ gemäß der hybriden Verteilung $H_n^{(i+1)}$ verteilt ist. Wir setzen:

$$q_{i+1} := \mathrm{Ws}\left[T\left(b_1, \ldots, b_i, h_{i+1}, \ldots, h_{\ell(n)}\right) = 1 \;\middle|\; b_{i+1} \neq h_{i+1}\right]$$

Die Bitfolge $(b_1, \ldots, b_i, h_{i+1}, \ldots, h_{\ell(n)})$ ist gemäß der hybriden Verteilung $H_n^{(i)}$ verteilt. Nach Definition (1.4) und wegen $h_{i+1} \in_{\mathrm{R}} \{0,1\}$ folgt:

$$p_i = \mathrm{Ws}\left[T\left(b_1, \ldots, b_i, h_{i+1}, \ldots, h_{\ell(n)}\right) = 1\right] = \tfrac{1}{2}\left(p_{i+1} + q_{i+1}\right) \tag{1.7}$$

Für die Analyse des Vorteils des Prediktors P_i zur Vorhersage des $(i+1)$-ten Bits unterscheiden wir zwei Fälle, nämlich ob das zufällige Bit h_{i+1} gleich b_{i+1} ist oder nicht:

- Falls h_{i+1} gleich b_{i+1} ist, sagt Algorithmus 1.4.1 genau dann das Bit b_{i+1} richtig voraus, wenn der statistische Test T den Wert 1 ausgibt.
- Falls h_{i+1} ungleich b_{i+1} ist, also $b_{i+1} = 1 - h_{i+1}$, sagt Algorithmus 1.4.1 genau dann das Bit b_{i+1} richtig voraus, wenn der statistische Test T den Wert 0 ausgibt.

Wir erhalten als Erfolgswahrscheinlichkeit des Prediktors P_i:

$$\begin{aligned}
\mathrm{Ws}\left[P_i\left(b_1, b_2, \ldots, b_{\ell(n)}\right) = b_{i+1}\right] &= \tfrac{1}{2} \cdot \overbrace{\mathrm{Ws}\left[T\left(b_1, \ldots, b_i, h_{i+1}, \ldots, h_{\ell(n)}\right) = 0 \;\middle|\; h_{i+1} \neq b_{i+1}\right]}^{=1-q_{i+1}} \\
&\quad + \tfrac{1}{2} \cdot \mathrm{Ws}\left[T\left(b_1, \ldots, b_i, h_{i+1}, \ldots, h_{\ell(n)}\right) = 1 \;\middle|\; h_{i+1} = b_{i+1}\right] \\
&= \tfrac{1}{2} - \tfrac{1}{2} q_{i+1} + \tfrac{1}{2} p_{i+1} \\
&= \tfrac{1}{2} + \tfrac{1}{2}\left(p_{i+1} - q_{i+1}\right)
\end{aligned}$$

Aus Gleichung (1.7) und der Voraussetzung (1.6) folgt

$$\tfrac{1}{2}\left(p_{i+1} - q_{i+1}\right) = p_{i+1} - \tfrac{1}{2}\left(p_{i+1} + q_{i+1}\right) = p_{i+1} - p_i \geq \frac{\delta(n)}{\ell(n)},$$

so daß der Prediktor P_i bei der Vorhersage des $(i+1)$-ten Bits des Ensembles $(B_n)_{n\in\mathbb{N}}$ den Vorteil $\epsilon(n) := \frac{\delta(n)}{\ell(n)}$ hat. ■

Insbesondere gibt es nach Lemma 1.4.5 für Ensemble vom Typ $\ell(n)$, das einen statistischenTest T mit $\delta(n)$-Toleranz nicht besteht, einen Prediktor P zur Vorhersage mit Vorteil $\frac{\epsilon(n)}{\ell(n)} = \frac{\delta(n)}{\ell(n)}^2$.

Wir fassen Lemma 1.4.4 und 1.4.5 zusammen und erhalten eine äquivalente Charakterisierung eines pseudozufälligen Ensembles:

Satz 1.4.6 (Yao 1982)
Ein Ensemble ist genau dann unvorhersagbar, wenn es pseudozufällig ist.

Beweis. Sei $(B_n)_{n\in\mathbb{N}}$ ein Ensemble vom Typ $\ell(n)$. Wir zeigen, daß $(B_n)_{n\in\mathbb{N}}$ genau dann vorhersagbar ist, wenn es nicht pseudozufällig ist.

„⇒“ Sei $(B_n)_{n\in\mathbb{N}}$ vorhersagbar, d.h. es existiert ein Prediktor mit Vorteil n^{-c} für ein festes $c \in \mathbb{N}$ bei der Vorhersage des $(i+1)$-ten Bits. Nach Lemma 1.4.4 existiert ein statistischer Test, den das Ensemble mit n^{-c}-Tolereranz nicht besteht.

„⇐“ Sei $(B_n)_{n\in\mathbb{N}}$ nicht pseudozufällig, d.h. es existiert ein statistischer Test, den das Ensemble mit n^{-c}-Toleranz für ein festes $c \in \mathbb{N}$ nicht besteht. Nach Lemma 1.4.5 gibt es einen Prediktor P mit Vorteil mindestens $\frac{n^{-c}}{\ell(n)^2}$. Da $\ell(n)$ durch ein Polynom beschränkt ist, existiert eine Konstante $d \in \mathbb{N}$ mit $\ell(n) \leq n^d$ für hinreichend große n. Der Prediktor P hat als Vorteil bei der Vorsage mindestens $n^{-(c+2d)}$ für feste $c, d \in \mathbb{N}$.

Dies war zu zeigen. ■

Wir haben die Vorhersagbarkeit nach rechts definiert. Ein Prediktor P_i^l zur Vorhersage des $(i-1)$-ten Bits nach links eines Ensembles $(B_n)_{n\in\mathbb{N}}$ des Typs $\ell(n)$ liest von der Eingabe $\left(b_1, b_2, \ldots, b_{\ell(n)}\right)$ nur die Bits $b_i, b_{i+1}, \ldots, b_{\ell(n)}$, um b_{i-1} zu berechnen. Den Vorteil des Prediktors definieren wir analog zur Definition 1.4.2 auf Seite 16. Es gilt:

Lemma 1.4.7
Sei $(B_n)_{n\in\mathbb{N}}$ ein Ensemble vom Typ $\ell(n)$, das einen statistischen Test T mit $\delta(n)$-Toleranz nicht besteht. Dann existieren eine Bitposition $i = i(n)$ mit $1 < i \leq \ell(n)+1$ und ein Prediktor P_i^l zur Vorhersage des $(i-1)$-ten Bits nach links mit Vorteil $\epsilon(n) := \frac{\delta(n)}{\ell(n)}$ und Laufzeit $|P_i^l| = |T| + \mathcal{O}(\ell(n))$.

Beweis. Wir betrachten das Ensemble $\left(B_n^{\mathrm{SP}}\right)_{n\in\mathbb{N}}$ der Verteilungen B_n^{SP}, die wir erhalten, wenn wir die Reihenfolge der gemäß B_n verteilten Bitfolge $\left(b_1, b_2, \ldots, b_{\ell(n)}\right)$ umkehren, also $\left(b_{\ell(n)}, b_{\ell(n)-1}, \ldots, b_1\right)$ betrachten. Das Ensemble $\left(B_n^{\mathrm{SP}}\right)_{n\in\mathbb{N}}$ besteht den statistischen Test

$$T^{\mathrm{SP}}\left(b_1^{\mathrm{SP}}, b_2^{\mathrm{SP}}, \ldots, b_{\ell(n)}^{\mathrm{SP}}\right) := T\left(b_{\ell(n)}^{\mathrm{SP}}, b_{\ell(n)-1}^{\mathrm{SP}}, \ldots, b_1^{\mathrm{SP}}\right)$$

mit $\delta(n)$-Toleranz nicht. Nach Lemma 1.4.5 existieren eine Bitposition i und ein Prediktor P_i, der zu $\left(b^{\mathrm{SP}}_{\ell(n)}, b^{\mathrm{SP}}_{\ell(n)-1}, \ldots, b^{\mathrm{SP}}_{i}\right)$ das $(i+1)$-te Bit, also b^{SP}_{i-1}, mit Vorteil $\epsilon(n) := \frac{\delta(n)}{\ell(n)}$ nach rechts voraus. Dieser Prediktor P_i sagt das $(i-1)$-te Bit des Ensembles $(B_n)_{n\in\mathbb{N}}$ mit Vorteil $\epsilon(n)$ nach links voraus. Der Prediktor P_i^l kehrt in $\mathcal{O}(\ell(n))$ Schritten die Reihenfolge der Eingabe um (genau die Bitpositionen i bis $\ell(n)$ und ergänzt $\ell(n) - i$ Nullbits) und ruft P_i auf. Wir erhalten

$$|P_i^l| = \underbrace{|T| + \mathcal{O}(\ell(n))}_{=|T^{\mathrm{SP}}|} + \mathcal{O}(\ell(n)) = |T| + \mathcal{O}(\ell(n))$$

als Laufzeit des Prediktors P_i^l. ■

Da sich die Konstruktion aus Lemma 1.4.4 unmittelbar auf Vorhersage nach links überträgt, gilt:

Korollar 1.4.8
Ein Ensemble ist genau dann pseudozufällig, wenn es nach links unvorhersagbar ist.

Wir betrachten Ensembles mit einer speziellen Eigenschaft. Sei $\ell(n)$ der Typ des Ensembles $(B_n)_{n\in\mathbb{N}}$. Für $i = 1, 2, \ldots, \ell(n) - 1$ soll jeweils jede zusammenhängende Teilfolge einer gemäß B_n verteilten Bitfolge die gleiche Wahrscheinlichkeitsverteilung haben. Die Verteilung einer zusammenhängenden Teilfolge bleibt invariant, wenn wir die Teilfolge um eine Position „nach rechts“ verschieben (engl. to shift). Ein solches Ensemble nennen wir shift-symmetrisch. Wir verallgemeinern diese Eigenschaft: Bei einem $k(n)$-shift-symmetrisch Ensemble bleibt die Verteilung einer zusammenhängenden Teilfolge invariant, wenn wir die Teilfolge um $k(n)$ Position „nach rechts“ verschieben.

Definition 1.4.9 (Shift-symmetrisches Ensemble)
Ein Ensemble $(B_n)_{n\in\mathbb{N}}$ des Typs $\ell(n)$ ist $k(n)$-shift-symmetrisch, wenn für jedes $n \in \mathbb{N}$ gilt, daß $\ell(n)$ ein Vielfaches von $k(n)$ ist und die (gemeinsamen) Verteilungen der Komponenten i und $i + k(n)$ für $1 \leq i \leq \ell(n) - k(n)$ identisch sind. Ein 1-shift-symmetrisches Ensembles nennen wir shift-symmetrisch.

Ein Beispiel für ein shift-symmetrisches Ensemble ist $(U_n)_{n\in\mathbb{N}}$, da jede Komponente mit Wahrscheinlichkeit $\frac{1}{2}$ Eins oder Null ist.

Wir lernen im weiteren eine allgemeine Klasse shift-symmetrischer Ensembles, die iterativ erzeugt werden, kennen. Zu einer Funktion $f : \{0,1\}^* \to \{0,1\}^*$ sei

$$f^i = \underbrace{f \circ f \circ \cdots \circ f}_{i\text{-mal}},$$

so daß $f^i(x)$ die Funktion f iterativ i-mal angewendet auf das Argument x ist. Die iterativen Ensembles, die die später betrachteten Pseudozufallsgeneratoren nach der Blum-Micali-Konstruktion aus Kapitel 1.5 erzeugen, sind shift-symmetrisch:

Definition 1.4.10 (Iteratives Ensemble)
Sei $\ell(n) \geq 1$ durch ein Polynom beschränkt. Sei $f : \{0,1\}^* \to \{0,1\}^*$ eine Funktion, die eingeschränkt auf $\{0,1\}^n$ eine Permutation ist und $h : \{0,1\}^* \to \{0,1\}$ ein Prädikat. Das Ensemble

$$\left(h(U_n),\ h(f(U_n)),\ h\left(f^2(U_n)\right),\ \ldots,h\left(f^{\ell(n)-1}(U_n)\right)\right)_{n\in\mathbb{N}}$$

vom Typ $\ell(n)$ heißt iterativ.

Da f eingeschränkt auf $\{0,1\}^n$ eine Permutation ist, sind die Verteilungen U_n und $f(U_n)$ identisch. Allgemein sind die Verteilungen $U_n, f(U_n), \ldots, f^{i+1}(U_n)$ gleich. Daher sind iterative Ensembles shift-symmetrisch. Diese Konstruktion kann auf $k(n)$-shift-symmetrische Ensembles erweitert werden. Dazu fordern wir, daß $h : \{0,1\}^* \to \{0,1\}^*$ längenregulär ist, also die Bilder von $x \in \{0,1\}^n$ jeweils die gleiche Länge $|h(1^n)|$ haben. Sei $k(n) := |h(1^n)|$ und $\ell(n)$ ein Vielfaches von $k(n)$. Dann ist

$$\left(h(U_n),\ h(f(U_n)),\ h\left(f^2(U_n)\right),\ \ldots,h\left(f^{\frac{\ell(n)}{k(n)}-1}(U_n)\right)\right)_{n\in\mathbb{N}}$$

ein $k(n)$-shift-symmetrisches Ensembles vom Typ $\ell(n)$.

Für $k(n)$-shift-symmetrische Ensembles des Typs $\ell(n)$ verschärfen wir die Aussage von Lemma 1.4.5 von Seite 18: Für die Bitposition zur Vorhersage müssen wir nur zwischen $k(n)$ statt $\ell(n)$ Positionen auswählen, so daß wir beim zufälligen Raten die Erfolgswahrscheinlichkeit mindestens $\frac{1}{k(n)}$ statt $\frac{1}{\ell(n)}$ haben. Für shift-symmetrische Ensembles wie zum Beispiel iterative Ensembles entfällt sogar das zufällige Raten der Bitposition zur Vorhersage.

Satz 1.4.11
Sei $(B_n)_{n\in\mathbb{N}}$ ein $k(n)$-shift-symmetrisches Ensemble vom Typ $\ell(n)$, das einen statistischen Test T mit $\delta(n)$-Toleranz nicht besteht. Dann existieren:

a) Eine Bitposition $i := i(n)$ mit $\ell(n) - k(n) \leq i(n) < \ell(n)$ und ein Prediktor P_i zur Vorhersage des $(i+1)$-ten Bits nach rechts und

b) eine Bitposition $i := i(n)$ mit $1 < i(n) \leq k(n) + 1$ und ein Prediktor P_i^l zur Vorhersage des $(i-1)$-ten Bits nach links

mit jeweils Vorteil $\epsilon(n) := \frac{\delta(n)}{\ell(n)}$ und Laufzeit $|T| + \mathcal{O}(\ell(n))$.

Beweis. Nach Lemma 1.4.5 existiert eine Bitposition $i_0 = i_0(n)$ mit $0 \leq i_0 < \ell(n)$ und ein Prediktor P'_{i_0}, der bei der Vorhersage des (i_0+1)-ten Bits den Vorteil $\epsilon(n) = \frac{\delta(n)}{\ell(n)}$ hat, mit Laufzeit $|P'_{i_0}| = |T| + \mathcal{O}(\ell(n))$ (siehe Algorithmus 1.4.1 auf Seite 19). Sei

$$i := \max\{i_0 + m \cdot k(n) \mid m \in \mathbb{N}_0, \quad i_0 + m \cdot k(n) < \ell(n)\}$$

d.h. wir erhöhen i_0 solange um $k(n)$, bis die Position $i+1$ in den letzten $k(n)$ Bitpositionen liegt. Abbildung 1.4.1 zeigt diese Situation schematisch.

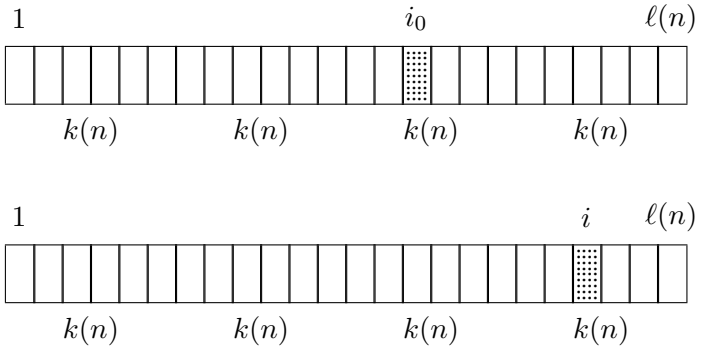

Abbildung 1.4.1: Skizze zum Beweis von Satz 1.4.11

Der Prediktor P_i zur Vorhersage des $(i+1)$-ten Bits erhält als Eingabe $b \in \{0,1\}^{\ell(n)}$, wobei er nur die ersten i Zeichen lesen darf. Er verschiebt die Eingabe b (genau die ersten i Bits) um $i-i_0$ Positionen nach vorne, der in Abbildung 1.4.1 schraffierten fällt weg. Sei b' das Resultat. Prediktor P_i ruft P'_{i_0} mit der Eingabe b' auf und gibt das Ergebnis aus. Da $(B_n)_{n\in\mathbb{N}}$ ein $k(n)$-shift-symmetrisches Ensemble ist, haben die zusammenhängenden Teilfolgen der Länge i_0, die in i_0 und i enden, die gleiche Verteilung. Wir erhalten die Behauptung a). Aussage b) folgt analog aus Lemma 1.4.7 auf Seite 20. ■

Aus Satz 1.4.11 folgt unmittelbar für shift-symmetrische, insbesondere iterative Ensembles:

Korollar 1.4.12
Sei $(B_n)_{n\in\mathbb{N}}$ ein shift-symmetrisches Ensemble vom Typ $\ell(n)$, das einen statistischen Test T mit $\delta(n)$-Toleranz nicht besteht. Dann existieren

a) ein Prediktor $P_{\ell(n)-1}$ zur Vorhersage des $\ell(n)$-ten Bits nach rechts und

b) ein Prediktor P_2^l zur Vorhersage des ersten Bits nach links

mit Vorteil $\epsilon(n) := \frac{\delta(n)}{\ell(n)}$ und Laufzeit $|T| + \mathcal{O}(\ell(n))$.

Beweis. Wende Satz 1.4.11 mit $k(n) = 1$ an. ■

1.5 Blum-Micali-Konstruktion und XOR-Bedingung

M. Blum und S. Micali [BM84] haben eine allgemeine Konstruktion für kryptographisch sichere Pseudozufallsgeneratoren aus Oneway-Permutationen und zugehörigem Hardcore-Prädikat vorgestellt. Die Konstruktion entspricht dem eines iterativen Ensembles, das wir in Definition 1.4.10 auf Seite 22 formuliert haben. Wir definieren zunächst Oneway-Funktionen und Oneway-Permutationen:

Definition 1.5.1 (Oneway-Funktion und Oneway-Permutation)
Eine Abbildung $f : \{0,1\}^* \to \{0,1\}^*$ heißt Oneway-Funktion, falls:

1. *Die Funktion f ist durch einen deterministischen Polynomialzeit-Algorithmus berechnenbar.*

2. *Für jeden probabilistischen Polynomialzeit-Algorithmus A und jedes $t \in \mathbb{N}$ existiert ein $n_0 \in \mathbb{N}$, so daß für alle $n \geq n_0$ gilt*

$$\mathrm{Ws}\left[\, A\,(1^n, f(U_n)) \in f^{-1}f(U_n)\right] < n^{-t},$$

wobei die Wahrscheinlichkeit über die Zufallsvariable und die internen Münzwürfe des Algorithmus' A gebildet wird.

Eine Oneway-Funktion f heißt Oneway-Permutation, falls $f(\{0,1\}^n) = \{0,1\}^n$ für alle $n \in \mathbb{N}$ ist.

Jeder Invertierungsalgorithmus zu einer Oneway-Funktion soll im Durchschnitt nur eine vernachlässigbare Erfolgswahrscheinlichkeit beim Finden eines Urbildes haben. Die zusätzliche Eingabe 1^n (n unär) stellt sicher, daß A in Polynomialzeit ein Urbild zur Eingabe $f(x)$ ausgeben kann, da sonst zum Beispiel $f(x) = \lfloor \log_2 x \rfloor$ eine triviale Oneway-Funktion wäre. Eine Oneway-Permutation ist eine Oneway-Funktion, die eingeschränkt auf $\{0,1\}^n$ jeweils eine Permutation ist.

Für eine Oneway-Funktion ist nicht auszuschließen, daß man effizient einzelne Bits (allgemeiner Prädikate) des Urbildes berechnen kann: Mit f ist zum Beispiel auch die Abbildung $g(x) := (\mathrm{lsb}(x), f(x))$ eine Oneway-Funktion, obwohl wir das niedrigstwertige Bit $\mathrm{lsb}(x)$ des Urbildes x kennen. Hardcore-Prädikate sollen den harten Kern von $f(x)$, an dem die Invertierung scheitert, darstellen.

Definition 1.5.2 (Hardcore-Prädikat)
Eine Abbildung $h : \{0,1\}^* \to \{0,1\}$ heißt Hardcore-Prädikat zur Funktion f, falls:

1. *Das Prädikat h ist durch einen deterministischen Polynomialzeit-Algorithmus berechnenbar.*

2. *Für jeden probabilistischen Polynomialzeit-Algorithmus A und jedes $t \in \mathbb{N}$ existiert ein $n_0 \in \mathbb{N}$, so daß für alle $n \geq n_0$ gilt*

$$\mathrm{Ws}[\, A\,(1^n, f(U_n)) = h(U_n)] < \tfrac{1}{2} + n^{-t},$$

wobei die Wahrscheinlichkeit über die Zufallsvariable und die internen Münzwürfe des Algorithmus' A gebildet wird.

Für alle bekannten Permutationen, von denen man annimmt, daß es Oneway-Funktionen sind, existieren einfache, individuelle Hardcore-Prädikate (zum Beispiel das niedrigstwertige Bit des Urbildes). Wir werden in Kapitel 1.6 ein allgemeines Hardcore-Prädikat für jede Oneway-Funktion kennenlernen.

Die Abbildung f zu einem Hardcore-Prädikat muß keine Oneway-Funktion sein. Zum Beispiel verwenden M. Naor und O. Reingold [NR95] das allgemeine Hardcore-Prädikat für

die Diffie-Hellman-Funktion [DH76], von der man voraussetzt, daß sie nicht in Polynomialzeit berechenbar ist.

Den Vorteil für die Berechnung eines Prädikats definieren wir analog zum Vorteil der Vorhersage eines Ensembles:

Definition 1.5.3 (Vorteil bei Berechnung eines Prädikats)
Ein probabilistischer Algorithmus A hat einen $\epsilon(n)$-Vorteil (gegenüber Raten) bei der Berechnung des Prädikats $h : \{0,1\}^* \to \{0,1\}$ zur Funktion f, wenn für unendlich viele n gilt

$$\mathrm{Ws}[\, A\,(1^n, f(U_n)) = h(U_n)] \geq \tfrac{1}{2} + \epsilon(n),$$

wobei die Wahrscheinlichkeit über die Zufallsvariable und die internen Münzwürfe des Algorithmus' A gebildet wird.

Für ein Hardcore-Prädikat h hat jeder Polynomialzeit-Algorithmus für die Berechnung des Prädikats h nur einen vernachlässigbaren Vorteil. Um zu beweisen, daß h ein Hardcore-Prädikat zu einer Oneway-Funktion f ist, nimmt man die Existenz eines Polynomialzeit-Algorithmus' mit Vorteil n^{-c} für das Hardcore-Prädikat h zu f an und zeigt, daß dann für unendlich viele n die Oneway-Funktion f effizient mit Wahrscheinlichkeit n^{-d} invertierbar ist (für feste $c, d \in \mathbb{N}$). Ein alternativer Beweis für geeignete Oneway-Funktion und Hardcore-Prädikat ist zu zeigen, daß wir sonst ein bekanntes Hardcore-Prädikat zu f mit nicht-vernachlässigbarem Vorteil berechnen können (siehe [FSt96, IN96]).

Algorithmus 1.5.1 Blum-Micali-Generator G^{BM} des Typs $\ell(n)$

EINGABE: zufälliger Startwert $x \in_{\mathrm{R}} \{0,1\}^n$

/* Sei f eine Oneway-Permutation, h ein Hardcore-Prädikat für f und $\ell(n) \geq n+1$ durch ein Polynom beschränkt. */

1. $x_0 := x$

2. FOR $j := 1$ TO $\ell(n)$ DO

2.1. $x_j := f(x_{j-1})$

2.2. $b_j := h(x_{j-1})$

END for

AUSGABE: $\big(b_1, b_2, \ldots, b_{\ell(n)}\big) \in \{0,1\}^{\ell(n)}$

Sei f eine Oneway-Permutation, h ein zugehöriges Hardcore-Prädikat und $\ell(n) \geq n+1$ durch ein Polynom beschränkt. Algorithmus 1.5.1 zeigt die Blum-Micali-Konstruktion für einen Pseudozufallsgenerator des Typs $\ell(n)$. Die in der Praxis verwendeten Generatoren, wie der RSA- oder der x^2-mod-N-Generator (siehe Kapitel 2 und 3) basieren auf dieser Konstruktion.

Lemma 1.5.4
Sei $\ell(n) \geq n+1$ durch ein Polynom beschränkt. Sei $f : \{0,1\}^* \to \{0,1\}^*$ eine Funktion, die eingeschränkt auf $\{0,1\}^n$ eine Permutation ist und $h : \{0,1\}^* \to \{0,1\}$ ein Prädikat. Das Ensemble

$$\left(h(U_n),\ h(f(U_n)),\ h\left(f^2(U_n)\right),\ \ldots,h\left(f^{\ell(n)-1}(U_n)\right)\right)_{n\in\mathbb{N}}$$

vom Typ $\ell(n)$ bestehe den statistischen Test T mit Toleranz $\delta(n)$ nicht. Dann existiert ein probabilistischer Algorithmus A mit Vorteil $\epsilon(n) := \frac{\delta(n)}{\ell(n)}$ für die Berechnung des Prädikats und Laufzeit $|A| = |T| + \mathcal{O}(\ell(n)\cdot(|f|+|h|))$, wobei $|f|$ und $|h|$ die Laufzeiten zur Berechnung der Funktionen seien.

Beweis. Das Ensemble ist iterativ und damit auch shift-symmetrisch. Nach Korollar 1.4.12 auf Seite 23 existiert ein Prediktor P_2^l, der in Zeit $|T| + \mathcal{O}(\ell(n))$ das erste Bit nach links mit Vorteil $\epsilon(n) = \frac{\delta(n)}{\ell(n)}$ vorhersagt, d.h. Algorithmus P_2^l bestimmt für unendlich viele n zu zufälligem $x \in_{\mathrm{R}} \{0,1\}^n$ aus

$$h(f(x)),\ h\left(f^2(x)\right),\ \ldots,h\left(f^{\ell(n)-1}(x)\right) \tag{1.8}$$

das Prädikat $h(x)$ mit Wahrscheinlichkeit mindestens $\frac{1}{2}+\epsilon(n)$. Der Algorithmus A erzeugt die Folge (1.8) in Zeit $\mathcal{O}(\ell(n)\cdot(|f|+|h|))$ und ruft den Prediktor P_2^l auf. Die Ausgabe des Algorithmus' A ist das Resultat des Prediktors. ■

Wir erhalten aus Lemma 1.5.4 die Korrektkeit der Konstruktion in Algorithmus 1.5.1. Es genügt für den Blum-Micali-Generator zu zeigen, daß jeder probabilistische Polynomialzeit-Algorithmus zur Berechnung von $h(x)$ bei gegebenem $f(x)$ nur einen vernachlässigbaren Vorteil hat:

Satz 1.5.5 (Blum, Micali 1982)
Sei $\ell(n) \geq n+1$ durch ein Polynom beschränkt und f eine Oneway-Permutation mit zugehörigem Hardcore-Prädikat h. Der Blum-Micali-Generator G^{BM}, Algorithmus 1.5.1, ist ein kryptographisch sicherer Pseudozufallsgenerator des Typs $\ell(n)$.

Beweis. Die Laufzeit des Generators G ist polynomiell in n, da f und h durch deterministische Polynomialzeit-Algorithmen berechbar sind. Da $\ell(n)$ durch ein Polynom beschränkt ist, gibt es eine Konstante $d \in \mathbb{N}$ mit $\ell(n) \leq n^d$ für hinreichend große n. Angenommen, das Ensemble $\left(G^{\mathrm{BM}}(U_n)\right)_{n\in\mathbb{N}}$ bestehe einen statistischen Test T mit Toleranz $\delta(n) := n^{-c}$ für ein festes $c \in \mathbb{N}$ nicht. Aus Lemma 1.5.4 erhalten wir einen Polynomialzeit-Algorithmus A, der für unendlich viele n zu $f(x)$ mit zufälligem $x \in_{\mathrm{R}} \{0,1\}^n$ das Hardcore-Prädikat $h(x)$ mit Vorteil $n^{-(c+d)}$ berechnet — Widerspruch zur Eigenschaft des Hardcore-Prädikats. ■

A. Yao hat 1982 auf der Konferenz „Symposium on Foundation of Computer Science" ein allgemeines Hardcore-Prädikat zu beliebiger Oneway-Funktion f vorgestellt. Man definiert eine Oneway-Funktion f_{xor}, die das Argument der Länge n^4 in n^3 Teilblöcke zu je n Bits zerlegt und auf diese die Oneway-Funktion f anwendet:

$$f_{\mathrm{xor}}\left(x_1, x_2, \ldots, x_{n^3}\right) := \left(f(x_1), f(x_2), \ldots, f(x_{n^3})\right)$$

Dann ist die XOR-Verknüpfung aller Bits,

$$h_{\text{xor}}\left(x_1, x_2, \ldots, x_{n^3}\right) := \sum_{i=1}^{n^3} \sum_{j=1}^{n} \text{bit}_j(x_i) \bmod 2,$$

ein Hardcore-Prädikat zu f_{xor}. Der Beweis basiert auf A. Yaos XOR-Lemma, das er zwar auf der Tagung vorgetragen hat, aber im Beitrag [Yao82] zum Konferenzband fehlt. Wir verweisen auf Theorem 6.12 in Kranakis' Buch [Kranakis86] und die Übersichtsarbeit von O. Goldreich, N. Nisan und A. Wigderson [GNW95]. In Verbindung mit der Blum-Micali-Konstruktion können wir mit Yaos Hardcore-Prädikat h_{xor} aus jeder Oneway-Permutation f einen Pseudozufallsgenerator erzeugen. Bei einem n-Bit-Seed wenden wir in jeder Iteration $n^{\frac{3}{4}}$-mal die Funktion f auf Eingaben der Länge $\sqrt[4]{n}$ an. Für einen praktikablen Pseudozufallsgenerator ist die Laufzeit zu groß, zumal wir n hinreichend groß wählen müssen, so daß f für Bitlänge $\sqrt[4]{n}$ in der Praxis nicht invertierbar ist. In Kapitel 1.6 lernen wir das allgemeine Harcore-Prädikat „inneres Produkt modulo 2“ von O. Goldreich und L.A. Levin kennen. Der Blum-Micali-Pseudozufallsgenerator zu einer Oneway-Permutation f und diesem Hardcore-Prädikat wendet in jeder Iteration f auf ein Argument mit $\frac{1}{2}n$ Bits an und gibt das innere Produkt modulo 2 der Vektoren bestehend aus den ersten $\frac{1}{2}n$ und den zweiten $\frac{1}{2}n$ Bits aus. Diese Konstruktion ist für einfach zu berechnende Oneway-Permutationen praktikabel, wie wir in Kapitel 3.2 sehen werden. Die Blum-Micali-Konstruktion setzt voraus, daß die Oneway-Funktion f eingeschränkt auf $\{0,1\}^n$ eine Permutation ist. J. Håstad, R. Impagliazzo, L.A. Levin und M.Luby [HILL91] zeigen, daß man aus jeder Oneway-Funktionen einen Pseudozufallsgenerator konstruieren kann. Die allgemeine Konstruktion ist aber zu aufwendig und daher wie Yaos Hardcore-Prädikat nur von theoretischer Bedeutung.

Eine für die Praxis relevante Erweiterung ist die Anzahl pro Iteration ausgegebener Bits, d.h. wir wollen bei einer Anwendung der Oneway-Permutation mehr als ein Bit ausgeben. In Algorithmus 1.5.1 geben wir das Hardcore-Prädikat des Urbildes aus. Wir erweitern die Definition des Hardcore-Prädikat auf mehrere Bits, die simultan sicher sein sollen:

Definition 1.5.6 (Hardcore-Funktion)
Eine Abbildung $h : \{0,1\}^* \to \{0,1\}^*$ heißt Hardcore-Funktion zur Funktion f, falls:

1. *Die Funktion h ist durch einen deterministischen Polynomialzeit-Algorithmus berechnenbar.*
2. *Die Funktion h ist längenregulär, d.h. für alle $x, y \in \{0,1\}^*$ mit $|x| = |y|$ gilt $|h(x)| = |h(y)|$.*
3. *Für jeden probabilistischen Polynomialzeit-Algorithmus A und jedes $t \in \mathbb{N}$ existiert ein $n_0 \in \mathbb{N}$, so daß für alle $n \geq n_0$ gilt*

$$\left|\text{Ws}\left[A\left(1^n, f(U_n), h(U_n)\right) = 1\right] - \text{Ws}\left[A\left(1^n, f(U_n), U_{|h(1^n)|}\right) = 1\right]\right| < n^{-t},$$

wobei die Wahrscheinlichkeit über die Zufallsvariablen und die internen Münzwürfe des Algorithmus' A gebildet wird.

Durch Eigenschaft 2 ist gewährleistet, daß für alle $x \in \{0,1\}^n$ der Funktionswert $h(x)$ die gleiche Bitlänge, nämlich $|h(1^n)|$, hat. Forderung 3 besagt, daß ein statistischer Test bei

gegebenem $f(U_n)$ das Ensemble $h(U_n)$ von echten Zufallsbits praktisch nicht unterscheiden kann, also $h(U_n)$ pseudozufällig ist.

Für alle bekannten Permutationen, von denen man annimmt, daß es Oneway-Funktionen sind, existieren einfache, individuelle Hardcore-Funktionen (zum Beispiel das niedrigstwertige Bit des Urbildes), die jeweils $\mathcal{O}(\log_2 n)$ Bits liefern. Das allgemeine Hardcore-Prädikat aus Kapitel 1.6 werden wir auch auf logarithmisch viele Bits erweitern. Wir verallgemeinern Lemma 1.5.4 auf Seite 25 von Hardcore-Prädikaten zu -Funktionen:

Lemma 1.5.7
Sei $\ell(n) \geq n+1$ durch ein Polynom beschränkt. Sei $f : \{0,1\}^* \to \{0,1\}^*$ eine Funktion, die eingeschränkt auf $\{0,1\}^n$ eine Permutation ist und $h(x) = \big(h_1(x), h_2(x), \ldots, h_{k(n)}(x)\big)$, wobei $\ell(n)$ ein Vielfaches von $k(n) \geq 1$ ist. Das Ensemble

$$\Big(h(U_n),\ h(f(U_n)),\ h\left(f^2(U_n)\right),\ \ldots,h\Big(f^{\frac{\ell(n)}{k(n)}-1}(U_n)\Big)\Big)_{n\in\mathbb{N}} \tag{1.9}$$

vom Typ $\ell(n)$ bestehe den statistischen Test T mit Toleranz $\delta(n)$ nicht. Dann existieren eine Bitposition $i := i(n)$ mit $1 < i < k(n)+1$ und ein probabilistischer Algorithmus A mit Vorteil $\epsilon(n) := \frac{\delta(n)}{\ell(n)}$ für die Vorhersage des $(i-1)$-ten Bits des Ensembles $h(U_n)$ nach links, also $h_{i-1}(U_n)$, bei gegebenem $f(U_n)$ und Laufzeit $|A| = |T| + \mathcal{O}\Big(\frac{\ell(n)}{k(n)} \cdot (|f| + |h|)\Big)$, wobei $|f|$ und $|h|$ die Laufzeiten zur Berechnung der Funktionen seien.

Beweis. Das Ensemble (1.9) ist $k(n)$-shift-symmetrisch. Nach Satz 1.4.11.b) auf Seite 22 existieren eine Bitposition $i = i(n)$ mit $1 < i(n) \leq k(n)+1$ und ein Prediktor P_i^l zur Vorhersage des $(i-1)$-ten Bits nach links mit Vorteil $\epsilon(n) = \frac{\delta(n)}{\ell(n)}$, d.h. Algorithmus P_i^l bestimmt für unendlich viele n zu zufälligem $x \in_{\mathrm{R}} \{0,1\}^n$ aus

$$\underbrace{h_i(x), h_{i+1}(x) \ldots, h_{k(n)}(x)}_{\text{Suffix von } h(x)}, h(f(x)),\ h\left(f^2(x)\right),\ \ldots,h\left(f^{\frac{\ell(n)}{k(n)}-1}(x)\right) \tag{1.10}$$

das Prädikat $h_{i-1}(x)$ mit Wahrscheinlichkeit mindestens $\frac{1}{2} + \epsilon(n)$. Algorithmus A erhält als Eingabe den Suffix von $h(x)$ sowie $f(x)$, erzeugt die Folge (1.10) in Zeit $\mathcal{O}\Big(\frac{\ell(n)}{k(n)} \cdot (|f| + |h|)\Big)$ und ruft den Prediktor P_2^l auf. Die Ausgabe des Algorithmus' A ist das Resultat des Prediktors. ■

Wir modifizieren die Blum-Micali-Konstruktion für Hardcore-Funktionen, indem wir statt des Hardcore-Prädikats sämtliche Bits der Hardcore-Funktion h ausgeben und nur $\frac{\ell(n)}{|h(1^n)|}$ Iterationen ausführen.

Satz 1.5.8
Sei $\ell(n) \geq n+1$ durch ein Polynom beschränkt und ein Vielfaches von $|h(1^n)|$. Der für eine Hardcore-Funktion modifizierte Blum-Micali-Generator G^{BM}, Algorithmus 1.5.1, ist ein kryptographisch sicherer Pseudozufallsgenerator des Typs $\ell(n)$.

Beweis. Folgt aus Lemma 1.5.7. ■

Um zu zeigen, daß h eine Hardcore-Funktion ist, genügt es nicht, die Sicherheit der einzelnen Bits nach zuweisen, sondern wir müssen zeigen, daß sie simultan sicher sind, also bei gegebenem Funktionswert effizient nicht von echten Zufallsbits zu unterscheiden sind. Nach Lemma 1.4.5 bzw. Lemma 1.4.7 genügt zum Nachweis, daß h eine Hardcore-Funktion ist, die Annahme der Existenz eines Prediktors zur Vorhersage des $(i+1)$-ten Bits nach rechts des Ensembles $(h(U_n))_{n\in\mathbb{N}}$ bei gegebenem $f(U_n)$ zu einem Widerspruch zu führen. Diese Beweise sind im allgemeinen schwierig zu führen, da wir die ersten i Bits der Hardcore-Funktion kennen müssen, um den Prediktor anwenden zu können.

Wir geben ein notwendiges und hinreichendes Sicherheitskriterium für eine Hardcore-Funktion mit logarithmisch vielen Bits, die *XOR-Bedingung* von U.V. Vazirani und V.V. Vazirani [VV84]. Wir wollen zunächst eine hinreichende Bedingung herleiten, wann zu einer Oneway-Funktion f die Abbildung $h : \{0,1\}^* \to \{0,1\}^2$ mit $h(x) := (h_1(x), h_2(x))$ eine Hardcore-Funktion ist, d.h. zu gegebenem $f(U_n)$ das Ensemble der Verteilung $h(U_n)$ pseudozufällig ist. Die Abbildungen h_1 und h_2 müssen Hardcore-Prädikate sein, da wir sonst effizient $h_1(x)$ oder $h_2(x)$ aus $f(x)$ berechnen können. Angenommen, ein Prediktor P kann zu $f(x)$ und $h_1(x)$ das Prädikat $h_2(x)$ mit Vorteil $\epsilon(n)$ vorhersagen. Betrachten wir die unendlich vielen n, für die der Prediktor mit Wahrscheinlichkeit mindestens $\frac{1}{2} + \epsilon(n)$ das Prädikat richtig berechnet. Wir konstruieren zwei probabilistische Polynomialzeit-Algorithmen

- A zur Vorhersage von $h_2(x)$ und
- B zur Vorhersage von $h_1(x) \oplus h_2(x)$ (XOR-Verknüpfung),

wobei beide Algorithmen als Eingabe nur $f(x)$ erhalten und einer den Vorteil $\frac{1}{2}\epsilon(n)$ hat.

Algorithmus 1.5.2 Algorithmus A bei XOR-Bedingung

EINGABE: $f(x)$ mit $x \in \{0,1\}^n$

/* Seien h_1, h_2 Hardcore-Prädikate zur Oneway-Funktion f und P ein Prediktor, der zu $f(x)$ und $h_1(x)$ das Prädikat $h_2(x)$ mit Vorteil $\epsilon(n)$ vorhersagen. */

1. Wähle zufällige Münzwürffolge $\omega \in \{0,1\}^{|P|}$ für P.

2. $v := P(\omega, 1^n, f(x), 0)$

3. $w := P(\omega, 1^n, f(x), 1)$

4. IF $v = w$ THEN gib v aus ELSE gib ein zufälliges Bit aus.

Wir modifizieren den Prediktor: Statt interner Münzwürfe verwendet er eine zusätzliche Eingabe aus $\{0,1\}^*$. Zur Vereinfachung setzen wir voraus, daß der Prediktor P in jedem Schritt einen Münzwurf ausführt und die Laufzeit $|P|$ nur von der Bitlänge n abhängt. Wir beschreiben die Münzwurffolge für Bitlänge n durch die auf $\{0,1\}^{|P|}$ gleichverteilte Zufallsvariable ω. Wir rufen den Prediktor P mit gleicher Münzwurffolge ω und $f(x)$ einmal mit 0 und einmal mit 1 für $h_1(x)$ auf:

$$v := P(\omega, 1^n, f(x), 0)$$
$$w := P(\omega, 1^n, f(x), 1)$$

Wir definieren die Ausgaben der Algorithmen A und B, wobei ein Algorithmus v und der andere ein zufälliges Bit ausgibt:

- A gibt v aus, wenn $v = w$ und sonst ein zufälliges Bit.
- B gibt v aus, wenn $v \neq w$ und sonst ein zufälliges Bit.

Algorithmus 1.5.2 zeigt exemplarisch A. Angenommen, die Ausgabe des Prediktors P ist für Eingabe ω, $f(x)$ und $h_1(x)$ richtig:

$$h_2(x) = P\left(\omega, 1^n, f(x), h_1(x)\right)$$

Dieser Fall tritt nach Voraussetzung mit Wahrscheinlichkeit (bezogen auf die zufällige Wahl von ω und x) mindestens $\frac{1}{2} + \epsilon(n)$ ein. In dieser Situation ist mindestens einer der Werte v und w gleich $h_2(x)$. Falls $v = w$, gilt $v = h_2(x)$ und A liefert das richtige Ergebnis. Für $v \neq w$ ist genau einer der Werte v, w korrekt und B liefert das richtige Ergebnis:

- Falls $v = h_2(x)$, ist $h_1(x) = 0$ und es gilt:

$$v = h_2(x) = 0 \oplus h_2(x) = h_1(x) \oplus h_2(x)$$

- Falls $w = h_2(x)$, ist $h_1(x) = 1$ und es gilt:

$$v = 1 \oplus w = 1 \oplus h_2(x) = h_1(x) \oplus h_2(x)$$

Wenn P das korrekte Resultat liefert, ist auch die Ausgabe eines der beiden Algorithmen A und B richtig und der andere gibt ein zufälliges Bit aus. Die Summe der Erfolgswahrscheinlichkeiten beider Algorithmen ist daher mindestens:

$$\tfrac{1}{2} + \epsilon(n) + \tfrac{1}{2} = 2 \cdot \left(\tfrac{1}{2} + \tfrac{1}{2}\epsilon(n)\right)$$

Einer der beiden Polynomialzeit-Algorithmen A oder B hat den Vorteil $\frac{1}{2}\epsilon(n)$. Wir fordern daher, daß h_1, h_2 und $h_1 \oplus h_2$ Hardcore-Prädikate zu f sind. Falls ein Prediktor aus $f(x)$ und $h_1(x)$ das Prädikat $h_2(x)$ mit Vorteil n^{-c} für ein festes $c \in \mathbb{N}$ vorhersagen kann, ist h_2 oder $h_1 \oplus h_2$ mit Vorteil $\frac{1}{2}n^{-c}$ vorhersagbar — Widerspruch zur Voraussetzung. Umgekehrt, falls eines der Prädikate h_1, h_2 oder $h_1 \oplus h_2$ kein Hardcore-Prädikat ist, erhalten wir unmitellbar einen statistischen Test, der $h_1(U_n), h_2(U_n)$ von U_2 mit nicht-vernachlässigbarer Toleranz unterscheiden kann.

Wir fordern im Fall einer Hardcore-Funktion $h = (h_1, h_2)$, daß jede nicht-leere Teilmenge von h_1 und h_2 ein Hardcore-Prädikat ist. In Satz 1.5.10 werden wir diese Idee auf mehrere Hardcore-Prädikate übertragen und in Satz 1.5.14 folgern, daß logarithmisch viele Hardcore-Prädikate eine Hardcore-Funktion bilden, wenn die XOR-Verknüpfung jeder nicht-leeren Teilmenge der Prädikate ein Hardcore-Prädikat ist.

Zuvor klären wir, wie man mit vernachlässigbarer Fehlerwahrscheinlichkeit feststellt, welcher der beiden konstruierten Polynomialzeit-Algorithmen A und B den Vorteil $\frac{1}{2}\epsilon(n)$ hat. Um die Erfolgswahrscheinlichkeit von Algorithmus A zur Berechnung von $h_2(x)$ aus $f(x)$ (approximativ) zu ermitteln, gehen wir wie folgt vor:

1. Wir wählen zufällig und unabhängig polynomiell viele Werte $x_1, x_2, \ldots, x_m$ aus $\{0,1\}^n$.
2. Rufe Algorithmus A mit Eingabe $f(x_i)$ auf und vergleiche das Ergebnis mit $h_2(x_i)$.

Wir definieren Indikatorvariablen $X_1, X_2, \ldots, X_m$, wobei X_i genau dann Eins sei, wenn Algorithmus A bei Eingabe $f(x_i)$ das Prädikat $h_2(x_i)$ richtig berechnet. Wenn der Algorithmus A die Erfolgswahrscheinlichkeit μ hat, sollten etwa μm der m Tests bestanden werden. Je größer die Anzahl m der Tests, desto genauer approximiert $\sum_{i=1}^{m} X_i$ den Wert μm. Wir wählen m so, daß mit hinreichend großer Wahrscheinlichkeit die Anzahl bestander Tests um maximal $\frac{1}{8}\epsilon(n)m$ vom Erwartungswert $\mu m \geq \left(\frac{1}{2} + \frac{1}{2}\epsilon(n)\right) m$ von $\sum_{i=1}^{m} X_i$ abweicht. Wir akzeptieren Algorithmus A, wenn er mindestens

$$m \cdot \left(\tfrac{1}{2} + \tfrac{1}{2}\epsilon(n) - \tfrac{1}{8}\epsilon(n)\right)$$

Tests besteht. Dann gilt mit großer Wahrscheinlichkeit, daß die Erfolgswahrscheinlichkeit von Algorithmus A höchstens um $\frac{1}{8}\epsilon(n)$ von $\frac{1}{2} + \frac{1}{2}\epsilon(n) - \frac{1}{8}\epsilon(n)$ abweicht, also Algorithmus A den Vorteil $\frac{1}{4}\epsilon(n) = \frac{1}{2}\epsilon(n) - 2 \cdot \frac{1}{8}\epsilon(n)$ hat. Aus der Chernoff-Schranke erhalten wir eine Fehlerschranke für unsere Wahl:

Fakt 1.5.9 (Chernoff-Schranke)
Seien $X_1, X_2, \ldots, X_m$ identisch verteilte, unabhängige Indikatorvariablen mit Erwartungswert $\mu := \mathrm{E}[X_i]$. Dann gilt für jedes $t > 0$:

$$\mathrm{Ws}\left[\left|\sum_{i=1}^{m} X_i - m\mu\right| \geq tm\right] \leq 2e^{-mt^2}$$

In unserem Beispiel ist $t := \frac{1}{8}\epsilon(n)$. Wir ermitteln durch $m := 64\epsilon(n)^{-2}(n+1) \cdot \ln 2$ Tests, ob Algorithmus A den Vorteil $\frac{1}{4}\epsilon(n)$ hat. Unser Ergebnis ist mit Wahrscheinlichkeit höchstens

$$2 \cdot e^{-m \cdot \frac{1}{64}\epsilon(n)^2} = 2 \cdot e^{-\ln 2^{n+1}} = 2^{-n}$$

falsch.

Der folgende Satz überträgt die Konstruktion auf Hardcore-Funktionen mit beliebig vielen Bits. Allerdings nimmt der Vorteil exponentiell in der Bitposition des Prediktors zur Vorhersage ab.

Satz 1.5.10 (Vazirani, Vazirani 1984)
Sei f eine Funktion und n die Länge des Arguments von f. Seien $h_1, h_2, \ldots, h_{k(n)}$ Prädikate zu f und $h(x) := \left(h_1(x), \ldots, h_{k(n)}(x)\right)$. Sei $i := i(n)$ und P_i ein Prediktor mit Vorteil $\epsilon(n)$ zur Vorhersage des $(i+1)$-ten Bits des Ensembles $(h(U_n))_{n\in\mathbb{N}}$ bei gegebenem $f(U_n)$ nach rechts. Dann existiert eine Folge $(H(n))_{n\in\mathbb{N}}$ nicht-leerer Teilmengen $H(n) \subseteq \{1, 2, \ldots, i(n)+1\}$ mit $i(n)+1 \in H(n)$ und ein Algorithmus C zur Berechnung des Prädikats

$$h_{H(n)}(x) := \sum_{t\in H(n)} h_t(x) \bmod 2$$

mit Vorteil $2^{-i}\epsilon(n)$ und Laufzeit $|C| = 2^i \cdot |P_i| + \mathcal{O}\left(i2^{i+1}\right)$.

Algorithmus 1.5.3 Beweis zu Satz 1.5.10

EINGABE: ▷ Oneway-Funktion f und Hardcore-Prädikate $h_1, \ldots, h_{k(n)}$ zu f

▷ Prediktor P_i mit Vorteil $\epsilon(n)$ zur Vorhersage des $(i+1)$-ten Bits des Ensembles $(h(U_n))_{n\in\mathbb{N}}$ bei gegebenem $f(U_n)$ nach rechts.

1. $H^{(i)} := \{i+1\}$, $P^{(i)} := P_i$

2. FOR $j := i$ DOWNTO 1 DO

/∗ Invariante: $P^{(j)}$ ist probabilistischer Algorithmus mit Vorteil $2^{j-i} \cdot \epsilon(n)$ zur Berechnung des Prädikats $h_{H^{(j)}}(x) = \sum_{t\in H^{(j)}} h_t(x) \bmod 2$ bei gegebenem $h_1(x), \ldots, h_j(x)$ und $f(x)$. Die Laufzeit ist $|P^{(j)}| = 2^{i-j} \cdot |P_i| + ci\left(2^{i-j+1} - 1\right)$ für eine von i und j unabhängige Konstante c. ∗/

2.1. Konstruiere zwei probabilistische Algorithmen

- $A^{(j-1)}$ zur Vorhersage von $h_{H^{(j)}}(x)$ und
- $B^{(j-1)}$ zur Vorhersage von $h_{H^{(j)}\cup\{j\}}(x) = h_{H^{(j)}}(x) \oplus h_j(x)$

bei gegebenem $h_1(x), h_2(x), \ldots, h_{j-1}(x)$ und $f(x)$. Einer der beiden Algorithmen hat den Vorteil $\frac{1}{2} \cdot 2^{j-i} \cdot \epsilon(n)$.

2.2. IF ($A^{(j-1)}$ hat Vorteil $2^{j-1-i} \cdot \epsilon(n)$)

THEN setze $H^{(j-1)} := H^{(j)}$ und $P^{(j-1)} := A^{(j-1)}$

ELSE setze $H^{(j-1)} := H^{(j)} \cup \{j\}$ und $P^{(j-1)} := B^{(j-1)}$.

END for

AUSGABE: Menge $H(n) := H^{(0)}$ und probabilistischer Algorithmus $C := P^{(0)}$

Beweis. Wir konstruieren die Menge $H(n)$ und den probabilistischen Algorithmus C gemäß Algorithmus 1.5.3. Wir beweisen durch Induktion über die Anzahl der Iterationen der Schleife in Schritt 2, daß bei Eintritt in den Schleifenrumpf Algorithmus $P^{(j)}$ den Vorteil $2^{j-i} \cdot \epsilon(n)$ zur Berechnung des Prädikats $h_{H^{(j)}}(x) = \sum_{t\in H^{(j)}} h_t(x) \bmod 2$ bei gegebenen $f(x)$ und $h_1(x), h_2(x), \ldots, h_j(x)$ hat, sowie die Laufzeit $|P^{(j)}| = 2^{i-j} \cdot |P_i| + ci\left(2^{i-j+1} - 1\right)$ für eine von i und j unabhängige Konstante c ist.

- Induktionsverankerung $j = i$. Gemäß Eingabe ist Algorithmus $P^{(j)}$ gleich P_i, also ein probabilistischer Algorithmus zur Vorhersage des Prädikats $h_{H^{(j)}}(x) = h_{j+1}(x) \bmod 2$ bei gegebenen $f(x)$ und $h_1(x), h_2(x), \ldots, h_j(x)$ mit Vorteil $2^0 \cdot \epsilon(n)$. Die Laufzeit ist:

$$|P^{(j)}| = 2^0 \cdot |P_i| + ci\left(2^1 - 1\right) = 2^{i-j} \cdot |P_i| + ci\left(2^{i-j+1} - 1\right)$$

- Induktionsschluß von j auf $j-1$. Wir konstruieren zwei Algorithmen
  - $A^{(j-1)}$ zur Vorhersage von $h_{H^{(j)}}(x)$ und
  - $B^{(j-1)}$ zur Vorhersage von $h_{H^{(j)}}(x) \oplus h_j(x)$

bei gegebenem $h_1(x), h_2(x), \ldots, h_{j-1}(x)$ und $f(x)$. Einer der beiden Algorithmen hat den Vorteil $\frac{1}{2} \cdot 2^{j-i} \cdot \epsilon(n)$.

Wir fixieren die Münzwurffolge ω des Algorithmus' $P^{(j)}$ und rufen ihn zu $f(x)$ und $h_1(x), h_2(x), \ldots, h_{j-1}(x)$ einmal mit 0 und einmal mit 1 für $h_j(x)$ auf:

$$v := P^{(j)}\left(\omega, 1^n, f(x), h_1(x), \ldots, h_{j-1}(x), 0\right)$$
$$w := P^{(j)}\left(\omega, 1^n, f(x), h_1(x), \ldots, h_{j-1}(x), 1\right)$$

Wir definieren die Ausgabe der Algorithmen $A^{(j-1)}$ und $B^{(j-1)}$, wobei ein Algorithmus v und der andere ein zufälliges Bit ausgibt:

- $A^{(j-1)}$ gibt v aus, wenn $v = w$ und sonst ein zufälliges Bit.
- $B^{(j-1)}$ gibt v aus, wenn $v \neq w$ und sonst ein zufälliges Bit.

Angenommen, die Ausgabe des probabilistischen Algorithmus' $P^{(j)}$ ist richtig:

$$h_{H^{(j)}}(x) = P^{(j)}\left(\omega, 1^n, f(x), h_1(x), \ldots, h_j(x)\right)$$

Nach Induktionsannahme tritt dies mit Wahrscheinlichkeit (bezogen auf die zufällige Wahl von ω und x) mindestens $\frac{1}{2} + 2^{j-i}\epsilon(n)$ ein. Falls $v = w$, gibt $A^{(j-1)}$ das richtige Ergebnis, nämlich $h_{H^{(j)}}(x)$, aus. Für $v \neq w$, ist genau ein Werte v oder w korrekt, und $B^{(j-1)}$ liefert das richtige Ergebnis:

- Falls $v = h_{H^{(j)}}(x)$, ist $h_j(x) = 0$ und es gilt:

$$v = h_{H^{(j)} \cup \{j\}}(x) \oplus 0 = h_{H^{(j)}}(x) \oplus h_j(x) = h_{H^{(j)} \cup \{j\}}(x)$$

- Falls $w = h_{H^{(j)}}(x)$, ist $h_j(x) = 1$ und es gilt:

$$v = h_{H^{(j)}}(x) \oplus 1 = h_{H^{(j)}}(x) \oplus h_j(x) = h_{H^{(j)} \cup \{j\}}(x)$$

Wenn $P^{(j)}$ das korrekte Resultat liefert, ist auch die Ausgabe einer der beiden Algorithmen $A^{(j-1)}$ und $B^{(j-1)}$ richtig und der andere gibt ein zufälliges Bit aus. Die Summe der Erfolgswahrscheinlichkeiten beider Algorithmen ist nach Induktionsannahme:

$$\tfrac{1}{2} + 2^{j-i} \cdot \epsilon(n) + \tfrac{1}{2} = 2 \cdot \left(\tfrac{1}{2} + 2^{j-1-i} \cdot \epsilon(n)\right)$$

Einer der beiden Algorithmen $A^{(j-1)}$ oder $B^{(j-1)}$ hat den Vorteil $2^{(j-1)-i} \cdot \epsilon(n)$ zur Vorhersage von $h_{H^{(j-1)}}(x)$ bei gegebenem $h_1(x), \ldots, h_{j-1}(x)$ und $f(x)$.

Für die Laufzeit beachten wir, daß die Algorithmen $A^{(j-1)}$ und $B^{(j-1)}$ jeweils zweimal den Algorithmus $P^{(j)}$, der nach Induktionsannahme die Laufzeit $2^{i-j} \cdot |P_i| + ci2^{i-j+1}$ hat, aufrufen und die Argumente in ci Schritten erzeugt werden können. Daher hat Algorithmus $P^{(j-1)}$ die Laufzeit:

$$|P^{(j-1)}| = 2^{i-j+1} \cdot |P_i| + ci\left(2^{i-j+2} - 1\right) + ci = 2^{i-(j-1)} \cdot |P_i| + ci2^{i-(j-1)+1}$$

Damit erfüllt die Ausgabe des Algorithmus' 1.5.3 die Behauptung. ■

In Satz 1.5.10 haben wir nur die Existenz der Folge $(H(n))_{n\in\mathbb{N}}$ und eines zugehörigen Algorithmus' zur Berechnung des Prädikats $h_{H(n)}$ behauptet. Wir können in Algorithmus 1.5.3 durch unabhängige Tests in Schritt 2.2 mit exponentiell kleiner Fehlerwahrscheinlichkeit entscheiden, ob Algorithmus $A^{(j)}$ den Vorteil $\frac{1}{4}\cdot 2^{j-i}\cdot\epsilon(n)$ hat. Insgesamt erhalten wir in Polynomialzeit mit exponentiell kleiner Fehlerwahrscheinlichkeit einen Algorithmus mit Vorteil $2^{-2i}\epsilon(n)$ zur Berechnung des Prädikats $h_{H(n)}$. Für $k(n)=\mathcal{O}(\log_2 n)$ können wir in Polynomialzeit alle maximal $2^{i(n)}-1$ nicht-leeren Teilmengen $H(n)\subseteq\{1,2,\ldots,i(n)+1\}$ mit $i(n)+1\in H(n)$ und zugehörige Algorithmen zur Berechnung von $h_{H(n)}(x)$ testen, ob diese den Vorteil $2^{-i-1}\epsilon(n)$ haben.

Algorithmus 1.5.4 Prediktor für Prädikat $h_{H(n)}$

EINGABE: ▷ $f(x)$ mit $x\in\{0,1\}^n$
▷ nicht-leere Teilmenge $H\subseteq\{1,2,\ldots,k(n)\}$

/∗ Seien $h_1,\ldots,h_{k(n)}$ Prädikate zur Funktion f und $h(x):=\big(h_1(x),\ldots,h_{k(n)}(x)\big)$. Sei T ein probabilistischer Algorithmus mit

$$\mathrm{Ws}\big[T\left(f(U_n),h(U_n)\right)\big]-\mathrm{Ws}\big[T\left(f(U_n),U_{k(n)}\right)\big]\geq\delta(n)$$

für unendlich viele n. ∗/

1. Wähle zufälliges $r=\big(r_1,r_2,\ldots,r_{k(n)}\big)\in_{\mathrm{R}}\{0,1\}^{k(n)}$.

2. IF $T(f(x),r)=\sum_{t\in H}r_t \bmod 2$ THEN gib 1 aus ELSE gib 0 aus.

/∗ Wir geben $T(f(x),r)\oplus\big(\sum_{t\in H}r_t \bmod 2\big)\oplus 1$ aus: ∗/

Wir geben einen alternativen Beweis aus [Goldreich95a, Goldreich95b]. Im Vergleich zu Algorithmus 1.5.3 und Satz 1.5.10 ist die Aussage des Satzes zwar schärfer, liefert aber keine Motivation.

Satz 1.5.11
Sei f eine Funktion und n die Länge des Arguments von f. Seien $h_1,h_2,\ldots,h_{k(n)}$ Prädikate zu f und $h(x):=\big(h_1(x),\ldots,h_{k(n)}(x)\big)$. Sei T ein probabilistischer Algorithmus mit

$$\mathrm{Ws}\big[T\left(f(U_n),h(U_n)\right)\big]-\mathrm{Ws}\big[T\left(f(U_n),U_{k(n)}\right)\big]\geq\delta(n) \tag{1.11}$$

für unendlich viele n. Dann bestimmt Algorithmus 1.5.4 in Zeit $|T|+\mathcal{O}(k(n))$ zu $f(x)$ und nicht-leerer Teilmenge $H(n)\subseteq\{1,2,\ldots,k(n)\}$ das Prädikat

$$h_{H(n)}(x):=\sum_{t\in H(n)}h_t(x) \bmod 2$$

mit Vorteil $\epsilon(n):=\frac{\delta(n)}{2^{k(n)-1}}$, wobei die Wahrscheinlichkeit sich auf die zufällige Wahl von $x\in_{\mathrm{R}}\{0,1\}^n$ und die der nicht-leeren Teilmenge $H(n)$ sowie auf die internen Münzwürfe des Algorithmus' bezieht.

Beweis. Sei $K(n)$ die Menge aller nicht-leeren Teilmengen von $\{1,2,\ldots,k(n)\}$, also:

$$K(n)=\mathrm{Pot}\left(\{1,2,\ldots,k(n)\}\right)\setminus\{\emptyset\}$$

Insbesondere gilt $|K(n)| = 2^{k(n)} - 1$. Sei H_n die Gleichverteilung auf $K(n)$ und A der Algorithmus 1.5.4. Wir betrachten den Vorteil $\epsilon(x)$ für ein beliebiges, fest gewähltes $x \in \{0,1\}^n$ und zeigen, daß für die unendlich vielen n mit Eigenschaft (1.11) gilt:

$$\epsilon(x) = \mathrm{Ws}\left[A\left(f(x), H_n\right) = \sum_{t \in H_n} h_t(x) \bmod 2\right] - \frac{1}{2} \geq \frac{\delta(x)}{2^{k(n)} - 1} \tag{1.12}$$

Da $\epsilon(n)$ und $\delta(n)$ die Mittelwerte aller $\epsilon(x)$ bzw. $\delta(x)$ über $x \in \{0,1\}^n$ sind, erhalten wir die Behauptung des Satzes aus Abschätzung (1.12). Die Laufzeit des Algorithmus' A ist offenbar $|A| = |T| + \mathcal{O}(k(n))$.

Um die Aussage (1.12) zu beweisen, führen wir folgende Notation ein: Zu $H \in K(n)$ und $a, b \in \{0,1\}^n$ schreiben wir $a \equiv_H b$ bzw. $a \not\equiv_H b$ gemäß:

$$a \equiv_H b \quad \Longleftrightarrow \quad \sum_{t \in H} h_t(a) = \sum_{t \in H} h_t(b) \bmod 2$$

Zu $h(x)$ mit festem $x \in \{0,1\}^n$ gilt genau für $2^{k(n)-1}$ der $r \in \{0,1\}^{k(n)}$, daß $r \equiv_H h(x)$ ist:

$$\mathrm{Ws}\left[U_{k(n)} \equiv_H h(x)\right] = \mathrm{Ws}\left[U_{k(n)} \not\equiv_H h(x)\right] = \tfrac{1}{2} \tag{1.13}$$

Algorithmus A gibt zu f und H genau in den folgenden beiden Fällen $\sum_{t \in H} h_t(x) \bmod 2$ aus:

a) $T\left(f(x), r\right) = 1$ und $r \equiv_H h(x)$

b) $T\left(f(x), r\right) = 0$ und $r \not\equiv_H h(x)$

Für festes $x \in \{0,1\}^n$ gilt wegen (1.13):

$$\begin{aligned}
\tfrac{1}{2} + \epsilon(x) &= \frac{1}{|K(n)|} \sum_{H \in K(n)} \mathrm{Ws}\left[A\left(f(x), H\right) = \sum_{t \in H} h_t(x) \bmod 2\right] \\
&= \frac{1}{2 \cdot |K(n)|} \sum_{H \in K(n)} \mathrm{Ws}\left[T\left(f(x), U_{k(n)}\right) = 1 \;\middle|\; U_{k(n)} \equiv_H h(x)\right] \\
&\quad + \frac{1}{2 \cdot |K(n)|} \sum_{H \in K(n)} \underbrace{\mathrm{Ws}\left[T\left(f(x), U_{k(n)}\right) = 0 \;\middle|\; U_{k(n)} \not\equiv_H h(x)\right]}_{=1 - \mathrm{Ws}\left[T\left(f(x), U_{k(n)}\right) = 1 \;\middle|\; U_{k(n)} \not\equiv_H h(x)\right]}
\end{aligned}$$

Wir erhalten:

$$\begin{aligned}
\epsilon(x) = &\quad \frac{1}{2 \cdot |K(n)| \cdot 2^{k(n)-1}} \sum_{H \in K(n)} \sum_{\substack{r \in \{0,1\}^{k(n)} \\ r \equiv_H h(x)}} \mathrm{Ws}[T\left(f(x), r\right) = 1] \\
&- \frac{1}{2 \cdot |K(n)| \cdot 2^{k(n)-1}} \sum_{H \in K(n)} \sum_{\substack{r \in \{0,1\}^{k(n)} \\ r \not\equiv_H h(x)}} \mathrm{Ws}[T\left(f(x), r\right) = 1]
\end{aligned} \tag{1.14}$$

Wir wollen die Reihenfolge der Summationsbedingungen vertauschen. Dazu setzen wir zu $r, z \in \{0,1\}^{k(n)}$:

$$G(r,z) := \{H \in K(n) \mid r \equiv_H z\}$$
$$V(r,z) := \{H \in K(n) \mid r \not\equiv_H z\}$$

Mit dieser Notation können wir (1.14) schreiben als:

$$\begin{aligned}\epsilon(x) = \quad & \frac{1}{|K(n)| \cdot 2^{k(n)}} \sum_{r \in \{0,1\}^{k(n)}} \sum_{H \in G(r,h(x))} \mathrm{Ws}[T(f(x),r) = 1] \\ & - \frac{1}{|K(n)| \cdot 2^{k(n)}} \sum_{r \in \{0,1\}^{k(n)}} \sum_{H \in V(r,h(x))} \mathrm{Ws}[T(f(x),r) = 1]\end{aligned} \tag{1.15}$$

Für $r \neq z$ gilt $|V(r,z)| = 2^{k(n)-1}$ und $|G(r,z)| = 2^{k(n)-1} - 1$. Ferner ist $V(r,r) = \emptyset$ und $G(r,r) = K(n)$. Aus (1.15) folgt:

$$\begin{aligned}\epsilon(x) = \quad & \frac{1}{|K(n)| \cdot 2^{k(n)}} \sum_{\substack{r \in \{0,1\}^{k(n)} \\ r \neq h(x)}} \left(2^{k(n)-1} - 1\right) \cdot \mathrm{Ws}[T(f(x),r) = 1] \\ & - \frac{1}{|K(n)| \cdot 2^{k(n)}} \sum_{\substack{r \in \{0,1\}^{k(n)} \\ r \neq h(x)}} 2^{k(n)-1} \cdot \mathrm{Ws}[T(f(x),r) = 1] \\ & + \frac{|K(n)|}{|K(n)| \cdot 2^{k(n)}} \cdot \mathrm{Ws}[T(f(x),h(x)) = 1]\end{aligned} \tag{1.16}$$

Wir erhalten:

$$\epsilon(x) = \frac{|K(n)|+1}{|K(n)| \cdot 2^{k(n)}} \cdot \mathrm{Ws}[T(f(x),h(x)) = 1] - \frac{1}{|K(n)| \cdot 2^{k(n)}} \sum_{r \in \{0,1\}^{k(n)}} \mathrm{Ws}[T(f(x),r) = 1]$$

Wegen $|K(n)| = 2^{k(n)} - 1$ folgt

$$\epsilon(x) = \frac{1}{2^{k(n)} - 1} \cdot \left(\mathrm{Ws}[T(f(x),h(x)) = 1] - \mathrm{Ws}\left[T\left(f(x), U_{k(n)}\right) = 1\right]\right)$$

und es gilt nach Voraussetzung für unendlich viele n die Behauptung (1.12). ■

Wir erhalten:

Korollar 1.5.12
Sei f eine Oneway-Funktion und n die Länge des Arguments von f. Seien $h_1, h_2, \ldots, h_{k(n)}$ mit $k(n) = \mathcal{O}(\log_2 n)$ Hardcore-Prädikate zu f und $h(x) := \left(h_1(x), \ldots, h_{k(n)}(x)\right)$. Sei T ein probabilistischer Polynomialzeit-Algorithmus mit

$$\mathrm{Ws}[T(1^n, f(U_n), h(U_n))] - \mathrm{Ws}\left[T\left(1^n, f(U_n), U_{k(n)}\right)\right] \geq \delta(n)$$

für unendlich viele n. Dann können wir bezüglich n und $\delta(n)^{-1}$ in Polynomialzeit mit Wahrscheinlichkeit mindestens $1-2^n$ eine nicht-leere Teilmenge $H(n) \subseteq \{1,2,\ldots,k(n)\}$ bestimmen, so daß Algorithmus 1.5.4 den Vorteil $\epsilon(n) := \frac{\delta(n)}{2^{k(n)+1}}$ bei der Vorhersage des Prädikats

$$h_{H(n)}(x) := \sum_{t \in H(n)} h_t(x) \bmod 2$$

hat, wobei sich die Wahrscheinlichkeit auf die zufällige Wahl von $x \in_{\mathrm{R}} \{0,1\}^n$ und die internen Münzwürfe des Algorithmus' bezieht. Insbesondere hat Algorithmus 1.5.4 polynomielle Laufzeit.

Beweis. Wir überprüfen jede der $2^{k(n)}-1$ nicht-leeren Teilmengen $H(n) \subseteq \{1,2,\ldots,k(n)\}$. Wegen $k(n) = \mathcal{O}(\log_2 n)$ sind dies $n^{\mathcal{O}(1)}$-viele Teilmengen. Wir bestimmen durch

$$m := 2^{2(k(n)+2)}\delta(n)^{-2} \cdot (n+k(n)) \ln 2$$

unabhängige Tests, ob Algorithmus 1.5.4 zur Berechnung von $h_{H(n)}$ den Vorteil $2^{-k(n)}\delta(n)$ hat. Wir geben $H(n)$ aus, wenn der Algorithmus mindestens

$$m \cdot \left(\tfrac{1}{2} + 2^{-k(n)}\delta(n) - \tfrac{1}{4} \cdot 2^{-k(n)}\delta(n)\right)$$

Tests besteht. Mit $t := 2^{-(k(n)+2)}\delta(n)$ folgt aus der Chernoff-Schranke 1.5.9 auf Seite 31, daß die Fehlerwahrscheinlichkeit unserer Entscheidung bezüglich einer Teilmenge $H(n)$ jeweils höchstens

$$2 \cdot e^{-mt^2} = 2 \cdot e^{-\ln 2^{n+k(n)}} = 2^{-(n+k(n)-1)} = 2^{-(k(n)-1)} \cdot 2^{-n}$$

ist. Da es insgesamt $2^{k(n)-1}$ Teilmengen gibt, folgt die behauptete Schranke 2^{-n} für die Fehlerwahrscheinlichkeit durch Summation.

Für die Laufzeit beachte, daß wegen $k(n) = \mathcal{O}(\log_2 n)$ wir $n^{\mathcal{O}(1)}$ viele Teilmengen betrachten und nach Voraussetzung die Funktionen f und h bezüglich n in Polynomialzeit berechnet werden können. ■

Wir definieren die XOR-Bedingung von U.V. Vazirani und V.V. Vazirani als Kriterium für die simultane Sicherheit von Prädikaten:

Definition 1.5.13 (XOR-Bedingung für Prädikate)
Sei f eine Oneway-Funktion und n die Länge des Arguments von f. Die $k(n) = \mathcal{O}(\log_2 n)$ Prädikate $h_1, h_2, \ldots, h_{k(n)}$ zu f erfüllen die XOR-Bedingung, falls jede Folge $(H(n))_{n\in\mathbb{N}}$ nicht-leerer Teilmengen $H(n) \subseteq \{1,2,\ldots,k(n)\}$ die XOR-Verknüpfung der Prädikate, also

$$h_{H(n)}(x) := \sum_{t \in H(n)} h_t(x) \bmod 2$$

ein Hardcore-Prädikat zu f ist.

Um zu zeigen, daß h mit $|h(1^n)| = \mathcal{O}(\log_2 n)$ eine Hardcore-Funktion ist, genügt der Nachweis der XOR-Bedingung. Der folgende Satz heißt in der Literatur auch XOR-Lemma von U.V. Vazirani und V.V. Vazirani.

Satz 1.5.14 (Vazirani, Vazirani 1984)
Sei f eine Oneway-Funktion und n die Länge des Arguments von f. Genau dann, wenn die $k(n) = \mathcal{O}(\log_2 n)$ Prädikate $h_1, h_2, \ldots, h_{k(n)}$ zu f die XOR-Bedingung erfüllen, ist

$$h(x) := \big(h_1(x), \ldots, h_{k(n)}(x)\big)$$

eine Hardcore-Funktion zur Oneway-Funktion f.

Beweis. Sei $k(n) \leq c \cdot \log_2 n$ für eine Konstante $c \in \mathbb{N}$ und hinreichend große n. Offenbar ist die XOR-Bedingung ein notwendiges Kriterium. Sie ist auch hinreichend, denn angenommen, die Prädikate $h_1, h_2, \ldots, h_{k(n)}$ zu f erfüllten die XOR-Bedingung, bilden aber keine Hardcore-Funktion. Dann existieren eine Konstante $d \in \mathbb{N}$ und ein probabilistischer Polynomialzeit-Algorithmus T mit

$$\mathrm{Ws}[T\,(1^n, f(U_n), h(U_n))] - \mathrm{Ws}\left[T\left(1^n, f(U_n), U_{k(n)}\right)\right] \geq n^{-d}$$

für unendlich viele n. Nach Satz 1.5.11 und Korollar 1.5.12 können wir in Polynomialzeit eine Teilmenge $H(n) \subseteq \{1, 2, \ldots, k(n)\}$ bestimmen, so daß Algorithmus 1.5.4 zur Berechnung des Prädikats

$$h_{H(n)}(x) := \sum_{t \in H(n)} h_t(x) \bmod 2$$

den Vorteil $n^{-(c+d)}$ hat — Widerspruch zur XOR-Bedingung. ■

1.6 Hardcore-Prädikat für beliebige Oneway-Funktion

In Kapitel 1.5 haben wir die Blum-Micali-Konstruktion für einen Pseudorandom-Generator aus einer Oneway-Permutation und zugehörigem Hardcore-Prädikat (bzw. -Funktion) kennengelernt. Wir stellen das allgemeine Hardcore-Prädikat „inneres Produkt modulo 2“ von O. Goldreich und L.A. Levin [GL89] sowie die Erweiterung auf eine Hardcore-Funktion mit logarithmisch vielen Bits vor.

Sei f eine beliebige Oneway-Funktion. Wir definieren zu f die Oneway-Funktion f_{ip} gemäß

$$f_{\mathrm{ip}}(x, r) := (f(x), r) \qquad \text{für } |x| = |r|,$$

d.h. man wendet die Funktion f auf die erste Hälfte der Eingabe an und behalten die zweite Hälfte bei. Wir zeigen in diesem Kapitel, daß das „innere Produkt modulo 2“, also

$$h_{\mathrm{ip}}(x, r) := \langle x, r \rangle \bmod 2 = \sum_{i=1}^{n} x_i r_i \bmod 2$$

ein Hardcore-Prädikat für die Oneway-Funktion f_{ip} bzw. f ist.

L.A. Levin [Levin87] hat diese Konstruktion vermutet und später zusammen mit O. Goldreich [GL89] bewiesen. Der ursprüngliche Beweis wurde später von C. Rackoff und R. Venkatesan [Levin93] (siehe auch [Knuth96, Goldreich95b]) durch die Idee paarweise unabhängiger Testpunkte aus [ACGS88, CG89] vereinfacht und verbessert. Wir geben einen effizienten Algorithmus, der die Oneway-Funktion mit Hilfe eines Algorithmus' für das innere Produkt modulo 2 invertiert, an. Mit der Tschebycheff-Ungleichung beschränken wir die Fehlerwahrscheinlichkeit der Mehrheitsentscheidungen nach oben:

Lemma 1.6.1
Seien $X_1, X_2, \ldots, X_m$ identisch verteilte, paarweise unabhängige Zufallsvariablen mit Erwartungwert $\mu := \mathrm{E}[X_i]$ und Varianz $\sigma^2 := \mathrm{Var}[X_i]$. Dann gilt für jedes $t > 0$:

$$\mathrm{Ws}\left[\left|\sum_{i=1}^{m} X_i - m\mu\right| \geq tm\right] \leq \frac{\sigma^2}{t^2 m}$$

Beweis. Da $X_1, X_2, \ldots, X_m$ paarweise unabhängige Zufallsvariablen sind, gilt für $i \neq j$:

$$\mathrm{E}[X_i X_j] = \mathrm{E}[X_i] \cdot \mathrm{E}[X_j] = \mu^2 \tag{1.17}$$

Sei $X := \sum_{i=1}^{m} X_i$. Wegen $\mathrm{E}[X] = m\mu$ erhalten wir:

$$\begin{aligned}
\mathrm{E}\Big[(X - m\mu)^2\Big] &= \mathrm{E}\big[X^2\big] + 2 \cdot (\mathrm{E}[X] - m\mu) - m^2\mu^2 \\
&= \sum_{i=1}^{m}\sum_{i=1}^{j} \mathrm{E}[X_i] \cdot \mathrm{E}[X_j] - m^2\mu^2 \\
&= m \cdot \mathrm{E}\big[X_j^2\big] + \sum_{i \neq j} \mathrm{E}[X_i] \cdot \mathrm{E}[X_j] - m^2\mu^2
\end{aligned}$$

Da es genau $m^2 - m$ Paare (i, j) mit $1 \leq i, j \leq m$ und $i \neq j$ gibt, gilt wegen Gleichung (1.17):

$$\begin{aligned}
\mathrm{E}\Big[(X - m\mu)^2\Big] &= m \cdot \mathrm{E}\big[X_j^2\big] + m^2\mu^2 - m\mu^2 - m^2\mu^2 \\
&= m \cdot \big(\mathrm{E}\big[X_j^2\big] - \mu^2\big) \\
&= m\sigma^2
\end{aligned} \tag{1.18}$$

Andererseits wissen wir:

$$\begin{aligned}
\mathrm{E}\Big[(X - m\mu)^2\Big] &= \sum_{k \in \mathbb{N}} k \cdot \mathrm{Ws}\big[(X - m\mu)^2 = k\big] \\
&\geq \sum_{k \geq t^2 m^2} t^2 m^2 \cdot \mathrm{Ws}\big[(X - m\mu)^2 = k\big] \\
&\geq t^2 m^2 \cdot \mathrm{Ws}\big[(X - m\mu)^2 \geq t^2 m^2\big]
\end{aligned}$$

In Verbindung mit (1.18) erhalten wir die Behauptung:

$$\mathrm{Ws}[\,|X - m\mu| \geq tm] = \mathrm{Ws}\big[(X - m\mu)^2 \geq t^2 m^2\big] \leq \frac{m\sigma^2}{t^2 m^2} = \frac{\sigma^2}{t^2 m}$$

■

Die Chernoff-Schranke aus Fakt 1.5.9 auf Seite 31 können wir nicht anwenden, da die Zufallsvariablen nur paarweise unabhängig sind.

Lemma 1.6.2
Sei f eine Funktion, n die Länge des Arguments von f und $f_{\mathrm{ip}}(x,r) := (f(x),r)$ für $|x| = |r|$. Sei A ein probabilistischer Algorithmus mit Vorteil $\epsilon(n)$ bei der Berechnung des Prädikats

$$h_{\mathrm{ip}}(x,r) = \langle x,r\rangle \bmod 2$$

zu $f_{\mathrm{ip}}(x,r)$, wobei die Wahrscheinlichkeit über die zufällige Wahl von $r \in_{\mathrm{R}} \{0,1\}^n$ und die internen Münzwürfe gebildet wird (insbesondere unabhängig von $x \in \{0,1\}^n$ ist). Dann generiert Algorithmus 1.6.1 zu $f(x)$ eine Menge $K \subseteq \{0,1\}^n$ mit 2^k Werten,

$$k := \left\lceil \log_2 \left(\tfrac{1}{2} n \epsilon(n)^{-2} + 1 \right) \right\rceil,$$

so daß für unendliche viele n mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ (bezogen auf die internen Münzwürfe) das Urbild x in K enthalten ist. Die Laufzeit ist $\mathcal{O}\left(nk2^k\right) + n2^k \cdot |A|$.

Als Konstante des $\mathcal{O}$-Ausdrucks der Laufzeit kann man 4 wählen. Beachte, daß Algorithmus 1.6.1 die Funktion f nicht auswertet.

Algorithmus 1.6.1 Invertieren mit Algorithmus A für „Inneres Produkt modulo 2“

EINGABE: $f(x)$ mit $x \in \{0,1\}^n$

/* Sei A ein probabilistischer Algorithmus mit Vorteil $\epsilon(n)$ bei der Berechnung des Prädikats $h_{\mathrm{ip}}(x,r) = \langle x,r\rangle \bmod 2$ zu $f_{\mathrm{ip}}(x,r)$. */

1. $k := \left\lceil \log_2 \left(\frac{1}{2} n \epsilon(n)^{-2} + 1 \right) \right\rceil, \quad m := 2^k - 1$

2. Wähle zufällige Matrix $R \in_{\mathrm{R}} \{0,1\}^{k \times n}$

3. FOR $i = 1$ TO n DO

3.1. FOR all $b \in \{0,1\}^k$ DO

/* Sei e_i der i-te Einheitsvektor und x_i das i-te Bit von x. Für $b = xR^{\mathrm{T}}$ gilt $\langle x, cR + e_i\rangle = \langle b,c\rangle + x_i$. Triff Mehrheitsentscheidung über alle $c \in \{0,1\}^k$ ungleich 0: */

3.1.1. $H_i(b) := \left\{ c \in \{0,1\}^k \setminus \{0\} \;\middle|\; A\left(f(x), cR + e_i\right) = \langle b,c\rangle \right\}$

3.1.2. $x_i(b) := \begin{cases} 0 & \text{falls } |H_i(b)| > \frac{1}{2}m \\ 1 & \text{falls } |H_i(b)| \leq \frac{1}{2}m \end{cases}$

END for b

END for i

4. FOR all $b \in \{0,1\}^k$ DO $x(b) := (x_1(b), x_2(b), \ldots, x_n(b))$

AUSGABE: $K := \left\{ x(b) \;\middle|\; b \in \{0,1\}^k \right\}$

Beweis. Wir zeigen, daß in der Ausgabemenge K des Algorithmus' 1.6.1 zu $f(x)$ mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ (bezogen auf die internen Münzwürfe) das Urbild x enthalten ist.

BEHAUPTUNG 1: Für $b := xR^{\mathrm{T}}$ gilt $\langle x, cR + e_i\rangle = \langle b, c\rangle + x_i$.

BEWEIS. Es gilt $\langle x, cR + e_i\rangle = \langle x, cR\rangle + \langle x, e_i\rangle = \langle xR^{\mathrm{T}}, c\rangle + x_i = \langle b, c\rangle + x_i$. □

BEHAUPTUNG 2: Die Vektoren $\{cR + e_i \mid c \in \{0,1\}^k \setminus \{0\}\}$ sind für eine zufällige Matrix $R \in_{\mathrm{R}} \{0,1\}^{k\times n}$ gleichverteilt in $\{0,1\}^n$ und paarweise unabhängig.

BEWEIS. Es genügt zu zeigen, daß die Vektoren $\{cR \mid c \in \{0,1\}^k \setminus \{0\}\}$ für eine zufällige Matrix $R \in_{\mathrm{R}} \{0,1\}^{k\times n}$ (insbesondere sind die Spalten unabhängig) gleichverteilt in $\{0,1\}^n$ und paarweise unabhängig sind. Wegen $c \neq 0$ ist für $i = 1, 2, \ldots, n$ der i-te Eintrag von cR modulo 2 gleich einer nicht-leeren Summe von Einträgen der i-ten Spalte von R und damit gleichverteilt in $\{0,1\}$.

Für den Beweis der paarweisen Unabhängigkeit müssen wir zu gegebenen $c, c' \in \{0,1\}^k$ mit $c \neq c'$ und $d, d' \in \{0,1\}^n$ zeigen, daß der Anteil alle Matrizen R aus $\{0,1\}^{k\times n}$ mit $(cR, c'R) = (d, d')$ gleich 2^{-2n} ist. Da cR und $c'R$ gleichverteilt in $\{0,1\}^n$ sind, gilt dann

$$\underbrace{\mathrm{Ws}[\,cR = d]}_{=2^{-n}} \cdot \underbrace{\mathrm{Ws}\left[\,c'R = d'\right]}_{=2^{-n}} = \mathrm{Ws}\left[\,(cR, c'R) = (d, d')\right],$$

so daß cR und $c'R$ unabhängig sind. Sei i die Position, in der sich c und c' unterscheiden. O.B.d.A. sei $c_i = 0$ und $c'_i = 1$. Wegen $c \neq 0$ existiert ein j mit $c_j = 1$. Wir fixieren zu einer Matrix $R \in \{0,1\}^{k\times n}$ alle bis auf die i-te Zeile $\vec{r}_i$ und die j-te Zeile $\vec{r}_j$. Wir zeigen, daß genau für ein Paar $(\vec{r}_i, \vec{r}_j)$ der 2^{2n} Möglichkeiten aus $\{0,1\}^n \times \{0,1\}^n$ gilt $(cR, c'R) = (d, d')$. Sei R^* die Matrix R, wenn man die beiden Zeilen durch 0 ersetzt. Wir suchen $(\vec{r}_i, \vec{r}_j)$ mit:

$$\begin{aligned} d &= cR^* + \vec{r}_j \\ d' &= c'R^* + c'_j\vec{r}_j + \vec{r}_i \end{aligned}$$

Aus der ersten Forderung erhalten wir $\vec{r}_j = d - cR^*$ und aus der zweiten $\vec{r}_i = d' - c'R^* - c'_j\vec{r}_j$. Beide Vektoren $\vec{r}_i$ und $\vec{r}_j$ aus $\{0,1\}^n$ sind damit zu fixierter Matrix R^* eindeutig bestimmt. □

BEHAUPTUNG 3: Sei $x \in \{0,1\}^n$ und $b := xR^{\mathrm{T}}$. Dann gilt für $i = 1, 2, \ldots, n$:

$$\mathrm{Ws}[\,x_i(b) \neq x_i] \leq \frac{1}{4\epsilon(n)^2 \cdot (2^k - 1)}$$

BEWEIS. Sei $C := \{0,1\}^k \setminus \{0\}$ mit $m = |C| = 2^k - 1$. Wir müssen die Fehlerwahrscheinlichkeit unserer Mehrheitsentscheidung, also die Wahrscheinlichkeit, daß mindestens für die Hälfte aller $c \in C$

$$A\left(f(x), cR + e_i\right) \neq \langle b, c\rangle + x_i \tag{1.19}$$

gilt, nach oben abschätzen. Zu jedem $c \in C$ definieren wir eine Indikatorvariable X_c für das Ereignis (1.19), d.h. die Ausgabe des Algorithmus' für $cR + e_i$ falsch ist. Nach Voraussetzung gilt:

$$\mathrm{Ws}[\,X_c = 1] \leq \tfrac{1}{2} - \epsilon(n)$$

Der Erwartungswert ist $\mu \leq \frac{1}{2} - \epsilon(n)$ und die Varianz:

$$\sigma^2 = \mathrm{E}\left[X_i^2\right] - \mathrm{E}[X_i]^2 \leq \max_{\xi \in [0,1]} \left\{\xi - \xi^2\right\} = \tfrac{1}{4} \tag{1.20}$$

Die m Zufallsvariablen X_c mit $c \in C$ hängen von der unabhängigen und zufälligen Wahl der internen Münzwürfe des Algorithmus' A und den nach Behauptung 2 gleichverteilten und paarweise unabhängigen $cR + e_i$ ab. Daher sind die m Zufallsvariablen X_c paarweise unabhängig und wir können Lemma 1.6.1 anwenden. Wegen $m\mu \leq \frac{1}{2}m - m\epsilon(n)$ gilt:

$$\mathrm{Ws}[x_i(b) \neq x_i] \leq \mathrm{Ws}\left[\sum_{c \in C} X_c \geq \frac{m}{2}\right] \leq \mathrm{Ws}\left[\sum_{c \in C} X_c - m\mu \geq m\epsilon(n)\right]$$

Mit $t := \epsilon(n)$ folgt aus Lemma 1.6.1:

$$\mathrm{Ws}[x_i(b) \neq x_i] \leq \mathrm{Ws}\left[\left|\sum_{c \in C} X_c - m\mu\right| \geq m\epsilon(n)\right] \leq \frac{\sigma^2}{t^2 m}$$

Wegen der Abschätzung (1.20) gilt

$$\mathrm{Ws}[x_i(b) \neq x_i] \leq \frac{\sigma^2}{t^2 m} \leq \frac{1}{4m\epsilon(n)^2}$$

Dies war zu zeigen. □

Behauptung 4: Schritt 3.1 gelingt in Zeit $(2k+1)2^k + 2^k \cdot |A|$.

Beweis. Wir beschreiben ein Verfahren zur Berechnung der 2^k Summen

$$h_i^*(b) := \sum_{c \in \{0,1\}^k \setminus \{0\}} (-1)^{\langle b,c \rangle} \cdot (-1)^{A(f(x), cR + e_i)} \qquad \text{für alle } b \in \{0,1\}^k \tag{1.21}$$

in Zeit $(2k+1)2^k + 2^k \cdot |A|$. Falls $A(f(x), cR + e_i) = \langle b, c \rangle \bmod 2$, ist der Summand $+1$ und sonst -1. Daher gilt folgende Äquivalenz:

$$|H_i(b)| > \tfrac{1}{2}m \quad \Longleftrightarrow \quad h_i^*(b) > 0$$

Es genügt für Schritt 3.1 die Berechnung der 2^k Summen (1.21). Wir setzen für $b, c \in \{0,1\}^k$:

$$w_k(b,c) := (-1)^{\langle b,c \rangle}$$

Mit $w_0 := +1$ gilt für $k \geq 1$:

$$w_k\left((b_1, \ldots, b_k), (c_1, \ldots, c_k)\right) = \begin{cases} -w_{k-1}\left((b_2, \ldots, b_k), (c_2, \ldots, c_k)\right) & \text{falls } b_1 = c_1 = 1 \\ +w_{k-1}\left((b_2, \ldots, b_k), (c_2, \ldots, c_k)\right) & \text{sonst.} \end{cases}$$

Um die 2^k Gleichungen (1.21) als Matrix-Vektor-Produkt darzustellen, schreiben wir w_k als $2^k \times 2^k$-Matrix

$$W_k = \left[w_k(b,c)\right]_{b,c \in \{0,1\}^k},$$

wobei die Zeilen durch b und die Spalten durch c jeweils in lexikographischer Ordnung indiziert sind. Es gilt $W_0 = [1]$ und für $k \geq 1$:

$$W_k = \begin{bmatrix} +W_{k-1} & +W_{k-1} \\ +W_{k-1} & -W_{k-1} \end{bmatrix} \tag{1.22}$$

Eine Matrix dieses Typs heißt Sylvestermatrix. Wir berechnen das Matrix-Vektor-Produkt

$$\begin{bmatrix} h_i^*(0,\ldots,0,0) \\ h_i^*(0,\ldots,0,1) \\ \vdots \\ h_i^*(1,\ldots,1,1) \end{bmatrix} = W_k \cdot \begin{bmatrix} y_i(0,\ldots,0,0) \\ y_i(0,\ldots,0,1) \\ \vdots \\ y_i(1,\ldots,1,1) \end{bmatrix} \tag{1.23}$$

mit $y_i(0) = 0$ und $y_i(c) = (-1)^{A(f(x),cR+e_i)}$ für $c \in \{0,1\}^k \setminus \{0\}$. Für die Berechnung von (1.23) im Fall $k \geq 1$ genügt wegen der Form (1.22) der Matrix W_k die Bestimmung von

$$W_{k-1} \cdot \begin{bmatrix} y_i(0,\ldots,0,0) \\ y_i(0,\ldots,0,1) \\ \vdots \\ y_i(0,1\ldots,1,1) \end{bmatrix} \quad \text{und} \quad W_{k-1} \cdot \begin{bmatrix} y_i(1,0,\ldots,0,0) \\ y_i(1,0\ldots,0,1) \\ \vdots \\ y_i(1,1\ldots,1,1) \end{bmatrix}$$

und zusätzlich 2^k Additionen und Subtraktionen. Dies legt ein Divide-&-Conquer-Verfahren nahe: Die beiden Matrix-Vektor-Produkte bestimmen wir durch einen rekursiven Aufruf des Algorithmus'. Seien $B(k)$ die Anzahl der Bitoperationen bei der Berechnung des Produkts der Matrix W_k mit dem Vektor y. Wegen

$$B(k) = \begin{cases} 1 & \text{falls k=0} \\ 2^k + 2 \cdot B(k-1) & \text{sonst} \end{cases}$$

ist $B(k) = (k+1) \cdot 2^k$. Da $y_i(0) = 0$ und $y_i(c) \in \{0, \pm 1\}$ für $c \in \{0,1\}^k \setminus \{0\}$, treten während der Berechnung Zahlen mit Bitlänge höchstens k auf. Wir können das Produkt in Zeit $(2k+1) \cdot 2^k$ bestimmen. Da die Berechung der 2^k $y_i(c)$-Werte in Zeit $2^k \cdot |A|$ gelingt, erhalten wir die angegebene Laufzeit. □

Aus Behauptung 3 folgern wir, daß der Algorithmus mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ das Urbild x in der Ausgabemenge K enthalten ist. Für festes $x \in \{0,1\}^n$ und $b := xR^{\mathrm{T}}$ gilt:

$$\begin{aligned} \mathrm{Ws}[\, x \in K] &\geq \mathrm{Ws}[\, x(b) = x] \\ &\geq 1 - \sum_{i=1}^{n} \mathrm{Ws}[\, x_i(b) \neq x] \\ &\geq 1 - \frac{n}{4n\epsilon(n)^2 \cdot (2^k - 1)} \end{aligned} \tag{1.24}$$

Nach Wahl von k ist $\frac{n}{2\epsilon(n)^2} \leq 2^k - 1$ und es gilt:

$$\frac{n}{4\epsilon(n)^2 \cdot (2^k - 1)} \leq \frac{2^k - 1}{2 \cdot (2^k - 1)} = \frac{1}{2}$$

Aus Abschätzung (1.24) folgt $\mathrm{Ws}[\, x \in K] \geq \frac{1}{2}$. Wir erhalten die angegebene Laufzeit

$$\mathcal{O}\left(nk2^k\right) + n2^k \cdot |A|$$

aus Behauptung 4, da wir Schritt 3.1 n-mal ausführen und Schritt 4 in Zeit $\mathcal{O}\left(n2^k\right)$ gelingt. ■

Wir werden in Kapitel 2.3.2 die Möglichkeit der Mehrheitsentscheidung durch eine Stichprobe kennenlernen. Da die Indikatorvariablen dann unabhängig sind, können wir die stärkere Chernoff-Schranke statt des Resultats aus Lemma 1.6.1 anwenden. In diesem Fall sind $\mathcal{O}\left(\epsilon(n)^{-2} n \log_2 n\right)$ statt $\mathcal{O}\left(\epsilon(n)^{-2} n^2\right)$ Aufrufe von Algorithmus A ausreichend.

Wir haben in Lemma 1.6.2 vorausgesetzt, daß die Erfolgswahrscheinlichkeit bei der Berechnung des Prädikats $h_{\mathrm{ip}}(x, r) = \langle x, r \rangle \bmod 2$ unabhängig von x ist. Nach Definition eines Hardcore-Prädikats hängt die Erfolgswahrscheinlichkeit des Prediktors jedoch von der gesamten, zufälligen Eingabe ab. Mit Hilfe des folgenden Lemmas können wir die Aussage von Lemma 1.6.2 nutzen:

Lemma 1.6.3
Seien X, Y endliche Mengen und U_X, U_Y die Gleichverteilung auf X bzw. Y. Sei $\psi : X \times Y \to \{0,1\}$ ein probabilistischer Algorithmus mit

$$\mathrm{Ws}[\, \psi\,(U_X, U_Y) = 1] \geq \tfrac{1}{2} + \epsilon,$$

wobei die Wahrscheinlichkeit über die Eingaben und die internen Münzwürfe des Algorithmus' gebildet wird. Dann existiert eine Teilmenge $G \subseteq X$ mit:

a) $\mathrm{Ws}[\, U_X \in G] \geq \epsilon$.

b) Für alle $x \in G$ gilt: $\mathrm{Ws}[\, \psi\,(x, U_Y) = 1] \geq \frac{1}{2} + \frac{1}{2}\epsilon$.

Beweis. Zu $x \in X$ setzen wir $\epsilon(x) := \mathrm{Ws}[\, \psi\,(x, U_Y) = 1] - \frac{1}{2}$ mit $\epsilon(x) \leq \frac{1}{2}$. Nach Voraussetzung gilt:

$$\frac{1}{|X|} \cdot \sum_{x \in X} 2\epsilon(x) = 2\left(\mathrm{Ws}[\, \psi\,(U_X, U_Y) = 1] - \tfrac{1}{2}\right) \geq 2\epsilon \tag{1.25}$$

Wir wählen:

$$G := \left\{ x \in X \;\middle|\; \epsilon(x) \geq \tfrac{1}{2}\epsilon \right\}$$

Es ist noch Eigenschaft a) nachzuweisen. Sei $\overline{G} := X \setminus G$. Aus $\overline{G} \subseteq X$ und $\epsilon(x) < \frac{1}{2}\epsilon$ für $x \in \overline{G}$ erhalten wir:

$$\frac{1}{|X|} \cdot \sum_{x \in \overline{G}} 2\epsilon(x) \leq \frac{1}{|X|} \cdot \sum_{x \in X} \epsilon = \epsilon \tag{1.26}$$

Wegen $2\epsilon(x) \leq 1$ und $G \subseteq X$ gilt:

$$\mathrm{Ws}[\, U_X \in G] = \frac{|G|}{|X|} \geq \frac{1}{|X|} \cdot \sum_{x \in G} 2\epsilon(x) = \frac{1}{|X|} \cdot \sum_{x \in X} 2\epsilon(x) - \frac{1}{|X|} \cdot \sum_{x \in \overline{G}} 2\epsilon(x)$$

Aus den Abschätzungen (1.25) und (1.26) erhalten wir Eigenschaft a):

$$\mathrm{Ws}[\,U_X \in G] \geq 2\epsilon - \epsilon = \epsilon$$

Dies war zu zeigen. ■

Es folgt:

Satz 1.6.4
Sei f eine Funktion, n die Länge des Arguments von f und $f_{\mathrm{ip}}(x,r) := (f(x),r)$ für $|x| = |r|$. Sei A ein probabilistischer Algorithmus mit Vorteil $\epsilon(n)$ bei der Berechnung des Prädikats

$$h_{\mathrm{ip}}(x,r) = \langle x,r\rangle \bmod 2$$

zu $f_{\mathrm{ip}}(x,r)$. Dann generiert Algorithmus 1.6.1 zu $f(x)$ eine Menge $K \subseteq \{0,1\}^n$ mit 2^k Werten,

$$k := \left\lceil \log_2 \left(\tfrac{1}{2} n \epsilon(n)^{-2} + 1\right)\right\rceil$$

so daß für unendliche viele n mit Wahrscheinlichkeit mindestens $\frac{1}{2}\epsilon(n)$ (bezogen auf die zufällige Wahl von $x \in_{\mathrm{R}} \{0,1\}^n$ und die internen Münzwürfe) das Urbild x in K enthalten ist. Die Laufzeit ist $\mathcal{O}\left(nk2^k\right) + n2^k \cdot |A|$.

Beweis. Wir betrachten nur die unendlich vielen n, für die der Algorithmus A das Prädikat h_{ip} mit Wahrscheinlichkeit mindestens $\frac{1}{2} + \epsilon(n)$ bestimmt. Aus Lemma 1.6.3 folgt mit $X = Y = \{0,1\}^n$ und $\psi(x,r)$ gemäß

$$\psi(x,r) = 1 \quad \Longleftrightarrow \quad A(f(x),r) = h_{\mathrm{ip}}(x,r),$$

daß eine Menge $G_n \subseteq \{0,1\}^n$ existiert mit

a) $\mathrm{Ws}[\,U_n \in G_n] \geq \epsilon(n)$ und

b) für alle $x \in G_n$ gilt $\mathrm{Ws}[\,A\,(f(x),U_n) = h_{\mathrm{ip}}(x,U_n)] \geq \frac{1}{2} + \frac{1}{2}\epsilon(n)$.

Nach Lemma 1.6.2 von Seite 39 gilt für alle $x \in G_n$: In der Ausgabemenge K des Algorithmus' 1.6.1 ist zu $f(x)$ mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ (bezogen auf die internen Münzwürfe) das Urbild x enthalten. Die behauptete Erfolgswahrscheinlichkeit folgt dann aus Eigenschaft a). ■

Wir erhalten als Korollar den Satz von O. Goldreich und L.A. Levin [GL89] über das Hardcore-Prädikat „inneres Produkt modulo 2“:

Korollar 1.6.5 (Goldreich, Levin 1989)
Sei f eine Oneway-Funktion und $f_{\mathrm{ip}}(x,r) := (f(x),r)$ für $|x| = |r|$. Dann ist das innere Produkt modulo 2, also

$$h_{\mathrm{ip}}(x,r) := \langle x,r\rangle \bmod 2,$$

ein Hardcore-Prädikat zu f_{ip} bzw. f.

Beweis. Angenommen, h_{ip} sei kein Hardcore-Prädikat, d.h. es existiert ein Polynomialzeit-Algorithmus, der das Prädikat h_{ip} zu f_{ip} mit Vorteil n^{-c} für ein festes $c \in \mathbb{N}$ bestimmt. Wir wenden Satz 1.6.4 mit $\epsilon(n) = n^{-c}$ an. Insbesondere ist:

$$k(n) \leq \left\lceil \log_2 \left(\tfrac{1}{2} n^{2c+1} + 1 \right) \right\rceil \leq (2c+1) \cdot \log_2 n$$

Wir erhalten, daß ein Polynomialzeit-Algorithmus existiert, der für unendlich viele n zu $f(x)$ mit Wahrscheinlichkeit mindestens $\frac{1}{2} n^{-c}$ eine Liste von 2^k Werten erzeugt, unter denen ein Urbild zu $f(x)$ ist. Wir wenden die Oneway-Funktion f auf die Werte der Liste an und geben einen Wert x' mit $f(x') = f(x)$ aus. Man kann in Polynomialzeit für unendliche viele n mit Wahrscheinlichkeit mindestens $n^{-(c+1)}$ ein Urbild zu $f(x)$ berechnen — Widerspruch, da f eine Oneway-Funktion ist. ■

Korollar 1.6.5 gilt auch, wenn die Funktion f nicht in Polynomialzeit berechenbar ist. In diesem Fall geben wir ein zufällig aus der Menge der möglichen 2^k Urbilder gezogenen Wert aus. Die Erfolgswahrscheinlichkeit $\epsilon \cdot 2^{-(k+1)}$ des Algorithmus' ist nicht-vernachlässigbar. Zum Beispiel verwenden M. Naor und O. Reingold [NR95] das innere Produkt modulo 2 als Hardcore-Prädikat in Verbindung mit der Diffie-Hellman-Funktion $(g^a, g^b) \mapsto g^{ab} \bmod p$ (für eine Primitivwurzel g modulo einer Primzahl p) [DH76], die vermutlich ohne Kenntnis eines der Exponenten a oder b nicht effizient berechenbar ist.

Wir möchten das allgemeine Hardcore-Prädikat zu einer Hardcore-Funktion erweitern. Sei f eine Oneway-Funktion und n die Länge des Arguments. Wir zeigen zunächst, daß für logarithmisch viele, zufällige und unabhängige Vektoren $\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_{k(n)}$ aus $\{0,1\}^n$ die Hardcore-Prädikate

$$h_{\text{ip}}(x, \vec{r}_1), \quad h_{\text{ip}}(x, \vec{r}_2), \quad \ldots, \quad h_{\text{ip}}\left(x, \vec{r}_{k(n)}\right)$$

simultan sicher sind. Dazu wenden wir die XOR-Bedingung aus Satz 1.5.14 auf Seite 38 an. Sei $H(n) \subseteq \{1, 2, \ldots, k(n)\}$ eine nicht-leere Teilmenge. Die XOR-Verknüpfung der entsprechenden Hardcore-Prädikate kann man schreiben als:

$$\sum_{t \in H(n)} h_{\text{ip}}(x, \vec{r}_t) = \sum_{t \in H(n)} \langle x, \vec{r}_t \rangle = \left\langle x, \sum_{t \in H(n)} \vec{r}_t \right\rangle = h_{\text{ip}}\left(x, \sum_{t \in H(n)} \vec{r}_t \right) \bmod 2$$

Da jeder Vektor $\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_{k(n)}$ gleichverteilt in $\{0,1\}^n$ ist, gilt dies auch für die nicht-leere Summe $\sum_{i \in H(n)} \vec{r}_i$. Nach Korollar 1.6.5 ist die XOR-Verknüpfung der entsprechenden Hardcore-Prädikate ebenfalls ein Hardcore-Prädikat. Aus Satz 1.5.14 folgt, daß die Prädikate eine Hardcore-Funktion bilden. In Form eines Vektor-Matrix-Produktes modulo 2 können wir diese Hardcore-Funktion als $xR \bmod 2$ schreiben, indem man die Vektoren $\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_{k(n)}$ als Spalten von R wählt. Wir erweitern eine gegebene, beliebige Oneway-Funktion f um $n \cdot k(n)$ Bits:

$$f_{\text{Matrix}}(x, r) := (f(x), r) \qquad \text{für } |x| \cdot k(|x|) = |r|$$

Die Funktion $\text{Matrix}_{n,k(n)}(r)$ mit $r \in \{0,1\}^{n \cdot k(n)}$ stellt die Bits von r als binäre $n \times k(n)$-Matrix dar. Für $k(n) = \mathcal{O}(\log_2 n)$ ist die Abbildung

$$h_{\text{Matrix}}(x, r) := x \cdot \text{Matrix}_{n,k(n)}(r) \bmod 2$$

eine Hardcore-Funktion für die Oneway-Funktion f_{Matrix} bzw. f.

O. Goldreich und L.A. Levin haben in ihrer Arbeit [GL89] eine ähnliche Hardcore-Funktion mit logarithmisch vielen Bits zu einer beliebigen Oneway-Funktion angegeben. Statt einer zufälligen Matrix verwenden sie eine *Töplitz-Matrix*. Eine Matrix $[r_{ij}]_{ij}$ heißt Töplitz-Matrix, wenn $r_{i+1,j+1} = r_{ij}$ gilt, also auf Positionen einer Diagonalen jeweils der gleiche Wert steht. Eine Töplitz-Matrix ist durch die Werte in der ersten Spalte und Zeile gegeben:

$$\text{Toeplitz}_{n,k(n)}\left(r_1, r_2, \ldots, r_{n+k(n)-1}\right) := \begin{bmatrix} r_{k(n)} & r_{k(n)-1} & \cdots & r_2 & r_1 \\ r_{k(n)+1} & r_{k(n)} & & r_3 & r_2 \\ r_{k(n)+2} & r_{k(n)+1} & & r_4 & r_3 \\ \vdots & \vdots & & \vdots & \vdots \\ r_{n+k(n)-1} & r_{n+k(n)-2} & & r_{n+1} & r_n \end{bmatrix}$$

Für eine zufällige Töplitz-Matrix muß die Funktion f nur um $n + k(n) - 1$ statt $n \cdot k(n)$ Bits erweitert werden. Wir definieren zu f die Oneway-Funktion f_{Toeplitz} gemäß

$$f_{\text{Toeplitz}}(x, r) := (f(x), r) \qquad \text{für } k(|x|) + |x| - 1 = |r|.$$

Wir zeigen, daß für $k(n) = \mathcal{O}(\log_2 n)$ das Produkt mit der $n \times k(n)$-Töplitz-Matrix, also

$$h_{\text{Toeplitz}}(x, r) := x \cdot \text{Toeplitz}_{n,k(n)}(r) \bmod 2$$

eine Hardcore-Funktion für die Oneway-Funktion f_{Toeplitz} bzw. f ist. Wie im Fall einer allgemeinen Matrix bezeichnen wir mit $\vec{r_t}$ die Einträge der t-ten Spalte der Matrix $\text{Toeplitz}_{n,k(n)}(r)$. Mit $h_t(x, r) := \langle x, \vec{r_t} \rangle \bmod 2$ kann man die Hardcore-Funktion h_{Toeplitz} schreiben als:

$$h_{\text{Toeplitz}}(x, r) = \big(h_1(x, r), h_2(x, r), \ldots, h_{k(n)}(x, r)\big)$$

Wir wenden ebenfalls die XOR-Bedingung aus Satz 1.5.14 an.

Algorithmus 1.6.2 Inneres Produkt modulo 2 berechnen

EINGABE: $f_{\text{ip}}(x, s) = (f(x), s)$ mit $x, s \in \{0,1\}^n$

/* Sei $(H(n))_{n\in\mathbb{N}}$ mit $H(n) \subseteq \{1, 2, \ldots, k(n)\}$ eine Folge nicht-leerer Teilmengen und A ein probabilistischer Algorithmus, der zu $f_{\text{Toeplitz}}(x, r)$ mit $n := |x|$ das Prädikat $h_{H(n)}(x, r) := \sum_{t\in H(n)} h_t(x, r)$ mit Vorteil $\epsilon(n)$ berechnet. Mit $\vec{r_t} \in \{0,1\}^n$ bezeichnen wir die Einträge der t-ten Spalte der Matrix $\text{Toeplitz}_{n,k(n)}(r)$. */

1. Wähle zufälliges $r \in \{0,1\}^{n+k(n)-1}$ mit $\sum_{t\in H(n)} \vec{r_t} = s$.

2. Gib $A\left(f_{\text{Toeplitz}}(x, r)\right)$ aus.

Im folgenden Lemma 1.6.6 zeigen wir, daß man das allgemeine Hardcore-Prädikat h_{ip} zu f_{ip} bzw. f effizient mit nicht-vernachlässigbarem Vorteil berechnen kann, falls die XOR-Bedingung für die logarithmisch vielen Prädikate $h_1(x, r), h_2(x, r), \ldots, h_{k(n)}(x, r)$ zu f_{Toeplitz} bzw. f nicht gilt.

Lemma 1.6.6
Sei $(H(n))_{n\in\mathbb{N}}$ mit $H(n) \subseteq \{1, 2, \dots, k(n)\}$ eine Folge nicht-leerer Teilmengen und A ein probabilistischer Algorithmus, der zu $f_{\text{Toeplitz}}(x, r)$ mit $n := |x|$ das Prädikat

$$h_{H(n)}(x, r) := \sum_{t\in H(n)} h_t(x, r)$$

mit Vorteil $\epsilon(n)$ berechnet. Dann bestimmt Algorithmus 1.6.2 das Hardcore-Prädikat $h_{\text{ip}}(x, s)$ zu $f_{\text{ip}}(x, s)$ mit Vorteil $\frac{\epsilon(n)}{2^{k(n)-1}}$ in Zeit $|A| + \mathcal{O}(nk(n))$.

Beweis. Wir wählen eine zufällige Töplitz-Matrix

$$\text{Toeplitz}_{n,k(n)}\left(r_1, r_2, \dots, r_{n+k(n)-1}\right) = \begin{bmatrix} r_{k(n)} & r_{k(n)-1} & \dots & r_2 & r_1 \\ r_{k(n)+1} & r_{k(n)} & & r_3 & r_2 \\ r_{k(n)+2} & r_{k(n)+1} & & r_4 & r_3 \\ \vdots & \vdots & & \vdots & \vdots \\ r_{n+k(n)-1} & r_{n+k(n)-1} & & r_{n+1} & r_n \end{bmatrix},$$

für die modulo 2 die Summe der Spalten der Indizes $H(n)$ gleich s^{T} ist, also $\sum_{t\in H(n)} \vec{r}_t = s$ gilt, wobei $\vec{r}_t \in \{0, 1\}^n$ die Einträge der t-ten Spalte sind. Dann gilt:

$$h_{H(n)}(x, r) = \sum_{t\in H(n)} h_t(x, r) = \left\langle x, \sum_{t\in H(n)} \vec{r}_t \right\rangle = \langle x, s\rangle = h_{\text{ip}}(x, s)$$

Wir müssen klären, wie man eine zufällige Töplitz-Matrix unter der gegebenen Nebenbedingung wählt und welchen Vorteil Algorithmus 1.6.2 bei der Berechnung des Prädikats h_{ip} hat.

Zur Vereinfachung sei zunächst $H(n) = \{1, 2, \dots, k(n)\}$. Zu gegebenem $s = (s_1, s_2, \dots, s_n)$ suchen wir eine zufällige Lösung $\left(r_1, r_2, \dots, r_{n+k(n)-1}\right) \in \{0, 1\}^{n+k(n)-1}$ des linearen Gleichungssystems

$$\begin{bmatrix} 1 & 1 & \dots & 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 1 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & & & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 1 & 1 & \dots & 1 & 0 \\ 0 & \dots & 0 & 0 & 1 & \dots & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_{n+k(n)-2} \\ r_{n+k(n)-1} \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{n-1} \\ s_n \end{bmatrix}$$

über dem Körper $\mathbb{Z}_2 = \{0, 1\}$. In jeder Zeile der Koeffizientenmatrix stehen genau $k(n)$ Einsen. Wir wählen zunächst zufällig und unabhängig Werte aus $\{0, 1\}$ für $r_1, r_2, \dots, r_{k(n)-1}$ und bestimmen dann $r_{k(n)}$, so daß die Summe modulo 2 gleich s_1 ist. In der zweiten Zeile bestimmen wir $r_{k(n)+1}$ aus den bekannten $r_2, r_3, \dots, r_{k(n)}$ und s_2, usw. Im allgemeinen Fall, also $H(n) \subseteq \{1, 2, \dots, k(n)\}$, ersetzt man für alle $t \notin H(n)$ in der Koeffizientenmatrix in jeder Zeile die $(k(n) - t + 1)$-te Eins durch Null. Wegen $H(n) \neq \emptyset$ steht in jeder Zeile und paarweise verschiedenen Spalten jeweils eine Eins. Wir bestimmen eine zufällige Lösung wie im Spezialfall. Dies gelingt in Zeit $\mathcal{O}(nk(n))$, so daß wir insgesamt die behauptete Laufzeit erhalten.

Wir schließen den Beweis mit einer Analyse des Vorteils von Algorithmus 1.6.2 zur Bestimmung des Hardcore-Prädikats $h_{\text{ip}}(x, s)$ zu $f_{\text{ip}}(x, s)$ bzw. f. Der Vorteil der Verfahren bezieht sich neben den internen Münzwürfen

- bei Algorithmus A auf die zufällige Wahl von $x \in \{0,1\}^n$ und der Töplitz-Matrix $\text{Toeplitz}_{n,k(n)}(r)$,
- bei Algorithmus 1.6.2 auf die zufällige Wahl von $x, s \in \{0,1\}^n$.

Da die Koeffizientenmatrix die Größe $n \times (n+k(n)-1)$ und den Rang n hat, gibt es zu jedem $s \in \{0,1\}^n$ genau $2^{k(n)-1}$ Töplitz-Matrizen und für $s, s' \in \{0,1\}^n$ mit $s \neq s'$ sind die beiden Mengen der Töplitz-Matrizen disjunkt. Algorithmus 1.6.2 hat den Vorteil $\frac{\epsilon(n)}{2^{k(n)-1}}$. ■

Wir erhalten, daß das Produkt mit einer zufälligen Töplitz-Matrix eine Hardcore-Funktion mit logarithmisch vielen Bits ist:

Satz 1.6.7 (Goldreich, Levin 1989)
Sei f eine Oneway-Funktion, n die Länge des Arguments von f und $f_{\text{Toeplitz}}(x,r) := (f(x), r)$ für $k(|x|) + |x| - 1 = |r|$. Dann ist für $k(n) = \mathcal{O}(\log_2 n)$ das Produkt mit einer Töplitz-Matrix modulo 2, also

$$h_{\text{Toeplitz}}(x,r) := x \cdot \text{Toeplitz}_{n,k(n)}(r) \bmod 2,$$

eine Hardcore-Funktion zu f_{Toeplitz} bzw. f.

Beweis. Sei $k(n) \leq c \cdot \log_2 n$ für eine Konstante $c \in \mathbb{N}$ und hinreichend große n. Angenommen, h_{Toeplitz} sei keine Hardcore-Funktion zu f_{Toeplitz} bzw. f. Dann existieren eine Konstante $d \in \mathbb{N}$ und ein probabilistischer Polynomialzeit-Algorithmus T mit

$$\text{Ws}[T(1^n, f(U_n), h(U_n))] - \text{Ws}\left[T\left(1^n, f(U_n), U_{k(n)}\right)\right] \geq n^{-d}$$

für unendlich viele n. Nach Satz 1.5.11 und Korollar 1.5.12 auf Seite 34 kann man in Polynomialzeit eine Teilmenge $H(n) \subseteq \{1, 2, \ldots, k(n)\}$ bestimmen, so daß Algorithmus 1.5.4 zur Berechnung des Prädikats

$$h_{H(n)}(x) := \sum_{t \in H(n)} h_t(x) \bmod 2$$

den Vorteil $n^{-(c+d)}$ hat. Aus Lemma 1.6.6 erhalten wir einen probabilistischen Algorithmus, der das Hardcore-Prädikat $h_{\text{ip}}(x,s)$ zu $f_{\text{ip}}(x,s)$ mit Vorteil $n^{-(2c+d)}$ in Polynomialzeit bestimmt — Widerspruch. ■

1.7 Existenz

Wir haben kryptographisch sichere Pseudozufallsgeneratoren definiert, aber nicht geklärt, ob solche überhaupt existieren. J. Boyar [Boyar89] hat zum Beispiel gezeigt, daß der klassische Lineare-Kongruenz-Generator aus Kapitel 1.1 nicht kryptographisch sicher ist. Wir zeigen, daß kryptographisch sichere Pseudozufallsgeneratoren genau dann existieren, wenn es Oneway-Funktionen gibt. Da die Existenz von Oneway-Funktionen bisher nur vermutet wird, kann man die Existenz von kryptographisch sicherer Pseudozufallsgeneratoren letztlich auch

nur vermuten. Allerdings ist die Oneway-Eigenschaft der vermuteten Oneway-Funktionen allgemein anerkannt.

Aus der Blum-Micali-Konstruktion (Kapitel 1.5) und dem inneren Produkt modulo 2 als Hardcore-Prädikat (Kapitel 1.6) folgt, daß die Existenz einer Oneway-Permutation hinreichend für die Existenz kryptographisch sicherer Pseudozufallsgeneratoren ist. Diese Bedingung kann abgeschwächt werden:

Satz 1.7.1 (Håstad, Impagliazzo, Levin, Luby 1989/90)
Die Existenz von Oneway-Funktionen ist hinreichend für die Existenz von kryptographisch sicheren Pseudozufallsgeneratoren.

Beweis. Siehe Originalarbeit [HILL91] oder M. Lubys Buch [Luby96, Kapitel 8-10]. ■

Wir beweisen, daß man aus einem kryptographisch sicheren Pseudozufallsgenerator eine Oneway-Funktion konstruieren kann. Dazu zeigen wir zunächst, daß man zu jedem durch ein Polynom beschränkten $m(n) \geq n+1$ aus einem Pseudozufallsgenerator des Typs $\ell(n)$ einen kryptographisch sicheren Generator des Typs $m(n)$ konstruieren kann.

Algorithmus 1.7.1 Pseudozufallsgenerator $G_{m(n)}$ vom Typ $m(n)$

EINGABE: zufälliger Startwert $x \in_{\mathrm{R}} \{0,1\}^n$

/* Sei G ein Pseudozufallsgenerator von Typ $\ell(n) = n+1$. */

1. $x_0 := x$

2. FOR $j := 1$ TO $m(n)$ DO

$$(b_j, x_j) := G(x_{j-1}) \text{ mit } b_j \in \{0,1\} \text{ und } x_j \in \{0,1\}^n$$

END for

AUSGABE: $\left(b_1, b_2, \ldots, b_{m(n)}\right) \in \{0,1\}^{m(n)}$

Zur Vereinfachung nehmen wir an, der Typ des gegebenen Pseudozufallsgenerators G sei $\ell(n) = n+1$. Dies können wir erreichen, indem gegebenenfalls die restlichen Bits der Ausgabe nicht beachten werden. Wir erhöhen mit Algorithmus 1.7.1 den Typ. Der Algorithmus wendet den Pseudozufallsgenerator G auf den (zufälligen) Startwert x_0 an und zerlegt die Ausgabe $G(x_0) \in \{0,1\}^{n+1}$ in das erste Bit b_1 und die restlichen n Bits x_1. Anschließend wendet er den Generator auf x_1 an und erhält Bit b_2 und x_2, mit dem der Algorithmus fortfährt.

Satz 1.7.2
Sei G ein kryptographisch sicherer Pseudozufallsgenerator vom Typ $\ell(n) = n+1$. Dann ist für jedes durch ein Polynom beschränktes $m(n) \geq n+1$ Algorithmus 1.7.1 ein kryptographisch sicherer Pseudozufallsgenerator $G_{m(n)}$ des Typs $m(n)$.

Beweis. Offenbar ist $G_{m(n)}$ ein deterministischer Polynomialzeit-Algorithmus. Um zu zeigen, daß $G_{m(n)}$ für jede Wahl von $m(n)$ ein kryptographisch sicherer Pseudozufallsgenerator

ist, wenden wir die Hybrid-Methode an. Wir definieren die hybriden Verteilungen:

$$H_n^{(j)} := \big(U_j \ , \ G_{m(n)-j}(U_n)\big) \qquad \text{für } j := j(n) = 0, 1, \dots, m(n)$$

Insbesondere gilt:

- $\big(G_{m(n)}(U_n)\big)_{n\in\mathbb{N}} = \big(H^{(0)}\big)_{n\in\mathbb{N}}$
- $\big(U_{m(n)}\big)_{n\in\mathbb{N}} = \big(H^{(m(n))}\big)_{n\in\mathbb{N}}$.

Angenommen, $G_{m(n)}$ sei kein Pseudozufallsgenerator, d.h. es existiert eine Konstante $c \in \mathbb{N}$ und ein statistischer Test $T_{m(n)}$ mit $\Delta_{T_{m(n)}}(G_{m(n)}(U_n)) \geq n^{-c}$ für unendlich viele n. Im weiteren betrachen wir nur diese n. Wir setzen für $j = 0, 1, \dots, m(n)$:

$$p_j := p_j(n) := \mathrm{Ws}\Big[\, T_{m(n)}\left(H_n^{(j)}\right) = 1\Big] \geq 0 \tag{1.27}$$

Nach Voraussetzung gilt für folgende Teleskopsumme:

$$\begin{aligned} \sum_{j=0}^{m(n)-1} (p_{j+1} - p_j) &= p_{m(n)} - p_0 \\ &= \mathrm{Ws}\big[\, T_{m(n)}\left(G_{m(n)}(U_n)\right) = 1\big] - \mathrm{Ws}\big[\, T\left(U_{m(n)}\right) = 1\big] \\ &\geq n^{-c} \end{aligned} \tag{1.28}$$

Wir erhalten, daß ein $i \in \{0, 1, \dots, m(n) - 1\}$ existiert mit

$$p_{i+1} - p_i \geq \frac{n^{-c}}{m(n)}, \tag{1.29}$$

denn sonst wäre die Summe in (1.28) kleiner als n^{-c}.

Wir konstruieren den statistischen Test T (Algorithmus 1.7.2) für die Generatorausgabe von G. Offenbar ist T ein probabilistischer Polynomialzeit-Algorithmus, da $m(n)$ durch ein Polynom in n beschränkt ist. Betrachten wir die Eingabe $x \in \{0,1\}^{n+1}$ des Tests T. Zunächst wählen wir i zufällige Bits $h_1, h_2, \dots, h_i$ und als weitere Bits übernehmen wir $G_{m(n)-i}(x)$. Je nach Verteilung der Eingabe x gilt:

- Für gemäß $G(U_n)$ verteilte Eingabe x ist h gemäß $H_n^{(i)} = \big(U_i \ , \ G_{m(n)-i}(U_n)\big)$ verteilt.
- Für gemäß U_{n+1} verteilte Eingabe x ist h gemäß $H_n^{(i+1)} = \big(U_{i+1} \ , \ G_{m(n)-i-1}(U_n)\big)$ verteilt.

Nach (1.29) gilt für unendlich viele n:

$$\Delta_T\left(G(U_n)\right) \geq \frac{1}{n^c \cdot m(n)}$$

Dies ist ein Widerspruch, da $m(n)$ durch ein Polynom beschränkt und G nach Voraussetzung ein kryptographisch sicherer Pseudozufallsgenerator ist. ■

Algorithmus 1.7.2 Statistischer Test T aus $T_{m(n)}$

EINGABE: $x \in \{0,1\}^{n+1}$

/* Sei $T_{m(n)}$ ein gegebener statistischer Test, den das Ensemble $\big(G_{m(n)}(U_n)\big)_{n\in\mathbb{N}}$ mit Toleranz n^{-c} nicht besteht. Dieser Algorithmus soll entscheiden, ob die Eingabe x gemäß $G(U_n)$ oder gemäß U_{n+1} verteilt ist. Sei i eine Bitposition mit (1.29). */

1. FOR $j := 1$ TO i DO $h_j \in_{\mathrm{R}} \{0,1\}$ /* wähle zufälliges Bit */

2. $(h_{i+1}, x_{i+1}) := x$ mit $h_{i+1} \in \{0,1\}$ und $x_{i+1} \in \{0,1\}^n$

3. FOR $j := i+2$ TO $m(n)$ DO

$$(h_j, x_j) := G(x_{j-1}) \text{ mit } h_j \in \{0,1\} \text{ und } x_j \in \{0,1\}^n$$

END for

4. $h := \big(h_1, h_2, \ldots, h_{m(n)}\big)$, d.h. $h = \big(h_1, h_2, \ldots, h_i, G_{m(n)-i}(x)\big)$.

5. Rufe statistischen Test $T_{m(n)}(h)$ auf und gib dessen Resultat aus.

Eine alternative Konstruktion zum Generator aus Satz 1.7.2, ist, den Pseudozufallsgenerator G $m(n)$-mal iterativ anzuwenden (Algorithmus 1.7.3). Um zu zeigen, daß $G'_{m(n)}$ ein Pseudozufallsgenerator ist, nimmt man an, es gäbe einen statistischen Test $T'_{m(n)}$, den das Ensemble $\Big(G'_{m(n)}(U_n)\Big)_{n\in\mathbb{N}}$ nicht besteht und erhält mit der Hybrid-Methode wie in Lemma 1.7.2 den Widerspruch, G sei kein Pseudozufallsgenerator. Als hybride Verteilungen nutzt man:

$$H^{(j)}_{m(n)} := \Big(U_j \ , \ G'_{m(n)-j}(U_n)\Big) \qquad \text{für } j := j(n) = 0, 1, \ldots, m(n)-n$$

Diese Konstruktion hat gegenüber der Konstruktion $G_{m(n)}$ aus Lemma 1.7.2 den Nachteil, daß der Algorithmus den Generator G auf Argumente der Länge $n, n+1, \ldots, m(n)-n$ anwendet, während $G_{m(n)}$ den Generator nur auf Argumente der Länge n insgesamt $m(n)$-mal anwendet.

Algorithmus 1.7.3 Pseudozufallsgenerator $G'_{m(n)}$ vom Typ $m(n)$

EINGABE: Startwert $x \in_{\mathrm{R}} \{0,1\}^n$

/* Sei G ein Pseudozufallsgenerator vom Typ $\ell(n) = n+1$. */

1. $x_0 := x$

2. FOR $j := 1$ TO $m(n)$ DO $x_j := G(x_{j-1}) \in \{0,1\}^{n+j}$

AUSGABE: $x_{m(n)-n} \in \{0,1\}^{m(n)}$

Nach Satz 1.7.2 folgt aus der Existenz eines Pseudozufallsgenerators, daß es insbesondere einen kryptographisch sicheren Generator des Typs $\ell(n) = 2n$ gibt.

Satz 1.7.3
Die Existenz von Oneway-Funktionen ist notwendig für die Existenz von kryptographisch sicheren Pseudozufallsgeneratoren.

Beweis. Sei G ein Pseudozufallsgenerator. Nach Satz 1.7.2 kann man voraussetzen, daß G vom Typ $\ell(n) = 2n$ ist. Wir zeigen, daß die Abbildung

$$f(x) := G(x) \in \{0,1\}^{2\cdot|x|}$$

eine Oneway-Funktion bildet. Offenbar ist f durch einen deterministischen Polynomialzeit-Algorithmus berechenbar.

Angenommen, f sei keine Oneway-Funktion, d.h. es gäbe eine Konstante $c \in \mathbb{N}$ und einen probabilistischen Polynomialzeit-Algorithmus A, so daß für unendlich viele n gilt:

$$\mathrm{Ws}[\, f\,(A(f(U_n))) = f(U_n)] \geq n^{-c} \tag{1.30}$$

Wir betrachten im weiteren nur diese n. Wir konstruieren einen statistischen Test T, den das nach Voraussetzung pseudozufällige Ensemble $(G(U_n))_{n\in\mathbb{N}}$ mit Toleranz $\delta(n) := n^{-(c+1)}$ nicht besteht:

$$T(x) := \begin{cases} 1 & \text{falls } f\,(A(x)) = x \\ 0 & \text{sonst} \end{cases}$$

Der statistische Test T gibt genau dann Eins aus, wenn A ein Urbild zu x findet. Wegen $f(U_n) = G(U_n)$ folgt aus Annahme (1.30):

$$\mathrm{Ws}[\, T\,(G(U_n)) = 1] = \mathrm{Ws}[\, f\,(A(f(U_n))) = f(U_n)] \geq n^{-c} \tag{1.31}$$

Da f durch einen deterministischen Algorithmus berechenbar ist, gilt $|f\,(\{0,1\}^n)| \leq 2^n$. Wir erhalten für beliebiges $x \in \{0,1\}^n$ die folgende Abschätzung:

$$\mathrm{Ws}[\, f(x) = U_{2n}] \leq \frac{|f\,(\{0,1\}^n)|}{|\{0,1\}^{2n}|} \leq \frac{2^n}{2^{2n}} = 2^{-n}$$

Dies bedeutet, daß für zufällige Eingabe aus $\{0,1\}^{2n}$ Algorithmus A nur mit Wahrscheinlichkeit höchstens 2^{-n} ein Urbild x unter f findet bzw. finden kann. Somit gilt:

$$\mathrm{Ws}[\, T\,(U_{2n}) = 1] = \mathrm{Ws}[\, f\,(A(U_{2n}))) = U_{2n}] \leq 2^{-n} \tag{1.32}$$

Aus (1.31) und (1.32) folgt für unendlich viele n:

$$\Delta_T(G) = \mathrm{Ws}[\, T\,(G(U_n)) = 1] - \mathrm{Ws}[\, T\,(U_{2n}) = 1] \geq n^{-c} - 2^{-n} \geq n^{-(c+1)}$$

Dies ist ein Widerspruch, da G nach Voraussetzung einen kryptographisch sicherer Pseudozufallsgenerator bildet. ■

Wir erhalten in Verbindung mit dem in Satz 1.7.1 auf Seite 50 zitierten Ergebnis von J.Håstad, R.Impagliazzo, L.A.Levin und M.Luby:

Korollar 1.7.4
Kryptographisch sichere Pseudozufallsgeneratoren existieren genau dann, wenn es Oneway-Funktionen gibt.

Die Existenz von Oneway-Funktionen wird zwar vermutet, konnte aber bisher nicht bewiesen werden. Man nimmt zum Beispiel an, daß die Exponentiation modulo einer Primzahl eine Oneway-Permutation ist. Sei p eine n-Bit Primzahl, $\alpha \in \mathbb{Z}_p^*$ eine primitive Wurzel modulo p und $x \in \mathbb{Z}_p^*$. Die Funktion bildet x auf $\alpha^x \bmod p$ ab:

$$\mathrm{Exp} : (p, \alpha, x) \mapsto (p, \alpha, \alpha^x \bmod p)$$

Es sind keine effizienten Algorithmen bekannt, um den *diskreten Logarithmus* modulo p zu berechnen, wenn $p-1$ einen großen Primfaktor enthält. Der Anteil der Primzahlen p, so daß $\frac{p-1}{2}$ nur aus kleinen Primfaktoren besteht, ist vernachlässigbar. M. Blum und S. Micali [BM84] wenden ihre Konstruktion für einen Pseudozufallsgenerator mit der (vermuteten) Oneway-Permutation Exponentiation modulo einer Primzahl p und dem Hardcore-Prädikat

$$h_{\mathrm{DL}}(p, \alpha, x) := \left[x > \tfrac{p-1}{2}\right]$$

zu $x \in \mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$ an. D.L. Long und A. Wigderson [LW88] haben dieses Hardcore-Prädikat zu einer Hardcore-Funktion mit logarithmisch vielen Bits erweitert. Die Sicherheit der unteren Bits hängt von der Form der Primzahl p ab. R. Peralta [Peralta85] zeigt u.a., daß für Primzahlen $p \equiv 3 \pmod 4$ das zweit niedrigstwertige Bit sicher ist (dies gilt zum Beispiel für $p = 2q + 1$, wobei q eine Primzahl ist). Das niedrigstwertige Bit von x entspricht dem Jacobi-Symbol von α^x modulo p und kann stets effizient berechnet werden. Im Fall, daß der Modul eine Blumzahl, d.h. das Produkt zweier Primzahlen kongruenz 3 modulo 4 ist (vergleiche Kapital 3.1), zeigen J.Håstad, A.W.Schrift und A.Shamir[HSS93] daß $\Theta(n)$ viele Bits simultan sicher sind. Die Sicherheit des diskreten Logarithmus' für diese Module beruht auf dem Problem der Faktorisierung.

In Kapitel 2 lernen wir den RSA- und in Kapitel 3 den x^2-mod-N-Generator kennen und geben einen neuen, verbesserten Sicherheitsbeweis. Diese beiden Generatoren sind populär und basieren auf den anerkannten RSA- und Rabin-Kryptosystemen, deren Sicherheit auf dem Problem der Faktorisierung großer Zahlen beruht. Dekodieren einer zufälligen Nachricht im Rabin-System ist äquivalent zum Zerlegen des Moduls in die beiden Primfaktoren, das RSA-System kann durch Faktorisieren des Moduls gebrochen werden (die Umkehrung der Aussage wird jedoch nur vermutet).

Ein weiterer Pseudozufallsgenerator stammt von J.B. Fischer und J. Stern [FSt96], der auf dem Problem der Syndrom-Dekodierung basiert. R. Impagliazzo und M. Noar [IN96] stellen einen Generator basierend auf dem Subsetsum-Problem vor. Die Hardcore-Funktionen für letzten beiden Abbildungen liefern sogar asymptotisch mehr als logaritmisch viele Bits.

Kapitel 2

RSA-Generator

Wir stellen den RSA-Pseudozufallsgenerator vor. Wir geben den bekannten Sicherheitsbeweis [ACGS88], der auf dem binären ggT-Algorithmus basiert, wieder und stellen den neuen Sicherheitsbeweis [FSch97] vor, der auf einer sukzessive verbesserten Approximation beruht. Wir schließen das Kapitel über den RSA-Generator mit Anmerkung zu Wahl der Parameter in der Praxis.

2.1 RSA-Kryptosystem und -Generator

R.L. Rivest, A. Shamir und L. Adleman [RSA78] haben 1978 ein Public-Key-Kryptosystem vorgestellt. Das System beruht auf elementarer Zahlentheorie und konnte bis heute trotz intensiver Forschung nicht gebrochen werden. Dieses System ist wegen der Einfachheit und der Effizienz populär, da sowohl Verschlüsseln als auch Entschlüsseln durch eine Exponentiation modulo N bewerkstelligt werden. Das Kryptosystem wird in Anlehnung an seine Erfinder RSA-System genannt.

Public-Key bedeutet, daß der Schlüssel zum Kodieren („to encrypt“) öffentlich bekannt ist, den Schlüssel zum Dekodieren („to decrypt“) aber nur der rechtmäßige Empfänger der verschlüsselten Nachricht kennt. Es muß praktisch unmöglich sein, aus dem öffentlichen Schlüssel (Public-Key) den geheimen Schlüssel (Secret-Key) zu berechnen. Dies setzt voraus, daß das Schlüsselpaar zufällig generiert wird.

Betrachten wir das RSA-System und die Wahl der Schlüssel. Zu gegebenem Sicherheitsparameter $n > 3$ wählen wir für das RSA-System einen zufälligen *RSA-Modul* N mit bekannter Faktorisierung aus der Menge

$$\mathcal{R}_n := \left\{ pq \;\middle|\; \begin{array}{l} p, q \text{ sind verschiedene Primzahlen mit} \\ 2^{\frac{n-2}{2}} < p < 2^{\frac{n-1}{2}} \text{ und } 2^{\frac{n}{2}} < q < 2^{\frac{n+1}{2}} \end{array} \right\}$$

mit Bitlänge n und Primfaktoren p und q (wir werden später klären, wie man dieses effizient bewerkstelligt). Da die beiden Primfaktoren p und q ungerade sind, ist auch N nicht durch 2 teilbar. Die Gruppe $\mathbb{Z}_N^*$ bildet den Nachrichtenraum und ist auch die Menge der verschlüsselten Nachrichten. Wir wählen zusätzlich einen zufälligen Kodierexponenten $e \in [3, 2^n)$, der teilerfremd zu $\varphi(N) = (p-1)(q-1)$ ist. Der Exponent $e = 1$ verschlüsselt die Nachricht nicht

und $e = 2$ scheidet aus, weil $\varphi(N)$ gerade ist. Mit Kenntnis der Primfaktoren p und q bestimmt man den Wert der Euler'sche Funktion $\varphi(N)$. Mit Hilfe des erweiterten Euklidischen Algorithmus' ermitteln wir den geheimen Dekodierexponenten $d \in [0, \varphi(N))$ mit $de \equiv 1 \bmod \varphi(N)$, so daß ein $z \in \mathbb{Z}$ mit

$$de = z \cdot \varphi(N) + 1 \tag{2.1}$$

existiert (An diese Stelle nutzen wir, daß e teilerfremd zu $\varphi(N)$ ist). Den öffentlichen Schlüssel bildet das Paar (N, e). Die Menge aller öffentlichen Schlüssel mit n-Bit-RSA-Modul sei:

$$\text{RSA-PK}_n := \left\{ (N, e) \;\middle|\; \begin{array}{l} N \in \mathcal{R}_n \text{ und } e \in [3, 2^n) \\ \text{teilerfremd zu } \varphi(N) \end{array} \right\}$$

Der geheime Schlüssel zu (N, e) ist der Dekodierexponent d mit Eigenschaft (2.1). Insbesondere sind beide Primfaktoren von N geheim.

Um eine Nachricht $x \in \mathbb{Z}_N^*$ bezüglich des öffentlichen Schlüssels (N, e) zu chiffrieren, erhebt man x in die e-te Potenz modulo N, wendet die Funktion $\text{RSA}_{N,e} : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ mit

$$\text{RSA}_{N,e}(x) := x^e \bmod N \tag{2.2}$$

an. Um eine verschlüsselte Nachricht $\text{RSA}_{N,e}(x)$ zu dechiffrieren, erhebt der rechtmäßige Empfänger die erhaltene Nachricht in die d-te Potenz modulo N:

$$\begin{aligned} (\text{RSA}_{N,e}(x))^d &\equiv x^{ed} \\ &\equiv \left(x^{\varphi(N)}\right)^z \cdot x && \text{(wegen (2.1))} \\ &\equiv 1^z \cdot x && \text{(Satz von Lagrange)} \\ &\equiv x \pmod N \end{aligned}$$

Die Abbildung $x \mapsto x^d \bmod N$ ist die inverse Funktion zu $\text{RSA}_{N,e}$. Wir erhalten:

Satz 2.1.1
Sei N ein RSA-Modul und $e \in \mathbb{N}$ teilerfremd zu $\varphi(N)$. Dann gilt:

a) Die Funktion $\text{RSA}_{N,e} : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ mit $\text{RSA}_{N,e}(x) := x^e \bmod N$ ist eine Permutation.

b) Für alle $x, y \in \mathbb{Z}_N^*$ gilt $\text{RSA}_{N,e}(xy) \equiv \text{RSA}_{N,e}(x) \cdot \text{RSA}_{N,e}(y) \pmod N$, d.h. die Abbildung ist ein Gruppen-Homomorphismus. Diese Kongruenz gilt auch für Werte aus $\mathbb{Z}_N$.

Beweis. Aussage a) ist eine Konsequenz der inversen Abbildung zu $\text{RSA}_{N,e}$, Behauptung b) zeigt man durch direktes Nachrechnen. ■

Die $\text{RSA}_{N,e}$-Funktion kann auf $\mathbb{Z}_N$ erweitert werden, allerdings ist diese Abbildung dann keine Permutation mehr.

Für die zufällige Wahl eines RSA-Moduls mit bekannter Primfaktorisierung genügt es, ein effizientes Verfahren für die zufällige Wahl einer Primzahl aus dem Intervall $\left[2^{n-1}, 2^n\right)$

anzugeben. Nach dem Primzahlsatz gilt für die Anzahl $\pi(x)$ der Primzahlen kleiner oder gleich x:

$$\lim_{x\to\infty} \pi(x) = \frac{x}{\ln x}$$

Wir können eine zufällige Zahl p aus dem Intervall $\left[2^{n-1}, 2^n\right)$ wählen und mit einem Primzahltest, wie zum Beispiel dem Verfahren von R. Solovay und V. Strassen [SoSt77, SoSt77], testen, ob p eine Primzahl ist. Falls p prim ist, besteht sie den Solovay-Strassen-Test und falls sie eine zusammengesetzte Zahl ist, nur mit Wahrscheinlichkeit maximal 2^{-n}. Nach dem Primzahlsatz müssen wir im Durchschnitt $\mathcal{O}(n)$ Zahlen testen, bis man eine Primzahl findet. Für große n ist dieses Vorgehen zu aufwendig. Statt dessen bietet es sich an, Primzahlen iterativ zu konstruieren:

1. Wir wählen Primzahlen $p_1, p_2, \ldots, p_r$ sowie Exponenten $e_1, e_2, \ldots, e_r$ und berechnen $F := \prod_{i=1}^{r} p_i^{e_i}$.

2. Wir wählen zufällige $R < F$, bis man $2RF + 1$ als prim nachweisen kann.

Dieser Algorithmus von U. Maurer [Maurer95] hat die erwartete Laufzeit $\mathcal{O}\left(\frac{n^{3,6}}{\log_2 n}\right)$ für die Erzeugung einer n-Bit-Primzahl. Einen speziellen Algorithmus für die Erzeugung *starker Primzahlen* p, d.h. $p = 2q + 1$ für eine weitere Primzahl q, stellt R.D Silverman [Sil97] vor.

Die Sicherheit des RSA-System basiert auf der Annahme, daß für zufälligen Public-Key $(N, e) \in_{\mathrm{R}} \text{RSA-PK}_n$ und zufällige Nachricht $x \in_{\mathrm{R}} \mathbb{Z}_N^*$ die Abbildung

$$\text{RSA}: \quad (N, e, x) \mapsto (N, e, \text{RSA}_{N,e}(x)) \tag{2.3}$$

eine Oneway-Funktion bzw. eine Oneway-Permutation ist. Wir können aus N und e den geheimen Dekodierexponenten $d = e^{-1} \bmod \varphi(N)$ bei Kenntnis von $\varphi(N)$ bestimmen. $\varphi(N)$ kann aus den beiden Primfaktoren p und q des RSA-Moduls N berechnet werden. Die Kenntnis von

$$\varphi(N) = (p-1)(q-1) = N - p - q + 1$$

ist aber auch äquivalent zur Kenntnis der beiden Primfaktoren, denn wenn man q durch $\frac{N}{p}$ ersetzt, folgt $N - p - \varphi(N) + 1 = \frac{N}{p}$ und wir erhalten den Primfaktor p als eine Lösung der quadratischen Gleichung:

$$p^2 - (N - \varphi(N) + 1)\, p + N = 0$$

Faktorisieren des RSA-Moduls ist trotz intensiver Forschung der einzige bekannte, allgemeine Angriff auf das RSA-System. Wir werden in Kapitel 2.4 untersuchen, wie groß der Sicherheitsparameter n gewählt werden muß, damit es beim heutigen Stand der Forschung und der Technik praktisch unmöglich ist, den Modul zu faktorisieren. Für eine Übersicht über spezielle Angriffe auf das RSA-Schema verweisen wir auf den Übersichtsartikel [KR95] von B. Kaliski und M. Robshaw.

Algorithmus 2.1.1 RSA-Pseudozufallsgenerator

EINGABE: ▷ zufälliger RSA-Public-Key $(N, e) \in_{\mathrm{R}} \text{RSA-PK}_n$
▷ zufälliger Startwert $x \in_{\mathrm{R}} \mathbb{Z}_N^*$

/* Sei $\ell(n) \geq n + 1$ durch ein Polynom beschränkt. */

1. $x_0 := x$

2. FOR $j := 1$ TO $\ell(n)$ DO

2.1. $x_j := \mathrm{RSA}_{N,e}(x_{j-1})$ /* $x_j = x_{j-1}^e \bmod N$ */

2.2. $b_j := \mathrm{lsb}_N(x_{j-1})$ /* unterstes Bit von $x_{j-1} \in [0, N)$ */

END for

AUSGABE: $\left(b_1, b_2, \ldots, b_{\ell(n)}\right) \in \{0,1\}^{\ell(n)}$

Unter der Annahme, daß die Funktion RSA aus (2.3) eine Oneway-Permutation ist, kann man mit einem Hardcore-Prädikat zu RSA bzw. $\mathrm{RSA}_{N,e}$ einen kryptographisch sicheren Pseudozufallsgenerator mit der Blum-Micali-Konstruktion aus Kapitel 1.5 angeben. Wir bezeichnen mit $\mathrm{bit}_{N,j}(x)$ das j-te Bit der Dualdarstellung von $x \bmod N$:

$$\mathrm{bit}_{N,j}(x) := \left\lfloor \frac{x \bmod N}{2^{j-1}} \right\rfloor \bmod 2$$

Speziell sei $\mathrm{lsb}_N(x) := \mathrm{bit}_{N,1}(x)$ das unterste Bit (engl. „least significant bit“). Wir zeigen in Kapitel 2.2 und 2.3:

Satz 2.1.2
Unter der Annahme, daß für zufälligen RSA-Public-Key $(N, e) \in_{\mathrm{R}} \text{RSA-PK}_n$ *und zufällige Nachricht* $x \in_{\mathrm{R}} \mathbb{Z}_N^*$ *die Abbildung*

$$\mathrm{RSA}: \quad (N, e, x) \mapsto (N, e, \mathrm{RSA}_{N,e}(x))$$

mit $\mathrm{RSA}_{N,e}(x) := x^e \bmod N$ *eine Oneway-Permutation ist, bildet für jedes* $j(n) \geq 1$ *mit* $j(n) = \mathcal{O}(\log_2 n)$ *die Funktion*

$$h_{j(n)}(N, e, x) := \mathrm{bit}_{N,j(n)}(x)$$

ein Hardcore-Prädikat zu RSA.

Für die Beweise in Kapitel 2.2 und 2.3 nehmen wir an, es gäbe einen probabilistischen Polynomialzeit-Algorithmus, der zu zufälligem RSA-Public-Key $(N, e) \in_{\mathrm{R}} \text{RSA-PK}_n$ und zufälliger Nachricht $x \in_{\mathrm{R}} \mathbb{Z}_N^*$ aus $(N, e, \mathrm{RSA}_{N,e}(x))$ das Bit $\mathrm{bit}_{N,j(n)}(x)$ mit Vorteil n^{-c} für ein festes $c \in \mathbb{N}$ berechnet. Wir zeigen, daß man dann in der Lage ist, für unendlich viele n mit nicht-vernachlässigbarer Wahrscheinlichkeit aus $(N, e, \mathrm{RSA}_{N,e}(x))$ die Nachricht x in Polynomialzeit zu berechnen. Dieses widerspricht der Annahme, daß die RSA-Abbildung eine Oneway-Permutation ist. Insbesondere folgt aus Satz 2.1.2 für den Generator 2.1.2:

Korollar 2.1.3
Sei $\ell(n) \geq n+1$ durch ein Polynom beschränkt. Unter der Annahme, daß die RSA*-Abbildung eine Oneway-Permutation ist, handelt es sich bei Algorithmus 2.1.2 um einen kryptographisch sicheren Pseudozufallsgenerator vom Typ $\ell(n)$.*

Beweis. Satz 1.5.5 auf Seite 26 gilt auch für die Argumente der RSA-Abbildung. In Verbindung mit Satz 2.1.2 erhalten wir die Behauptung. ■

Korollar 2.2.11 auf Seite 75 und alternativ Korollar 2.3.10 auf Seite 93 zeigen, wie schnell wir die RSA-Funktion invertieren können, wenn das vom RSA-Generator erzeugte Ensemble einen statistischen Test mit $\delta(n)$-Toleranz nicht besteht.

Wir zeigen ferner in Kapitel 2.2 und 2.3, daß die logarithmisch vielen untersten Bits simultan sicher sind. Wir bezeichnen mit $\mathrm{Bits}_{N,k}(x)$ die k untersten Bits der Dualdarstellung von $x \bmod N$:

$$\mathrm{Bits}_{N,k}(x) := (x \bmod N) \bmod 2^k$$

Nach folgendem Satz können in jeder Iteration des RSA-Generators 2.1.1 $k(n) = \mathcal{O}(\log_2 n)$ Bits, nämlich $\mathrm{Bits}_{N,k(n)}(x_{j-1})$, ausgeben werden:

Satz 2.1.4
Unter der Annahme, daß für zufälligen RSA-Public-Key $(N,e) \in_{\mathrm{R}}$ RSA-PK$_n$ und zufällige Nachricht $x \in_{\mathrm{R}} \mathbb{Z}_N^*$ die Abbildung

$$\mathrm{RSA}: \quad (N,e,x) \mapsto (N,e,\mathrm{RSA}_{N,e}(x))$$

mit $\mathrm{RSA}_{N,e}(x) := x^e \bmod N$ eine Oneway-Permutation ist, bildet für $k(n) \geq 1$ mit $k(n) = \mathcal{O}(\log_2 n)$ die Funktion

$$h_{k(n)}(N,e,x) := \mathrm{Bits}_{N,k(n)}(x)$$

eine Hardcore-Funktion zu RSA*.*

Korollar 2.2.16 auf Seite 78 zeigt, wie schnell wir die RSA-Funktion invertieren können, wenn das vom RSA-Generator mit der Hardcore-Funktion erzeugte Ensemble einen statistischen Test mit $\delta(n)$-Toleranz nicht besteht.

Wir haben in diesem Kapitel die theoretischen Ergebnisse der Kapitel 2.2 und 2.3 zusammengefaßt. In Kapitel 2.4 untersuchen wir, wie man die Bitlänge n des Moduls wählen soll, um den Anforderungen der Praxis zu genügen.

2.2 Sicherheitsbeweis mit binärem ggT-Algorithmus

Wir stellen den Sicherheitsbeweis von W. Alexi, B. Chor, O. Goldreich und C.P. Schnorr [ACGS88] vor, wobei sich unsere Darstellung an [Knuth96] anlehnt. Wir nehmen an, es gäbe einen probabilistischen Algorithmus A, der für zufälligen RSA-Public-Key $(N,e) \in_{\mathrm{R}}$ RSA-PK$_n$ und zufällige Nachricht $x \in_{\mathrm{R}} \mathbb{Z}_N^*$ zu gegebenem $(N,e,\mathrm{RSA}_{N,e}(x))$ das j-te Bit

$\mathrm{bit}_{N,j}(x)$ mit Vorteil $\epsilon(n)$ berechnet. Wir betrachten nur diese n und zeigen, daß falls A ein Polynomialzeit-Algorithmus ist und $j = \mathcal{O}(\log_2 n)$ gilt, wir bezüglich n und $\epsilon(n)^{-1}$ in Polynomialzeit für diese unendlich vielen n die RSA-Funktion mit nichtvernachlässigbarer Wahrscheinlichkeit invertieren können (wir setzen voraus, daß $\epsilon(n)^{-1} = n^{\mathcal{O}(1)}$ ist). Dies widerspricht der Annahme, daß die RSA-Funktion eine Oneway-Permutation sei. Unser Beweis gliedert sich in zwei Schritte:

1. Wir definieren den Absolutbetrag modulo N und zeigen, wie man mit einem Algorithmus B zur Berechnung des j-ten Bits des Urbildabsolutbetrags modulo N die RSA-Funktion invertiert.
2. Wir konstruieren aus Algorithmus A zur Berechnung des j-ten Bits modulo N den Algorithmus B.

Zur Vereinfachung sei zunächst für die betrachteten n die Erfolgswahrscheinlichkeit von Algorithmus A unabhängig von N und e. Wir untersuchen den Spezialfall $j = 1$ (d.h. das unterste Bit) und verallgemeinern dann unseren Ansatz.

$$\begin{array}{lll} & 0 & \\ N-1 \equiv -1 & & 1 \\ N-2 \equiv -2 & & 2 \end{array}$$

$$N - \tfrac{N-1}{2} \equiv -\tfrac{N-1}{2} \qquad \tfrac{N-1}{2}$$

Abbildung 2.2.1: Absolutbetrag modulo N

Wir betrachten Zahlen im Ring $\mathbb{Z}$ der ganzen Zahlen als auch im Faktorring $\mathbb{Z}_N$. Um die Ringe zu unterscheiden, führen wir für die Zahlen bzw. Restklassen des Faktorrings $\mathbb{Z}_N$ andere Notationen ein. Wir stellen die Restklassen modulo N sowohl durch ihren kleinsten, nicht-negativen Repräsentanten, d.h. $[0, N-1]$, als auch durch ihre betragsmäßig kleinsten Repräsentanten, d.h. $\left[-\frac{N-1}{2}, +\frac{N-1}{2}\right]$, dar. Um beide Darstellungen zu kennzeichnen, nutzen wir folgende Notation

$$\begin{aligned} [z]_N &:= (z \bmod N) \in \{0, 1, \ldots, N-1\} \\ [z]_{\pm N} &:= (z \underline{\bmod} N) \in \left\{-\tfrac{N-1}{2}, -\tfrac{N-3}{2}, \ldots, +\tfrac{N-1}{2}\right\}, \end{aligned}$$

wobei $z \underline{\bmod} N = z - \left\lfloor \frac{z}{N} + \frac{1}{2} \right\rfloor N$ ist. Alle arithmetischen Operationen sind in $\mathbb{Z}$. Wir definieren den Absolutbetrag abs_N modulo N zu $z \in \mathbb{Z}$ gemäß:

$$\mathrm{abs}_N(z) := \left|[z]_{\pm N}\right| = \begin{cases} [z]_N & \text{falls } [z]_N \leq \frac{N-1}{2} \\ N - [z]_N & \text{sonst} \end{cases}$$

Der Absolutbetrag $\mathrm{abs}_N(z)$ entspricht dem Abstand von $[z]_N$ zur 0 (vergleiche Abbildung 2.2.1). Insbesondere gilt $\mathrm{abs}_N(z) = \mathrm{abs}_N(-z)$, und falls $\mathrm{abs}_N(z)$ gerade ist, folgt:

$$\tfrac{1}{2} \cdot \mathrm{abs}_N(z) = \mathrm{abs}_N(2^{-1}z)$$

Wir bezeichnen mit $\mathrm{abs}_{N,j}(z)$ das j-te Bit des Absolutbetrags $\mathrm{abs}_N(z)$:

$$\mathrm{abs}_{N,j}(z) := \left\lfloor \frac{\mathrm{abs}_N(z)}{2^{j-1}} \right\rfloor \bmod 2$$

Statt in der Gruppe $\mathbb{Z}_N^*$ arbeiten wir im Ring $\mathbb{Z}_N$. Wir können bei bezüglich der Modullänge n polynomieller Laufzeit der Algorithmen die Wahrscheinlichkeit, daß ein in $\mathbb{Z}_N$ gleichverteilter Wert nicht invertierbar modulo N ist, vernachlässigen:

Lemma 2.2.1
Sei $N \in \mathcal{R}_n$. Die Wahrscheinlichkeit, daß ein zufälliger Wert aus $\mathbb{Z}_N$ den Modul N teilt, ist vernachlässigbar:

$$\frac{|\mathbb{Z}_N^*|}{|\mathbb{Z}_N|} = \frac{\varphi(N)}{N} \geq 1 - 2^{-\frac{n-3}{2}}$$

Beweis. Sei $N = pq$ die Primfaktorzerlegung mit $p < q$. Wegen $\varphi(N) = (p-1)(q-1)$ gilt:

$$\frac{\varphi(N)}{N} = \frac{(p-1)(q-1)}{N} = \frac{pq - p - q + 1}{N} \geq \frac{N - 2p}{N} = 1 - \frac{2p}{N}$$

Nach Definition von $\mathcal{R}_n$ ist $N \geq 2^{\frac{n-2}{2}} \cdot 2^{\frac{n}{2}} = 2^{n-1}$ und $p \leq 2^{\frac{n-1}{2}}$, so daß gilt:

$$\frac{\varphi(N)}{N} \geq 1 - \frac{2^{\frac{n+1}{2}}}{2^{n-1}} = 1 - 2^{-\frac{n-3}{2}}$$

Die Wahrscheinlichkeit, daß ein zufälliger Wert aus $\mathbb{Z}_N^*$ den Modul $N \in \mathcal{R}_n$ teilt, ist vernachlässigbar. ■

2.2.1 Binärer ggT-Algorithmus

Neben dem RSA-Modul N und dem Kodierexponenten e kennen wir auch die verschlüsselte Nachricht $\mathrm{RSA}_{N,e}(x)$. Unser Ziel (im Fall $j = 1$, d.h. des untersten Bits) ist die Bestimmung von $a \in \mathbb{Z}_N^*$ mit $\mathrm{RSA}_{N,e}(ax) = \pm 1 \bmod N$, denn da die $\mathrm{RSA}_{N,e}$-Funktion eine Permutation ist, gilt

$$\mathrm{RSA}_{N,e}(ax) \equiv \pm 1 \pmod N \quad \Longleftrightarrow \quad [ax]_{\pm N} = \pm 1 \quad \Longleftrightarrow \quad x = \left[\pm a^{-1}\right]_N$$

und wir erhalten das Urbild $x \in \mathbb{Z}_N^*$. Man kann zu $\mathrm{RSA}_{N,e}(x)$ für jedes $a \in \mathbb{Z}_N$ den Wert $\mathrm{RSA}_{N,e}(ax)$ ohne Kenntnis von x effizient berechnen, da nach Satz 2.1.1.a) auf Seite 56 gilt:

$$\mathrm{RSA}_{N,e}(ax) \equiv a^e \cdot \mathrm{RSA}_{N,e}(x) \pmod N$$

Allgemeiner sind wir in der Lage, zu $a, b \in \mathbb{Z}_N$ und $\mathrm{RSA}_{N,e}(x)$ zu berechnen:

$$\mathrm{RSA}_{N,e}\left((a \pm b)x\right) \equiv (a \pm b)^e \cdot \mathrm{RSA}_{N,e}(x) \pmod N \tag{2.4}$$

Wenn wir $a, b \in \mathbb{Z}_N$ zufällig und unabhängig wählen, sind $[ax]_{\pm N}$ und $[ab]_{\pm N}$ zufällig und unabhängig in $\left[-\frac{N-1}{2}, +\frac{N-1}{2}\right]$ verteilt. Nach einem bekannten Satz von G.L. Dirichlet (1849) [Knuth81, Kapitel 4.5.2, Theorem D] konvergiert die Wahrscheinlichkeit, daß $[ax]_{\pm N}$ und $[bx]_{\pm N}$ teilerfremd sind, für N gegen unendlich gegen $\frac{6}{\pi} \approx 0,61$. Auf M. Ben-Or, B. Chor und A. Shamir [BCS83] geht die Idee zurück, daß man den größten gemeinsamen Teiler von $[ax]_{\pm N}$ und $[bx]_{\pm N}$ mit einem binären ggT-Algorithmus berechnen kann.

Algorithmus 2.2.1 Binärer ggT-Algorithmus von R.P. Brent und H.T. Kung

---

EINGABE: $u, v \in \mathbb{Z}$ mit u ungerade

1. $c := 0$ /* c dient als Anhaltspunkt für die Differenz $\log_2 |u| - \log_2 |v|$ */

2. WHILE $v \neq 0$ DO

2.1. WHILE (v ist gerade) DO $v := \frac{1}{2}v$ und $c := c + 1$.

2.2. IF $c > 0$ THEN vertausche u und v, setze $c := -c$.

/* u, v sind ungerade und es gilt $c \leq 0$. */

2.3. $w := \frac{u+v}{2}$

2.4. IF (w ist gerade) THEN $v := w$ ELSE $v := w - v$. /* v ist gerade. */

END while

AUSGABE: Absolutbetrag von u.

Algorithmus 2.2.1 zeigt den binären ggT-Algorithmus von R.P. Brent und H.T. Kung [Knuth96, neue Aufgabe 37 zu Kapitel 4.5.3]. Dieses Verfahren hat gegenüber anderen binären ggT-Algorithmen den Vorteil, daß es das Vorzeichen von $u - v$ d.h. ob $u \geq v$ ist, nicht auswertet.

Lemma 2.2.2
Sei u, v die Eingabe für den Brent-Kung-Algorithmus 2.2.1 mit $k := \log_2 \max\{|u|, |v|\}$. Dann gilt:

a) Wir erhöhen in Schritt 2.1 den Zähler c maximal $2k$-mal.

b) Wir wiederholen die Schleife in Schritt 2 maximal $(1 + 2k)$-mal.

c) Der Betrag der auftretenden Werte für u, v, w ist höchstens $\max\{|u|, |v|\}$.

d) Die Ausgabe von Algorithmus 2.2.1 ist gleich $\mathrm{ggT}(u, v)$.

Beweis. Sei m die Anzahl der Erhöhungen von c in Schritt 2.1. Durch Induktion über die Anzahl der Iterationen zeigt man die Schleifeninvariante:

$$|u| \leq 2^{k - \frac{m-c}{2}} \qquad |v| \leq 2^{k - \frac{m+c}{2}}$$

Wir erhalten $m \leq 2k$ und die Behauptung a). Mit Ausnahme der ersten Iteration ist v zu Beginn gerade, so daß in Schritt 2.1 dann jeweils c erhöht wird. Wir führen die Schleife

maximal $(1 + 2k)$-mal aus (Behauptung b)). Behauptung c) kann man induktiv für jeden Schritt der Schleife nachweisen (für Schritt 2.4 beachte, daß $w - v = \frac{u-v}{2}$ ist). Ebenso zeigt man, daß der größte gemeinsame Teiler von u und v in jedem Schritt erhalten bleibt und damit am Ende gleich $\mathrm{ggT}(u, 0)$ ist. ■

Wir nehmen an, daß uns ein Algorithmus zur Bestimmung des untersten Bits des Absolutbetrags modulo N von z bei gegebenem $\mathrm{RSA}_{N,e}(z)$ mit Wahrscheinlichkeit $1-\frac{1}{12n-9}$, also nahe bei 1, zur Verfügung steht. Mit diesem Algorithmus können wir zu $\mathrm{RSA}_{N,e}(z)$ entscheiden, ob $[z]_{\pm N}$ gerade ist.

Algorithmus 2.2.2 RSA-Invertierung mit binärem ggT-Algorithmus

---

EINGABE: ▷ RSA-Public-Key $(N, e) \in \text{RSA-PK}_n$
▷ $\mathrm{RSA}_{N,e}(x)$ mit $x \in \mathbb{Z}_N^*$

/∗ Sei B ein Algorithmus mit $\mathrm{Ws}\left[B\left(N, e, \mathrm{RSA}_{N,e}\left(U_{\mathbb{Z}_N^*}\right)\right) = \mathrm{abs}_{N,1}\left(U_{\mathbb{Z}_N^*}\right)\right] \geq 1 - \frac{1}{12n-9}$. ∗/

1. Wähle $a, b \in \mathbb{Z}_N$ zufällig und unabhängig.

/∗ Berechne den größten gemeinsamen Teiler von $[ax]_{\pm N}$ und $[bx]_{\pm N}$ mit binärem ggT-Algorithmus 2.2.1 und mit Hilfe des Algorithmus' B als Test, ob $[ax]_{\pm N}$ bzw. $[bx]_{\pm N}$ gerade ist. ∗/

2. $c := 0$

3. WHILE $(\mathrm{RSA}_{N,e}(bx) \neq 0)$ DO

/∗ Verlasse die While-Schleife vorzeitig, wenn Algorithmus B zum $(4n - 2)$-ten Male aufgerufen werden soll. ∗/

3.1. WHILE $B\left(N, e, \mathrm{RSA}_{N,e}(bx)\right) = 0$ DO $b := \left[2^{-1}b\right]_N$ und $c := c + 1$

3.2. IF $c > 0$ THEN vertausche a und b, setze $c := -c$.

3.3. $w := \left[2^{-1}(a + b)\right]_N$

3.4. IF $B\left(N, e, \mathrm{RSA}_{N,e}(wx)\right) = 0$ THEN $b := w$ ELSE $b := w - a$.

3.5. $b := \left[2^{-1}b\right]_N$ und $c := c + 1$.

END while

/∗ Falls $[ax]_{\pm N}$ und $[bx]_{\pm N}$ bei der Wahl in Schritt 1 teilerfremd waren und $[ax]_{\pm N}$ ungerade, gilt bei richtigen Resultaten von B jetzt $[ax]_{\pm N} = \pm 1$ und wir erhalten das Urbild x. ∗/

4. IF $\mathrm{ggT}(a, N) \neq 1$ THEN berechne geheimen Schlüssel d und gib $\left[\mathrm{RSA}_{N,e}(x)^d\right]_N$ aus.

5. IF $\mathrm{RSA}_{N,e}\left(+a^{-1}\right)\big) = \mathrm{RSA}_{N,e}(x)$ THEN gib $\left[+a^{-1}\right]_N$ aus

6. IF $\mathrm{RSA}_{N,e}\left(-a^{-1}\right)\big) = \mathrm{RSA}_{N,e}(x)$ THEN gib $\left[-a^{-1}\right]_N$ aus

---

Um zu $\mathrm{RSA}_{N,e}(x)$ das Urbild zu ermitteln, wählen wir $a, b \in \mathbb{Z}_N$ zufällig und unabhängig und berechnen mit Verfahren 2.2.2 wie beim binären ggT-Algorithmus 2.2.1 den größten gemeinsamen Teiler von $[ax]_{\pm N}$ und $[bx]_{\pm N}$. Da wir x nicht kennen, entscheiden wir mit Algorithmus B, ob das Argument gerade ist. Durch die zusätzlichen Faktoren a und b kann man jede Operation des ggT-Algorithmus' nachvollziehen (siehe Gleichung (2.4) auf Seite

61). Falls $[ax]_{\pm N}$ und $[bx]_{\pm N}$ teilerfremd sind sowie $[ax]_{\pm N}$ ungerade ist, erhalten wir eine Darstellung $[\bar{a}x]_{\pm N} = \pm 1$ mit bekanntem $\bar{a} \in \mathbb{Z}_N$ und schließlich das Urbild x aus

$$x = \left[\pm \bar{a}^{-1}\right]_N .$$

Falls $\bar{a}$ kein multiplikatives Inverses modulo N hat, erhalten wir aus $\mathrm{ggT}(\bar{a}, N)$ einen Primfaktor von N und können mit der Primfaktorzerlegung des Moduls N den geheimen Schlüssel d berechnen.

Nach Lemma 2.2.2.c) sind die auftretenden Werte sämtlich absolut kleiner als $\frac{1}{2}N$, so daß man durch die Rechnung modulo N die Werte nicht ändert. Der Vorteil des Brent-Kung-Verfahren gegenüber anderen binären ggT-Algorithmen ist, daß man das Vorzeichen von $[ax]_{\pm N} - [bx]_{\pm N}$ nicht bestimmen müssen, zumal wir den Wert $[x]_N$ nicht kennen.

Wir modifzieren den binären ggT-Algorithmus 2.2.1. Da nicht auszuschließen ist, daß Algorithmus B ein falsches Ergebnis liefert, führen wir einen Zähler für die Anzahl der Aufrufe von B ein, um diese in der Laufzeitanalyse zu beschränken. Durch eine weitere Änderung reduzieren wir die Anzahl der Anrufe von B: Da nach Schritt 3.4 in Algorithmus 2.2.2 stets $[bx]_{\pm N}$ gerade ist, multiplizieren wir b mit 2^{-1} modulo N und sparen dadurch einen Aufruf von Algorithmus B in Schritt 3.1 bei der nachfolgenden Iteration.

Satz 2.2.3
Sei $(N, e) \in \mathrm{RSA\text{-}PK}_n$ *ein RSA-Public-Key und* B *ein Algorithmus mit*

$$\mathrm{Ws}\left[B\left(N, e, \mathrm{RSA}_{N,e}\left(U_{\mathbb{Z}_N^*}\right)\right) = \mathrm{abs}_{N,1}\left(U_{\mathbb{Z}_N^*}\right)\right] \geq 1 - \frac{1}{12n-9}.$$

Dann bestimmt Algorithmus 2.2.2 in Zeit $(4n-3)\left(|B| + \mathcal{O}\left(n^3\right)\right)$ *das Urbild zu* $\mathrm{RSA}_{N,e}(x)$ *mit Wahrscheinlichkeit für große* N *bzw.* n *etwa* $\frac{1}{4}$*, wobei die Wahrscheinlichkeit nur über die internen Münzwürfe gebildet wird.*

Beweis. Algorithmus 2.2.2 liefert offenbar das Urbild, wenn

a) die Werte $[ax]_{\pm N}$ und $[bx]_{\pm N}$ teilerfremd sind und $[ax]_{\pm N}$ ungerade ist sowie

b) die Aufrufe von B jeweils das richtige Resultat liefern und $4n-3$ Aufrufe ausreichend sind.

Zu untersuchen ist die Wahrscheinlichkeit, mit der diese beiden Ereignisse mindestens eintreten und daß $4n-3$ Aufrufe von Algorithmus B ausreichend sind.

Für Punkt a) ist die Wahrscheinlichkeit zu analysieren, daß zufällig und unabhängig aus $[+K, -K]$ gezogene Werte u, v teilerfremd sind und u ungerade ist. Wir skizzieren, daß für K gegen unendlich die Wahrscheinlichkeit gegen $\frac{4}{\pi^2}$ konvergiert. Für einen formalen Beweis verweisen wir auf die Übungsaufgaben 10 und 13 zu Theorem D in Kapitel 4.5.2 von [Knuth81]. Wir nehmen an, daß der Grenzwert der Wahrscheinlichkeit, daß $\mathrm{ggT}(u, v) = 1$ ist, existiert und bezeichnen ihn mit p. Für festes $d \in \mathbb{N}$ gilt $\mathrm{ggT}(u, v) = d$ mit Wahrscheinlichkeit $\frac{p}{d^2}$, denn:

$$\mathrm{ggT}(u, v) = d \quad \Longleftrightarrow \quad d \mid u, \quad d \mid v \quad \text{und} \quad \mathrm{ggT}\left(\frac{u}{d}, \frac{v}{d}\right) = 1$$

Durch Summation über alle $d \in \mathbb{N}$ erhalten wir

$$1 = \sum_{d \geq 1} \frac{p}{d^2} = p \underbrace{\left(1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \frac{1}{25} + \cdots\right)}_{=H_\infty^2 = \frac{\pi^2}{6}}$$

und das bekannte Resultat $p = \frac{6}{\pi^2}$ von G.L. Dirichlet. Wenn man u, v zufällig wählt, ist mit Wahrscheinlichkeit $\frac{1}{2}$ die Zahl u ungerade. Wir nehmen weiter an, daß der Grenzwert der Wahrscheinlichkeit, daß $\mathrm{ggT}(u, v) = 1$ unter der Bedingung u oder v ungerade, existiert und wollen ihn mit $\bar{p}$ bezeichnen. Da genau dann der größte gemeinsame Teiler ungerade ist, wenn eines der beiden Argumente ungerade ist, erhalten wir:

$$1 = \sum_{\substack{d \geq 1 \\ d \text{ ungerade}}} \frac{\bar{p}}{d^2} = \bar{p} \left(1 + \frac{1}{9} + \frac{1}{25} + \frac{1}{36} + \cdots\right)$$

Es folgt $\bar{p} = \frac{8}{\pi^2}$ wegen:

$$1 + \frac{1}{9} + \frac{1}{25} + \frac{1}{36} + \cdots = \underbrace{1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \frac{1}{25} + \cdots}_{=\frac{\pi^2}{6} = \frac{4\pi^2}{24}} - \underbrace{\frac{1}{4}\left(1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \cdots\right)}_{=\frac{\pi^2}{24}} = \frac{\pi^2}{8}$$

Der Grenzwert der Wahrscheinlichkeit, daß u und v teilerfremd sind und u ungerade ist, beträgt $\frac{1}{2}\bar{p} = \frac{4}{\pi^2}$.

Für Punkt b) schätzen wir die Anzahl der Aufrufe von Algorithmus B bei richtigen Resultaten nach oben ab. Aus Lemma 2.2.2 folgt, da die Absolutbeträge von $[ax]_{\pm N}$ und $[ax]_{\pm N}$ maximal 2^{n-1} sind, daß die Anzahl der Iterationen höchstens $1 + 2(n-1) = 2n - 1$ ist. Die Anzahl der Aufrufe von Algorithmus B in Schritt 3.4 ist daher durch $2n - 1$ beschränkt. Für die Anzahl der Aufrufe von B in Schritt 3.1 müssen wir unsere Modifikation in Vergleich zum binären ggT-Algorithmus 2.2.1 auf Seite 62 beachten: Mit Ausnahme der ersten Iteration ist die Anzahl der Tests, ob das Argument gerade ist, um eins geringer, da wir in Schritt 3.5 eine Halbierung vorwegnehmen. Im binären ggT-Algorithmus hätten wir nach Lemma 2.2.2a) den Zähler c maximal $(2n-2)$-mal erhöht und den Test, ob das Argument noch gerade ist, in jeder Iteration der äußeren Schleife zusätzlich einmal ausgeführt. Wir rufen zum Invertieren der $\mathrm{RSA}_{N,e}$-Funktion Algorithmus B in Schritt 3.1 maximal $(2n-1)$-mal auf. Insgesamt wird Algorithmus B $(4n-3)$-mal aufgerufen. Da wir a und b zufällig wählen, sind alle Argumente beim Aufruf von Algorithmus B modulo N gleichverteilt, so daß mit Wahrscheinlichkeit höchstens

$$(4n-3) \cdot \frac{1}{12n-9} = \frac{1}{3}$$

ein Resultat von Algorithmus B während der ggT-Bestimmung falsch ist.

Wir nehmen an, die Werte $[ax]_{\pm N}$ und $[bx]_{\pm N}$ sind teilerfremd und ist $[ax]_{\pm N}$ ungerade (Punkt a)). Dies tritt mit Wahrscheinlichkeit für N bzw. n gegen unendlich mit Wahrscheinlichkeit $\frac{4}{\pi^2}$. Mit Wahrscheinlichkeit $1 - \frac{1}{3} = \frac{1}{3}$ sind alle Resultate von Algorithmus B richtig. Algorithmus 2.2.2 liefert daher für N bzw. n gegen unendlich mit Wahrscheinlichkeit

$$\frac{2}{3} \cdot \frac{4}{\pi^2} = \frac{8}{3\pi^2} \approx 0,27$$

das Urbild zu $\mathrm{RSA}_{N,e}(x)$. Für die Laufzeit des Algorithmus' beachte, daß wir Algorithmus B maximal $(4n-3)$-mal aufrufen und die Berechnung der $\mathrm{RSA}_{N,e}$-Funktion mit einem Divide-&-Conquer-Verfahren in $\mathcal{O}\left(n^3\right)$ Schritten gelingt. ■

Wir können den allgemeinen Fall, daß Algorithmus B das j-te Bit des Absolutbetrags modulo N berechnet mit $j = \mathcal{O}(\log_2 n)$, auf unser Vorgehen im Fall des untersten Bits zurückführen. Falls der größte gemeinsame Teiler von $[ax]_{\pm N}$ und $[ax]_{\pm N}$ gleich 2^{j-1} ist, treten in der ggT-Berechnung nur Werte auf, deren Absolutbetrag Vielfache von 2^{j-1} sind, also die untersten $j-1$ Bits jeweils Null sind. Wir können in diesem Fall mit dem Algorithmus für die Berechnung des Bits j des Absolutbetrags modulo N ebenfalls den größte gemeinsame Teiler berechnen. Falls dieser 2^{j-1} ist, erhalten wir das Urbild x aus der Darstellung $[ax]_{\pm N} = \pm 2^{j-1}$:

Korollar 2.2.4
Sei $(N,e) \in \text{RSA-PK}_n$ ein RSA-Public-Key und $j := j(n) \geq 1$ mit $j(n) = \mathcal{O}(\log_2 n)$. Sei B ein Algorithmus mit

$$\mathrm{Ws}[\,B\,(N,e,\mathrm{RSA}_{N,e}(z)) = \mathrm{abs}_{N,j}(z)] \geq 1 - \frac{1}{12n-9}$$

für zufälliges $z \in_{\mathrm{R}} \left\{ z \in \mathbb{Z}_N^* \;\middle|\; [z]_{\pm N} \equiv 0 \pmod{2^{j-1}} \right\}$. Dann kann man in Zeit

$$(4n-3)\left(|B| + \mathcal{O}\left(n^3\right)\right)$$

das Urbild zu $\mathrm{RSA}_{N,e}(x)$ mit Wahrscheinlichkeit für große N bzw. n etwa 2^{-2j} bestimmen, wobei die Wahrscheinlichkeit nur über die internen Münzwürfe gebildet wird.

Beweis. Wir verwenden einen modifizierten Algorithmus 2.2.2 von Seite 63. In Schritt 5 bzw. 6 ersetzen wir $\pm a^{-1}$ durch $\pm 2^{-(j-1)}a^{-1}$. Mit Algorithmus B bestimmt man, ob der Wert durch 2^{j-1} teilbar ist. Wenn die Startwerte $[ax]_{\pm N}$ und $[bx]_{\pm N}$ durch 2^{j-1} teilbar sind, berechnet der modifizierte Algorithmus den größten gemeinsamen Teiler. Falls gilt, daß

a) $[ax]_{\pm N}$ und $[bx]_{\pm N}$ sind durch 2^{j-1} teilbar,

b) der größte gemeinsame Teiler von $[ax]_{\pm N}$ und $[bx]_{\pm N}$ ist 2^{j-1},

c) $\left[2^{-(j-1)}ax\right]_{\pm N}$ ist ungerade und

d) alle Resultate des Algorithmus' B korrekt sind,

liefert der modifizierte Algorithmus 2.2.2 das Urbild x. Mit Wahrscheinlichkeit $2^{-2(j-1)}$ sind $[ax]_{\pm N}$ und $[bx]_{\pm N}$ durch 2^{j-1} teilbar. Für K mit

$$[-K,+K] = \left[-\tfrac{N-1}{2},+\tfrac{N-1}{2}\right] \cap 2^{j-1}\mathbb{Z}$$

gilt nach Voraussetzung, daß für N gegen unendlich auch K gegen unendlich geht. Wegen

$$\mathrm{ggT}(u,v) = 2^{j-1} \quad \Longleftrightarrow \quad 2^{j-1} \mid u, \quad 2^{j-1} \mid v \quad \text{und} \quad \mathrm{ggT}\left(\frac{u}{2^{j-1}},\frac{v}{2^{j-1}}\right) = 1$$

argumentieren wir wie im vorherigen Fall, daß zwei zufällig und unabhängig aus $[-K,+K]$ gewählte Zahlen u,v teilerfremd sind und u ungerade ist (siehe Satz 2.2.3): Für N bzw. K

gegen unendlich konvergiert die Wahrscheinlichkeit, daß zwei zufällig und unabhängig aus $\left[-\frac{N-1}{2}, +\frac{N-1}{2}\right]$ gewählte Werte u, v durch 2^{j-1} teilbar sind, $\mathrm{ggT}(u,v) = 2^{j-1}$ und $2^{-(j-1)}u$ ungerade ist, gegen $2^{-2(j-1)}\frac{4}{\pi^2}$. Analog zu Satz 2.2.3 erhalten wir, daß der Algorithmus für N bzw. n gegen unendlich mit Wahrscheinlichkeit

$$\frac{2}{3} \cdot \frac{4}{\pi^2} \cdot 2^{-2(j-1)} \approx \frac{2^{-2(j-1)}}{4} = \frac{2^{-2j+2}}{2^2} = 2^{-2j}$$

das Urbild bestimmt. ■

Wir erhalten aus Satz 2.2.3 und Korollar 2.2.4 für die simultane Sicherheit der Bits des Absolutbetrags modulo N:

Korollar 2.2.5
Sei $(N,e) \in \text{RSA-PK}_n$ ein RSA-Public-Key und $j := j(n) \geq 1$ mit $j(n) = \mathcal{O}(\log_2 n)$. Sei B ein Algorithmus mit

$$\mathrm{Ws}[\,B\,(N, e, \mathrm{RSA}_{N,e}(z), \mathrm{abs}_{N,1}(z), \ldots, \mathrm{abs}_{N,j-1}(z)) = \mathrm{abs}_{N,j}(z)] \geq 1 - \frac{1}{12n-9}$$

für zufälliges $z \in_{\mathrm{R}} \left\{ z \in \mathbb{Z}_N^* \;\middle|\; [z]_{\pm N} \equiv 0 \pmod{2^{j-1}} \right\}$. Dann kann man in Zeit

$$(4n-3)\left(|B| + \mathcal{O}\left(n^3\right)\right)$$

das Urbild zu $\mathrm{RSA}_{N,e}(x)$ mit Wahrscheinlichkeit für große N bzw. n etwa 2^{-2j} bestimmen, wobei die Wahrscheinlichkeit nur über die internen Münzwürfe gebildet wird.

Beweis. Wir übernehmen die Modifikation aus Korollar 2.2.4. Wenn die Startwerte $[ax]_{\pm N}$ und $[bx]_{\pm N}$ durch 2^{j-1} teilbar sind, gilt dies für alle Werte während der Berechnung des größten gemeinsamen Teilers. Wir können die $j-1$ untersten Bits des Absolutbetrags modulo N beim Aufruf von Algorithmus B gleich 0 setzen. ■

2.2.2 Invertieren mit Algorithmus für unterstes Bit

Wir haben in Satz 2.2.3 und Korollar 2.2.5 angenommen, daß uns ein Algorithmus B für das j-te Bit des Absolutbetrags modulo N zur Verfügung steht. Wir konstruieren B aus Algorithmus A mit

$$\mathrm{Ws}\left[\, A\left(N, e, \mathrm{RSA}_{N,e}\left(U_{\mathbb{Z}_N^*}\right)\right) = \mathrm{bit}_{N,j}\left(U_{\mathbb{Z}_N^*}\right)\right] = \tfrac{1}{2} + \epsilon(n),$$

wobei $U_{\mathbb{Z}_N^*}$ eine auf $\mathbb{Z}_N^*$ gleichverteilte Zufallsvariable sei.

Neben dem RSA-Modul $N \in \mathcal{R}_n$ und dem Kodierexponenten $e \in [3, 2^n)$ erhalten wir $\mathrm{RSA}_{N,e}(dx)$ und $d \in \mathbb{Z}_N$ als Eingabe. Das Ziel ist, bezüglich n und $\epsilon(n)^{-1}$ in Polynomialzeit das j-te Bit des Absolutbetrags modulo N von dx mit Wahrscheinlichkeit $1 - \frac{1}{12n-9}$ zu bestimmen. Wir wollen eine Einschränkung machen: Der Absolutbetrag modulo N von dx soll nicht größer als $\frac{1}{2}\epsilon(n)N$ sein. Allgemein nennen wir eine Zahl $z \in \mathbb{Z}$ κ-klein modulo N, wenn $\mathrm{abs}_N(z) \leq \kappa N$ ist. Ein zufälliger Wert aus $\mathbb{Z}_N$ ist κ-klein modulo N mit Wahrscheinlichkeit 2κ.

Betrachten wir wieder zunächst den Fall $j = 1$, d.h. wir sollen zu gegebenem $\mathrm{RSA}_{N,e}(dx)$ und $d \in \mathbb{Z}_N$ entscheiden, ob der Absolutbetrag modulo N von dx gerade ist. Vergleichen wir die Situation in $\mathbb{Z}$. Eine Zahl $d \in \mathbb{Z}$ ist genau dann gerade (ihr Absolutbetrag durch 2 dividierbar), wenn für alle $r \in \mathbb{Z}$ folgende Äquivalenz gilt:

$$2 \mid r \quad \Longleftrightarrow \quad 2 \mid (d + r)$$

Wir wollen diese Beobachtung auf $\mathbb{Z}_N$ übertragen. Allerdings muß die Reduktion modulo N beachtet werden. Das folgende Lemma bildet die Grundlage unserer Mehrheitsentscheidung.

Lemma 2.2.6
Sei N ein RSA-Modul und $d \in \mathbb{Z}_N$ κ-klein modulo N. Für zufälliges $r \in_{\mathrm{R}} \mathbb{Z}_N$ gilt dann mit Wahrscheinlichkeit mindestens $1 - \kappa$:

$$\mathrm{abs}_N(d) \equiv [d + r]_N + [r]_N \pmod 2 \tag{2.5}$$

Beweis. Der Beweis basiert auf zwei Beobachtungen. Nach Definition des Absolutbetrags modulo N gilt wegen $-1 \equiv +1 \pmod 2$:

$$\mathrm{abs}_N(d) \equiv [d]_N + N \cdot \underbrace{\Big[\, [d]_{\pm N} < 0 \,\Big]}_{\text{Wert des logischen Ausdrucks}} \pmod 2$$

Für das Produkt N zweier ungerader Primzahlen gilt $N \equiv 1 \pmod 2$ und da ferner

$$[d + r]_N \equiv [d]_N + [r]_N + N \cdot \Big[\, [d]_N + [r]_N \geq N \Big] \pmod 2,$$

ist, können wir (2.5) schreiben als:

$$\Big[\, [r]_{\pm N} < 0 \Big] = \Big[\, [d]_N + [r]_N \geq N \Big] \tag{2.6}$$

Um zu zeigen, daß diese Kongruenz von allen $r \in \mathbb{Z}_N$ mit Ausnahme von $\mathrm{abs}_N(d)$-vielen erfüllt wird, unterscheiden wir zwei Fälle:

a) Es gilt $[d]_{\pm N} \geq 0$. Wegen $[d]_{\pm N} = \mathrm{abs}_N(d)$ erfüllen in diesem Fall die $r \in \mathbb{Z}_N$ mit

$$[r]_N \in [0, N - \mathrm{abs}_N(d))$$

die Gleichung (2.6).

b) Es gilt $[d]_{\pm N} < 0$. Wegen $[d]_{\pm N} = -\mathrm{abs}_N(d)$ erfüllen in diesem Fall die $r \in \mathbb{Z}_N$ mit

$$[r]_N \in [\mathrm{abs}_N(e), N)$$

die Gleichung (2.6).

Die Behauptung folgt aus der Voraussetzung $\mathrm{abs}_N(d) \leq \kappa N$. ■

Wir erhalten folgenden Ansatz für eine Mehrheitsentscheidung: Wähle am Anfang zufällig und unabhängig $r_1, r_2, \ldots, r_m$ aus $\mathbb{Z}_N$ und nimm an, daß $\mathrm{abs}_N(dx)$ gerade ist, wenn für die Mehrheit der m Werte

$$A\left(N, e, \mathrm{RSA}_{N,e}\left((d+r_i)x\right)\right) \equiv A\left(N, e, \mathrm{RSA}_{N,e}\left(r_i x\right)\right) \pmod 2 \tag{2.7}$$

gilt. Angenommen, Algorithmus A habe wie in [BCS83] den Vorteil $\frac{1}{4} + \epsilon(n)$, d.h. liefere mit Wahrscheinlichkeit maximal $\frac{1}{4} - \epsilon(n)$ ein falsches Resultat. Da r_i ein zufälliger Wert ist, sind insgesamt beide Resultate von A mit Wahrscheinlichkeit maximal $\frac{1}{2} - 2\epsilon(n)$ falsch. Ein weitere Fehlerquelle ist, daß die Kongruenz

$$\mathrm{abs}_N(dx) \equiv \left[(d+r_i)x\right]_N + \left[r_i x\right]_N \pmod 2$$

nicht gilt. Wenn wir fordern, dx sei $\epsilon(n)$-klein, tritt dieser Fehler nach Lemma 2.2.6 mit Wahrscheinlichkeit maximal $\epsilon(n)$ ein. Insgesamt ist die i-te Entscheidung mit Wahrscheinlichkeit mindestens $\frac{1}{2} + \epsilon(n)$ richtig. Wir können aber nur voraussetzen, daß Algorithmus A mit Wahrscheinlichkeit maximal $\frac{1}{2} - \epsilon(n)$ ein falsches Resultat liefert. Daher sind beide Resultate von A mit Wahrscheinlichkeit maximal $1 - 2\epsilon(n)$ falsch und insgesamt ist die i-te Entscheidung nur mit Wahrscheinlichkeit mindestens $\epsilon(n)$ richtig. Zum Vergleich: Bei einem zufälligen Münzwurf ist die i-te Entscheidung mit Wahrscheinlichkeit $\frac{1}{2}$ richtig. Um das Problem, daß sich in Bedingung (2.7) die Fehlerwahrscheinlichkeiten addieren (*Error-Doubling*-Phänomen), zu vermeiden, modifizieren wir unseren Ansatz.

Wir erzeugen zu gegebenem $\mathrm{RSA}_{N,e}(dx)$ und $d \in \mathbb{Z}_N$ modulo N gleichverteilte Punkte $r_1, r_2, \ldots, r_m$, wobei wir jeweils mit Wahrscheinlichkeit mindestens $1 - \frac{1}{4}\epsilon(n)$ das unterste Bit modulo N von $(d+r_i)x$ kennen, so daß man dieses nicht mehr mit Algorithmus A bestimmen muß. Wir nehmen an, daß $\mathrm{abs}_N(dx)$ gerade sei, wenn für die Mehrheit der m Werte gilt:

$$\underbrace{\mathrm{bit}_{N,1}\left(r_i x\right)}_{\text{bekannt}} \equiv A\left(N, e, \mathrm{RSA}_{N,e}\left((d+r_i)x\right)\right) \pmod 2 \tag{2.8}$$

Um die Punkte $r_1, r_2, \ldots, r_m$ samt zugehörige, unterste Bits $\mathrm{bit}_{N,1}\left((d+r_i)x\right)$ zu bestimmen, wählen wir u, v zufällig sowie unabhängig aus $\mathbb{Z}_N$ und setzen:

$$r_i := \left[iu + v\right]_N \quad \text{für } i = 1, 2, \ldots, m$$

Diese Technik heißt *Two-Point-Based-Sampling*, da man basierend auf den zwei Punkten u und v Testpunkte $r_1, r_2, \ldots, r_m$ erzeugt (siehe auch [CG89]). Die Werte $r_1, r_2, \ldots, r_m$ sind gleichverteilt in $\mathbb{Z}_N$ und wir werden sehen, daß sie modulo N paarweise unabhängig ist. Angenommen, wir kennen $\mathrm{bit}_{N,1}(ux)$, $\mathrm{bit}_{N,1}(vx)$ sowie ein $z_i \in \mathbb{N}_0$ mit:

$$z_i N \leq i \cdot \left[ux\right]_N + \left[vx\right]_N < (z_i + 1)N$$

Dann gilt für r_i:

$$\begin{aligned} \mathrm{bit}_{N,1}(r_i x) &\equiv \left[(iu+v)x\right]_N \\ &\equiv i \cdot \left[ux\right]_N + \left[vx\right]_N - z_i N \\ &\equiv i \cdot \mathrm{bit}_{N,1}(ux) + \mathrm{bit}_{N,1}(vx) - z_i N \pmod 2 \end{aligned}$$

Der Wert z_i soll mit Wahrscheinlichkeit mindestens $1 - \frac{1}{4}\epsilon(n)$ aus Approximationen von $\left[ux\right]_N$ und $\left[vx\right]_N$ berechnet werden. Wir kennen aber weder $\mathrm{bit}_{N,1}(ux)$ und $\mathrm{bit}_{N,1}(vx)$, noch

Approximationen für $[ux]_N$ und $[vx]_N$. Wenn es bezüglich n und $\epsilon(n)^{-1}$ nur polynomiell viele Möglichkeiten gibt, probieren wir alle und versuchen jeweils mit dieser Wahl das Urbild x zu bestimmen. Für die richtige Wahl erhalten man mit Wahrscheinlichkeit mindestens $1 - \frac{1}{12n-9}$ durch eine Mehrheitsentscheidung jeweils das unterste Bit des Absolutbetrags modulo N und mit Algorithmus 2.2.2 auf Seite 63 das Urbild.

Formalisieren wir unser Vorgehen und analysieren die Erfolgswahrscheinlichkeit. Wir beweisen zunächst Eigenschaften der Testpunkte $r_1, r_2, \ldots, r_m$:

Lemma 2.2.7
Sei $N \in \mathcal{R}_n$ ein RSA-Modul mit Primfaktoren p, q und $m < \frac{1}{2}\min\{p,q\}$. Seien d, u, v zufällig und unabhängig modulo N verteilt. Dann gilt für die m Punkte

$$r_i := [iu + v]_N \quad \textit{für } i = 1, 2, \ldots, m,$$

daß die m Werte $d + r_i$ modulo N gleichverteilt und paarweise unabhängig sind.

Beweis. Da d ein von r_i unabhängiger Wert ist, genügt es zu zeigen, daß die Testpunkte $r_1, r_2, \ldots, r_m$ modulo N gleichverteilt und paarweise unabhängig sind. Mit u und v ist auch r_i gleichverteilt in $\mathbb{Z}_N$. Für i, j mit $i \neq j$ wollen wir zeigen, daß r_i und r_j modulo N unabhängig verteilt sind. Die Transformationsmatrix aus

$$\begin{bmatrix} r_i \\ r_j \end{bmatrix} \equiv \begin{bmatrix} i & 1 \\ j & 1 \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix} \pmod{N}$$

hat die Determinante $i - j \neq 0$. Wegen $i, j < \frac{1}{2}\min\{p,q\}$ gilt $0 < |i-j| < \min\{p,q\}$ und die Differenz hat modulo N ein Inverses, so daß die Transformation regulär ist. Für alle $c_1, c_2 \in \mathbb{Z}_N$ gilt somit:

$$\mathrm{Ws}\big[\,[r_i]_N = c_1 \text{ und } [r_j]_N = c_2\big] = \frac{1}{N^2} = \mathrm{Ws}[\,[r_i]_N = c_1] \cdot \mathrm{Ws}\big[\,[r_j]_N = c_2\big]$$

Für $i \neq j$ sind r_i und r_j modulo N unabhängig. ■

Für die Berechnung von $\mathrm{bit}_{N,1}(r_i x)$ der m Testpunkte verwenden wir Approximationen von $[ux]_N$ und $[vx]_N$. Dazu unterteilen man $[0, N-1]$ in $4m\epsilon(n)^{-1}$ bzw. $4\epsilon(n)^{-1}$ gleich große Intervalle:

Lemma 2.2.8
Seien die Werte wie in Lemma 2.2.7. Gegeben seien fernen $b^{(u)} := \mathrm{bit}_{N,1}(ux)$ und $b^{(v)} := \mathrm{bit}_{N,1}(vx)$ sowie $k^{(u)}, k^{(v)}$ mit $0 \le k^{(u)} < 4m\epsilon(n)^{-1}$, $0 \le k^{(v)} < 4\epsilon(n)^{-1}$ und:

$$[ux]_N \in \left[k^{(u)} \cdot \frac{\epsilon(n)N}{4m}, \left(k^{(u)} + 1\right) \cdot \frac{\epsilon(n)N}{4m}\right)$$
$$[vx]_N \in \left[k^{(v)} \cdot \frac{\epsilon(n)N}{4}, \left(k^{(v)} + 1\right) \cdot \frac{\epsilon(n)N}{4}\right)$$

Dann können wir für festes i mit $1 \le i \le m$ das unterste Bit $s_i := \mathrm{bit}_{N,1}(r_i x)$ von $[r_i x]_N$ mit Wahrscheinlichkeit mindestens $1 - \frac{1}{4}\epsilon(n)$ bestimmen.

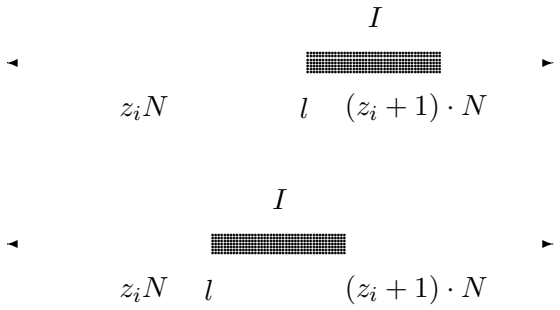

Abbildung 2.2.2: Skizze zu Beweis von Lemma 2.2.8

Beweis. Wir wollen das $z_i \in \mathbb{N}_0$ ermitteln mit

$$[r_i x]_N = i \cdot [ux]_N + [vx]_N - z_i N,$$

da wir wegen $N \equiv 1 \pmod 2$ dann $\mathrm{bit}_{N,1}(r_i x)$ erhalten aus:

$$\begin{aligned} \mathrm{bit}_{N,1}(r_i x) &\equiv i \cdot \mathrm{bit}_{N,1}(ux) + \mathrm{bit}_{N,1}(vx) - w\,\mathrm{bit}_0(N) \\ &\equiv i \cdot b^{(u)} + b^{(v)} - z_i \pmod 2 \end{aligned}$$

Für die Bestimmung von z_i beachte:

$$i \cdot [ux]_N + [vx]_N \in I := \left[\frac{(ik^{(u)} + k^{(v)}m)\epsilon(n)N}{4m}, \frac{(ik^{(u)} + 1 + k^{(v)}m + m)\epsilon(n)N}{4m} \right)$$

Wir kennen nach Voraussetzung die Grenzen dieses Intervalls der Länge:

$$|I| \leq \frac{(1+m)\epsilon(n)N}{4m} \leq \frac{\epsilon(n)N}{2} < \frac{N}{2}$$

Insbesondere liegt in I höchstens ein ganzzahliges Vielfaches von N. Sei l die unter Grenze des Intervalls I. Wir setzen $\overline{z}_i := \left\lfloor \frac{l}{N} \right\rfloor$ und falls in I ein ganzzahliges Vielfaches von N liegt, addieren wir zufällig 0 der 1 zu $\overline{z}_i$ (vergleiche Abbildung 2.2.2). Nach Lemma 2.2.7 ist $[r_i x]_N$ gleichverteilt in $\mathbb{Z}_N$, so daß mit Wahrscheinlichkeit $\frac{1}{2}\epsilon(n)N$ im Intervall I ein ganzzahliges Vielfaches von N liegt. In dieser Situation ist mit Wahrscheinlichkeit $\frac{1}{2}$ unsere Wert $\overline{z}_i$ für z_i richtig. Falls kein ganzzahliges Vielfaches von N in I liegt, gilt stets $\overline{z}_i = z_i$. ■

Wir erhalten aus einem Algorithmus zur Berechnung des untersten Bit ein Verfahren zur Berechnung des untersten Bits des Absolutbetrags modulo N, dessen Fehlerwahrscheinlichkeit man nach oben abschätzen kann durch:

Lemma 2.2.9
Seien p, q die beiden Primfaktoren des RSA-Moduls N, u, v zufällig und unabhängig aus $\mathbb{Z}_N$ sowie

$$r_i := [iu + v]_N \quad \textit{für } i = 1, 2, \ldots, m$$

mit $m < \frac{1}{2}\min\{p,q\}$. *Seien* $b^{(u)} := \mathrm{bit}_{N,1}(ux)$, $b^{(v)} := \mathrm{bit}_{N,1}(vx)$ *und* $k^{(u)}, k^{(v)}$ *wie in Lemma 2.2.8 und* $s_1, s_2, \ldots, s_i$ *die berechneten Werte für* $\mathrm{bit}_{N,1}(r_i)$. *Sei* A *ein Algorithmus zur Bestimmung des untersten Bit des Urbildes mit*

$$\mathrm{Ws}\left[A\left(N, e, \mathrm{RSA}_{N,e}\left(U_{\mathbb{Z}_N^*}\right)\right) = \mathrm{bit}_{N,1}\left(U_{\mathbb{Z}_N^*}\right)\right] \geq \tfrac{1}{2} + \epsilon(n),$$

wobei die Wahrscheinlichkeit über die zufällige Nachricht $U_{\mathbb{Z}_N^*}$ *und die internen Münzwürfe gebildet wird. Sei* d *unabhängig von* u *und* v. *Dann gilt für alle* dx, *die* $\frac{1}{2}\epsilon(n)$*-klein modulo* N *sind,*

$$(2.9)\quad \left[\mathrm{abs}_{N,1}(dx) = 0\right] \neq \left[\left|\{i \in \{1,2,\ldots,m\} \,:\, s_i = A\left(N, e, \mathrm{RSA}_{N,e}\left((d+r_i)x\right)\right)\}\right| \geq \tfrac{1}{2}m\right]$$

mit Wahrscheinlichkeit maximal $\frac{4}{m\epsilon(n)^2}$.

Beweis. Wir schätzen zunächst für festes i die Wahrscheinlichkeit nach oben ab, daß

$$[\mathrm{abs}_{N,1}(dx) = 0] \neq [s_i = A\left(N, e, \mathrm{RSA}_{N,e}\left((d+r_i)x\right)\right)]$$

gilt, also die i-te Entscheidung falsch ist. Es gibt drei Fehlerquellen:

a) Das Resultat von Algorithmus A ist falsch.

b) Es gilt $\mathrm{abs}_N(dx) \not\equiv [(d+r_i)x]_N + [r_i x]_N \pmod 2$.

c) Der berechnete Wert s_i für $\mathrm{bit}_{N,1}(r_i)$ ist falsch.

Da $(d+r_i)x$ gleichverteilt in $\mathbb{Z}_N$ ist, hat Ereignis a) nach Voraussetzung die Wahrscheinlichkeit maximal $\frac{1}{2} - \epsilon(n)$. Weil dx $\frac{1}{2}\epsilon(n)$-klein modulo N und $r_i x$ gleichverteilt in $\mathbb{Z}_N$ ist, gilt nach Lemma 2.2.6 auf Seite 68, daß Ereignis b) die Wahrscheinlichkeit maximal $\frac{1}{2}\epsilon(n)$ hat. Aus Lemma 2.2.8 folgt, daß Ereignis c) mit Wahrscheinlichkeit maximal $\frac{1}{4}\epsilon(n)$ eintritt. Die i-te Entscheidung ist mit Wahrscheinlichkeit maximal

$$\tfrac{1}{2} - \epsilon(n) + \tfrac{1}{2}\epsilon(n) + \tfrac{1}{4}\epsilon(n) = \tfrac{1}{2} - \tfrac{1}{4}\epsilon(n)$$

falsch.

Wir definieren Indikatorvariablen $X_1, X_2, \ldots, X_m$ für das Ereignis, daß die i-te Entscheidung falsch ist. Die m Zufallsvariablen $X_1, X_2, \ldots, X_m$ hängen von den internen Münzwürfen des Algorithmus' A, von der Wahl der s_i und den Werten $d + r_i$ ab. Da d nach Voraussetzung von u, v unabhängig ist, folgt aus Lemma 2.2.7 auf Seite 70, daß die Werte $d + r_i$ für $i = 1, 2, \ldots, m$ modulo N gleichverteilt und paarweise unabhängig sind. Die Indikatorvariablen sind daher identisch verteilt und paarweise unabhängig. Wegen

$$\mathrm{Ws}[\,X_i = 1] \leq \tfrac{1}{2} - \tfrac{1}{4}\epsilon(n)$$

ist der Erwartungswert $\mu \leq \frac{1}{2} - \frac{1}{4}\epsilon(n)$. Wegen $m\mu \leq \frac{1}{2}m - \frac{1}{4}m\epsilon(n)$ gilt für das Ereignis (2.9) aus der Behauptung:

$$\mathrm{Ws}[\,(2.9)] \leq \mathrm{Ws}\left[\sum_{i=1}^{m} X_i \geq \frac{m}{2}\right] \leq \mathrm{Ws}\left[\sum_{i=1}^{m} X_i - \mu m \geq \frac{m\epsilon(n)}{4}\right]$$

Wir schätzen die Varianz σ^2 nach oben ab durch:

$$\sigma^2 = \mathrm{E}\left[X_i^2\right] - \mathrm{E}[X_i]^2 \leq \max_{\xi\in[0,1]} \left\{\xi - \xi^2\right\} = \tfrac{1}{4} \tag{2.10}$$

Aus Lemma 1.6.1 auf Seite 39 folgt mit $t := \frac{1}{4}\epsilon(n)$:

$$\mathrm{Ws}[(2.9)] \leq \mathrm{Ws}\left[\left|\sum_{i=1}^{m} X_i - \mu m\right| \geq \frac{m\epsilon(n)}{4}\right] \leq \frac{\sigma^2}{t^2 m} \tag{2.11}$$

Mit der Abschätzung aus (2.10) gilt:

$$\mathrm{Ws}[(2.9)] \leq \frac{4}{m\epsilon(n)^2}$$

Dies war zu zeigen. ■

Wir erhalten Algorithmus 2.2.3 zum Invertieren der RSA-Funktion, wenn uns ein Algorithmus A zur Bestimmung des untersten Bits des Urbildes wie in Lemma 2.2.9 zur Verfügung steht.

Satz 2.2.10
Sei $(N,e) \in \text{RSA-PK}_n$ ein RSA-Public-Key. Sei A ein Algorithmus zur Bestimmung des untersten Bit des Urbildes mit

$$\mathrm{Ws}\left[A\left(N, e, \mathrm{RSA}_{N,e}\left(U_{\mathbb{Z}_N^*}\right)\right) = \mathrm{bit}_{N,1}\left(U_{\mathbb{Z}_N^*}\right)\right] \geq \tfrac{1}{2} + \epsilon(n),$$

wobei die Wahrscheinlichkeit über die zufällige Nachricht $U_{\mathbb{Z}_N^*}$ und die internen Münzwürfe gebildet wird. Falls $\epsilon(n)^{-1} = n^{\mathcal{O}(1)}$ ist, bestimmt für hinreichend große N bzw. n Algorithmus 2.2.3 in Zeit

$$2^{19{,}2} n^3 \epsilon(n)^{-6}\left(|A| + \mathcal{O}\left(n^3\right)\right)$$

zu $\mathrm{RSA}_{N,e}(x)$ das Urbild x mit Wahrscheinlichkeit für große N bzw. n etwa $\frac{1}{4}\epsilon(n)^2$, wobei die Wahrscheinlichkeit nur über die internen Münzwürfe gebildet wird.

Beweis. Sei $m := 48n\epsilon(n)^{-2}$. Wegen $\epsilon(n)^{-1} = n^{\mathcal{O}(1)}$ können wir für hinreichend große N bzw. n die Wahrscheinlichkeit, daß ein modulo N gleichverteilter Wert nicht teilerfremd zu N ist, vernachlässigen. Unser Verfahren zu Bestimmung von $\mathrm{abs}_{N,1}(dx)$ in Schritt 4.2 hat nach Lemma 2.2.9 auf Seite 71 maximal die Fehlerwahrscheinlichkeit

$$\frac{4}{m\epsilon(n)^2} = \frac{1}{12n} \leq \frac{1}{12n-9},$$

wenn dx $\frac{1}{2}\epsilon(n)$-klein modulo N ist. Ein zufälliger Wert ax ist $\frac{1}{2}\epsilon(n)$-klein modulo N mit Wahrscheinlichkeit $\epsilon(n)$. Wenn für die zufällige Wahl von a und b in Algorithmus 2.2.2 auf Seite 63 gilt, daß ax und bx $\frac{1}{2}\epsilon(n)$-klein modulo N sind, gilt dies nach Lemma 2.2.2.c) auf Seite 62 auch für alle während der ggT-Berechnung auftretenden Werte. Nach Satz 2.2.3 auf Seite 64 folgt, daß wir

Algorithmus 2.2.3 RSA-Invertierung mit Algorithmus für unterstes Bit

EINGABE: ▷ RSA-Public-Key $(N, e) \in \text{RSA-PK}_n$
▷ $\text{RSA}_{N,e}(x)$ mit $x \in \mathbb{Z}_N^*$

/∗ Sei A Algorithmus mit $\text{Ws}\left[A\left(N, e, \text{RSA}_{N,e}\left(U_{\mathbb{Z}_N^*}\right)\right) = \text{bit}_{N,1}\left(U_{\mathbb{Z}_N^*}\right)\right] \geq \frac{1}{2} + \epsilon(n)$, wobei die Wahrscheinlichkeit über die zufällige Nachricht $U_{\mathbb{Z}_N^*}$ und die internen Münzwürfe gebildet wird. ∗/

/∗ Erzeugt m Testpunkte für Mehrheitsentscheidung: ∗/

1. Wähle $u, v \in \mathbb{Z}_N$ zufällig und unabhängig.

2. $m := 48n\epsilon(n)^{-2}$

3. FOR $i := 1$ TO m DO $r_i := [iu + v]_N$.

/∗ Probiere alle möglichen Approximationen und unterste Bits: ∗/

4. FOR all $(k^{(u)}, k^{(v)}, b^{(u)}, b^{(v)}) \in \left[0, 4m\epsilon(n)^{-1}\right) \times \left[0, 4\epsilon(n)^{-1}\right) \times \{0,1\}^2$ DO

4.1. FOR $i := 1$ TO m DO

Berechne mit $(k^{(u)}, k^{(v)}, b^{(u)}, b^{(v)})$ wie in Lemma 2.2.8 auf Seite 70 das unterste Bit s_i von $[r_i x]_N$.

4.2. Rufe Algorithmus 2.2.2 auf Seite 63 auf. Falls der Algorithmus das Urbild x liefert, gib x aus und stoppe. Verwende als Algorithmus B zur Bestimmung von $\text{abs}_{N,1}(dx)$ die folgende Mehrheitsentscheidung: Gib genau dann 0 aus, wenn

$$|\{i \in \{1, 2, \ldots, m\} \,:\, s_i = A\left(N, e, \text{RSA}_{N,e}\left((d + r_i)x\right)\right)\}| \geq \tfrac{1}{2}m$$

ist (vergleiche (2.8) auf Seite 69). Dieser Algorithmus arbeitet nur korrekt, wenn für die zufällig aus $\mathbb{Z}_N$ in Algorithmus 2.2.2 gewählten a, b die Zahlen ax und bx $\frac{1}{2}\epsilon(n)$-klein Modulo N sind.

END for

a) für die richtige Wahl in Schritt 4 sowie

b) wenn ax und bx $\frac{1}{2}\epsilon(n)$-klein modulo N sind,

für große N bzw. n das Urbild x mit Wahrscheinlichkeit $\frac{1}{4}\epsilon(n)^2$ erhalten.

Analysieren wir die Laufzeit. Schritt 3 gelingt in Zeit $m \cdot \mathcal{O}(n)$. Betrachten wir die Laufzeit von Schritt 4.2. Unsere Implementierung des Algorithmus' B hat die Laufzeit:

$$|B| = m \cdot \left(|A| + \mathcal{O}\left(n^3\right)\right)$$

Schritt 4.2 gelingt nach Satz 2.2.3 auf Seite 64 in Zeit $4nm\left(|A| + \mathcal{O}\left(n^3\right)\right)$. Da es in Schritt 4 insgesamt $2^6 m\epsilon(n)^{-2}$ Möglichkeiten gibt, erhalten wir als Laufzeit von Algorithmus 2.2.3:

$$m \cdot \mathcal{O}(n) + 2^6 m\epsilon(n)^{-2} \cdot 4nm\left(|A| + \mathcal{O}\left(n^3\right)\right) = 2^{19,2} n^3 \epsilon(n)^{-6}\left(|A| + \mathcal{O}\left(n^3\right)\right)$$

■

Fassen wir unsere Resultate über das unterste Bit zu einem Sicherheitsbeweis des RSA-Generators 2.1.2 auf Seite 58 zusammen:

Korollar 2.2.11
Sei $\ell(n) \geq n+1$ durch ein Polynom beschränkt. Falls das vom RSA-Generator 2.1.2 erzeugt Ensemble des Typs $\ell(n)$ einen statistischen Test T mit nicht-vernachlässigbarer Toleranz $\delta(n)$ nicht besteht, kann man die $\mathrm{RSA}_{N,e}$-Funktion für unendlich viele n und zufälligen RSA-Public-Key $(N,e) \in_{\mathrm{R}} \text{RSA-PK}_n$ in Zeit

$$2^{25,2} n^3 \delta(n)^{-6} \ell(n)^6 \left(|T| + \ell(n) \cdot \mathcal{O}\left(n^3\right) \right)$$

mit Wahrscheinlichkeit für große N bzw. n etwa $\frac{\delta(n)^3}{16 \cdot \ell(n)^3}$ invertieren.

Beweis. Nach Lemma 1.5.4 auf Seite 25 erhalten wir einen Prediktor A zur Vorhersage des Prädikats $\mathrm{bit}_{N,1}(x)$ zu gegebenem $\mathrm{RSA}_{N,e}(x)$ mit Vorteil $\epsilon(n) := \frac{\delta(n)}{\ell(n)}$ und Laufzeit $|T| + \ell(n) \cdot \mathcal{O}\left(n^3\right)$. Wir betrachten nur die unendlich vielen n, für die A das unterste Bit mit Wahrscheinlichkeit mindestens $\epsilon(n)$ bestimmt. Aus Lemma 1.6.3 auf Seite 44 folgt mit $X = \text{RSA-PK}_n$ und $Y = \mathbb{Z}_N^*$, daß eine Teilmenge $G_n \subseteq \text{RSA-PK}_n$ existiert mit:

a) $\mathrm{Ws}[U_{\text{RSA-PK}_n} \in G_n] \geq \epsilon(n)$.

b) Für alle $(N,e) \in G_n$ gilt: $\mathrm{Ws}\left[A\left(N, e, \mathrm{RSA}_{N,e}\left(U_{\mathbb{Z}_N^*}\right)\right) = \mathrm{bit}_{N,1}\left(U_{\mathbb{Z}_N^*}\right)\right] \geq \frac{1}{2} + \frac{1}{2}\epsilon(n)$.

Wir wählen n hinreichend groß und betrachten nur die öffentlichen Schlüssel $(N,e) \in G_n$, für die der Prediktor A mit Wahrscheinlichkeit $\frac{1}{2}\epsilon(n)$ (unabhängig von N und e) das unterste Bit bestimmt. Aus Satz 2.2.10 folgt die Behauptung des Korollars, denn wir erhalten für $(N,e) \in G_n$ in Zeit

$$2^{19,2} n^3 2^6 \epsilon(n)^{-6} \left(|A| + \mathcal{O}\left(n^3\right) \right)$$

zu $\mathrm{RSA}_{N,e}(x)$ das Urbild x mit Wahrscheinlichkeit für große N bzw. n etwa $\frac{1}{4} \cdot \frac{1}{4}\epsilon(n)^2 = \frac{1}{16}\epsilon(n)^2$. ∎

2.2.3 Invertieren mit Algorithmus für j-tes Bit

Wir können den allgemeinen Fall, daß Algorithmus A das j-te Bit des Urbildes berechnet, auf unser Vorgehen im Fall des untersten Bits wie im Kapitel 2.2.2 zurückführen. Wir formulieren zunächst Lemma 2.2.6 von Seite 68 für den Fall des j-ten Bits des Absolutbetrags modulo N:

Lemma 2.2.12
Sei j mit $1 \leq j < \lfloor \log_2 N \rfloor$ und $d \in \mathbb{Z}_N$ κ-klein modulo N, so daß die $j-1$ untersten Bits von $[d]_N$ Null sind. Für zufälliges $r \in_{\mathrm{R}} \mathbb{Z}_N$ gilt dann mit Wahrscheinlichkeit mindestens $1-\kappa$:

$$\mathrm{abs}_{N,j}(d) \equiv \mathrm{bit}_{N,j}(d+r) + \mathrm{bit}_{N,j}(r) \pmod 2 \tag{2.12}$$

Beweis. Wir zeigen, daß alle $r \in \mathbb{Z}_N$ mit Ausnahme von $\mathrm{abs}_N(d)$-vielen die Kongruenz (2.12) erfüllen. Wie in Lemma 2.2.6 unterscheidet man zwei Fälle:

a) Es gilt $[d]_{\pm N} \geq 0$. Nach Definition des Absolutbetrags modulo N folgt $\mathrm{abs}_N(d) = [d]_N$ und somit $\mathrm{abs}_{N,j}(d) = \mathrm{bit}_{N,j}(d)$. Für alle $[r]_N \in [0, N - \mathrm{abs}_N(d))$ ist:

$$[d + r]_N = [d]_N + [r]_N$$

Da die $j-1$ untersten Bits von $[d]_N$ Null sind, tritt bei der Addition von $[d]_N$ und $[r]_N$ kein Überschlag in die j-te Bitposition auf, so daß gilt:

$$\mathrm{bit}_{N,j}(d + r) \equiv \mathrm{bit}_{N,j}(d) + \mathrm{bit}_{N,j}(r) \pmod 2$$

Wir erhalten für alle $[r]_N \in [0, N - \mathrm{abs}_N(d))$:

$$\begin{aligned}\mathrm{abs}_{N,j}(d) &\equiv \mathrm{bit}_{N,j}(d) + \mathrm{bit}_{N,j}(r) + \mathrm{bit}_{N,j}(r) \\ &\equiv \mathrm{bit}_{N,j}(d + r) + \mathrm{bit}_{N,j}(r) \pmod 2\end{aligned}$$

b) Es gilt $[d]_{\pm N} < 0$. Nach Definition des Absolutbetrags modulo N folgt $\mathrm{abs}_N(d) = N - [d]_N$. Sei $\mathrm{bit}_j(N)$ die j-te Bitposition der Dualdarstellung von N. Weil die $j-1$ untersten Bits von $[d]_N$ Null sind, gilt wegen $-1 \equiv +1 \pmod 2$:

$$\mathrm{abs}_{N,j}(d) = \mathrm{bit}_{N,j}(d) + \mathrm{bit}_j(N) \pmod 2$$

Für alle $[r]_N \in [\mathrm{abs}_N(d), N)$ ist:

$$[d + r]_N = [d]_N + [r]_N - N$$

Da die $j-1$ untersten Bits von $[d]_N$ Null sind, tritt bei der Addition von $[d]_N$ und $[r]_N$ kein Überschlag in die j-te Bitposition auf, so daß gilt:

$$\mathrm{bit}_{N,j}(d + r) \equiv \mathrm{bit}_{N,j}(d) + \mathrm{bit}_{N,j}(r) + \mathrm{bit}_j(N) \pmod 2$$

Wir erhalten für alle $[r]_N \in [\mathrm{abs}_N(d), N)$:

$$\begin{aligned}\mathrm{abs}_{N,j}(d) &\equiv \mathrm{bit}_{N,j}(d) + \mathrm{bit}_j(N) \\ &\equiv \mathrm{bit}_{N,j}(d) + \mathrm{bit}_{N,j}(r) + \mathrm{bit}_{N,j}(r) + \mathrm{bit}_j(N) \\ &\equiv \mathrm{bit}_{N,j}(d + r) + \mathrm{bit}_{N,j}(r) \pmod 2\end{aligned}$$

Die Behauptung folgt aus der Voraussetzung $\mathrm{abs}_N(d) \leq \kappa N$. ■

Wir verwenden wie im Spezialfall des untersten Bits als Testpunkte $r_i := [iu + v]_n$, $i = 1, 2, \ldots, m$, für zufällig und unabhängig aus $\mathbb{Z}_N$ gewählte u und v. Für die Berechnung von $\mathrm{bit}_{N,j}(r_i x)$ der m Testpunkte verwenden wir Approximationen von $[ux]_N$ und $[vx]_N$ und nehmen an, daß wir die j untersten Bits von $[ux]_N$ und $[vx]_N$ kennen. Lemma 2.2.8 von Seite 70 lautet in der verallgemeinerten Fassung:

Lemma 2.2.13
Seien die Werte wie in Lemma 2.2.7. Gegeben seien fernen $B^{(u)} := \mathrm{Bits}_{N,j}(ux)$ *und* $B^{(v)} := \mathrm{Bits}_{N,j}(vx)$ *sowie* $k^{(u)}, k^{(v)}$ *mit* $0 \leq k^{(u)} < 4m\epsilon(n)^{-1}$, $0 \leq k^{(v)} < 4\epsilon(n)^{-1}$ *und:*

$$[ux]_N \in \left[k^{(u)} \cdot \frac{\epsilon(n)N}{4m}, \left(k^{(u)} + 1\right) \cdot \frac{\epsilon(n)N}{4m}\right)$$

$$[vx]_N \in \left[k^{(v)} \cdot \frac{\epsilon(n)N}{4}, \left(k^{(v)} + 1\right) \cdot \frac{\epsilon(n)N}{4}\right)$$

Dann können wir für festes i *mit* $1 \leq i \leq m$ *das Bit* $s_i := \mathrm{bit}_{N,j}(r_i x)$ *von* $[r_i x]_N$ *mit Wahrscheinlichkeit mindestens* $1 - \frac{1}{4}\epsilon(n)$ *bestimmen.*

Beweis. Wir ermitteln wie im Beweis zu Lemma 2.2.8 das $z_i \in \mathbb{N}_0$, so daß

$$[r_i x]_N = i \cdot [ux]_N + [vx]_N - z_i N$$

ist, mit Wahrscheinlichkeit mindestens $1 - \frac{1}{4}\epsilon(n)$. Um das j-te Bit von $[r_i x]_N$ zu berechnen, betrachten wir nur die j untersten Bits der Gleichung:

$$\begin{aligned}[r_i x]_N &\equiv i \cdot [ux]_N + [vx]_N - z_i N \\ &\equiv i \cdot \mathrm{Bits}_{N,j}(ux) + \mathrm{Bits}_{N,j}(vx) - z_i N \pmod{2^j - 1}\end{aligned}$$

Wir erhalten mit Wahrscheinlichkeit mindestens $1 - \frac{1}{4}\epsilon(n)$ das j-te Bit von $[r_i x]_N$, indem man die j-te Bitposition von $i \cdot \mathrm{Bits}_{N,j}(ux) + \mathrm{Bits}_{N,j}(vx) - z_i N$ ausgibt. ■

Für Lemma 2.2.13 ist es nötig, die gesamten unteren Bits zu kennen und nicht nur die j-te Bitposition, da bei der Addition ein Überschlag von der $(j-1)$-ten Bitposition ins j-te Bit auftreten kann.

Wir verallgemeinern Algorithmus 2.2.3 von Seite 74 auf die j-te Bitposition. In Schritt 4 wählt man

$$\left(k^{(u)}, k^{(v)}, B^{(u)}, B^{(v)}\right) \in \left[0, 4m\epsilon(n)^{-1}\right) \times \left[0, 4\epsilon(n)^{-1}\right) \times \{0,1\}^j \times \{0,1\}^j$$

und ruft den für die j-te Bitposition modifizierten Algorithmus 2.2.2 auf (vergleiche Korollar 2.2.4 auf Seite 66). Als Verallgemeinerung von Satz 2.2.10 gilt:

Satz 2.2.14
Sei $(N, e) \in \text{RSA-PK}_n$ ein RSA-Public-Key und $j := j(n) \geq 1$ mit $j(n) = \mathcal{O}(\log_2 n)$. Sei A ein Algorithmus zur Bestimmung des j-ten Bits des Urbildes mit

$$\mathrm{Ws}\left[A\left(N, e, \mathrm{RSA}_{N,e}\left(U_{\mathbb{Z}_N^*}\right)\right) = \mathrm{bit}_{N,j}\left(U_{\mathbb{Z}_N^*}\right)\right] \geq \tfrac{1}{2} + \epsilon(n),$$

wobei die Wahrscheinlichkeit über die zufällige Nachricht $U_{\mathbb{Z}_N^*}$ und die internen Münzwürfe gebildet wird. Falls $\epsilon(n)^{-1} = n^{\mathcal{O}(1)}$ ist, bestimmt der verallgemeinerte Algorithmus 2.2.3 für hinreichend große N bzw. n in Zeit

$$2^{17,2+2j} n^3 \epsilon(n)^{-6} \left(|A| + \mathcal{O}\left(n^3\right)\right)$$

zu $\mathrm{RSA}_{N,e}(x)$ das Urbild x mit Wahrscheinlichkeit für große N bzw. n etwa $2^{-2j}\epsilon(n)^2$, wobei die Wahrscheinlichkeit nur über die internen Münzwürfe gebildet wird.

Beweis. Der Beweis ist wie in Satz 2.2.10 auf Seite 73. Wir beachten die nach Korollar 2.2.4 auf Seite 66 geringere Erfolgswahrscheinlichkeit und daß es jetzt in Schritt 4 um den Faktor 2^{2j} mehr Möglichkeiten als im ursprünglichen Algorithmus gibt. ■

Fassen wir unsere Resultate über die j-te Bitposition zusammen und verallgemeinern Korollar 2.2.11 von Seite 75:

Korollar 2.2.15
Sei $j := j(n) \geq 1$ mit $j(n) = \mathcal{O}(\log_2 n)$ und $\ell(n) \geq n+1$ durch ein Polynom beschränkt. Falls das vom RSA-Generator 2.1.1 auf Seite 58 mit Prädikat $\mathrm{bit}_{N,j}(x)$ statt $\mathrm{bit}_{N,1}(x)$ erzeugte Ensemble des Typs $\ell(n)$ einen statistischen Test T mit nicht-vernachlässigbarer Toleranz $\delta(n)$ nicht besteht, kann man die $\mathrm{RSA}_{N,e}$-Funktion für unendlich viele n und zufälligen RSA-Public-Key $(N,e) \in_{\mathrm{R}} \text{RSA-PK}_n$ in Zeit

$$2^{23,2+2j} n^3 \delta(n)^{-6} \ell(n)^6 \left(|T| + \ell(n) \cdot \mathcal{O}\left(n^3\right)\right)$$

mit Wahrscheinlichkeit für große N bzw. n etwa $2^{-2(j+1)} \frac{\delta(n)^3}{\ell(n)^3}$ invertieren.

Beweis. Wie Beweis zu Korollar 2.2.11 auf Seite 75. ■

Um die Effizienz des RSA-Generators 2.1.2 auf Seite 58 zu erhöhen, kann man (für hinreichend großes große N bzw. n) die logarithmisch vielen untersten Bits simultan ausgeben. Angenommen, wir geben die untersten $k(n)$ Bits aus. Wenn das erzeugte Ensemble von Typ $\ell(n)$ ist, genügen $\frac{\ell(n)}{k(n)}$ Iterationen. Wir erhalten aus Korollar 2.2.15 für die simultane Sicherheit der logarithmisch vielen untersten Bits:

Korollar 2.2.16
Sei $k := k(n) \geq 1$ mit $k(n) = \mathcal{O}(\log_2 n)$ und $\ell(n) \geq n+1$ durch ein Polynom beschränkt. Falls das vom RSA-Generator 2.1.1 auf Seite 58 mit Funktion

$$\mathrm{Bits}_{N,k}(x) = (\mathrm{bit}_{N,k}(x), \mathrm{bit}_{N,k-1}(x), \ldots, \mathrm{bit}_{N,1}(x))$$

statt $\mathrm{bit}_{N,1}(x)$ erzeugte Ensemble des Typs $\ell(n)$ einen statistischen Test T mit nicht-vernachlässigbarer Toleranz $\delta(n)$ nicht besteht, kann man die $\mathrm{RSA}_{N,e}$-Funktion für unendlich viele n und zufälligen RSA-Public-Key $(N,e) \in_{\mathrm{R}} \text{RSA-PK}_n$ in Zeit

$$2^{23,2+2k} n^3 \delta(n)^{-6} \ell(n)^6 \left(|T| + \frac{\ell(n)}{k(n)} \cdot \mathcal{O}\left(n^3\right)\right)$$

mit Wahrscheinlichkeit für große N bzw. n mindestens etwa $2^{-2(k+1)} \frac{\delta(n)^3}{k \cdot \ell(n)^3}$ invertieren.

Beweis. Nach Lemma 1.5.7 auf Seite 28 existiert eine Bitposition $j := j(n)$ mit $1 \leq j \leq k(n)$ und ein Prediktor zur Vorhersage des Bits $\mathrm{bit}_{N,j}(x)$ zu gegebenen $\mathrm{Bits}_{N,j-1}(x)$ und $\mathrm{RSA}_{N,e}(x)$ mit Vorteil $\frac{\delta(n)}{\ell(n)}$ und Laufzeit $|T| + \frac{\ell(n)}{k(n)} \cdot \mathcal{O}\left(n^3\right)$. Wir wählen die Position j zufällig und erhalten aus Korollar 2.2.15 die Behauptung. ■

2.3 Sicherheitsbeweis mit sukzessiver Approximation

In diesem Kapitel stellen wir den Sicherheitsbeweis aus [FSch97] für den RSA-Generator vor. Im Vergleich zu dem Beweis aus Kapitel 2.2 ist dieser konzeptionell einfacher. Die Laufzeit zum Invertieren der RSA-Funktion und insbesondere die Anzahl der Aufrufe des Algorithmus', der das Hardcore-Prädikate mit Vorteil $\epsilon(n)$ vorhersagt, wird deutlich reduziert.

2.3.1 Invertieren durch sukzessive Approximation

Wir betrachten zunächst wieder den Fall $j = 1$, d.h. das unterste Bit modulo N, und verwenden wie in Kapitel 2.2 die Notation $[z]_N$ für den kleinsten, nicht-negativen Repräsentanten der Restklasse von z modulo N:

$$[z]_N := (z \bmod N) \in \{0, 1, \ldots, N-1\}$$

Sei A ein Algorithmus, der zu gegebenem $\mathrm{RSA}_{N,e}(x)$ einen $\epsilon(n)$-Vorteil zur Vorhersage von $\mathrm{bit}_{N,1}(x)$ hat. Zur Vereinfachung sei zunächst wieder für die betrachteten n die Erfolgswahrscheinlichkeit von Algorithmus A unabhängig von N und e. Mit diesem Algorithmus wollen wir bezüglich n und $\epsilon(n)^{-1}$ zu gegebenem $\mathrm{RSA}_{N,e}(x)$ das Urbild $x \in \mathbb{Z}_N^*$ berechnen. Wir setzen ferner wieder $\epsilon(n)^{-1} = n^{\mathcal{O}(1)}$ voraus, um für hinreichend große N bzw. n die Wahrscheinlichkeit, daß ein modulo N gleichverteilter Wert nicht teilerfremd zu N ist, zu vernachlässigen (vergleiche Lemma 2.2.1 auf Seite 61).

Wir greifen eine Idee von S. Goldwasser, S. Micali und P. Tong [GMT82] auf: Man wählt $a_1 \in \mathbb{Z}_N$ zufällig und beginnt mit einer Approximation $k_1 N$ von $[a_1 x]_N$, so daß

$$\Big| [a_1 x]_N - k_1 N \Big| \leq \kappa N$$

für ein gegebenes $\kappa \in [0, 1]$ gilt. Mit Hilfe des Algorithmus' A bestimmen wir $b_1 := \mathrm{bit}_{N,1}(a_1 x)$ und setzen $a_2 := 2^{-1} a_1 \bmod N$. Wir berechnen aus $k_1 N$ und b_1 eine Approximation $k_2 N$ zu $[a_2 x]_N$ mit

$$\Big| [a_2 x]_N - k_2 N \Big| \leq \tfrac{1}{2} \kappa N.$$

Dazu unterscheiden wir zwei Fälle:

- Es gilt $b_1 = 0$. Dann ist $[a_1 x]_N$ gerade und wir erhalten $[a_2 x]_N = \frac{1}{2} [a_1 x]_N$. Mit $k_2 = \frac{1}{2} k_1$ folgt:

$$\begin{aligned} \Big| [a_2 x]_N - k_2 N \Big| &= \Big| \tfrac{1}{2} [a_1 x]_N - \tfrac{1}{2} k_1 N \Big| \\ &= \tfrac{1}{2} \cdot \Big| [a_1 x]_N - k_1 N \Big| \\ &\leq \tfrac{1}{2} \kappa N \end{aligned}$$

- Es gilt $b_1 = 1$. Dann ist $[a_1 x]_N$ ungerade und, da der RSA-Modul N das Produkt zwei ungerader Primzahlen ist, erhalten wir, daß $N + [a_1 x]_N$ gerade ist. Wegen

$$N \leq N + [a_1 x]_N < 2N$$

gilt:

$$[a_2 x]_N = \left[2^{-1}(N + a_1 x)\right]_N = \tfrac{1}{2} \left(N + [a_1 x]_N\right)$$

Aus der Voraussetzung $\Big| [a_1 x]_N - k_1 N \Big| \leq \kappa N$ folgt

$$\Big| N + [a_1 x]_N - (k_1 + 1) N \Big| \leq \kappa N$$

und mit $k_2 := \frac{1}{2}(k_1 + 1)$ erhalten wir:

$$\begin{aligned}\Big|\,[a_2x]_N - k_2N\Big| &= \Big|\tfrac{1}{2}(N + [a_1x]_N) - \tfrac{1}{2}(k_1+1)N\Big| \\ &= \tfrac{1}{2}\cdot\Big|N + [a_1x]_N - (k_1+1)N\Big| \\ &\le \tfrac{1}{2}\kappa N\end{aligned}$$

In beiden Fällen verbessern wir unsere Approximation mit $k_2 := \frac{1}{2}(k_1 + b_1)$ um den Faktor 2. Mit a_2x wird der Vorgang wiederholt und nach t Iterationen kennen wir a_t und k_t mit:

$$\Big|\,[a_tx]_N - k_tN\Big| \le \frac{\kappa N}{2^t}$$

Wenn $\Big|\,[a_tx]_N - k_tN\Big| < \frac{1}{2}$ gilt, ist $[a_tx]_N$ die nächste ganze Zahl bei k_tN und wir erhalten das gesuchte Urbild x als:

$$x \equiv a_t^{-1}\left\lfloor k_tN + \tfrac{1}{2}\right\rfloor \pmod N$$

Für die Mehrheitsentscheidung wählt man ähnlich wie in Kapitel 2.2 zwei Werte u, v unabhängig und zufällig aus $\mathbb{Z}_N$, um modulo N gleichverteilte und paarweise unabhängige $m := n\epsilon(n)^{-2}$ Testpunkte

$$r_i := [iu + v]_N \qquad \text{für } i = \pm1, \pm2, \ldots, \pm\tfrac{1}{2}m$$

zu erhalten. Man überlege sich, daß der Beweis zu Lemma 2.2.7 von Seite 70 auch für diese Punkte gilt. Durch eine Mehrheitsentscheidung bestimmen wir, ob $\mathrm{bit}_{N,1}(a_tx) = 0$ ist. Dazu setzen wir die Kenntnis von $\mathrm{bit}_{N,1}(ux)$ und $\mathrm{bit}_{N,1}(ux)$ sowie ganzer Zahlen $w_{t,i}$, die jeweils mit Wahrscheinlichkeit mindestens $1 - \frac{1}{4}\epsilon(n)$

$$[(a_t + r_i)x]_N = [a_tx]_N + i\cdot[ux]_N + [vx]_N - w_{t,i}N \tag{2.13}$$

erfüllen ($i = \pm1, \pm2, \ldots, \pm\frac{1}{2}m$), voraus. Einen Wert $w_{t,i}$ mit Eigenschaft (2.13) heißt *richtig*. Wegen $N \equiv 1 \pmod N$ nehmen wir an, daß $\mathrm{bit}_{N,1}(a_tx) = 0$ bzw. $[a_tx]_N \equiv 0 \pmod 2$ ist, wenn für die Mehrheit der m Testpunkte $r_i = [iu + v]_N$ ($i = \pm1, \pm2, \ldots, \pm\frac{1}{2}m$) gilt:

$$A\left(N, e, \mathrm{RSA}_{N,e}\left((a_t + r_i)x\right)\right) \equiv i\cdot[ux]_N + [vx]_N - \lfloor w_{t,i}\rfloor \pmod 2$$

Neben einem falschen Resultat von Algorithmus A ist ein fehlerhaftes $w_{t,i}$ eine Fehlerquelle für die i-te Entscheidung. Im Vergleich zum ggT-Verfahren aus Kapitel 2.2 setzen wir nicht mehr voraus, daß a_t modulo N $\epsilon(n)$-klein ist, um Reduktionen modulo N nicht berücksichtigen zu müssen.

Wir zeigen zunächst, wie man aus Approximationen von $[a_tx]_N$, $[ux]_N$ und $[vx]_N$ den richtigen Wert $w_{t,i}$ mit Wahrscheinlichkeit mindestens $1 - \frac{1}{4}\epsilon(n)$ bestimmen kann:

Lemma 2.3.1
Sei $t \geq 1$ und $r_i = [iu + v]_N$ modulo N gleichverteilt. Gegeben seien $k_t \in \frac{\epsilon(n)}{2^t \cdot 4}\left[0, 2^t \cdot 4\epsilon(n)^{-1}\right)$, $k^{(u)} \in \frac{\epsilon(n)^3}{4n}\left[0, 4n\epsilon(n)^{-3}\right)$ und $k^{(v)} \in \frac{\epsilon(n)}{4}\left[0, 4\epsilon(n)^{-1}\right)$ mit:

$$\begin{aligned}\Big|\,[a_t x]_N - k_t N\Big| &\leq \frac{\epsilon(n)^3}{2^t \cdot 8} N \\ \Big|\,[ux]_N - k^{(u)} N\Big| &\leq \frac{\epsilon(n)^3}{8n} N \\ \Big|\,[vx]_N - k^{(v)} N\Big| &\leq \frac{\epsilon(n)}{8} N\end{aligned}$$

Für festes i mit $0 < |i| \leq \frac{1}{2} n\epsilon(n)^{-2}$ erfüllt dann $w_{t,i} := \left\lfloor k_t + i \cdot k^{(u)} + k^{(v)} \right\rfloor$ die Gleichung

$$[(a_t + r_i)x]_N = [a_t x]_N + i \cdot [ux]_N + [vx]_N - w_{t,i} N \tag{2.14}$$

mit Wahrscheinlichkeit mindestens $1 - \frac{1}{4}\epsilon(n)$.

Beweis. Wir definieren $\overline{w}_{t,i} := k_t + i \cdot k^{(u)} + k^{(v)}$, so daß $w_{t,i} = \lfloor \overline{w}_{t,i} \rfloor$ gilt. Wegen $w_{t,i} = \left\lfloor k_t + i \cdot k^{(u)} + k^{(v)} \right\rfloor$ folgt:

$$\begin{aligned}&\Big|\,[a_t x]_N + i \cdot [ux]_N + [vx]_N - \overline{w}_{t,i} N\Big| \\ \leq\ &\Big|\,[a_t x]_N - k_t N\Big| + |i| \cdot \Big|\,[ux]_N - i \cdot k^{(u)} N\Big| + \Big|\,[vx]_N - k^{(v)} N\Big|\end{aligned}$$

Durch Einsetzen der drei gegebenen Approximationen $k_t, k^{(u)}$ und $k^{(v)}$ erhalten wir:

$$\begin{aligned}\Big|\,[a_t x]_N + i \cdot [ux]_N + [vx]_N - \overline{w}_{t,i} N\Big| &\leq \frac{\epsilon(n)^3}{2^t \cdot 8} N + \frac{|i| \cdot \epsilon(n)^3}{8n} N + \frac{\epsilon(n)}{8} N \\ &= \frac{\epsilon(n)}{8}\left(\frac{1}{2^t} + \frac{|i| \cdot \epsilon(n)^2}{n} + 1\right) N\end{aligned} \tag{2.15}$$

Aus $t \geq 1$ und $|i| \leq \frac{1}{2} n\epsilon(n)^{-2}$ folgt, der Ausdruck innerhalb der Klammern ist maximal 2, so daß gilt:

$$\Big|\,[a_t x]_N + i \cdot [ux]_N + [vx]_N - \overline{w}_{t,i} N\Big| \leq \tfrac{1}{4}\epsilon(n) N \tag{2.16}$$

Der Wert $[a_t x]_N + i \cdot [ux]_N + [vx]_N$ liegt im schraffierten Bereich der Zahlengerade eines der drei Fälle in Abbildung 2.3.1, wobei wir zur Vereinfachung zunächst den Fall $\overline{w}_{t,i} \in \mathbb{Z}$ ausschließen.

Aus Abschätzung (2.16) folgt, daß dieses Intervall höchstens $\frac{1}{2}\epsilon(n)N < \frac{1}{2}N$ lang ist. Da die Länge kürzer als N ist, kann es maximal ein Vielfaches von N enthalten. Genau dann, wenn die Gleichung (2.14), also

$$[(a_t + r_i)x]_N = [a_t x]_N + i \cdot [ux]_N + [vx]_N - \lfloor \overline{w}_{t,i} \rfloor \cdot N,$$

nicht gilt, ist der Wert $[a_t x]_N + i \cdot [ux]_N + [vx]_N$ entweder kleiner als $\lfloor \overline{w}_{t,i} \rfloor \cdot N$ oder größer (bzw. gleich) $\lceil \overline{w}_{t,i} \rceil \cdot N$. Diese Teile des Intervalls haben nach Abschätzung (2.16) maximal die Länge $\frac{1}{4}\epsilon(n)N$. Da nach Voraussetzung $r_i = [iu + v]_N$ modulo N gleichverteilt ist, gilt dies auch für $[(a_t + r_i)x]_N$, d.h. mit Wahrscheinlichkeit maximal $\frac{1}{4}\epsilon(n)$ ist die Wahl $w_{t,i} = \lfloor \overline{w}_{t,i} \rfloor$ falsch. Falls $\overline{w}_{t,i}$ ganzzahlig ist, verwenden wir die gleiche Argumentation jedoch mit $\overline{w}_{t,i} + 1$ statt $\lceil \overline{w}_{t,i} \rceil$. ■

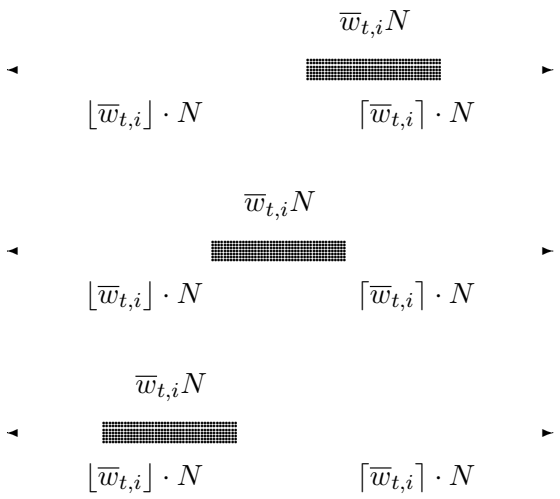

Abbildung 2.3.1: Skizze zu Beweis von Lemma 2.3.1

Nur wenn für die Nachkommastellen von $\overline{w}_{t,i} := k_t + i \cdot k^{(u)} + k^{(v)}$ gilt

$$1 - \tfrac{1}{4}\epsilon(n) < \overline{w}_{t,i} \bmod 1 < 1,$$

kann die Gleichung (2.14) verletzt sein, da wir in diesem Fall eventuell fälschlicherweise abrunden.

Durch Verwenden der Indizies $i = \pm 1, \pm 2, \ldots, \pm\frac{1}{2}m$ können wir $|i|$ in Ungleichung (2.15) durch $\frac{1}{2}m$ nach oben abschätzen und im Vergleich zur Wahl $i = 1, 2, \ldots, m$ die Approximationen um den Faktor 2 schlechter wählen. Wir analysieren die Laufzeit des ersten Ansatzes, Algorithmus 2.3.1 (vergleiche mit Satz 2.2.10 auf Seite 73):

Satz 2.3.2
Sei $(N, e) \in \text{RSA-PK}_n$ *ein RSA-Public-Key. Sei* A *ein Algorithmus zur Bestimmung des untersten Bits des Urbildes mit*

$$\text{Ws}\left[A\left(N, e, \text{RSA}_{N,e}\left(U_{\mathbb{Z}_N^*}\right)\right) = \text{bit}_{N,1}\left(U_{\mathbb{Z}_N^*}\right)\right] \geq \tfrac{1}{2} + \epsilon(n),$$

wobei die Wahrscheinlichkeit über die zufällige Nachricht $U_{\mathbb{Z}_N^*}$ *und die internen Münzwürfe gebildet wird. Falls* $\epsilon(n)^{-1} = n^{\mathcal{O}(1)}$ *ist, bestimmt für hinreichend große* N *bzw.* n *Algorithmus 2.3.1 in Zeit*

$$2^9 n^3 \epsilon(n)^{-7}\left(|A| + \mathcal{O}\left(n^3\right)\right)$$

zu $\text{RSA}_{N,e}(x)$ *das Urbild* x *mit Wahrscheinlichkeit mindestens* $\frac{1}{2}$*, wobei die Wahrscheinlichkeit nur über die internen Münzwürfe gebildet wird.*

Beweis. Betrachten wir zunächst die Wahrscheinlichkeit, daß eine Mehrheitsentscheidung richtig ist. Wir definieren Indikatorvariable $X_1, X_2, \ldots, X_m$ für das Ereignis, daß die i-te Entscheidung falsch ist. Die m Zufallsvariablen $X_1, X_2, \ldots, X_m$ hängen von den internen Münzwürfen des Algorithmus' A und den m Werten $a_t + r_i$, $i = \pm 1, \pm 2, \ldots, \pm\frac{1}{2}m$, ab.

Algorithmus 2.3.1 RSA-Invertierung durch sukzessive Approximation (erster Ansatz)

EINGABE: ▷ RSA-Public-Key $(N, e) \in \text{RSA-PK}_n$
▷ $\text{RSA}_{N,e}(x)$ mit $x \in \mathbb{Z}_N^*$

/∗ Sei A Algorithmus mit $\text{Ws}\left[A\left(N, e, \text{RSA}_{N,e}\left(U_{\mathbb{Z}_N^*}\right)\right) = \text{bit}_{N,1}\left(U_{\mathbb{Z}_N^*}\right)\right] \geq \frac{1}{2} + \epsilon(n)$, wobei die Wahrscheinlichkeit über die zufällige Nachricht $U_{\mathbb{Z}_N^*}$ und die internen Münzwürfe gebildet wird. ∗/

/∗ Erzeugt m Testpunkte für Mehrheitsentscheidung: ∗/

1. Wähle $u, v, a_0 \in \mathbb{Z}_N$ zufällig und unabhängig.

2. $m := n\epsilon(n)^{-2}$

3. FOR $i = 1$ TO m DO $r_i := [iu + v]_N$.

/∗ Probiere alle möglichen Approximationen und untersten Bits: ∗/

4. FOR all $(k_0, k^{(u)}, k^{(v)}, b_0, b^{(u)}, b^{(v)}) \in \frac{\epsilon(n)}{4}\left[0, 4\epsilon(n)^{-1}\right)^2 \times \frac{\epsilon(n)}{4m}\left[0, 4m\epsilon(n)^{-1}\right) \times \{0,1\}^3$ DO

4.1. FOR $t := 1$ TO n DO

4.1.1. $a_t := \left[2^{-1} a_{t-1}\right]_N$

4.1.2. $k_t := \frac{1}{2}\left(k_{t-1} + b_{t-1}\right)$

4.1.3. FOR $i = 1$ TO m DO $w_{t,i} := \left\lfloor k_t + i \cdot k^{(u)} + k^{(v)} \right\rfloor$

4.1.4. $M := \left\{ i \in \{\pm 1, \ldots, \pm\frac{1}{2}m\} \;\middle|\; \begin{array}{ll} & A\left(N, e, \text{RSA}_{N,e}\left((a_t + r_i)x\right)\right) \\ \equiv & i \cdot [ux]_N + [vx]_N - w_{t,i} \pmod 2 \end{array} \right\}$

4.1.5. IF $|M| \geq \frac{1}{2}m$ THEN $b_t := 0$ ELSE $b_t := 1$

END for t

4.2. IF $\text{RSA}_{N,e}\left(a_n \cdot \left\lfloor k_n N + \frac{1}{2} \right\rfloor\right) = \text{RSA}_{N,e}(x)$ THEN gib $\left[a_n \cdot \left\lfloor k_n N + \frac{1}{2} \right\rfloor\right]_N$ aus

END for all

Wir wählen u, v, a_0 zufällig und unabhängig. Wegen $a_t = \left[2^{-t} a_0\right]_N$ folgt aus Lemma 2.2.7 auf Seite 70, daß die Werte $a_t + r_i$ modulo N gleichverteilt und paarweise unabhängig sind. Die Indikatorvariablen sind identisch verteilt und paarweise unabhängig. Aus Lemma 2.3.1 erhalten wir

$$\text{Ws}[X_i = 1] \leq \tfrac{1}{2} - \epsilon(n) + \tfrac{1}{4}\epsilon(n) = \tfrac{1}{2} - \tfrac{3}{4}\epsilon(n)$$

und der Erwartungswert ist $\mu \leq \frac{1}{2} - \frac{3}{4}\epsilon(n)$. Aus $m\mu \leq \frac{1}{2}m - \frac{3}{4}m\epsilon(n)$ folgt:

$$\text{Ws}\left[\begin{array}{l}\text{Mehrheitsent-}\\ \text{scheidung falsch}\end{array}\right] \leq \text{Ws}\left[\sum_{i=1}^{m} X_i \geq \frac{m}{2}\right] \leq \text{Ws}\left[\sum_{i=1}^{m} X_i - \mu m \geq \frac{3m\epsilon(n)}{4}\right] \tag{2.17}$$

Wir schätzen die Varianz σ^2 nach oben ab durch:

$$\sigma^2 = \text{E}\left[X_i^2\right] - \text{E}[X_i]^2 \leq \max_{\xi \in [0,1]} \left\{\xi - \xi^2\right\} = \tfrac{1}{4} \tag{2.18}$$

Aus Lemma 1.6.1 auf Seite 39 folgt mit $t := \frac{3}{4}\epsilon(n)$:

$$\mathrm{Ws}\left[\begin{array}{l}\text{Mehrheitsent-}\\ \text{scheidung falsch}\end{array}\right] \leq \mathrm{Ws}\left[\left|\sum_{i=1}^{m} X_i - \mu m\right| \geq \frac{3m\epsilon(n)}{4}\right] \leq \frac{\sigma^2}{t^2 m}$$

Mit der Abschätzung aus (2.18) gilt:

$$\mathrm{Ws}\left[\begin{array}{l}\text{Mehrheitsent-}\\ \text{scheidung falsch}\end{array}\right] \leq \frac{9}{4m\epsilon(n)^2} < \frac{1}{2n}$$

In der Schleife von Schritt 4.1 wird zu jeder Wahl von Schritt 4 n-mal eine Mehrheitsentscheidung getroffen. Für die richtige Wahl in Schritt 4 sind mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ die Mehrheitsentscheidungen richtig.

Durch Induktion zeigt man, daß für die richtige Wahl in Schritt 4 und richtige Mehrheitsentscheidungen als Schleifeninvariante

$$\Big|\,[a_{t-1}x]_N - k_{t-1}N\Big| \leq \frac{\epsilon(n)}{2^{t-1}\cdot 16}N$$

gilt. Beim Verlassen der Schleife ist wegen $t = n+1$ und $\epsilon(n) < 1$

$$\Big|\,[a_n x]_N - k_n N\Big| < \tfrac{1}{2},$$

und $[a_t x]_N$ ist die nächste ganze Zahl bei $k_t N$. Wir erhalten das gesuchte Urbild x als:

$$x \equiv a_n^{-1}\left\lfloor k_n N + \tfrac{1}{2}\right\rfloor \pmod N$$

Analysieren wir die Laufzeit von Algorithmus 2.3.1. Die Anzahl der Iterationen der Schleife in Schritt 4 ist wegen $m = n\epsilon(n)^{-2}$:

$$2^6\epsilon(n)^{-2}\cdot m\epsilon(n)^{-1}\cdot 2^3 = 2^9\epsilon(n)^{-3}m = 2^9 n\epsilon(n)^{-5}$$

Wir wiederholen die Schleife in Schritt 4.1 n-mal. In jeder Iteration ruft man m-mal Algorithmus A auf, so daß die Anzahl der Aufrufe von Algorithmus A gleich

$$2^9 n\epsilon(n)^{-5}\cdot n\cdot m = 2^9 n\epsilon(n)^{-5}\cdot n\cdot n\epsilon(n)^{-2} = 2^9 n^3\epsilon(n)^{-7}$$

ist. Bei jedem Aufruf berechnen wir eine RSA-Kodierung, was mit einem Divide-&-Conquer-Verfahren in $\mathcal{O}\left(n^3\right)$ Schritten gelingt. Die Zahlen $k_t, k^{(u)}, k^{(v)}$ sind durch $\mathcal{O}\left(\log_2(n\epsilon(n)^{-1})\right)$ Bits darstellbar. ■

Wir verbessern Algorithmus 2.3.1. Die Argumente von Algorithmus A beim Aufruf in Schritt 4.1.4 sind unabhängig von der Wahl $(k_0, k^{(u)}, k^{(v)}, b_0, b^{(u)}, b^{(v)})$ in Schritt 4. Man kann daher nach Schritt 3 bereits Algorithmus A aufrufen:

$$\begin{array}{ll} b_{t,i} := A\left(N, e, \mathrm{RSA}_{N,e}\left((a_t + r_i)x\right)\right) & \text{für} \quad i = \pm 1, \pm 2, \ldots, \pm\frac{1}{2}m \\ & \text{und} \quad t = 1, 2, \ldots, n \end{array}$$

Wir senken die Anzahl der Aufrufe des Algorithmus' A von $2^9 n^3\epsilon(n)^{-7}$ auf $n^2\epsilon(n)^{-2}$. Dies ist möglich, da die Argumente $(a_t + r_i)x$ im Gegensatz zur Invertieren mit dem ggT-Algorithmus

in Kapitel 2.2 unabhängig von den Approximationen und den vorherigen Resultaten des Algorithmus' A sind.

Um die Laufzeit von Algorithmus 2.3.1 zusätzlich zu senken, setzen wir $a_0 := u$ in Schritt 1, anstatt a_0 zufällig und unabhängig aus $\mathbb{Z}_N$ zu wählen. In Schritt 4 wählt man nur $(k^{(u)}, k^{(v)}, b^{(u)}, b^{(v)})$ und setzen anschließend $k_0 = k^{(v)}$ und $b_0 = b^{(v)}$. Nachdem wir b_t durch eine Mehrheitsentscheidung bestimmt haben, können wir u durch a_t ersetzen, denn das Paar (k_t, b_t) hat die Eigenschaften für $(k^{(u)}, b^{(u)})$ und zusätzlich gilt:

$$\left| [a_t x]_N - k_t N \right| = 2^{t-i} \cdot \left| [a_i x]_N - k_i N \right| \qquad \text{für } i = 0, 1, \ldots, t-1 \tag{2.19}$$

Dadurch verbessert sich in jeder Iteration die Approximationen von $[ux]_N$, so daß die Fehlerwahrscheinlichkeit bei der Berechnung der $w_{t,i}$ abnimmt (vergleiche Lemma 2.3.1). Wir wählen zu Beginn eine schlechtere Approximation und verbessern diese sukzessive. Da die Fehlerwahrscheinlichkeit bei der Berechnung der $w_{t,i}$ von der Güte der Approximationen und von i, also der Anzahl der Testpunkte, abhängt, verwenden wir zu Beginn weniger Testpunkte und erhöhen parallel zur Verbesserung der Approximation von $[ux]_N$ die Anzahl der Testpunkte. Die größere Fehlerwahrscheinlichkeit bei der Mehrheitsentscheidung der ersten Iterationen gleichen man mit der kleineren Fehlerwahrscheinlichkeit in den nachfolgenden Iterationen aus. Wir setzen $u_t := a_t$ für $t = 0, 1, \ldots, n$ und bilden in Iteration $t = 1, 2, \ldots, n$ die Mehrheitsentscheidung jeweils über

$$m_t := \min\left\{2^t, 2n\right\} \cdot \epsilon(n)^{-2} = \begin{cases} 2^t \epsilon(n)^{-2} & \text{falls } t \leq 1 + \log_2 n \\ 2n\epsilon(n)^{-2} & \text{sonst} \end{cases}$$

Testpunkte. Sei $M_t := \left\{ i \in \mathbb{Z} \setminus \{0\} : \left|i + \frac{1}{2}\right| \leq \frac{1}{2} m_t \right\}$. Um die Approximation k_t und das unterste Bit b_t von $[a_t x]_N$ zu bestimmen, verwenden wir die Testpunkte

$$r_{t,i} := [i u_{t-1} + v]_N = \left[2^{-1} i a_t + v\right]_N$$

für $i \in M_t$. Bei Eintritt in die Iteration t kennen wir:

$$\begin{aligned} \left| [u_{t-1} x]_N - k_{t-1}^{(u)} N \right| &\leq \frac{\epsilon(n)^3}{2^{t-1} \cdot 8} N \\ \left| [vx]_N - k^{(v)} N \right| &\leq \frac{\epsilon(n)}{8} N \end{aligned} \tag{2.20}$$

Für $\overline{w}_{t,i} := k_t + i \cdot k_{t-1}^{(u)} + k^{(v)}$ gilt wegen $u_t = a_t$, $k_{t-1}^{(u)} = k_t$ und Gleichung (2.19) auf Seite 85:

$$\begin{aligned} & \left| [a_t x]_N + [r_{t,i} x]_N - \overline{w}_{t,i} N \right| \\ \leq & \left| [a_t x]_N + i \cdot [u_{t-1} x]_N + [vx]_N - k_t N - i \cdot k_{t-1}^{(u)} \cdot N - k^{(v)} \cdot N \right| \\ \leq & \left| \tfrac{1}{2} [u_{t-1} x]_N - \tfrac{1}{2} k_{t-1}^{(u)} N + i \cdot [u_{t-1} x]_N - i \cdot k_{t-1}^{(u)} N \right| + \left| [vx]_N - k^{(v)} N \right| \end{aligned}$$

Wir erhalten

$$\left| [a_t x]_N + [r_{t,i} x]_N - \overline{w}_{t,i} N \right| \leq \left|i + \tfrac{1}{2}\right| \cdot \left| [u_{t-1} x]_N - k_{t-1}^{(u)} N \right| + \left| [vx]_N - k^{(v)} N \right|$$

und durch Einsetzen der beiden Approximationen aus (2.20) folgt:

$$\left| [a_t x]_N + [r_{t,i} x]_N - \overline{w}_{t,i} N \right| \leq \frac{\left|i + \frac{1}{2}\right| \cdot \epsilon(n)^3}{2^{t-1} \cdot 8} N + \frac{\epsilon(n)}{8} N = \frac{\epsilon(n)}{8} \left(\frac{\left|i + \frac{1}{2}\right| \cdot \epsilon(n)^2}{2^{t-1}} + 1 \right) N$$

Wegen $\left|i + \frac{1}{2}\right| \leq 2^{t-1} \epsilon(n)^{-2}$ für $i \in M_t$ erhalten wir:

$$\left| [a_t x]_N + [r_{t,i} x]_N - \overline{w}_{t,i} N \right| \leq \tfrac{1}{4} \epsilon(n) N \tag{2.21}$$

Wir sind in der gleichen Situation (2.16) wie im Beweis zu Lemma 2.3.1 auf Seite 81. Der $w_{t,i}$-Wert kann nur falsch sein, wenn für die Nachkommastellen von $\overline{w}_{t,i} := k_t + i \cdot k^{(u)} + k^{(v)}$ gilt:

$$1 - \tfrac{1}{4} \epsilon(n) < \overline{w}_{t,i} \bmod 1 < 1 \tag{2.22}$$

Man kann jedes $w_{t,i}$ mit Wahrscheinschlichkeit $1 - \frac{1}{4}\epsilon(n)$ bestimmen, wenn für die Testpunkte $r_{t,i}$ gilt, daß die Werte $a_t + r_{t,i}$ modulo N gleichverteilt sind. Für die Analyse der Fehlerwahrscheinlichkeit benötigen wir fener, daß die Werte modulo N paarweise unbhängig sind:

Lemma 2.3.3
Sei N ein RSA-Modul mit Primfaktoren p, q und $\frac{1}{2}m < \min\{p, q\}$. Seien u_{t-1}, v zufällig und unabhängig modulo N verteilt sowie $a_t = \left[2^{-1} u_{t-1}\right]_N$. Dann gilt für die m Punkte

$$r_{t,i} := [i u_{t-1} + v]_N \qquad \text{für } i = \pm 1, \pm 2, \ldots, \pm \tfrac{1}{2} m,$$

daß für festes $t \geq 1$ die m Werte $a_t + r_{t,i}$ modulo N gleichverteilt und paarweise unabhängig sind.

Beweis. Da u_{t-1} und v gleichverteilt in $\mathbb{Z}_N$ sind, ist auch $a_t + r_{t,i}$ modulo N gleichverteilt. Für i, j mit $i \neq j$ zeigen wir, daß $a_t + r_{t,i}$ und $a_t + r_{t,j}$ modulo N unabhängig verteilt sind. Die Transformationsmatrix

$$\begin{bmatrix} a_t + r_{t,i} \\ a_t + r_{t,j} \end{bmatrix} \equiv \begin{bmatrix} 2^{-1} + i & 1 \\ 2^{-1} + j & 1 \end{bmatrix} \cdot \begin{bmatrix} u_{t-1} \\ v \end{bmatrix} \pmod{N}$$

hat die Determinante $i - j \neq 0$. Wegen $i, j < \frac{1}{2} \min\{p, q\}$ gilt $0 < |i - j| < \min\{p, q\}$ und die Differenz hat modulo N ein Inverses, so daß die Transformation regulär ist. Für $i \neq j$ sind $a_t + r_{t,i}$ und $a_t + r_{t,j}$ modulo N unabhängig. ■

Wir nehmen diese Verbesserungen in Algorithmus 2.3.1 auf, Algorithmus 2.3.2 zeigt das verbesserte Verfahren zum Invertieren der RSA-Funktion durch sukzessive Approximation:

Satz 2.3.4
Sei $(N, e) \in \text{RSA-PK}_n$ ein RSA-Public-Key. Sei A ein Algorithmus zur Bestimmung des untersten Bits des Urbildes mit

$$\text{Ws}\left[A\left(N, e, \text{RSA}_{N,e}\left(U_{\mathbb{Z}_N^*}\right)\right) = \text{bit}_{N,1}\left(U_{\mathbb{Z}_N^*}\right)\right] \geq \tfrac{1}{2} + \epsilon(n),$$

Algorithmus 2.3.2 RSA-Invertierung durch sukzessive Approximation

EINGABE: ▷ RSA-Public-Key $(N, e) \in \text{RSA-PK}_n$
▷ $\text{RSA}_{N,e}(x)$ mit $x \in \mathbb{Z}_N^*$

/∗ Sei A Algorithmus mit $\text{Ws}\left[A\left(N, e, \text{RSA}_{N,e}\left(U_{\mathbb{Z}_N^*}\right)\right) = \text{bit}_{N,1}\left(U_{\mathbb{Z}_N^*}\right)\right] \geq \frac{1}{2} + \epsilon(n)$, wobei die Wahrscheinlichkeit über die zufällige Nachricht $U_{\mathbb{Z}_N^*}$ und die internen Münzwürfe gebildet wird. ∗/

1. Wähle $u_0, v \in \mathbb{Z}_N$ zufällig und unabhängig. Setze $a_0 := u_0$.

2. FOR $t := 1$ TO n DO $a_t := u_t := \left[2^{-t} u_0\right]_N$

3. FOR $t := 1$ TO n /∗ Bestimme mit Algorithmus A unterste Bits ∗/

3.1. $m_t := \min\left\{2^t, 2n\right\} \cdot \epsilon(n)^{-2}$

3.2. $M_t := \left\{i \in \mathbb{Z} \setminus \{0\} \ : \ \left|i + \frac{1}{2}\right| \leq \frac{1}{2} m_t\right\}$

3.3. FOR all $i \in M_t$ DO

3.3.1. $r_{t,i} := [i u_{t-1} + v]_N$

3.3.2. $b_{t,i} := A\left(N, e, \text{RSA}_{N,e}\left((a_t + r_{t,i})x\right)\right)$

END for i

END for t

/∗ Probiere alle möglichen Approximationen und untersten Bits: ∗/

4. FOR all $\left(k_0^{(u)}, k^{(v)}, b_0^{(u)}, b^{(v)}\right) \in \frac{1}{4}\epsilon(n)^3 \left[0, 4\epsilon(n)^{-3}\right) \times \frac{1}{4}\epsilon(n) \left[0, 4\epsilon(n)^{-1}\right) \times \{0,1\}^2$ DO

4.1. $k_0 := k_0^{(u)}$ und $b_0 := b_0^{(u)}$.

4.2. FOR $t := 1$ TO n DO

4.2.1. $k_t := \frac{1}{2}\left(k_{t-1} + b_{t-1}\right)$

4.2.2. FOR all $i \in M_t$ DO $w_{t,i} := \left\lfloor k_t + i \cdot k_{t-1}^{(u)} + k^{(v)} \right\rfloor$

4.2.3. $M := \{i \in M_t \mid b_{t,i} \equiv i \cdot [u_{t-1} x]_N + [vx]_N - w_{t,i} \pmod 2\}$

4.2.4. IF $|M| \geq \frac{1}{2} m_t$ THEN $b_t := 0$ ELSE $b_t := 1$

4.2.5. $k_t^{(u)} := k_t$

END for t

4.3. IF $\text{RSA}_{N,e}\left(a_n \cdot \left\lfloor k_n N + \frac{1}{2} \right\rfloor\right) = \text{RSA}_{N,e}(x)$ THEN gib $\left[a_n \cdot \left\lfloor k_n N + \frac{1}{2} \right\rfloor\right]_N$ aus

END for all

wobei die Wahrscheinlichkeit über die zufällige Nachricht $U_{\mathbb{Z}_N^*}$ und die internen Münzwürfe gebildet wird. Falls $\epsilon(n)^{-1} = n^{\mathcal{O}(1)}$ ist, bestimmt für hinreichend große N bzw. n Algorithmus 2.3.2 in Zeit

$$2^6 n^2 \epsilon(n)^{-6} + 2n^2 \epsilon(n)^{-2}\left(|A| + \mathcal{O}\left(n^3\right)\right)$$

zu $\mathrm{RSA}_{N,e}(x)$ *das Urbild* x *mit Wahrscheinlichkeit mindestens* $\frac{1}{3}$*, wobei die Wahrscheinlichkeit nur über die internen Münzwürfe gebildet wird.*

Beweis. Nach unseren Vorüberlegungen müssen wir noch die Fehlerwahrscheinlichkeit bei richtiger Wahl von $\left(k_0^{(u)}, k^{(v)}, b_0^{(u)}, b^{(v)}\right)$ analysieren, d.h. die Fehlerwahrscheinlichkeiten der einzelnen Iterationen abschätzen. Wir definieren wie im Beweis zu Satz 2.3.2 auf Seite 82 paarweise unabhängige Indikatorvariablen $X_1, X_2, \ldots, X_{m_t}$ mit Erwartungswert $\mu \leq \frac{1}{2} - \frac{3}{4}\epsilon(n)$. Es gilt:

$$\text{(2.23)}\quad \mathrm{Ws}\left[\begin{array}{l} t\text{-te Mehrheitsent-} \\ \text{scheidung falsch} \end{array}\right] \leq \mathrm{Ws}\left[\sum_{i=1}^{m_t} X_i \geq \frac{m}{2}\right] \leq \mathrm{Ws}\left[\sum_{i=1}^{m_t} X_i - \mu m_t \geq \frac{3m_t\epsilon(n)}{4}\right]$$

Aus Lemma 1.6.1 auf Seite 39 folgt mit $t := \frac{3}{4}\epsilon(n)$:

$$\mathrm{Ws}\left[\begin{array}{l} t\text{-te Mehrheitsent-} \\ \text{scheidung falsch} \end{array}\right] \leq \mathrm{Ws}\left[\left|\sum_{i=1}^{m_t} X_i - \mu m\right| \geq \frac{3m_t\epsilon(n)}{4}\right] \leq \frac{\sigma^2}{4t^2 m_t}$$

Wir schätzen die Varianz σ^2 durch $\frac{1}{4}$ nach oben ab durch und erhalten:

$$\text{(2.24)}\qquad \mathrm{Ws}\left[\begin{array}{l} t\text{-te Mehrheitsent-} \\ \text{scheidung falsch} \end{array}\right] \leq \frac{4}{9\epsilon(n)^2 \cdot m_t}$$

Wegen

$$m_t = \begin{cases} 2^t\epsilon(n)^{-2} & \text{falls } t \leq 1 + \log_2 n \\ 2n\epsilon(n)^{-2} & \text{sonst} \end{cases}$$

ist die Mehrheitsententscheidung der Iterationen $t \leq 1 + \log_2 n$ jeweils mit Wahrscheinlichkeit maximal $\frac{4}{2^t 9}$ falsch, die der Iterationen $t > 1 + \log_2 n$ jeweils mit Wahrscheinlichkeit maximal $\frac{2}{9n}$. Bei richtiger Wahl von $\left(k_0^{(u)}, k^{(v)}, b_0^{(u)}, b^{(v)}\right)$ liefert der Algorithmus mit Wahrscheinlichkeit mindestens

$$1 - \sum_{t=1}^{1+\log_2 n} \frac{4}{2^t 9} - \sum_{t=\log_2 n+2}^{n} \frac{2}{9n} \geq 1 - \frac{4}{9}\sum_{t=1}^{\infty} \frac{1}{2^t} - \frac{2n}{9n} \geq 1 - \frac{4}{9} - \frac{2}{9} = \frac{1}{3}$$

das Urbild x. Analysieren wir die Laufzeit von Algorithmus 2.3.2. Die Anzahl der Aufrufe des Algorithmus' A in Schritt 3 ist:

$$\sum_{t=1}^{n} m_t = \sum_{t=1}^{n} \min\left\{2^t, 2n\epsilon(n)^{-2}\right\} \cdot \epsilon(n)^{-2} \leq 2n^2\epsilon(n)^{-2}$$

Bei jedem Aufruf berechnen wir eine RSA-Kodierung, was mit einem Divide-&-Conquer-Verfahren in Zeit $\mathcal{O}\left(n^3\right)$ gelingt. In Schritt 4 gibt es für $\left(k_0^{(u)}, k^{(v)}, b_0^{(u)}, b^{(v)}\right)$ insgesamt

$$4\epsilon(n)^{-1} \cdot 4\epsilon(n)^{-3} \cdot 2^2 = 2^6\epsilon(n)^{-4}$$

Möglichkeiten. Jede der n Iterationen von Schritt 4.2 gelingt in $n\epsilon(n)^{-2}$ Schritten, so daß insgesamt Algorithmus 2.3.2 in Zeit

$$2^6 n^2\epsilon(n)^{-6} + 2n^2\epsilon(n)^{-2}\left(|A| + \mathcal{O}\left(n^3\right)\right)$$

mit Wahrscheinlichkeit mindestens $\frac{1}{3}$ das Urbild x bestimmt. ■

Wir vergleichen dieses Resultat mit der Invertierung durch den binären ggT-Algorithmus in Kapitel 2.2. In Satz 2.2.10 auf Seite 73 haben das Urbild in Zeit

$$2^{19,2}n^3\epsilon(n)^{-6}\left(|A|+\mathcal{O}\left(n^3\right)\right)$$

mit Wahrscheinlichkeit etwa $\frac{1}{4}$ bestimmt. Die neue Methode, Algorithmus 2.3.2, ermittelt das Urbild in Zeit

$$2^6n^2\epsilon(n)^{-6}+2n^2\epsilon(n)^{-2}\left(|A|+\mathcal{O}\left(n^3\right)\right)$$

mit Wahrscheinlichkeit mindestens $\frac{1}{3}$. Insbesondere konnte die Anzahl der Aufrufe von Algorithmus A von $2^{19,2}n^3\epsilon(n)^{-6}$ auf $2n^2\epsilon(n)^{-2}$ reduziert werden. Wir senken in Abschnitt 2.3.2 diese um den Faktor $\frac{n}{\log_2 n}$, indem man statt aller Testpunkte nur eine Stichprobe auswertet. Die Laufzeit neben den Aufrufen von Algorithmus A reduzieren wir in Abschnitt 2.3.3, indem die Approximationen und untersten Bits nicht mehr getrennt, sondern gemeinsam behandelt werden.

Wir verallgemeinern Algorithmus 2.3.2 und Satz 2.3.4 auf die j-te Bitposition. In Schritt 4 wählen wir neben den Approximationen jetzt statt der untersten Bits von $[ux]_N$ und $[vx]_N$ jeweils die j untersten Bits:

$$\left(k_0^{(u)},k^{(v)},B_0^{(u)},B^{(v)}\right)\in\tfrac{1}{4}\epsilon(n)^3\left[0,4\epsilon(n)^{-3}\right)\times\tfrac{1}{4}\epsilon(n)\left[0,4\epsilon(n)^{-1}\right)\times\{0,1\}^j\times\{0,1\}^j$$

Die Mehrheitsentscheidung lautet

$$A\left(N,e,\mathrm{RSA}_{N,e}\left((a_t+r_i)x\right)\right)\equiv\left\lfloor\frac{i\cdot[ux]_N+[vx]_N-w_{t,i}N}{2^{j-1}}\right\rfloor\pmod 2,$$

wobei wir die Werte $A\left(N,e,\mathrm{RSA}_{N,e}\left((a_t+r_i)x\right)\right)$ zu Beginn berechnen und den Faktor $w_{t,i}$ mit Hilfe der Approximationen bestimmen. Die j untersten Bits von $i\cdot[ux]_N$ und $[vx]_N$ kennt man aus $B_0^{(u)}$ und $B^{(v)}$, so daß insgesamt gilt:

Satz 2.3.5
Sei $(N,e)\in\text{RSA-PK}_n$ ein RSA-Public-Key und $j:=j(n)\geq 1$. Sei A ein Algorithmus zur Bestimmung des j-ten Bits des Urbildes mit

$$\mathrm{Ws}\left[A\left(N,e,\mathrm{RSA}_{N,e}\left(U_{\mathbb{Z}_N^*}\right)\right)=\mathrm{bit}_{N,j}\left(U_{\mathbb{Z}_N^*}\right)\right]\geq\tfrac{1}{2}+\epsilon(n),$$

wobei die Wahrscheinlichkeit über die zufällige Nachricht $U_{\mathbb{Z}_N^*}$ und die internen Münzwürfe gebildet wird. Falls $\epsilon(n)^{-1}=n^{\mathcal{O}(1)}$ ist, bestimmt für hinreichend große N bzw. n der modifizierte und auf die j-te Bitposition verallgemeinerte Algorithmus 2.3.2 in Zeit

$$2^{4+2j}n^2\epsilon(n)^{-6}+2n\epsilon(n)^{-2}\left(|A|+\mathcal{O}\left(n^3\right)\right)$$

zu $\mathrm{RSA}_{N,e}(x)$ das Urbild x mit Wahrscheinlichkeit mindestens $\frac{1}{3}$, wobei die Wahrscheinlichkeit nur über die internen Münzwürfe gebildet wird.

Wir vergleichen dieses Resultat mit dem entsprechenden Satz aus Kapitel 2.2. Bei der Methode basierend auf dem binären ggT-Algorithmus bestimmt man in Zeit

$$2^{17,2+2j}n^3\epsilon(n)^{-6}\left(|A|+\mathcal{O}\left(n^3\right)\right)$$

mit Wahrscheinlichkeit etwa $2^{-2j}\epsilon(n)^2$, für große N bzw. n zu $\mathrm{RSA}_{N,e}(x)$ das Urbild x (Satz 2.2.14 auf Seite 77). Die Anzahl der Aufrufe von Algorithmus A konnten wir im neuen Verfahren reduzieren, da wir Algorithmus A unabhängig von den Approximationen und untersten Bits von $[ux]_N$ und $[vx]_N$ aufrufen.

Bei der simultanen Sicherheit nehmen wir an, daß Algorithmus A die j-te Bitposition bei gegebenen Bits vorhersagt, d.h. Algorithmus A erhält als Argumente nicht nur $\mathrm{RSA}_{N,e}(a_t + r_{t,i})$, sondern auch $\mathrm{Bits}_{N,j-1}(a_t + r_{t,i})$. Diese untersten Bits basieren auf den Approximationen und untersten Bits von $[ux]_N$ und $[vx]_N$. Wir können somit die Aufrufe von Algorithmus A nicht mehr zu Beginn getrennt von der Behandlung der möglichen Approximationen und untersten Bits ausführen. Wenn wir für jede Möglichkeit die entsprechenden Bits $\mathrm{Bits}_{N,j-1}(a_t + r_{t,i})$ bestimmen, ist die Laufzeit des Verfahrens nahezu identisch mit der Methode basierend auf dem binären ggT-Algorithmus, wenngleich die Erfolgswahrscheinlichkeit $\frac{1}{3}$ statt $2^{-2j}\epsilon(n)^2$ ist:

Korollar 2.3.6
Sei $(N, e) \in \mathrm{RSA\text{-}PK}_n$ ein RSA-Public-Key und $j := j(n) \geq 1$ mit $j(n) = \mathcal{O}(\log_2 n)$. Sei A ein Algorithmus zur Vorhersage des j-ten Bits des Urbildes mit

$$\mathrm{Ws}\left[A\left(N, e, \mathrm{RSA}_{N,e}\left(U_{\mathbb{Z}_N^*}\right), \mathrm{Bits}_{N,j-1}\left(U_{\mathbb{Z}_N^*}\right)\right) = \mathrm{bit}_{N,j}\left(U_{\mathbb{Z}_N^*}\right)\right] \geq \tfrac{1}{2} + \epsilon(n),$$

wobei die Wahrscheinlichkeit über die zufällige Nachricht $U_{\mathbb{Z}_N^*}$ und die internen Münzwürfe gebildet wird. Falls $\epsilon(n)^{-1} = n^{\mathcal{O}(1)}$ ist, kann man für hinreichend große N bzw. n in Zeit

$$2^{4+2j} n^2 \epsilon(n)^{-6} \cdot \left(|A| + \mathcal{O}\left(n^3\right)\right)$$

zu $\mathrm{RSA}_{N,e}(x)$ das Urbild x mit Wahrscheinlichkeit mindestens $\frac{1}{3}$ bestimmen, wobei die Wahrscheinlichkeit nur über die internen Münzwürfe gebildet wird.

Durch Anwenden des XOR-Lemmas aus Kapitel 1.5, (Satz 1.5.11 auf Seite 34) erreicht man die Laufzeit

$$2^{(6+6j} n^2 \epsilon(n)^{-6} + 2^{2j+2} n^2 \epsilon(n)^{-2} \left(|A| + \mathcal{O}\left(n^3\right)\right),$$

indem wir Algorithmus A in einen Prediktor einer nicht-leeren Teilmenge der j unteren Bits inklusive Bitposition j mit Vorteil $2^{-j}\epsilon(n)$ transformieren. Insbesondere erhält dieser Prediktor neben der RSA-Verschlüsselung keine zusätzlichen Informationen, so daß die Anzahl der Aufrufe unabhängig von den geratenen Approximationen und Bits ist.

2.3.2 Mehrheitsentscheidung durch Stichprobe

Unser Ziel ist, die Anzahl der Aufrufe des Algorithmus' A in Satz 2.3.4 um den Faktor $\frac{n}{\log_2 n}$ zu senken. Wir haben das Ereignis, daß die i-te Eintscheidung falsch ist, durch Indikatorvariablen $X_1, X_2, \ldots, X_m$ modelliert. Da diese nur paarweise unabhängig sind, kann für die Fehlerwahrscheinlichkeit der Mehrheitsentscheidung nur die Schranke aus Lemma 1.6.1 von Seite 39 angewenden werden, aber nicht die additive Form der stärkeren Chernoff-Schranke:

Fakt 2.3.7 (Additive Form der Chernoff-Schranke)
Seien $X_1, X_2, \ldots, X_m$ identisch verteilte, unabhängige Indikatorvariablen mit Erwartungswert $\mu := \mathrm{E}[X_i]$. Dann gilt für jedes $t > 0$:

$$\mathrm{Ws}\left[\sum_{i=1}^{m} X_i - m\mu \geq tm\right] \leq e^{-2mt^2}$$

Wir zeigen, daß es genügt, eine zufällige Stichprobe (Subsample) zu wählen und nur diese für die Mehrheitsentscheidung zu betrachten. Für die Stichprobe wählt man zufällig und unabhängig m'-viele Positionen

$$\mathrm{SProbe}(1), \mathrm{SProbe}(2), \ldots, \mathrm{SProbe}\left(m'\right)$$

aus der Menge $\left\{\pm 1, \pm 2, \ldots, \pm\frac{1}{2}m'\right\}$ und bilden die Mehrheitsentscheidung nur über diese Positionen. Insbesondere müssen wir Algorithmus A anstatt für alle m nur für die Werte an den m' Positionen der Stichprobe aufrufen. Diese Mehrheitsentscheidung über eine Stichprobe nennen wir *Subsample-Majority-Decision* (SMAJ):

Lemma 2.3.8 (Subsample-Majority-Decision)
Seien $X_1, X_2, \ldots, X_m$ identisch verteilte, paarweise unabhängige Indikatorvariablen für fehlerhafte Entscheidungen mit Erwartungswert $\mu \leq \frac{1}{2} - \frac{3}{4}\epsilon(n)$. Dann ist die Mehrheitsentscheidung durch eine zufällige Stichprobe der Größe m' höchstens mit Wahrscheinlichkeit

$$\frac{4}{m\epsilon(n)^2} + e^{-\frac{1}{2}m'\epsilon(n)^2}$$

falsch.

Beweis. Aus Lemma 1.6.1 auf Seite 39 folgt mit $t := \frac{1}{4}\epsilon(n)$

$$\mathrm{Ws}\left[\left|\sum_{i=1}^{m} X_i - \mu m\right| \geq \frac{m\epsilon(n)}{4}\right] \leq \frac{\sigma^2}{t^2 m} \leq \frac{4}{m\epsilon(n)^2}, \tag{2.25}$$

wobei wir die Varianz σ^2 von X_i nach oben durch $\frac{1}{4}$ beschränkt haben. Wir zeigen, daß unter der Bedingung

$$\sum_{i=1}^{m} X_i - \mu m \leq \tfrac{1}{4}m\epsilon(n) \tag{2.26}$$

die Mehrheitsentscheidung durch die zufällige Stichprobe der Größe m' mit Wahrscheinlichkeit höchstens $e^{-2m'\cdot\left(\frac{1}{2}\epsilon(n)\right)^2}$ falsch ist, d.h. es gilt:

$$\sum_{i=1}^{m'} X_{\mathrm{SProbe}(i)} \geq \tfrac{1}{2}m'$$

Angenommen, Ereignis (2.26) gilt. Weil wir die Positionen der Stichprobe zufällig wählen, folgt dann mit $\mu \leq \frac{1}{2} - \frac{3}{4}\epsilon(n)$:

$$\mu' := \mathrm{E}\left[X_{\mathrm{SProbe}(i)}\right] = \tfrac{1}{m}\sum_{i=1}^{m} \mathrm{E}[X_i] \leq \mu + \tfrac{1}{4}\epsilon(n) \leq \tfrac{1}{2} - \tfrac{1}{2}\epsilon(n)$$

Da die Positionen der Stichprobe unabhängig gewählt werden, können wir die additive Form der Chernoff-Schranke mit $t := \frac{1}{2}\epsilon(n)$ anwenden:

$$\begin{aligned}\mathrm{Ws}\left[\sum_{i=1}^{m'} X_{\mathrm{SProbe}(i)} \geq \tfrac{1}{2}m' \;\middle|\; (2.26)\right] &\leq \mathrm{Ws}\left[\sum_{i=1}^{m'} X_{\mathrm{SProbe}(i)} - \mu' \cdot m' \geq \tfrac{1}{2}\epsilon(n)m' \;\middle|\; (2.26)\right] \\ &\leq e^{-2m't^2} + \mathrm{Ws}[\,(2.26)] \\ &\leq e^{-2m' \cdot \left(\frac{1}{2}\epsilon(n)\right)^2} + \mathrm{Ws}[\,(2.26)]\end{aligned}$$

Da das Ereignis (2.26) nach Abschätzung (2.25) mit Wahrscheinlichkeit höchstens $\frac{4}{m\epsilon(n)^2}$ eintritt, können wir die Fehlerwahrscheinlichkeit der Subsample-Majority-Decision nach oben beschränken durch:

$$\frac{4}{m\epsilon(n)^2} + e^{-\frac{1}{2}m'\epsilon(n)^2}$$

Dies war zu zeigen. ■

Wir wenden die Technik der Mehrheitsentscheidung über eine Stichprobe auf Algorithmus 2.3.2 von Seite 87 an. Ab Iteration $t \geq 4 + \log_2 n$ verwenden wir die Subsample-Majority-Decision aus Lemma 2.3.8 über eine Stichprobe der Größe $m'_t = 2\epsilon(n)^{-2}\log_2 n$ über $m_t = 2^4\epsilon(n)^{-2}n$ Testpunkte. Wir wählen für die Iterationen ab $t \geq 4 + \log_2 n$ jeweils die gleichen Indizes als Stichprobe, da diese nur für jede Mehrheitsentscheidung unabhängig sein müssen. In Schritt 3 ersetze für $t \geq 4 + \log_2 n$ die Menge M_t durch die Stichprobe M'_t.

Satz 2.3.9
Sei $(N, e) \in \text{RSA-PK}_n$ ein RSA-Public-Key und $j := j(n)$. Sei A ein Algorithmus zur Bestimmung des j-ten Bit des Urbildes mit

$$\mathrm{Ws}\left[\, A\left(N, e, \mathrm{RSA}_{N,e}\left(U_{\mathbb{Z}_N^*}\right)\right) = \mathrm{bit}_{N,j}\left(U_{\mathbb{Z}_N^*}\right)\right] \geq \tfrac{1}{2} + \epsilon(n),$$

wobei die Wahrscheinlichkeit über die zufällige Nachricht $U_{\mathbb{Z}_N^*}$ und die internen Münzwürfe gebildet wird. Falls $\epsilon(n)^{-1} = n^{\mathcal{O}(1)}$ ist, bestimmt für hinreichend große N bzw. n der durch die Mehrheitsentscheidung über eine Stichprobe modifizierte und auf die j-te Bitposition verallgemeinerte Algorithmus 2.3.2 in Zeit

$$2^{6+2j}n\epsilon(n)^{-6}\log_2 n + 2^2\epsilon(n)^{-2}n\log_2 n \cdot \left(|A| + \mathcal{O}\left(n^3\right)\right)$$

zu $\mathrm{RSA}_{N,e}(x)$ das Urbild x mit Wahrscheinlichkeit mindestens $\frac{2}{9}$, wobei die Wahrscheinlichkeit nur über die internen Münzwürfe gebildet wird.

Beweis. Wir bilden die Mehrheitsentscheidung

a) in den Iterationen t mit $1 \leq t \leq 1 + \log_2 n$ über $m_t = 2^t\epsilon(n)^{-2}$ Testpunkte,

b) in den Iterationen t mit $2 + \log_2 n \leq t \leq 7 + \log_2 n$ über $m_t = 2n\epsilon(n)^{-2}$ Testpunkte und

c) ab Iteration $t \geq 4 + \log_2 n$ verwenden wir die Subsample-Majority-Decision über eine Stichprobe der Größe $m'_t = 2\epsilon(n)^{-2}\log_2 n$ über $m_t = 2^4\epsilon(n)^{-2}n$ Testpunkte.

Nach Lemma 2.3.8 ist für $n \geq 2^9 = 512$ die Stichprobe maximal mit Wahrscheinlichkeit

$$\frac{4}{m_t \epsilon(n)^2} + e^{-\frac{1}{2} m'_t \epsilon(n)^2} \leq \frac{4}{16n} + e^{2^3 \log_2 n} \leq \frac{1}{4n} + n^{-1{,}142} \leq \frac{1}{3n}$$

falsch. Bei richtiger Wahl von $\left(k_0^{(u)}, k^{(v)}, b_0^{(u)}, b^{(v)}\right)$ ist die Fehlerwahrscheinlichkeit maximal (vergleiche mit Abschätzung (2.24) auf Seite 88):

$$\sum_{t=1}^{1+\log_2 n} \frac{4}{2^t 9} + \sum_{t=2+\log_2 n}^{7+\log_2 n} \frac{1}{9n} + \sum_{t=3+\log_2 n}^{n} \frac{1}{3n} \leq \sum_{t=1}^{\infty} \frac{4}{2^t 9} + \frac{1}{3} \leq \frac{4}{9} + \frac{3}{9} = \frac{7}{9}$$

Der Algorithmus liefert für $n \geq 2^9$ das Urbild x mit Wahrscheinlichkeit mindestens $\frac{2}{9}$. Die Anzahl der Aufrufe von Algorithmus A ist maximal:

$$\begin{aligned}
\sum_{t=1}^{3+\log_2 n} m_t + \sum_{t=4+\log_2 n}^{n} m'_t &\leq \epsilon(n)^{-2} \left(\sum_{t=1}^{1+\log_2 n} 2^t + \sum_{t=2+\log_2 n}^{7+\log_2 n} 2n + \sum_{t=8+\log_2 n}^{n} 2 \log_2 n \right) \\
&\leq \epsilon(n)^{-2} \left(2^{2+\log_2 n} + 12n + 2n \log_2 n \right) \\
&\leq \epsilon(n)^{-2} \left(4n + 12n + 2n \log_2 n \right) \\
&\leq 2\epsilon(n)^{-2} \left(8n + n \log_2 n \right)
\end{aligned}$$

Für $n \geq 2^9$ ist die Anzahl der Aufrufe von Algorithmus A nach oben beschränkt durch:

$$\sum_{t=1}^{3+\log_2 n} m_t + \sum_{t=4+\log_2 n}^{n} m'_t \leq 4\epsilon(n)^{-2} n \log_2 n \tag{2.27}$$

Bei jedem Aufruf berechnen wir eine RSA-Kodierung, was mit einem Divide-&-Conquer-Verfahren in $\mathcal{O}\left(n^3\right)$ Schritten gelingt. Da es für

$$\left(k_0^{(u)}, k^{(v)}, B_0^{(u)}, B^{(v)}\right) \in \tfrac{1}{4}\epsilon(n)^3 \left[0, 4\epsilon(n)^{-3}\right) \times \tfrac{1}{4}\epsilon(n) \left[0, 4\epsilon(n)^{-1}\right) \times \{0,1\}^j \times \{0,1\}^j$$

$2^{4+2j}\epsilon(n)^{-4}$ Möglichkeiten gibt und nach Abschätzung (2.27) jeweils Zeit $2^2\epsilon(n)^{-2} n \log_2 n$ ausreicht, erhalten wir als Laufzeit:

$$2^{6+2j} n \epsilon(n)^{-6} \log_2 n + 2^2 \epsilon(n)^{-2} n \log_2 n \cdot \left(|A| + \mathcal{O}\left(n^3\right) \right)$$

Dies war zu zeigen. ■

Fassen wir unsere Resultate zusammen:

Korollar 2.3.10
Sei $j := j(n) \geq 1$ mit $j = \mathcal{O}(\log_2 n)$ und $\ell(n) \geq n+1$ durch ein Polynom beschränkt. Falls das vom RSA-Generator 2.1.1 auf Seite 58 mit Hardcore-Prädikat $\mathrm{bit}_{N,j}(x)$ erzeugte Ensemble des Typs $\ell(n)$ einen statistischen Test T mit nicht-vernachlässigbarer Toleranz $\delta(n)$ nicht besteht, kann man die $\mathrm{RSA}_{N,e}$-Funktion für unendlich viele n und zufälligen RSA-Public-Key $(N,e) \in_{\mathrm{R}} \text{RSA-PK}_n$ in Zeit

$$2^{12+2j} n \delta(n)^{-6} \ell(n)^6 \log_2 n + 2^4 \delta(n)^{-2} \ell(n)^2 n \log_2 n \cdot \left(|T| + \ell(n) \cdot \mathcal{O}\left(n^3\right) \right)$$

mit Wahrscheinlichkeit mindestens $\frac{2 \cdot \delta(n)}{9 \cdot \ell(n)}$ invertieren.

Beweis. Nach Lemma 1.5.4 auf Seite 25 erhalten wir einen Prediktor A zur Vorhersage des Prädikats $\mathrm{bit}_{N,1}(x)$ zu gegebenem $\mathrm{RSA}_{N,e}(x)$ mit Vorteil $\epsilon(n) := \frac{\delta(n)}{\ell(n)}$ und Laufzeit $|T| + \ell(n) \cdot \mathcal{O}\left(n^3\right)$. Wir betrachten nur die unendlich vielen n, für die A das j-te Bit mit Wahrscheinlichkeit mindestens $\epsilon(n)$ bestimmt. Aus Lemma 1.6.3 auf Seite 44 folgt mit $X = \text{RSA-PK}_n$ und $Y = \mathbb{Z}_N^*$, daß eine Teilmenge $G_n \subseteq \text{RSA-PK}_n$ existiert mit:

a) $\mathrm{Ws}[U_{\text{RSA-PK}_n} \in G_n] \geq \epsilon(n)$.

b) Für alle $(N,e) \in G_n$ gilt: $\mathrm{Ws}\left[A\left(N, e, \mathrm{RSA}_{N,e}\left(U_{\mathbb{Z}_N^*}\right)\right) = \mathrm{bit}_{N,1}\left(U_{\mathbb{Z}_N^*}\right)\right] \geq \frac{1}{2} + \frac{1}{2}\epsilon(n)$.

Wir wählen n hinreichend groß und betrachten nur die öffentlichen Schlüssel $(N,e) \in G_n$, für die der Prediktor A mit Wahrscheinlichkeit $\frac{1}{2}\epsilon(n)$ (unabhängig von N und e) das unterste Bit bestimmt. Aus Satz 2.3.9 folgt die Behauptung des Korollars, denn wir erhalten für $(N,e) \in G_n$ in Zeit

$$2^{6+2j} n \log_2 n \cdot 2^6 \epsilon(n)^{-6} + 2^2 \cdot 2^2 \epsilon(n)^{-2} n \log_2 n \cdot \left(|A| + \mathcal{O}\left(n^3\right)\right)$$

zu $\mathrm{RSA}_{N,e}(x)$ das Urbild x mit Wahrscheinlichkeit für große N bzw. n etwa $\frac{2}{9}\epsilon(n)$. ■

Wir vergleichen dieses Resultat mit dem entsprechenden Ergebnis bei der Invertierung mit Hilfe des binären ggT-Algorithmus' aus Kapitel 2.2. In Korollar 2.2.15 auf Seite 78 hat man das Urbild in Zeit

$$2^{23,2+2j} n^3 \delta(n)^{-6} \ell(n)^6 \left(|T| + \ell(n) \cdot \mathcal{O}\left(n^3\right)\right)$$

mit Wahrscheinlichkeit für große N bzw. n etwa $2^{-2(j+1)}\delta(n)^3\ell(n)^3$ bestimmt. Mit dem neuen Sicherheitsbeweis haben wir insbesondere die Anzahl der Aufrufe des Algorithmus' A deutlich reduzieren.

2.3.3 Simultane Behandlung aller Approximationen

Unser Ziel ist, die eigentliche Laufzeit von Algorithmus 2.3.2 von Seite 87 (d.h. ohne die Aufrufe von A) für das unterste Bit im Fall $\epsilon(n)^{-1} = \Omega(n)$ asymptotisch zu verbessern. Nach Satz 2.3.9 auf Seite 92 ist die eigentliche Laufzeit $2^8 n \epsilon(n)^{-6} \log_2 n$. Algorithmus 2.3.2 bearbeitet in Schritt 4 alle $2^4\epsilon(n)^{-4}$ möglichen $k_0^{(u)} \in \frac{1}{4}\epsilon(n)^3\left[0, 4\epsilon(n)^{-3}\right)$ und $k^{(v)} \in \frac{1}{4}\epsilon(n)\left[0, 4\epsilon(n)^{-1}\right)$ getrennt. Wir werden sehen, daß durch gemeinsame Behandlung aller Paare man als eigentliche Laufzeit $\mathcal{O}\left(n^2\epsilon(n)^{-4}\log_2(n\epsilon(n)^{-1})\right)$ erreicht.

Wir wollen in jeder Iteration t die Mehrheitsentscheidung über die gleiche Stichprobe $M_t' \subseteq \{\pm 1, \pm 2, \ldots, \pm n\epsilon(n)^{-2}\}$ der Größe $m_t' = 2\epsilon(n)^{-2}\log_2 n$ treffen. Um bereits zu Beginn $2\epsilon(n)^{-2}\log_2 n$ Testpunkte zu verwenden, bedarf es einer besserer Approximation $k_0^{(u)}$. Für die simultane Behandlung aller Approximationen erweist es sich als vorteilhaft, in jeder Stufe die gleiche Approximationsgüte zu verwenden, da dadurch die Abhängigkeit von der Iteration t aufgehoben wird. Vorab berechnen wir mit Algorithmus A unabhängig von den Approximationen $k_0^{(u)}$ und $k^{(v)}$:

$$b_{t,i} := A\left(N, e, \mathrm{RSA}_{N,e}\left((a_t + r_{t,i})x\right)\right) \quad \begin{array}{ll} \text{für} & i \in M_t' \\ \text{und} & t = 1, 2, \ldots, n \end{array}$$

Für die Mehrheitsentscheidung in Iteration t untersuchen wir, ob mehr als die Hälfte der $i \in M'_t$ die folgende Kongruenz erfüllen:

$$b_{t,i} \equiv i \cdot b_t^{(u)} + b^{(v)} + \underbrace{\left\lfloor k_t^{(u)} + ik_{t-1}^{(u)} + k^{(v)} \right\rfloor}_{=w_{t,i}} \pmod 2 \tag{2.28}$$

Wir wollen in (2.28) den Ausdruck innerhalb der $\lfloor\cdot\rfloor$-Klammern in die zwei Summanden $k_t^{(u)}+k^{(v)}$ und $ik_{t-1}^{(u)}$ aufspalten und getrennt auf den ganzzahligen Anteil abrunden. Allgemein gilt für $\alpha, \beta \geq 0$:

$$\lfloor \alpha + \beta \rfloor = \lfloor \alpha \rfloor + \lfloor \beta \rfloor + \underbrace{\Big[(\alpha \bmod 1) + (\beta \bmod 1) \geq 1\Big]}_{\text{Wert des logischen Ausdrucks}}$$

Man kann die Bedingung (2.28) getrennt für $k_t^{(u)} + k^{(v)}$ und $ik_{t-1}^{(u)}$ auswerten, wenn die Nachkommastellen zusätzlich vermerkt werden. Bei der Analyse der Fehlerwahrscheinlichkeit haben wir zugelassen, daß für $\overline{w}_{t,i} := k_t^{(u)} + ik_{t-1}^{(u)} + k^{(v)}$ nach Abschätzung (2.16) auf Seite 81 gilt

$$\Big| [a_t x]_N + i \cdot [ux]_N + [vx]_N - \overline{w}_{t,i} N \Big| \leq \tfrac{1}{4}\epsilon(n) N,$$

und $w_{t,i} := \lfloor \overline{w}_{t,i} \rfloor$ gesetzt. Unsere Analyse bleibt richtig, wenn man bei der Berechnung von $w_{t,i}$ die Summanden $k_t^{(u)} + k^{(v)}$ und $ik_{t-1}^{(u)}$ modulo 2 als Vielfache von $\frac{1}{4}\epsilon(n)$ nährungsweise darstellt. Die Nachkommastellen repräsentieren wir durch $s_1, s_2 \in \big[0, 4\epsilon(n)^{-1}\big)$:

$$\begin{aligned} s_1 &:= \Big\lfloor \big((k_t^{(u)} + k^{(v)}) \bmod 1 \big) 4\epsilon(n)^{-1} \Big\rfloor \\ s_2 &:= \Big\lfloor \big(ik_{t-1}^{(u)} \bmod 1 \big) 4\epsilon(n)^{-1} \Big\rfloor \end{aligned} \tag{2.29}$$

Sofern $s_1 + s_2 \not\equiv -1 \pmod{4\epsilon(n)^{-1}}$, können wir die Bedingung, daß die Summe der Nachkommastellen mindestens 1 ist, durch $\big[s_1 + s_2 \geq 4\epsilon(n)^{-1}\big]$ oder äquivalent durch $\left\lfloor \frac{s_1+s_2}{4\epsilon(n)^{-1}} \right\rfloor$ charakterisieren:

$$\Big\lfloor k_t^{(u)} + ik_{t-1}^{(u)} + k^{(v)} \Big\rfloor \equiv \Big\lfloor k_t^{(u)} + k^{(v)} \Big\rfloor + \Big\lfloor ik_{t-1}^{(u)} \Big\rfloor + \left\lfloor \frac{s_1 + s_2}{4\epsilon(n)^{-1}} \right\rfloor \pmod 2$$

Im Fall $s_1 + s_2 \equiv -1 \pmod{4\epsilon(n)^{-1}}$ kann es sein, daß die Summe der Nachkommastellen zwar mindestens 1 ist, durch das Abrunden von s_1 und s_2 auf das nächst kleinere Vielfache von $4\epsilon(n)^{-1}$ die Summe $s_1 + s_2$ jedoch kleiner als $4\epsilon(n)^{-1}$ ist. Diese Ausnahme schränkt uns aber nicht ein, da nach Bedingung (2.22) auf Seite 86 in diesem Fall auch das berechnete $w_{t,i}$ falsch sein kann. Für $s_1 + s_2 \not\equiv -1 \pmod{4\epsilon(n)^{-1}}$ lautet die Bedingung (2.28):

$$b_{t,i} \equiv (i \bmod 2) \cdot b_t^{(u)} + b^{(v)} + \Big\lfloor k_t^{(u)} + k^{(v)} \Big\rfloor + \Big\lfloor ik_{t-1}^{(u)} \Big\rfloor + \left\lfloor \frac{s_1 + s_2}{4\epsilon(n)^{-1}} \right\rfloor \pmod 2 \tag{2.30}$$

Für die gleichzeitige Behandlung ist es hinderlich, daß in jeder Iteration die Approximationen für u verbessert werden. Wir wollen zunächst in Bedingung (2.30) die Approximation $k_{t-1}^{(u)}$ durch einen Ausdruck in $k_t^{(u)}$ ersetzen. Aus der Identität $k_t^{(u)} = \frac{1}{2}\big(k_{t-1}^{(u)} + b_{t-1}\big)$ und $b_{t-1} \in \{0,1\}$ folgt

$$k_{t-1}^{(u)} \equiv 2k_t^{(u)} - b_{t-1} \equiv 2k_t^{(u)} \pmod 1,$$

so daß wegen $k_{t-1}^{(u)} \in [0,1)$ man aus $k_t^{(u)}$ den vorherigen Wert durch $2k_t^{(u)} \bmod 1$ erhält. Unser zentrales Ziel ist die Berechnung der Funktion

$$\Gamma\left(k^{(u)}, k^{(v)}, b^{(u)}, b^{(v)}, t\right) := \left| \left\{ i \in M_t' \;\middle|\; \begin{array}{l} i \text{ erfüllt die Gleichung (2.28) mit} \\ k_t^{(u)} := k^{(u)} \text{ und } k_{t-1}^{(u)} := 2 \cdot k_t^{(u)} \bmod 1 \end{array} \right\} \right|$$

für alle $k^{(u)}, k^{(v)}, b^{(u)}, b_0$ und t. Durch die Werte der Γ-Funktion können wir dann die Iterationen nachvollziehen.

Sei $K := \frac{1}{2^6 n}\epsilon(n)^3 \left[0, 2^6 n \epsilon(n)^{-3}\right)$. Bei der simultanen Behandlung aller Approximationen soll nicht mehr iterativ jeweils eine Approximation $k_0^{(u)}$ verwendet und sukzessive deren Verbesserung, sondern nur den aktuellen Wert für $k_t^{(u)}$ und den vorherigen Wert $k_{t-1}^{(u)}$, betrachtet werden. Dazu stellen wir die Approximation $k_t^{(u)}$ durch einen Wert aus K dar. Die Qualität der $w_{t,i}$ aus Abschätzung (2.21) auf Seite 86 bleibt wegen $\left|i + \frac{1}{2}\right| \leq 2 \cdot n\epsilon(n)^{-2}$ erhalten:

$$\left| [a_t x]_N + [r_{t,i} x]_N - \overline{w}_{t,i} N \right| \leq \frac{\left|i + \frac{1}{2}\right| \cdot \epsilon(n)^3}{2^6 n} N + \frac{\epsilon(n)}{8} N \leq \frac{\epsilon(n)}{4} N$$

Wir beschreiben im folgenden, wie man die Werte der Γ-Funktion effizient berechnen kann. Da auf jeder Stufe t die gleiche Approximationsgüte für $k_t^{(u)}$ verwendet wird, lassen wir im weiteren den Index weg und assozieren den Vorgängerwert $k_{t-1}^{(u)}$ stets mit

$$k_{\text{vor}}^{(u)} := 2 \cdot k^{(u)} \bmod 1,$$

wobei $k_{\text{vor}}^{(u)}$ in der Menge $K_{\text{vor}} = \frac{1}{2^5 n}\epsilon(n)^3 \left[0, 2^5 n \epsilon(n)^{-3}\right)$ liegt. Die Berechnung der Werte der Γ-Funktion unterteilen wir in drei Schritte:

1. Für alle $\nu \in \{0,1\}$, $c, s_2 \in \left[0, 4\epsilon(n)^{-1}\right)$, $k_{\text{vor}}^{(u)} \in K_{\text{vor}}$ und $t \in [1,n]$ berechne die Werte der Funktion:

$$\Psi_\nu\left(c, s_2, k_{\text{vor}}^{(u)}, t\right) := \left\{ i \in M_t' \;\middle|\; \begin{array}{ll} s_2 \equiv \left\lfloor (i \cdot k_{\text{vor}}^{(u)} \bmod 1) 4\epsilon(n)^{-1} \right\rfloor & (\text{mod } 4\epsilon(n)^{-1}) \\ c \equiv i & (\text{mod } 4\epsilon(n)^{-1}) \\ \nu = b_{t,i} & \end{array} \right\}$$

   Für festes $k_{\text{vor}}^{(u)}$ und t ordne die Indizes $i \in M_t'$ dem jeweils entsprechenden Tupel (ν, c, s) zu und zähle die Zuordnungen. Die Anzahl der Paare $(k_{\text{vor}}^{(u)}, t)$ ist $|K_{\text{vor}}| \cdot n = 2^5 n^2 \epsilon(n)^{-3}$. Wir erhalten unmittelbar ein Verfahren mit Laufzeit $m_t' \cdot 2^5 n^2 \epsilon(n)^{-3} = \mathcal{O}\left(n^2 \epsilon(n)^{-5} \log_2 n\right)$. Um die Berechnung zu beschleunigen, zerlegen wir $k_{\text{vor}}^{(u)}$ in:

$$k_{\text{vor}}^{(u)} = \bar{k}_{\text{vor}}^{(u)} + j\frac{\epsilon(n)}{4} \qquad \text{mit } \bar{k}_{\text{vor}}^{(u)} \in \tfrac{\epsilon(n)^3}{2^5 n}\left[0, 2^3 n \epsilon(n)^{-2}\right) \text{ und } j \in \left[0, 4\epsilon(n)^{-1}\right) \tag{2.31}$$

   Wegen $(j\frac{\epsilon(n)}{4} \bmod 1) 4\epsilon(n)^{-1} = j \bmod 4\epsilon(n)^{-1}$ und der Kongruenz $i \equiv c \pmod{4\epsilon(n)^{-1}}$ in der Definition der Ψ-Funktion verwendet man folgende Identität:

$$\Psi_\nu\left(c, s_2, \bar{k}_{\text{vor}}^{(u)} + j\tfrac{\epsilon(n)}{4}, t\right) = \Psi_\nu\left(c, (s_2 - jc) \bmod 4\epsilon(n)^{-1}, \bar{k}_{\text{vor}}^{(u)}, t\right)$$

   Wir betrachten nur die $2^3 n^2 \epsilon(n)^{-2}$ Paare $(k_{\text{vor}}^{(u)}, t)$ statt der $2^5 n^2 \epsilon(n)^{-3}$ Paare $(k_{\text{vor}}^{(u)}, t)$. Insgesamt kann man die Γ-Werte in Zeit $m_t' \cdot 2^3 n^2 \epsilon(n)^{-2} = \mathcal{O}\left(n^2 \epsilon(n)^{-4} \log_2 n\right)$ ermitteln.

2. Wir verwenden die Zerlegung aus (2.31). Für alle $s_2 \in \left[0, 4\epsilon(n)^{-1}\right)$, $\bar{k}^{(u)}_{\text{vor}} \in K_{\text{vor}}$, $j \in \left[0, 4\epsilon(n)^{-1}\right)$, $t \in [1, n]$ und $\sigma \in \{0, 1\}$ berechne die Werte der Funktion:

$$\Psi'_\nu\left(\sigma, s_2, \bar{k}^{(u)}_{\text{vor}} + j\tfrac{\epsilon(n)}{4}, t\right) := \sum_{c \equiv \sigma \pmod 2} \Psi_\nu\left(c, s_2, \bar{k}^{(u)}_{\text{vor}} + j\tfrac{\epsilon(n)}{4}, t\right)$$

Im Gegensatz zur Ψ-Funktion bezieht sich die Ψ'-Funktion auf die Summe aller i in der Restklasse σ modulo 2. Man überlegt sich leicht folgende Identität:

$$\Psi'_\nu\left(\sigma, s_2, \bar{k}^{(u)}_{\text{vor}} + j\tfrac{\epsilon(n)}{4}, t\right) = \sum_{c \equiv 0 \pmod 2} \Psi_\nu\left(c + \sigma, (s_2 - j\sigma - jc) \bmod 4\epsilon(n)^{-1}, \bar{k}^{(u)}_{\text{vor}}, t\right)$$

Wir beschleunigen die Berechnung durch zwei Beobachtungen:

- Für $j \equiv j' \pmod{2\epsilon(n)^{-1}}$ und $s_2 - j\sigma = s'_2 - j'\sigma$ gilt:

$$\Psi'_\nu\left(\sigma, s_2, \bar{k}^{(u)}_{\text{vor}} + j\tfrac{\epsilon(n)}{4}, t\right) = \Psi'_\nu\left(\sigma, s'_2, \bar{k}^{(u)}_{\text{vor}} + j'\tfrac{\epsilon(n)}{4}, t\right)$$

- Es genügt, die Ψ'_ν-Funktion nur für $\nu = 1$ auszuwerten, denn die Summe der Funktionswerte Ψ'_0 und Ψ'_1 hängt nur von n und $\epsilon(n)$, aber nicht von den Resultaten $b_{t,i}$ des Algorithmus' A ab.

Wir beschränken uns daher auf die Berechnung der insgesamt $2^7 n^2 \epsilon(n)^{-4}$ Funktionswerte $\Psi'_1\left(\sigma, s_2, \bar{k}^{(u)}_{\text{vor}} + j\tfrac{\epsilon(n)}{4}, t\right)$, d.h. ein Viertel der Funktionswerte. Unser Algorithmus zur Bestimmung dieser Werte entspricht einem Butterfly-Netzwerk. Für $c_2 \in [0, 2^e)$ tritt die Teilsumme

$$\sum_{c \equiv 0 \pmod{2^e}} \Psi_\nu\left(c + c_2, (s_2 - jc_2 - jc) \bmod 4\epsilon(n)^{-1}, \bar{k}^{(u)}_{\text{vor}}, t\right) \tag{2.32}$$

in den 2^e Summen $\Psi'_1(\sigma, s'_2, \bar{k}^{(u)}_{\text{vor}} + j'\tfrac{\epsilon(n)}{4}, t)$ mit

- $c_1 \equiv \sigma \pmod{2^e}$,
- $j \equiv j' \pmod{4\epsilon(n)^{-1}2^{-e}}$ und
- $s' = j'c_2 = s - jc_2$

auf. Wir kennen die Werte der Summe (2.32) für $e = 0$ aus Schritt 1 und berechnen nun sukzessive die Teilsummen (2.32) für $e = 1, 2, \ldots, \log_2(2\epsilon(n)^{-1})$. Dies gelingt insgesamt in $2^7 n^2 \epsilon(n)^{-4} \log_2(2\epsilon(n)^{-1})$ Schritten.

3. Für alle $k^{(u)} \in K$, $k^{(v)} \in \frac{\epsilon(n)}{4}\left[0, 4\epsilon(n)^{-1}\right)$, $b^{(u)}, b^{(v)} \in \{0, 1\}$ sowie $t \in [1, n]$ und s_1, s_2 wie in Definition (2.29) bestimme die Werte der Γ-Funktion gemäß:

$$\Gamma\left(k^{(u)}, k^{(v)}, b^{(u)}, b^{(v)}, t\right) = \sum_{(\nu,\sigma,s_2)} \Psi'_\nu\left(\sigma, s_2, \bar{k}^{(u)}_{\text{vor}}, t\right)$$
$$+ \left| \left\{ i \in M'_t \;\middle|\; \begin{array}{l} s_1 + \left\lfloor (i \cdot \bar{k}^{(u)}_{\text{vor}} \bmod 1) 4\epsilon(n)^{-1} \right\rfloor \equiv -1 \pmod{4\epsilon(n)^{-1}} \\ b_{t,i} \equiv i \cdot b^{(u)} + b^{(v)} - \left\lfloor k^{(u)} + i \cdot \bar{k}^{(u)}_{\text{vor}} + k^{(v)} \right\rfloor \pmod 2 \end{array} \right\} \right|,$$

wobei die Summe über alle Tripel (ν, σ, s) mit

- $s_1 + s_2 \not\equiv -1 \pmod{4\epsilon(n)^{-1}}$ und
- $\nu \equiv \sigma \cdot b^{(u)} + b^{(v)} + \left\lfloor \frac{s_1+s_2}{4\epsilon(n)^{-1}} \right\rfloor \pmod 2$

gebildet wird. Für festes $k^{(u)}$ und t können wir die erste Summe für alle $\sigma \in \{0,1\}$ und alle $s_1 \in [0, 4\epsilon(n)^{-1})$ in $2^3\epsilon(n)^{-1}$ Schritten berechnen. Der zweite Summand entspricht dem Fall $s_1 + s_2 \not\equiv -1 \pmod{4\epsilon(n)^{-1}}$ und da in dieser Situation das berechnete $w_{t,i}$ und somit die Mehrheitsenscheidung falsch sein kann, ersetzen wir die Bedingung

$$b_{t,i} \equiv i \cdot b^{(u)} + b^{(v)} - \left\lfloor k^{(u)} + i \cdot \bar{k}^{(u)}_{\text{vor}} + k^{(v)} \right\rfloor \pmod 2$$

durch einen Münzwürf. Wir bestimmen die Werte der Γ-Funktion in $\mathcal{O}\left(n^2\epsilon(n)^{-1}\right)$ Schritten.

Schritt 2 dominiert die Laufzeit, so daß insgesamt die Auswertung der Γ-Funktion in

$$\mathcal{O}\left(n^2\epsilon(n)^{-4}\log_2\left(n\epsilon(n)^{-1}\right)\right)$$

Schritten gelingt. Mit Hilfe der Werte der Γ-Funktion können wir die jeweils n Iterationen für alle Approximationen $k_0^{(u)} \in K$, $k^{(v)} \in \frac{\epsilon(n)}{4}\left[0, 4\epsilon(n)^{-1}\right)$ und untersten Bits $b^{(u)}, b^{(v)} \in \{0,1\}$ in $\mathcal{O}\left(n^2\epsilon(n)^{-4}\right)$ Schritten nachvollziehen. Nach Lemma 2.3.8 auf Seite 2.3.8 ist für $n \geq 512$ die Mehrheitsentscheidung über $m'_t = 2n\epsilon(n)^{-1}$ Testpunkte durch eine Stichprobe der Größe $m'_t = 2n\epsilon(n)^{-2}\log_2 n$ maximal mit Wahrscheinlichkeit

$$\frac{4}{m_t\epsilon(n)^2} + e^{-\frac{1}{2}m'_t\epsilon(n)^2} \leq \frac{4}{16n} + e^{-\frac{1}{2}\log_2 n} \leq \frac{1}{4n} + n^{-1{,}142} \leq \frac{1}{3n}$$

falsch. Die Erfolgswahrscheinlichkeit des gesamten Verfahrens ist mindestens $\frac{2}{3}$. Zu Beginn rufen wir Algorithmus A für $n \cdot 2\epsilon(n)^{-2}\log_2 n$ Werte auf.

Satz 2.3.11
Sei $(N,e) \in \text{RSA-PK}_n$ ein RSA-Public-Key. Sei A ein Algorithmus zur Bestimmung des untersten Bits des Urbildes mit

$$\text{Ws}\left[A\left(N, e, \text{RSA}_{N,e}\left(U_{\mathbb{Z}_N^*}\right)\right) = \text{bit}_{N,1}\left(U_{\mathbb{Z}_N^*}\right)\right] \geq \tfrac{1}{2} + \epsilon(n),$$

wobei die Wahrscheinlichkeit über die zufällige Nachricht $U_{\mathbb{Z}_N^*}$ und die internen Münzwürfe gebildet wird. Falls $\epsilon(n)^{-1} = n^{\mathcal{O}(1)}$ ist, kann für hinreichend große N bzw. n in Zeit

$$\mathcal{O}\left(n^2\epsilon(n)^{-4}\log_2\left(n\epsilon(n)^{-1}\right)\right) + 2\epsilon(n)^{-2}n\log_2 n \cdot \left(|A| + \mathcal{O}\left(n^3\right)\right)$$

zu $\text{RSA}_{N,e}(x)$ das Urbild x mit Wahrscheinlichkeit mindestens $\frac{2}{3}$ bestimmt werden, wobei die Wahrscheinlichkeit nur über die internen Münzwürfe gebildet wird.

Korollar 2.3.10 auf Seite 93 gilt ensprechend:

Korollar 2.3.12
Sei $\ell(n) \geq n+1$ durch ein Polynom beschränkt. Falls das vom RSA-Generator 2.1.1 auf Seite 58 erzeugte Ensemble des Typs $\ell(n)$ einen statistischen Test T mit nicht-vernachlässigbarer Toleranz $\delta(n)$ nicht besteht, kann man die $\mathrm{RSA}_{N,e}$-Funktion für unendlich viele n und zufälligen RSA-Public-Key $(N,e) \in_{\mathrm{R}} \text{RSA-PK}_n$ in Zeit

$$\mathcal{O}\left(n^2\ell(n)^4\delta(n)^{-4}\log_2\left(n\ell(n)\delta(n)^{-1}\right)\right) + 2^3\delta(n)^{-2}\ell(n)^2 n\log_2 n \cdot \left(|T| + \ell(n)\cdot\mathcal{O}\left(n^3\right)\right)$$

mit Wahrscheinlichkeit mindestens $\frac{2\cdot\delta(n)}{3\cdot\ell(n)}$ invertieren.

Nach einer Beobachtung von O. Goldreich [FSch97] ist die Anzahl der Aufrufe von Algorithmus A asymptotisch bis auf den Faktor $\log_2 n$ optimal.

2.3.4 Ausblick

Wir geben einen kurzen Ausblick auf Möglichkeiten, wie man die beweisbare Sicherheit des RSA-Generators noch verbessern könnte. Die Anzahl der Aufrufe des Algorithmus' A zur Vorhersage des untersten Bits des Urbildes der $\mathrm{RSA}_{N,e}$-Funktion ist bis auf einen Faktor $\log_2 n$ optimal, so daß wir uns auf Einsparungen bei der zusätzlichen Laufzeit konzentrieren.

Durch die simultane Behandlung aller Approximationen konnten die zusätzliche Laufzeit reduzieren werden. Diese sollte weiterhin asymptotisch reduzierbar sein, wenn man statt die Iterationen für alle Approximationen anhand der Werte der Γ-Funktion nachvollzieht, iterativ jeweils (mindestens) die Hälfte der Approximationen als falsch erkennt und aussondert. Nach $\mathcal{O}\left(\log_2 \epsilon(n)^{-1}\right)$ Schritten kennen wir die richtige Approximation. Für die richtigen Approximationen (und untersten Bits) gilt die Kongruenz

$$b_i \equiv i\cdot b^{(u)} + b^{(v)} + \left\lfloor k^{(u)} + ik^{(u)}_{\mathrm{vor}} + k^{(v)} \right\rfloor \pmod 2 \tag{2.33}$$

mit Wahrscheinlichkeit $\frac{1}{2} + \epsilon(n)$ (den exakten $\epsilon(n)$-Wert kann gegebenenfalls zu Beginn mit polynomiell vielen Versuchen bis auf eine exponentiell kleine Abweichung bestimmt werden). Nehmen wir zur Vereinfachung an, Algorithmus A haben für Zahlen kleiner als $\frac{1}{2}N$ die gleiche Erfolgswahrscheinlichkeit wir für die Werte überhalb von $\frac{1}{2}N$. Da $a_t + r_{t,i}$ modulo N gleichverteilt ist, wissen wir, daß

$$k := k^{(u)} + ik^{(u)}_{\mathrm{vor}} + k^{(v)} \bmod 1$$

gleichverteilt im Intervall $[0,1)$ ist. Falls zum Beispiel für korrekte Approximationen $k^{(u)}$ und $k^{(u)}_{\mathrm{vor}}$ der Wert $k^{(v)}$ um $\frac{1}{2}$ von der korrekten Approximation abweicht, gilt die Kongruenz (2.33) nur mit Wahrscheinlichkeit $\frac{1}{2} + \frac{1}{2}\epsilon(n)$, denn genau für alle $k \in \left[0, \frac{1}{2}\right)$ haben wir den Wert der rechten Seite der Kongruenz (2.33) invertiert.

Die Möglichkeit, falsche Approximationen zu erkennen, hat insbesondere Konsequenzen für die beweisbare, simultane Sicherheit, da man in diesem Fall für jede Approximation den Algorithmus A aufruft und wir so die Anzahl der Aufrufe senken können.

2.4 Wahl der Parameter

... in general nothing but frustration
can be expected to come from an attack
on a number of 25 or more digits.

— John Brillhart und John Selfridge (1967)

Wir untersuchen, wie der RSA-Modul gewählt werden soll, damit man mit den bekannten Angriffen, insbesondere Faktorisierungsalgorithmen, das RSA-Schema nicht brechen kann. Wir folgern aus diesen Resultaten, wie sich die Wahl der Parameter auf die exakte Sicherheit des RSA-Generators auswirkt.

2.4.1 Brechen des RSA-Systems durch Faktorisieren des Moduls

Für den RSA-Generator müssen wir den öffentlichen Schlüssel $(N, e) \in \text{RSA-PK}_n$ und den Startwert $x \in \mathbb{Z}_N^*$ so wählen, daß das RSA-Kryptosystem nicht effizient gebrochen werden kann. B. Kaliski und M. Robshaw [KR95] haben alle bekannten Angriffe auf das RSA-System analysiert und kommen zu dem Schluß, das für zufällige RSA-Public-Keys $(N, e) \in_{\text{R}} \text{RSA-PK}_n$ das System sicher ist, wenn der Modul N bzw. dessen Bitlänge n zu groß zum Faktorisieren ist. Die Autoren schließen bei zufälliger Wahl des öffentlichen Kodierexponten e auch M. Wieners Angriff [Wiener90], mit Hilfe einer Kettenbruch-Approximation den geheimen Schlüssel d ermittelt, aus.

B. Kaliski und M. Robshaw [KR95] sowie M.Blum [Blum81] wählen einen Modul N, der aus etwa zwei gleich großen Primfaktoren p und q bestehen. D.E. Knuth [Knuth96] und M. Wiener [Wiener90] schlagen vor, daß der Abstand zwischen beiden Primzahlen asymptotisch größer als $\sqrt{p} \cdot (\log_2 p)^{\mathcal{O}(1)}$ ist, da der Modul sonst mit einem Siebalgorithmus basierend auf einer Idee von Fermat [Knuth81, Algorithmus 4.5.4D] effizient faktorisiert werden kann. Fermats Ansatz ist, N als Differenz zweier Quadratzahlen darstellen:

$$N = a^2 - b^2 = (a - b)(a + b)$$

Wir beginnen mit $a := \lfloor \sqrt{N} \rfloor$, testen, ob $a^2 - N$ eine Quadratzahl ist und falls nicht, inkrementieren wir a. Die Laufzeit ist etwa $\frac{p+q}{2} - \sqrt{N}$ (Mit einem Siebverfahren kann man die Schrittweite erhöhen und somit die Laufzeit senken). Für die Module $N \in \mathcal{R}_n$ ist wegen $\sqrt{p} \leq 2^{\frac{n-1}{4}}$ die Differenz zwischen beiden Primfaktoren hinreichend groß:

$$q - p \geq 2^{\frac{n}{2}} - 2^{\frac{n-2}{2}} = 2^{\frac{n-2}{2}}$$

Nach R.D Silverman [Sil97] gewährt hingegen die Anforderung bezüglich des Abstands zwischen beiden Primzahlen keine zusätzliche Sicherheit, denn für einen beliebigen Startwert a ist allgemein die Laufzeit $\frac{p+q}{2} - a$.

Die Forschung nach schnellen Verfahren in der (algorithmischen) Zahlentheorie wurde in den siebziger und achziger Jahren intensiviert, da diese Resultate u.a. Anwendung in Kryptographie haben. Einer der ersten, modernen Faktorisierungsalgorithms, stammt von C. Pomerance [Pom84]. Der *Quadratic-Sieve-Algorithmus* hat die Laufzeit $L_N\left[\frac{1}{2}, 1\right]$ mit:

$$L_N[v, a] := \exp\left((a + \mathrm{o}(1)) \cdot (\ln N)^v \cdot (\ln \ln N)^{1-v}\right)$$

H.W. Lenstra [Lenstra87] hat die *Ellipic-Curve-Methode* (ECM) vorgestellt, die elliptischen Kurven zum Faktorisieren verwenden. Unter heuristischen Annahmen ist die erwartete Laufzeit zum Faktorisieren eines RSA-Moduls $N = pq$ mit etwa gleichen großen Primfaktoren p und q:

$$\exp\left((2 + o(1))\sqrt{(\ln p)(\ln \ln p)}\right)$$

Die Erfolgswahrscheinlichkeit der Ellipic-Curve-Methode hängt nicht ab von der Form der Primfaktoren. Im Hinblick auf spezielle Faktorisierungsalgorithmen, die voraussetzen, daß die Primfaktoren p und q bestimmte Formen haben (z.B. $p-1$ hat nur kleine Primfaktoren), halten B. Kaliski und M. Robshaw [KR95] bei zufälliger Wahl des Moduls die Form der Primfaktoren im Vergleich zur Größe für von untergeordneter Bedeutung. U. Maurer [Maurer95] äußert sich ähnlich, fordert aber mit Hinweis auf *Super-Encryption* (d.h. wiederholtes verschlüsseln, bis der Ausgangswert erreicht wird), daß $p - 1$ und $q - 1$ keine kleinen Primfaktoren enthalten sollen. B. Kaliski und M. Robshaw [KR95] halten jedoch diesen Angriff für große, zufällige Primfaktoren für nicht praktikabel.

Der schnellste, bekannte Faktorisierungsalgorithmus für große Zahlen ist das *General-Number-Field-Sieve* (GNFS), das J. Pollard in Zusammenarbeit mit A.K. Lenstra, H.W. Lenstra und M.S. Manasse entwickelt hat. Für mehrere Arbeiten über das Number-Field-Sieve verweisen wir auf das Buch [LenLen93]. Unter heuristischen Annahmen ist die erwartete Laufzeit $L_N\left[\frac{1}{3}, c_0\right]$ mit $c_0 \approx 1,9229$. Für eine (allerdings nicht praktikable) Variante von D. Coppersmith kann die Konstante auf $c_1 \approx 1,9018$ gesenkt werden. Nach A.K. Lenstra ist das General-Number-Field-Sieve für $N \geq 2^{372}$ dem optimierten Quadratic-Sieve-Algorithmus überlegen. Für Zahlen der Form $N = a^k \pm b$ gibt es eine spezielle Version des Number-Field-Sieve-Algorithmus: *Special-Number-Field-Sieve* (SNFS). Im Fall, daß a, b klein und k groß sind, ist die erwartete Laufzeit des Special-Number-Field-Sieve $L_N\left[\frac{1}{2}, c_2\right]$ mit $c_2 \approx 1,5262$. Für eine Übersicht über weitere Faktorisierungsalgorithmen verweisen wir auf die Übersichtsarbeit [LenLen90] von A.K. Lenstra und H.W. Lenstra.

Zur Betrachtung der Rechenleistung, um eine Zahl mit dem Number-Field-Sieve zu faktorisieren, verwenden wir als Einheit MIPS-Jahre (MJ). Ein MIPS-Jahr ist die Rechenkraft eines Computer, der pro Sekunde eine Million Operationen ausführt, über den Zeitraum eines Jahres (dies entspricht einer VAX 11/780 der Firma DEC). Ein MIPS-Jahr sind etwa $3,16 \cdot 10^{13}$ Rechenschritte. Den Aufwand zum Faktorisieren mit dem Number-Field-Sieve gibt A.M. Odlyzko [Odlyzko95] wie folgt an:

Faktorisieren mit Number-Field-Sieve		
Bitlänge N	GNFS (in MJ)	SNFS (in MJ)
512	$3 \cdot 10^4$	$3 \cdot 10^5$
768	$2 \cdot 10^8$	$3 \cdot 10^7$
1024	$1 \cdot 10^{14}$	$3 \cdot 10^9$
1536	$3 \cdot 10^{16}$	$2 \cdot 10^{11}$
2048	$3 \cdot 10^{20}$	$4 \cdot 10^{14}$

A.M. Odlyzko [Odlyzko95] schätzt die im Jahr 2004 weltweit zur Faktorisierung eines RSA-Moduls stehende Rechnerleistung auf $2 \cdot 10^9$ MIPS-Jahre, wenn man annimmt, daß die Anzahl der Rechner und ihre Leistung wie in der Vergangenheit zunimmt. Für das Jahr 2014 prognostiziert er zwischen 10^{11} und 10^{13} MIPS-Jahre.

2.4.2 RSA-Generator in der Praxis

Wir haben in Korollar 2.3.10 auf Seite 93 gesehen, daß wenn das vom RSA-Generator 2.1.1 auf Seite 58 mit zufälligem $(N, e) \in_{\mathrm{R}} \text{RSA-PK}_n$ und $x \in_{\mathrm{R}} \mathbb{Z}_N^*$ erzeugte Ensemble des Typs $\ell(n)$ einen statistischen Test T mit Toleranz $\delta(n)$ nicht besteht, können wir für unendlich viele n die RSA-Funktion in Zeit

$$2^{14} n \delta(n)^{-6} \ell(n)^6 \log_2 n + 2^4 \delta(n)^{-2} \ell(n)^2 n \log_2 n \cdot \left(|A| + \ell(n) \cdot \mathcal{O}\left(n^3\right)\right)$$

mit Wahrscheinlichkeit mindestens $\frac{2 \cdot \delta(n)}{9 \cdot \ell(n)}$ invertieren. Wenn wir als Toleranz weniger ein Prozent, also $\delta(n) := 2^{-7}$, und als Modullänge $n := 2^{11} = 2048$ wählen, ist die Laufzeit zum Invertieren etwa:

$$2^{31{,}5} \ell^6 + 2^{32{,}5} \ell^2 \cdot |A| + 2^{65{,}5} \ell^3$$

Die Laufzeit zum Invertieren mit Hilfe eines statistischen Tests T mit Laufzeit $|T| \leq 2^{40}$ ist etwa maximal:

$$2^{31{,}5} \ell^6 + 2^{72{,}5} \ell^2 + 2^{65{,}5} \ell^3$$

Wie groß können wir die Ausgabelänge ℓ wählen, d.h. wieviele pseudozufällige Bits können wir erzeugen, wenn das General-Number-Field-Sieve der beste Faktorisierungsalgorithmus mit nicht vernachlässigbarer Erfolgswahrscheinlichkeit ist? Die Laufzeit des GNFS-Algorithmus' ist $3 \cdot 10^{20} \approx 2^{68}$ MIPS-Jahre, was zirka 2^{113} Rechenschritten entspricht. Wir können etwa bis zu eine Million, also $\ell \approx 2^{20}$, pseudozufälliger Bits erzeugen.

Wenn wir als obere Schranke für die Laufzeit des statistischen Tests T die von A.M. Odlyzko [Odlyzko95] für das Jahr 2004 prognostizierte, zur Verfügung stehende Rechenleistung von $2 \cdot 10^9 \approx 2^{31}$ MIPS-Jahren verwenden, ist die Laufzeit zum Invertieren des RSA-Funktion mit Hilfe eines statistischen Tests größer als die zum Faktorisieren des Moduls. Statistische Tests mit dieser Laufzeitschranke liefern für Bitlänge $n = 5000$ einen schnelleren Faktorisierungsalgorithmus als das General-Number-Field-Sieve.

Kapitel 3

x^2-mod-N-Generator

Wir stellen den bekannten x^2-mod-N-Pseudozufallsgenerator sowie die Variante des Muddle-Square-Generators vor. Wir skizzieren den Sicherheitsbeweis [VV84, ACGS88] und leiten den neuen Beweis aus [FSch97] her.

3.1 Rabin-Funktion und x^2-mod-N-Generator

L. Blum, M. Blum und M. Shub [BBS86] haben 1982 auf der Crypto-Konferenz einen einfachen Pseudozufallsgenerator vorgestellt, der einen Wert x modulo einer zusammengesetzen n-Bit-Zahl N quadriert und neben $x^2 \bmod N$ zusätzlich das unterste Bit von x ausgibt. Diesen Pseudozufallsgenerator nennt man x^2-mod-N- oder nach den drei Autoren auch BBS-Generator. Im Vergleich zur RSA-Funktion ist das Quadrieren modulo N in $\mathcal{O}\left(n^2\right)$ statt in $\mathcal{O}\left(n^3\right)$ Schritten möglich und Wurzelziehen ist äquivalent zum Faktorisieren des Moduls.

Der Generator basiert auf einem Public-Key-Kryptosystem von M.O. Rabin [Rabin79]. Der Teilnehmer wählt ähnlich wie beim RSA-System zwei verschiedene Primzahlen p, q und verwendet $N = pq$ als seinen öffentlichen sowie das Paar (p, q) als geheimen Schlüssel. Wir beschränken uns auf den Fall $p \equiv q \equiv 3 \pmod 4$, d.h. zu gegebenem Sicherheitsparameter $n > 3$ wählt der Teilnehmer einen zufälligen Modul N aus der Menge

$$\mathcal{B}_n := \left\{ pq \;\middle|\; \begin{array}{l} p, q \text{ sind verschiedene Primzahlen} \\ \text{mit } p \equiv q \equiv 3 \pmod 4 \text{ sowie} \\ 2^{\frac{n-2}{2}} < p < 2^{\frac{n-1}{2}} \text{ und } 2^{\frac{n}{2}} < q < 2^{\frac{n+1}{2}} \end{array} \right\} \subseteq \mathcal{R}_n$$

mit Bitlänge n. Diese Zahlen nennt man *Blumzahlen* nach M. Blum [Blum81], der auf eine wichtige Eigenschaft für kryptographische Anwendungen hingewiesen hat (Siehe Satz 3.1.5 auf Seite 106). Nach De-la-Vallée-Poisson gilt für die Anzahl $\pi_m(x)$ der Primzahlen $p \leq x$ in einer Restklasse von $\mathbb{Z}_m^*$ (unabhängig von der Restklasse):

$$\pi_m(x) \sim \frac{1}{\varphi(m)} \cdot \frac{x}{\ln x}$$

Da 3 teilerfremd zu $m := 4$ ist, folgt wegen $\varphi(4) = 2$ in Verbindung mit dem Primzahlsatz $\pi(x) \sim \frac{x}{\ln x}$, daß für große Zahlen etwa die Hälfte aller Primzahlen kongruent 3 modulo 4 sind.

Die Gruppe $\mathbb{Z}_N^*$ bildet den Nachrichtenraum und die Menge der verschlüsselten Nachrichten des Rabin-System. Um eine Nachricht $x \in \mathbb{Z}_N^*$ bezüglich des öffentlichen Schlüssels N zu chiffrieren, wenden wir die Funktion $\text{Rabin}_N : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ mit

$$\text{Rabin}_N(x) := x^2 \bmod N \tag{3.1}$$

an. Um eine verschlüsselte Nachricht $\text{Rabin}_N(x)$ zu dechiffrieren, berechnet der rechtmäßige Empfänger die je zwei Wurzeln modulo p und q, um anschließend die Lösungen mit Hilfe des Chinesischen Restsatzes $\mathbb{Z}_N^* \cong \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ zusammenzusetzen. Der Empfänger wählt aus den vier Wurzeln modulo N die Nachricht (zum Beispiel mit Hilfe redundanter Informationen in x) aus. Man überlege sich, daß wie bei der $\text{RSA}_{N,e}$-Funktion für alle $x, y \in \mathbb{Z}_N^*$ gilt

$$\text{Rabin}_N(xy) \equiv \text{Rabin}_N(x) \cdot \text{Rabin}_N(y) \pmod{N},$$

so daß $\text{Rabin}_N : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ ein Gruppen-Homomorphismus ist. Diese Kongruenz gilt auch für Werte aus $\mathbb{Z}_N$.

Die verschlüsselte Nachricht hat die Form x^2 modulo N. Reste dieses Typs, d.h. die modulo N als Quadrate von Zahlen aus $\mathbb{Z}_N^*$ darstellbar sind, nennt man quadratische Reste. Da 1 ein quadratischer Rest sowie mit a^2 und b^2 auch $(ab^{-1})^2$ ein quadratischer Rest ist, bilden diese eine Untergruppe von $\mathbb{Z}_N^*$.

Definition 3.1.1 (Quadratische Reste QR_N)
Sei $N \in \mathbb{N}$ ein beliebiger Modul. Die Menge der quadratischen Reste modulo N

$$\text{QR}_N := \left\{ x^2 \bmod N \;\middle|\; x \in \mathbb{Z}_N^* \right\}$$

bilden eine Untergruppe von $\mathbb{Z}_N^*$.

Falls ein $y \in \mathbb{Z}_N^*$ keine Darstellung als Quadrat modulo N hat, nennt man es einen quadratischen Nichtrest modulo N. Wir bezeichnen mit $\text{NQR}_N := \mathbb{Z}_N \setminus \text{QR}_N$ die Menge der quadratischen Nichtreste modulo N, die allerdings keine Untergruppe von $\mathbb{Z}_N^*$ bildet.

Wir wollen untersuchen, wie man zu $y \in \text{QR}_N$ eine Wurzel, d.h. ein $x \in \mathbb{Z}_N^*$ mit $x^2 \equiv y \pmod{N}$ bestimmen kann. Im Fall eines Primzahlmoduls p mit $p \equiv 3 \pmod 4$ erhalten wir eine Wurzel modulo p von $y \in \text{QR}_p$ durch Erheben von y in die $\frac{p+1}{4}$-te Potenz:

Lemma 3.1.2
Sei p eine Primzahl mit $p \equiv 3 \pmod 4$. Dann gilt:

a) Der quadratische Rest $y \in \text{QR}_p$ hat genau die beiden Wurzeln $x_{1,2} := \pm y^{\frac{p+1}{4}}$ modulo p.

b) QR_p ist eine Untergruppe von $\mathbb{Z}_p^*$ mit Index 2, d.h. modulo p gibt es genau $\frac{p-1}{2}$ quadratische Reste.

Beweis. Wir beweisen zunächst Aussage a) und folgern dann Aussage b). Wegen $y \in \text{QR}_p$ existiert ein $z \in \mathbb{Z}_N^*$ mit $y \equiv z^2 \pmod p$. Aus dem kleinen Satz von Fermat folgt $z^{p-1} \equiv 1 \pmod p$, so daß $y^{\frac{p-1}{2}} \equiv 1 \pmod p$ ist. Wegen

$$(x_{1,2})^2 \equiv \left(\pm y^{\frac{p+1}{4}}\right)^2 \equiv y^{\frac{p+1}{2}} \equiv y \cdot y^{\frac{p-1}{2}} \equiv y \bmod p$$

sind $x_{1,2}$ Wurzeln modulo p von y. Wir müssen zeigen, daß es keine weiteren gibt. Sei x eine beliebige Wurzeln modulo p von y. Da es in $\mathbb{Z}_p^*$ keine Nullteiler gibt, folgt aus

$$(x_{1,2} - x) \cdot (x_{1,2} + x) \equiv x_{1,2}^2 - x^2 \equiv 0 \pmod{p},$$

daß $x = x_1$ oder $x = x_2$ ist. Da modulo p Quadrieren eine 2:1-Abbildung ist, gibt es wegen $\left|\mathbb{Z}_p^*\right| = p - 1$ genau $\frac{p-1}{2}$ quadratische Reste modulo p. ■

Die Sicherheit des Rabin-Systems basiert auf der Annahme, daß für zufälligen Public-Key bzw. zufällige Blumzahl $N \in_{\mathrm{R}} \mathcal{B}_n$ und zufällige Nachricht $x \in_{\mathrm{R}} \mathbb{Z}_N^*$ die Abbildung

$$\text{Rabin}: \quad (N, x) \mapsto (N, \text{Rabin}_N(x)) \tag{3.2}$$

eine Oneway-Funktion ist. Mit Kenntnis der beiden Primfaktoren p und q der Blumzahl N können wir die Rabin_N-Funktion invertieren: Wir berechnen mit dem Chinesischen Restsatz zu $\text{Rabin}_N(x)$ das Urbild $x \in \text{QR}_N$ mit:

$$x \equiv \text{Rabin}_N(x)^{\frac{p+1}{4}} \pmod{p}$$
$$x \equiv \text{Rabin}_N(x)^{\frac{q+1}{4}} \pmod{q}$$

Umgekehrt ist die Blumzahl N effizient zu faktorisieren, wenn wir die Rabin_N-Funktion invertieren können:

Satz 3.1.3
Sei A ein Algorithmus, der zu $N \in \mathcal{B}_n$ und $\text{Rabin}_N(x)$ für zufälliges $x \in_{\mathrm{R}} \mathbb{Z}_N^*$ mit Wahrscheinlichkeit $\epsilon(n)$ ein Urbild von $\text{Rabin}_N(x)$ ausgibt. Dann können wir die Blumzahl N mit Wahrscheinlichkeit $\frac{1}{2}\epsilon(n)$ in Zeit $|A| + \mathcal{O}\left(n^3\right)$ faktorisieren.

Beweis. Wir wollen mit dem Algorithmus zur Berechnung des Urbildes von $\text{Rabin}_N(x)$ den Modul $N \in \mathcal{B}_n$ faktorisieren. Seien p und q die beiden (unbekannten) Primfaktoren von N. Wir wählen ein zufälliges $x \in_{\mathrm{R}} \mathbb{Z}_N^*$. Nach Lemma 3.1.2.a) und wegen des Chinesischen Restsatzes $\mathbb{Z}_N^* \cong \mathbb{Z}_p^* \times \mathbb{Z}_q^*$ sind die vier Wurzeln von x^2 modulo N:

$$x_1 := (+x \bmod p, +x \bmod q)$$
$$x_2 := (-x \bmod p, -x \bmod q) \equiv -x_1 \pmod{N}$$
$$x_3 := (-x \bmod p, +x \bmod q)$$
$$x_4 := (+x \bmod p, -x \bmod q) \equiv -x_3 \pmod{N}$$

Wir rufen den Algorithmus mit N und $\text{Rabin}_N(x) = x^2 \bmod N$ auf, sei y das Resultat. Da x eine zufällige Wurzeln modulo N von x^2 ist, gilt mit Wahrscheinlichkeit $\frac{1}{2}$:

$$y \not\equiv \pm x \pmod{N}$$

Insbesondere ist dann N kein Teiler von $y \pm x$, so daß wegen

$$(y + x)(y - x) \equiv y^2 - x^2 \equiv 0 \pmod{N}$$

p oder q einen der beiden Faktoren $(y + x)$ oder $(y - x)$ teilt. Wir berechnen $\text{ggT}(y + x, N)$ sowie $\text{ggT}(x - y, N)$ und erhalten, wenn der Algorithmus eine korrekte Wurzel modulo N von x^2 liefert, mit Wahrscheinlichkeit $\frac{1}{2}$ einen der Primfaktoren, so daß man die Faktorisierung von N kennt. Da für den Euklidischen Algorithmus $\mathcal{O}\left(n^3\right)$ Bitoperationen ausreichen (siehe u.a. [Knuth81]), erhalten wir als Laufzeit $|A| + \mathcal{O}\left(n^3\right)$. ■

Das Faktorisieren einer Blumzahl und das Wurzelziehen modulo einer Blumzahl sind äquivalent im Sinne, daß wir die Probleme aufeinander probabilistisch in Polynomialzeit reduzieren können. Da bisher trotz intensiver Forschung (vergleiche Kapitel 2.4) keine effizienten Faktorisierungsalgorithmen bekannt sind, rechtfertigt Satz 3.1.3 die Annahme, daß die Rabin-Funktion eine Oneway-Funktion ist.

Die Blum-Micali-Konstruktion für einen Pseudozufallsgenerator aus Kapitel 1.5 setzt voraus, daß die Oneway-Funktion eine Permutation ist. Die Rabin-Funktion ist aber eine 4:1-Abbildung. Auf M. Blum [Blum81] geht die Beobachtung zurück, daß für eine Blumzahl N die Rabin-Funktion eingeschänkt auf Gruppe der quadratischen Reste modulo N eine Permutation ist. Wir beweisen zunächst folgendes Lemma:

Lemma 3.1.4
Für Primzahlen p mit $p \equiv 3 \pmod 4$ gilt $-1 \notin \mathrm{QR}_p$.

Beweis. Angenommen, es sei $-1 \in \mathrm{QR}_p$. Dann gibt es ein $x \in \mathbb{Z}_n^*$ mit $(\pm x)^2 \equiv -1 \pmod p$. Nach Lemma 3.1.2.a) existieren keine weiteren Wurzeln modulo p. Wegen des kleinen Satzes von Fermat gilt $(\pm x)^{p-1} \equiv 1 \pmod p$. Aus $-1 \not\equiv +1 \pmod p$ und

$$(-1)^{\frac{p-1}{2}} \equiv (\pm x)^{p-1} \equiv x^{p-1} \equiv +1 \pmod p$$

folgt, daß $\frac{p-1}{2}$ gerade, also $p-1$ durch 4 teilbar ist — Widerspruch, da $p - 1 \equiv 2 \pmod 4$. ■

Wir erhalten, daß Quadrieren auf den quadratischen Resten modulo einer Blumzahl eine 1:1-Abbildung ist. M. Blum [Blum81] hat als erstes auf diese Eigenschaft hingewiesen und erste kryptographische Anwendungen gegeben.

Satz 3.1.5
Sei N eine Blumzahl. Dann ist die Funktion $\mathrm{Rabin}_N : \mathrm{QR}_N \to \mathrm{QR}_N$ mit $\mathrm{Rabin}_N(x) := x^2 \bmod N$ eine Permutation.

Beweis. Wir müssen zeigen, daß jeder quadratische Rest $y \in \mathrm{QR}_N$ genau eine Wurzel $x \in \mathrm{QR}_N$ hat. Seien p und q die beiden Primfaktoren des Moduls N. Nach Lemma 3.1.2 hat y modulo p die Wurzeln $x_1^{(p)}$ und $x_2^{(p)}$ mit:

$$x_1^{(p)} \equiv +y^{\frac{p+1}{4}} \pmod p$$
$$x_2^{(p)} \equiv -y^{\frac{p+1}{4}} \pmod p$$

Da QR_p eine Gruppe ist sowie nach Lemma 3.1.4 gilt $-1 \notin \mathrm{QR}_p$, folgt $x_1^{(p)} \in \mathrm{QR}_p$ und $x_2^{(p)} \notin \mathrm{QR}_p$. Mit dem gleichen Argument zeigt man, daß genau eine Wurzel modulo q von y in QR_q liegt. Nach dem chinesischen Restsatz gilt $\mathrm{QR}_N \cong \mathrm{QR}_p \times \mathrm{QR}_q$, so daß nur die Wurzel x mit $x \in \mathrm{QR}_p$ und $x \in \mathrm{QR}_q$ ein quadratischer Rest modulo N ist. ■

Sei $\mathrm{bit}_{N,j}(x)$ das j-te Bit der Dualdarstellung von $x \bmod N$. Speziell sei $\mathrm{lsb}_N(x) := \mathrm{bit}_{N,1}(x)$ das unterste Bit. Wir zeigen in Kapitel 3.5:

Algorithmus 3.1.1 x^2-mod-N-Pseudozufallsgenerator

EINGABE: ▷ zufällige Blumzahl $N \in_{\mathrm{R}} \mathcal{B}_n$
▷ zufälliger Startwert $x \in_{\mathrm{R}} \mathrm{QR}_N$

/* Sei $\ell(n) \geq n+1$ durch ein Polynom beschränkt. */

1. $x_0 := x$

2. FOR $j := 1$ TO $\ell(n)$ DO

2.1. $x_j := x^2 \bmod N$

2.2. $b_j := \mathrm{lsb}_N(x_{j-1})$ /* unterstes Bit von $x_{j-1} \in [0, N)$ */

END for

AUSGABE: $\left(b_1, b_2, \ldots, b_{\ell(n)}\right) \in \{0,1\}^{\ell(n)}$

Satz 3.1.6
Unter der Annahme, daß für zufällige Blumzahl $N \in_{\mathrm{R}} \mathcal{B}_n$ und zufälligen quadratischen Rest $x \in_{\mathrm{R}} \mathrm{QR}_N$ die Abbildung

$$\mathrm{Rabin}: \quad (N, x) \mapsto (N, \mathrm{Rabin}_N(x))$$

mit $\mathrm{Rabin}_N(x) := x^2 \bmod N$ *eine Oneway-Permutation ist, bildet für jedes $j(n) \geq 1$ mit $j(n) = \mathcal{O}(\log_2 n)$ die Funktion*

$$h_{j(n)}(N, x) := \mathrm{bit}_{N,j(n)}(x)$$

ein Hardcore-Prädikat zur Rabin-*Funktion eingeschränkt auf quadratische Reste.*

Für den Beweis in Kapitel 3.5 nehmen wir an, es gäbe einen probabilistischen Polynomialzeit-Algorithmus, der zu zufälliger Blumzahl $N \in_{\mathrm{R}} \mathcal{B}_n$ und zufälligem quadratischen Rest $x \in_{\mathrm{R}} \mathrm{QR}_N$ aus $(N, \mathrm{Rabin}_N(x))$ das Bit $\mathrm{bit}_{N,j(n)}(x)$ mit Vorteil n^{-c} für ein festes $c \in \mathbb{N}$ berechnet. Wir zeigen, daß man dann in der Lage ist, für unendlich viele n mit nicht-vernachlässigbarer Wahrscheinlichkeit aus $(N, \mathrm{Rabin}_N(x))$ eine Wurzel modulo N in Polynomialzeit zu berechnen. Dieses widerspricht der Annahme, daß die Rabin-Abbildung eine Oneway-Permutation ist bzw. das Blumzahlen nicht effizient faktorisierbar sind. Insbesondere folgt aus Satz 2.1.2 für den Generator 2.1.2:

Korollar 3.1.7
Sei $\ell(n) \geq n+1$ durch ein Polynom beschränkt. Unter der Annahme, die Rabin-*Abbildung eingeschränkt auf quadratische Reste eine Oneway-Permutation ist, handelt es sich bei Algorithmus 3.1.1 um einen kryptographisch sicheren Pseudozufallsgenerator vom Typ $\ell(n)$.*

Beweis. Satz 1.5.5 auf Seite 26 gilt auch für die Argumente der Rabin-Abbildung. In Verbindung mit Satz 3.1.6 erhalten wir die Behauptung. ■

T.W. Cusick [Cusick95] die Ausgabefolge des x^2-mod-N-Pseudozufallsgenerators mit Hilfe von Sätzen aus der Zahlentheorie untersucht (d.h. nicht empirisch durch Computerexperimente). Er zeigt ohne kryptographische Annahmen, daß die durchschnittliche Differenz der Anzahl der Nullen und Einsen im Grenzwert für Folgenlänge gegen unendlich nicht schlechter als die einer Folge zufälliger, unabhängiger Münzwürfe ist. Für eine genau Formulierung der Aussage und Beweise verweisen wir auf die Originalarbeit.

Wie beim RSA-Generator werden wir in Kapitel 3.5 zeigen, daß die logarithmisch vielen untersten Bits simultan sicher sind, also in jeder Iteration des x^2-mod-N-Generators $\mathcal{O}(\log_2 n)$ Bits simultan ausgeben werden können:

Satz 3.1.8
Unter der Annahme, daß für zufällige Blumzahl $N \in_{\mathrm{R}} \mathcal{B}_n$ und zufälligen quadratischen Rest $x \in_{\mathrm{R}} \mathrm{QR}_N$ die Abbildung

$$\text{Rabin}: \quad (N, x) \mapsto (N, \text{Rabin}_N(x))$$

mit $\text{Rabin}_N(x) := x^2 \bmod N$ *eine Oneway-Permutation ist, bildet für $k(n) \geq 1$ mit $k(n) = \mathcal{O}(\log_2 n)$ die Funktion*

$$h_{k(n)}(N, x) := \text{Bits}_{N,k(n)}(x)$$

ein Hardcore-Funktion zur Rabin-*Funktion eingeschränkt auf quadratische Reste.*

In Kapitel 3.2 lernen wir den Muddle-Square-Generator kennen, der statt des untersten Bits das allgemeine Hardcore-Prädikat „inneres Produkt modulo 2“ von O. Goldreich und L.A. Levin verwendet. Zu dieser Variante des x^2-mod-N-Generators kennt man bessere Sicherheitsbeweise als zum dem ursprünglichen Generator.

Bevor wir den Sicherheitsbeweis des RSA-Generators auf den x^2-mod-N-Generator übertragen, beschäftigen wir uns in Kapitel 3.3 intensiver mit den quadratischen Resten modulo einer Blumzahl. Die Rabin-Funktion eingeschränkt auf die quadratischen Reste eine Permutation ist. In Kapitel 3.3 werden wir sehen, daß man auch auf andere Weise aus der Rabin-Funktion durch geringfüge Modifikation eine Permutation erhält.

3.2 Muddle-Square-Generator

If you generate random bits with the muddle-square method (...), you either get numbers that pass all reasonable statistic tests, or you get the fame and fortune for discovering a new factorization algorithm.

— Donald E. Knuth (1996)

D.E. Knuth [Knuth96] stellt in den überarbeiteten und erweiterten Kapiteln zu seinem Standardwerk [Knuth81] eine Variante des x^2-mod-N-Generators vor. Als Oneway-Permutation verwendet er ebenfalls die Rabin-Funktion eingeschränkt auf quadratische Reste, jedoch statt

des untersten Bits verwendet er das allgemeine Hardcore-Prädikat „inneres Produkt modulo 2“ von O. Goldreich und L.A. Levin [GL89, Levin93], das wir in Kapitel 1.6 kennengelernt haben.

Algorithmus 3.2.1 Muddle-Square-Pseudozufallsgenerator

EINGABE: ▷ zufällige Blumzahl $N \in_{\mathrm{R}} \mathcal{B}_n$
▷ zufälliger Startwert $x \in_{\mathrm{R}} \mathrm{QR}_N$
▷ zufälliger Vektor $r \in_{\mathrm{R}} \{0,1\}^n$

/* Sei $\ell(n) \geq n+1$ durch ein Polynom beschränkt. */

1. $x_0 := x$

2. FOR $j := 1$ TO $\ell(n)$ DO

2.1. $x_j := x^2 \bmod N$

2.2. $b_j := \langle x_{j-1}, r \rangle$ /* x_{j-1} als Bitvektor aus $\{0,1\}^n$ */

END for

AUSGABE: $\left(b_1, b_2, \ldots, b_{\ell(n)}\right) \in \{0,1\}^{\ell(n)}$

Neben der zufälligen Blumzahl $N \in_{\mathrm{R}} \mathcal{B}_n$ wählt man einen zufälligen Vektor $r \in_{\mathrm{R}} \{0,1\}^n$. Wir fassen die Dualdarstellung des quadratischen Rests $x \in_{\mathrm{R}} \mathrm{QR}_N$ (genauer den kleinsten, nicht-negativen Repräsentanten) als Bitvektor in $\{0,1\}^n$ auf und geben das innere Produkt $\langle x, r \rangle$ modulo 2 aus. Algorithmus 3.2.1 zeigt den Pseudozufallsgenerator. D.E. Knuth nennt diese Erzeugung Muddle-Square-Methode, da die Bits des quadratischen Restes ungeachtet ihrer Wertigkeit addiert werden („to muddle“ heißt wörtlich übersetzt „durcheinander bringen“). In der klassischen Theorie von Zufallsgeneratoren kennt man eine ähnliche Methode: J. von Neuman hat 1946 die bekannte Middle-Square-Methode vorgeschlagen: Man quadriert den vorherigen Wert und gibt die mittleren Ziffern des Quadrates aus.

Die Muddle-Square-Methode bildet unter der Annahme, daß Blumzahlen effizient nicht faktorisierbar sind, einen Pseudozufallsgenerator im Sinne von A. Yao bzw. M. Blum und S. Micali:

Lemma 3.2.1
Sei A ein probabilistischer Algorithmus mit Vorteil $\epsilon(n)$ zur Berechnung des Prädikats

$$h_{\mathrm{ip}}(N, x, r) := \langle x, r \rangle \bmod 2$$

zu $f_{\mathrm{ip}} : (N, x, r) = (\mathrm{Rabin}_N(x), r)$ mit $N \in_{\mathrm{R}} \mathcal{B}_N$, $x \in_{\mathrm{R}} \mathrm{QR}_N$ und $r \in_{\mathrm{R}} \{0,1\}^n$. Dann können wir für unendlich viele n für zufällige Blumzahlen $N \in_{\mathrm{R}} \mathcal{B}_n$ und zufälligen quadratischen Rest $x \in_{\mathrm{R}} \mathrm{QR}_N$ zu $x^2 \bmod N$ eine Wurzel modulo N in Zeit

$$\mathcal{O}\left(n^2\epsilon(n)^{-2} \cdot \log_2\left(n\epsilon(n)^{-2}\right) + n^3\epsilon(n)^{-2}\right) + 2n^2\epsilon(n)^{-2} \cdot |A|$$

mit Wahrscheinlichkeit mindestens $\frac{1}{2}\epsilon(n)$ berechnen.

Beweis. Wir betrachten nur die unendlich vielen n, für die Algorithmus A das Prädikat h_{ip} mit Wahrscheinlichkeit $\frac{1}{2} + \epsilon(n)$ bestimmt. Es gibt nach Lemma 1.6.3 auf Seite 44 eine Teilemenge G_n der Paare $(N, x) \in \mathcal{B}_n \times \mathrm{QR}_N$, so daß

a) ein zufälliges Paar $(N, x) \in_{\mathrm{R}} \mathcal{B}_n \times \mathrm{QR}_N$ mit Wahrscheinlichkeit mindestens $\epsilon(n)$ in G_n liegt und

b) für alle $(x, N) \in G_n$ gilt $\mathrm{Ws}[\, A\,(N, \mathrm{Rabin}_N(x), U_n) = h_{\mathrm{ip}}(N, x, U_n)] \geq \frac{1}{2} + \frac{1}{2}\epsilon(n)$.

Nach Lemma 3.2.1 auf Seite 39 können wir für $(N, x) \in G_n$ in Zeit $\mathcal{O}\left(nk2^k + n^2 2^k\right) + n2^k \cdot |A|$, wobei $k := \left\lceil \log_2 \left(2n\epsilon(n)^{-2} + 1\right)\right\rceil$, mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ zu $\mathrm{Rabin}_N(x)$ das Urbild $x \in \mathrm{QR}_N$ bestimmen, indem man für alle möglichen 2^k Urbilder in $\mathcal{O}\left(n^2\right)$ testet, ob das Quadrat modulo N mit x^2 übereinstimmt. Wir können allerdings nicht garantieren, daß die gefundene Wurzel ein quadratischer Rest ist, da eventuell in der Ausgabemenge neben der Wurzel $x \in \mathrm{QR}_N$ weitere enthalten sein können. ■

Falls das vom Muddle-Square-Generator 3.2.1 erzeugte Ensemble einen statistischen Test mit Tolerenz n^{-c} für festes $c \in \mathbb{N}$ nicht besteht, können wir den Modul effizient mit nicht-vernachlässigbarer Wahrscheinlichkeit faktorisieren:

Satz 3.2.2
Sei $\ell(n) \geq n + 1$ durch ein Polynom beschränkt. Falls das vom Muddle-Squadre-Generator 3.2.1 erzeugte Ensemble des Typs $\ell(n)$ einen statistischen Test T mit Toleranz $\delta(n)$ nicht besteht, kann man für unendlich viele n eine zufällige Blumzahl $N \in_{\mathrm{R}} \mathcal{B}_n$ in Zeit

$$\mathcal{O}\left(n^2\ell(n)^2\delta(n)^{-2} \cdot \log_2\left(n\ell(n)^2\delta(n)^{-2}\right)\right) + 2n^2\ell(n)^2\delta(n)^{-2} \cdot \left(|A| + \ell(n) \cdot \mathcal{O}\left(n^2\right)\right)$$

mit Wahrscheinlichkeit mindestens $\frac{\delta(n)}{4\ell(n)}$ faktorisieren.

Für $\delta(n) \geq 2^{-\ell(n)}$ kann die Laufzeitschranke zu $\mathcal{O}\left(n^4\ell(n)^3\delta(n)^{-2} \cdot |A|\right)$ vereinfacht werden.

Beweis. Nach Lemma 1.5.4 auf Seite 25 erhalten wir einen Prediktor A zur Vorhersage des inneren Produkts zu gegebenem $\mathrm{Rabin}_N(x)$ und r mit Vorteil $\epsilon(n) := \frac{\delta(n)}{\ell(n)}$ und Laufzeit $|A| = |T| + \ell(n) \cdot \mathcal{O}\left(n^2\right)$, da Quadrieren modulo N in $\mathcal{O}\left(n^2\right)$ Schritten möglich ist. Nach Lemma 3.2.1 kann man zu gegebenem $x^2 \bmod N$ für unendlich viele n in Zeit

$$\mathcal{O}\left(\frac{n^2}{\epsilon(n)^2} \cdot \log_2 \frac{n}{\epsilon(n)^2} + \frac{n^3}{\epsilon(n)^2}\right) + \frac{2n}{\epsilon(n)^2} \cdot |A|$$

mit Wahrscheinlichkeit mindestens $\frac{1}{2}\epsilon(n)$ eine Wurzel modulo N bestimmen. Aus Satz 3.1.3 auf Seite 105 folgt, daß wir eine zufällige Blumzahlen $N \in_{\mathrm{R}} \mathcal{B}_n$ mit Wahrscheinlichkeit $\frac{1}{2} \cdot \frac{1}{2}\epsilon(n)$ in zusätzlichen $\mathcal{O}\left(n^3\right)$ Schritten faktorisieren können. ■

Wir erhalten:

Korollar 3.2.3
Sei $\ell(n) \geq n+1$ durch ein Polynom beschränkt. Unter der Annahme, daß für zufällige Blumzahl $N \in_{\mathrm{R}} \mathcal{B}_n$ und zufälligen quadratischen Rest $x \in_{\mathrm{R}} \mathrm{QR}_N$ die Abbildung

$$\mathrm{Rabin}: \quad (N, x) \mapsto (N, \mathrm{Rabin}_N(x))$$

mit $\mathrm{Rabin}_N(x) := x^2 \bmod N$ eine Oneway-Permutation ist, bildet der Muddle-Square-Generator 3.2.1 einen kryptographisch sicheren Pseudozufallsgenerator.

Wir haben in Satz 1.6.7 auf Seite 49 die Verallgemeinerung des Hardcore-Prädikats auf die Hardcore-Funktion der Töplitz-Matrix kennengelernt. Man kann statt des inneren Produkts das Produkt mit einer zufälligen Töplitz-Matrix berechnen und in jeder Iteration logarithmisch vielen Bits ausgeben. Wenn wir in jeder Iteration $k(n)$-viele Bits ausgeben und die Generatorausgabe einen statistischen Test mit Toleranz $\delta(n)$ nicht besteht, erhält man aus dem Beweis zu Satz 1.6.7 einen Prediktor zur Vorhersage des inneren Produkts mit Vorteil $\epsilon(n) := \frac{\delta(n)}{k(n)^2}$.

3.3 Quadratische Reste und modifizierter Generator

In diesem Kapitel untersuchen wir die quadratischen Reste modulo einer Blumzahl N. Dazu betrachten wir zunächst den Fall einer Primzahl $p > 2$ als Modul. Um die quadratischen Reste modulo p zu charaktisieren, definiert man zu $x \in \mathbb{Z}_p^*$ das *Legrende-Symbol*:

$$\left(\frac{x}{p}\right) := \begin{cases} +1 & \text{falls } x \in \mathrm{QR}_p \\ -1 & \text{sonst} \end{cases}$$

Da die quadratischen Reste eine Untergruppe von $\mathbb{Z}_p^*$ bilden, gilt für $x, y \in \mathbb{Z}_p^*$

$$\left(\frac{xy}{p}\right) = \left(\frac{x}{p}\right) \cdot \left(\frac{y}{p}\right)$$

und $\left(\frac{1}{p}\right) = +1$. Insbesondere haben $x \in \mathbb{Z}_p^*$ und das Invererse $y = x^{-1} \in \mathbb{Z}_p^*$ das gleiche Legrende-Symbol. Für Primzahlen $p > 2$ kann man effizient mit dem sog. *Euler-Kriterium* zu $x \in \mathbb{Z}_p^*$ entscheiden, ob x ein quadratischen Reste modulo p ist:

$$\left(\frac{x}{p}\right) \equiv x^{\frac{p-1}{2}} \pmod{p}$$

Aus dem Euler-Kriterium folgt das „erste Ergänzungsgesetz“, wonach -1 genau dann ein quadratischer Rest modulo $p > 2$ ist, wenn $p \equiv 1 \pmod 4$ gilt (vergleiche Lemma 3.1.4):

$$\left(\frac{-1}{p}\right) \equiv (-1)^{\frac{p-1}{2}} \pmod{p}$$

Nach dem ersten Ergänzungsgesetz ist genau dann -1 modulo $p > 2$ ein quadratischer Rest, wenn $p \equiv 3 \pmod 4$ gilt. Das „zweite Ergänzungsgesetz“ besagt, daß 2 genau dann ein quadratischer Rest modulo $p > 2$ ist, wenn $p \equiv \pm 1 \pmod 8$ oder $p \equiv \pm 7 \pmod 8$ gilt:

$$\left(\frac{2}{p}\right) \equiv 2^{\frac{p^2-1}{8}} \pmod{p}$$

Das *Jacobi-Symbol* ist die Verallgemeinerung des Legrende-Symbols auf ungerade Zahlen N. Sei $N = \prod_{i=1}^{r} p_i$ die Zerlegung in Primzahlen, wobei Primzahlen auch mehrfach auftreten können. Zu $x \in \mathbb{Z}_N^*$ ist das Jacobi-Symbol das Produkt der Legrende-Symbole $\left(\frac{x}{p_i}\right)$:

$$\left(\frac{x}{N}\right) = \left(\frac{x}{p_1}\right) \cdot \left(\frac{x}{p_1}\right) \cdots \left(\frac{x}{p_r}\right)$$

Betrachten wir den Fall, daß N eine Blumzahl mit den beiden Primfaktoren p und q ist:

$$\left(\frac{x}{N}\right) = \left(\frac{x}{p}\right) \cdot \left(\frac{x}{q}\right)$$

Falls $x \in \mathrm{QR}_N$ ist, gilt $\left(\frac{x}{N}\right) = (+1)(+1) = +1$. Im Gegensatz zum Legrende-Symbol ist aber die Umkehrung falsch. Nach Lemma 3.1.4 auf Seite 106 oder dem ersten Ergänzungsgesetz ist -1 modulo p und modulo q ein quadratischer Nichtrest, nach Chinesischen Restsatz $\mathrm{QR}_N \cong \mathrm{QR}_p \times \mathrm{QR}_q$ folgt, daß -1 auch modulo N ein quadratischer Nichtrest ist, wenngleich für das Jacobi-Symbol gilt:

$$\left(\frac{-1}{N}\right) = \left(\frac{-1}{p}\right) \cdot \left(\frac{-1}{q}\right) = (-1) \cdot (-1) = +1$$

Allgemein hat $x \in \mathbb{Z}_N^*$ das Jacobi-Symbol $+1$, falls $x \in \mathrm{QR}_N$ ist oder x modulo p und q ein quadratischer Nichtrest ist. Da modulo einer Primzahl sowohl x und das Inverse x^{-1} das gleiche Legrende-Symbol haben, gilt für das Jacobi-Symbol $x \in \mathbb{Z}_N^*$:

$$\left(\frac{x}{N}\right) = \left(\frac{x}{p}\right) \cdot \left(\frac{x}{q}\right) = \left(\frac{x^{-1}}{p}\right) \cdot \left(\frac{x^{-1}}{q}\right) = \left(\frac{x^{-1}}{N}\right)$$

Wegen $\left(\frac{1}{N}\right) = +1$ bilden die Elemente mit Jacobi-Symbol $+1$ eine Untergruppe von $\mathbb{Z}_N^*$ mit Index 2:

$$\mathbb{Z}_N(+1) := \left\{ x \in \mathbb{Z}_N^* \;\middle|\; \left(\frac{x}{N}\right) = +1 \right\}$$

Die quadratischen Reste QR_N bilden ihrerseits eine Untergruppe von $\mathbb{Z}_N(+1)$ mit Index 2.

Obwohl wir bei der Definition des Jacobi-Symbols die Faktorisierung des Moduls N verwendet haben, kann man das Jacobi-Symbol $\left(\frac{x}{N}\right)$ effizient ohne Kenntnis der Faktorisierung von N berechnen:

Satz 3.3.1
Sei $N \in \mathcal{B}_n$ eine Blumzahl und $x \in \mathbb{Z}_N^*$. Dann kann in $\mathcal{O}(n^3)$ Schritten das Jacobi-Symbol $\left(\frac{x}{N}\right)$ ohne Kenntnis der Faktorisierung von N berechnet werden.

Beweis. Ein Algorithmus zur Berechnung des Jacobi-Symbols mit angegebener Laufzeit folgt aus [Knuth81, Aufgabe 23a, Kapitel 4.5.4]. ■

Für die Variante der Rabin-Funktion benötigen wir den Absolutbetrag modulo N, wie wir ihn in Kapitel 2.2 eingeführt haben. Zu $z \in \mathbb{Z}$ ist der Absolutbetrag abs_N modulo N definiert als:

$$\mathrm{abs}_N(z) := \begin{cases} [z]_N & \text{falls } [z]_N \leq \frac{N-1}{2} \\ N - [z]_N & \text{sonst} \end{cases}$$

Der Absolutbetrag $\mathrm{abs}_N(z)$ entspricht dem Abstand von $[z]_N$ zur 0 (vergleiche Abbildung 2.2.1 auf Seite 60).

Satz 3.3.2
Sei $N \in \mathcal{B}_n$ und $S_N(+1) := \left\{ x \in \mathbb{Z}_N(+1) \;\middle|\; [x]_N \leq \frac{N-1}{2} \right\}$. Dann ist die Abbildung $\mathrm{Rabin}_N^* : S_N(+1) \to S_N(+1)$ mit $\mathrm{Rabin}_N^*(x) := \mathrm{abs}_N(x^2)$ eine Permutation.

Beweis. Wir müssen zeigen, daß zu $y \in \mathrm{QR}_N$ genau eine Wurzel modulo N in S_N liegt. Sei x die Wurzel von y in QR_N. Wegen $\left(\frac{-1}{N}\right) = +1$ ist $[-x]_N = N - x$ die andere Wurzel mit Jacobi-Symbol +1, so daß genau einer der beiden Werte in $S_N(+1)$ liegt. ■

Wie Satz 3.1.3 zeigt man:

Satz 3.3.3
Sei A ein Algorithmus, der zu $N \in \mathcal{B}_n$ und $\mathrm{Rabin}_N^*(x)$ für ein zufälliges $x \in_{\mathrm{R}} \mathbb{Z}_N^*$ mit Wahrscheinlichkeit $\epsilon(n)$ ein Urbild von $\mathrm{Rabin}_N^*(x)$ ausgibt. Dann können wir die Blumzahl N mit Wahrscheinlichkeit $\frac{1}{2}\epsilon(n)$ in Zeit $|A| + \mathcal{O}\left(n^3\right)$ faktorisieren.

Dieser Satz rechtfertigt die Annahme, daß für zufällige Blumzahl $N \in_{\mathrm{R}} \mathcal{B}_n$ und zufälligen quadratischen Rest $x \in_{\mathrm{R}} \mathrm{QR}_N$ die Abbildung

$$\mathrm{Rabin}^* : \quad (N, x) \mapsto (N, \mathrm{Rabin}_N^*(x))$$

mit $\mathrm{Rabin}_N^*(x) := x^2 \bmod N$ eingeschränkt auf $S_N(+1)$ eine Oneway-Permutation ist.

Als Hardcore-Prädikate verwenden wir die gleichen wie im Fall der ursprünglichen Rabin_N-Funktion, d.h. der zugehörige Generator gibt in jeder Iteration das unterste Bit des Absolutbetrags der Wurzel in $S_N(+1)$ aus. Wir werden in Kapitel 3.4 die Sicherheit dieses Generators aus den Sicherheitsbeweisen zum RSA-Generator folgern.

Algorithmus 3.3.1 Modifizierter x^2-mod-N-Pseudozufallsgenerator

---

EINGABE: ▷ zufällige Blumzahl $N \in_{\mathrm{R}} \mathcal{B}_n$
▷ zufälliger Startwert $x \in_{\mathrm{R}} S_N(+1) = \left\{ x \in \mathbb{Z}_N(+1) \;\middle|\; [x]_N \leq \frac{N-1}{2} \right\}$

/* Sei $\ell(n) \geq n + 1$ durch ein Polynom beschränkt. */

1. $x_0 := x$

2. FOR $j := 1$ TO $\ell(n)$ DO

2.1. $x_j := \mathrm{abs}_N(x^2)$

2.2. $b_j := \mathrm{lsb}_N(x_{j-1})$ /* unterstes Bit von $x_{j-1} \in [0, N)$ */

END for

AUSGABE: $\left(b_1, b_2, \ldots, b_{\ell(n)}\right) \in \{0,1\}^{\ell(n)}$

Wir können zu gegebenem $x \in \mathbb{Z}_N^*$ nach Satz 3.3.1 effizient entscheiden, ob x in der Untergruppe $\mathbb{Z}_N(+1)$ liegt. Kann man zu gegebener Blumzahl $N \in \mathcal{B}_n$ und $x \in \mathbb{Z}_N(+1)$ ohne Kenntnis der Faktorisierung von N auch effizient bestimmen, ob x ein quadratischer Rest modulo N ist? Obwohl sich bereits C.F. Gauß Anfang des 19-ten Jahrhunderts mit diesem Problem beschäftigt hat, sind bis heute keine Polynomialzeit-Algorithmen mit Erfolgswahrscheinlichkeit $\frac{1}{2} + n^{-c}$ mit konstantem $c \in \mathbb{N}$ kannt. Falls es einen Algorithmus mit Erfolgswahrscheinlichkeit $\frac{1}{2}+\epsilon(n)$ gibt, können wir durch eine Mehrheitsentscheidung die Fehlerwahrscheinlichkeit auf 2^{-n} senken: Um zu entscheiden, ob $x \in \mathbb{Z}_N(+1)$ ein quadratischer Rest ist, wählen wir zufällige $r_1, r_2, \ldots, r_m \in_{\mathrm{R}} \mathrm{QR}_N$ und nehmen an, daß $x \in \mathrm{QR}_N$ ist, wenn der Algorithmus für die Mehrheit der Testpunkte $r_i x \in \mathbb{Z}_N(+1)$ ausgibt, daß es quadratische Reste seien. Wir nutzen, daß die qudratischen Reste modulo N eine Gruppe bilden, also gilt:

$$x \in \mathrm{QR}_N \qquad \Longleftrightarrow \qquad r_i x \in \mathrm{QR}_N$$

Mit $m := \frac{1}{2}\epsilon(n)^{-2}n$ und $t = \epsilon(n)$ folgt aus der additiven Chernoff-Schranke (siehe Fakt 2.3.7 auf Seite 91), daß die Fehlerwahrscheinlichkeit höchstens 2^{-n} ist. Um ein zufälliges $r_i \in_{\mathrm{R}} \mathrm{QR}_N$ zu ziehen, wählen wir einfach ein zufälliges $r \in_{\mathrm{R}} \mathbb{Z}_N^*$ und setzen $r_i := r^2 \bmod N$. Da jedes Quadrat modulo einer Blumzahl eine 4:1-Abbildung ist, erhält man einen zufälligen quadratischen Rest modulo N.

Man nimmt an, daß es keinen probabilistischen Polynomialzeit-Algorithmus gibt, der zu zufälliger Blumzahl $N \in_{\mathrm{R}} \mathcal{B}$ und $x \in_{\mathrm{R}} \mathbb{Z}_N(+1)$ mit Wahrscheinlicheit $\frac{1}{2} + n^{-c}$ (für ein beliebiges, festes $c \in \mathbb{N}$) entscheidet, ob x ein quadratischer Rest modulo N ist. Diese *quadratische Residuozitätsannahme* ist aber stärker als die Vermutung, daß man Blumzahl nicht in Polynomialzeit faktorisieren kann, denn mit Hilfe der Primfaktorzerlegung $N = pq$ können wir durch Auswerten der Legrende-Symbole modulo p und modulo q entscheiden, ob $x \in \mathbb{Z}_N(+1)$ ein quadratischer Rest ist. Umgekehrt ist offen, ob zum Entscheiden die Kenntnis der Primfaktorzerlegung des Moduls Voraussetzung ist.

Satz 3.3.4
Sei A ein Algorithmus, der für zufälliges $(N, x) \in_{\mathrm{R}} \mathcal{B}_n \times \mathrm{QR}_N$ einen Vorteil $\epsilon(n)$ bei der Bestimmung des untersten Bits des Urbildes in QR_N von $\mathrm{Rabin}_N(x)$ zu gegebener Blumzahl N und $\mathrm{Rabin}_N(x)$ hat. Dann kann man in Zeit $|A| + \mathcal{O}\left(n^2\right)$ für zufälliges $N \in_{\mathrm{R}} \mathcal{B}_n$ und $z \in_{\mathrm{R}} \mathbb{Z}_N(+1)$ mit Wahrscheinlichkeit mindestens $\frac{1}{2}+\epsilon(n)$ entscheiden, ob z ein quadratischer Rest modulo N ist.

Beweis. Sei $x \in \mathrm{QR}_N$ die Wurzel von z^2, die ein quadratischer Rest modulo N ist. Da N eine Blumzahl ist, gilt entweder $x = z$ oder $x = -z$. Wir unterscheiden zwei Fälle:

- Falls z ein quadratischer Rest ist, gilt $x = z$ und insbesondere $\mathrm{bit}_{N,1}(z) = \mathrm{bit}_{N,1}(x)$.
- Falls z kein quadratischer Rest ist, gilt $-x \equiv z \pmod N$. Wegen $N \equiv 1 \pmod 2$ folgt

  $$\mathrm{bit}_{N,1}(z) \equiv [z]_N \equiv [-x]_N \equiv N - [x]_N \equiv 1 - \mathrm{bit}_{N,1}(x) \pmod 2,$$

  so daß $\mathrm{bit}_{N,1}(z) \neq \mathrm{bit}_{N,1}(x)$ ist.

Wir nehmen $z \in \mathrm{QR}_N$ an, wenn $A(N, z^2) = \mathrm{bit}_{N,1}(z)$ gilt.

Für ein korrektes Resultat von A ist unsere Entscheidung ebenfalls richtig. Da mit Wahrscheinlichkeit mindestens $\frac{1}{2} + \epsilon(n)$ Algorithmus A ein richtiges Resultat liefert, folgt die behauptete Erfolgswahrscheinlichkeit. Für die Laufzeitabschätzung $|A| + \mathcal{O}\left(n^2\right)$ beachte, daß man mit in $\mathcal{O}\left(n^2\right)$ Schritten das Quadrat modulo N berechnen kann. ■

Auf diesem Satz und der quadratischen Residuozitätsannahme basiert der ursprüngliche Sicherheitsbeweis von L. Blum, M. Blum und M. Shub [BBS86] zum x^2-mod-N-Pseudozufallsgenerator 3.1.1 auf Seite 107:

Korollar 3.3.5 (Blum, Blum, Shub 1982)
Unter der quadratische Residuozitätsannahme ist $(N, x) \mapsto \mathrm{lsb}_N(x)$ *ein Hardcore-Prädikat zur* Rabin-*Funktion eingeschränkt auf quadratische Reste und für jedes durch ein Polynom beschränktes* $\ell(n) \geq n+1$ *ist der* x^2*-mod-*N*-Generator 3.1.1 ein kryptographisch sicher Pseudozufallsgenerator vom Typ* $\ell(n)$*.*

Beweis. Angenommen, $(N, x) \mapsto \mathrm{lsb}_N(x)$ bilde kein Hardcore-Prädikat, d.h. es gäbe einen probabilistischen Polynomialzeit-Algorithmus, der mit Vorteil n^{-c} für festes $c \in \mathbb{N}$ das Prädikat $\mathrm{lsb}_N(x)$ für zufälliges $(N, x) \in_{\mathrm{R}} \mathcal{B}_n \times \mathrm{QR}_N$ berechnet. Aus Satz 3.3.4 folgt ein Widerspruch zur quadratischen Residuozitätsannahme. Wir erhalten, daß der x^2-mod-N-Generator 3.1.1 ein kryptographisch sicherer Pseudozufallsgenerator vom Typ $\ell(n)$ ist. ■

3.4 Sicherheitsbeweis für modifizierten Generator

Wir betrachten zunächst den Fall $j = 1$, also das unterste Bit. Sei A^* ein Algorithmus, der zu gegebenem $\mathrm{Rabin}_N^*(z)$ mit $z \in S_N(+1)$ einen $\epsilon(n)$-Vorteil zur Vorhersage von $\mathrm{bit}_{N,1}(z)$ hat, wobei $\mathrm{Rabin}_N^* : S_N(+1) \to S_N(+1)$ mit

$$S_N(+1) := \left\{ z \in \mathbb{Z}_N(+1) \;\middle|\; [z]_N \leq \tfrac{N-1}{2} \right\}$$

und $\mathrm{Rabin}_N^*(z) = \mathrm{abs}_N(z^2)$. Zur Vereinfachung sei zunächst für die betrachteten n die Erfolgswahrscheinlichkeit von Algorithmus A^* unabhängig von N. Mit Hilfe von A^* wollen wir bezüglich n und $\epsilon(n)^{-1}$ zu gegebenem Funktionswert $\mathrm{Rabin}_N^*(x)$ eine Wurzel modulo N von x^2 berechnen. Wir setzen $\epsilon(n)^{-1} = n^{\mathcal{O}(1)}$ voraus, um für hinreichend große N bzw. n die Wahrscheinlichkeit, daß ein modulo N gleichverteilter Wert nicht teilerfremd zu N ist, zu vernachlässigen.

Wir übernehmen das Verfahren aus Kapitel 2.3, d.h. Algorithmus 2.3.2 von Seite 87, Algorithmus 3.4.1 zeigt für die Rabin-Funktion angepaßte Version. Wir verbessern dieses Verfahren wieder durch die Mehrheitsentscheidung über eine Stichprobe. Sei

$$S_N := \left\{ z \in \mathbb{Z}_N^* \;\middle|\; [z]_N \leq \tfrac{N-1}{2} \right\}$$

so daß $S_N(+1) = S_N \cap \mathbb{Z}_N(+1)$. Algorithmus A^* liefert zu $\mathrm{Rabin}_N^*(z)$ das unterste Bit der Wurzel in $S_N(+1)$. Drei Situationen sind zu unterscheiden:

- Falls $z \in S_N(+1)$ ist, bezieht sich das Resultat von A^* auf z.

- Falls $z \notin S_N(+1)$ aber $\left(\frac{z}{N}\right) = +1$ ist, bezieht sich das Resultat von A^* auf $[-z]_N \in S_N(+1)$.
- Im Fall $\left(\frac{z}{N}\right) = -1$ bezieht sich das Resultat von A^* auf eine inkongruente Wurzel, die sich nicht nur im Vorzeichen unterscheidet. In Satz 3.3.3 bzw. im Beweis des Satzes 3.1.3 auf Seite 105 haben wir gesehen, daß man mit beiden Wurzeln auch den Modul N faktorisieren kann. Unter unserer Voraussetzung, daß die Rabin*-Abbildung eine Oneway-Funktion bzw. Faktorisieren einer Blumzahl effizient nicht möglich ist, kennen wir die Wurzel, auf die sich das Resultat bezieht, nicht.

Wir sondern bereits zu Beginn die Testpunkte $r_{t,i} = [iu_{t-1} + v]_N$ aus, für die $a_t + r_{t,i}$ das Jacobi-Symbol -1 hat. Da für das Urbild $x \in S_N(+1)$ das Jacobi-Symbol $+1$ ist, erhält man genau die Testpunkte, für welche $(a_t + r_{t,i})\,x$ das Jacobi-Symbol $+1$ hat. Wir können aber die Mehrheitsentscheidungen nicht direkt übernehmen, denn beim Vergleich

$$A^*\left(N, \mathrm{Rabin}_N^*\left((a_t + r_{t,i})x\right)\right) \equiv i \cdot [u_{t-1}x]_N + [vx]_N - w_{t,i} \pmod 2 \tag{3.3}$$

ist zu beachten, daß für $(a_t + r_{t,i})x \notin S_N(+1)$ das Resultat von A^* sich nicht auf $(a_t + r_{t,i})x$ sondern auf $-(a_t + r_{t,i})x$ bezieht. Wir haben im Beweis zu Satz 3.3.4 auf Seite 114 gesehen, daß allgemein das unterste Bit von z und $-z$ modulo N unterschiedlich sind, also

$$\mathrm{bit}_{N,1}(z) \equiv 1 + \mathrm{bit}_{N,1}(-z) \pmod 2$$

gilt. Wir ändern daher den Vergleich (3.3) zu:

$$A^*(N, \mathrm{Rabin}_N^*((a_t + r_{t,i})x)) \equiv i \cdot [u_{t-1}x]_N + [vx]_N - \underbrace{w_{t,i} + \left[\,[(a_t + r_{t,i})x]_N > \tfrac{N+1}{2}\right]}_{=:w_{t,i}^*} \pmod 2$$

Da wir nur $\mathrm{Rabin}_N^*\left((a_t + r_{t,i})x\right)$, aber nicht x kennen, verwenden wir zur Berechnung von $w_{t,i}^*$ die Approximationen, die bereits zur Bestimmung von $w_{t,i}$ genutzen werden. Um $w_{t,i}$ zu bestimmen, berechnen wir $\overline{w}_{t,i} := k_t + i \cdot k_{t-1}^{(u)} + k^{(v)}$ aus den Approximationen, so daß nach Abschätzung (2.21) auf Seite 86 gilt

$$\left|\,[a_t x]_N + [r_{t,i}x]_N - \overline{w}_{t,i}N\right| \le \tfrac{1}{4}\epsilon(n)N$$

und setzen $w_{t,i} := \lfloor \overline{w}_{t,i} \rfloor$. Für die Rabin_N^*-Funktion lautet die Zuweisung:

$$w_{t,i}^* := \lfloor \overline{w}_{t,i} \rfloor + \left[\overline{w}_{t,i} \bmod 1 > \tfrac{1}{2}\right]$$

Das berechnete $w_{t,i}$ kann nur falsch sein, wenn

$$1 - \tfrac{1}{4}\epsilon(n) < \overline{w}_{t,i} \bmod 1 < 1$$

ist. In diesem Fall haben wir fälschlicherweise abgerundet, so daß man zu $w_{t,i}$ eins addieren müßten. Dieser Fehler hebt sich aber weg, da wegen $\overline{w}_{t,i} \bmod 1 > \frac{1}{2}$ fälscherlicherweise eins addieren wird. Bei der Berechnung des zweiten Summanden von $w_{t,i}^*$ kann ähnlich wir bei $w_{t,i}$ ein Fehler auftreten, wenn die Nachkommastellen von $\overline{w}_{t,i}$ zwar kleiner als $\frac{1}{2}$ sind, aber $[(a_t + r_{t,i})x]_N$ größer als $\frac{N-1}{2}$ ist. Dies kann nur eintreten, wenn

$$\tfrac{1}{2} - \tfrac{1}{4}\epsilon(n) < \overline{w}_{t,i} \bmod 1 < \tfrac{1}{2}.$$

Algorithmus 3.4.1 Rabin*-Invertierung durch sukzessive Approximation

EINGABE: ▷ Blumzahl $N \in \mathcal{B}_n$
▷ $\text{Rabin}_N^*(x)$ mit $x \in S_N(+1) := \left\{ z \in \mathbb{Z}_N(+1) \;\middle|\; [z]_N \leq \frac{N-1}{2} \right\}$

/* Sei A^* Algorithmus mit $\text{Ws}\left[A^*\left(N, \text{Rabin}_N^*\left(U_{S_N(+1)}\right)\right) = \text{bit}_{N,1}\left(U_{S_N(+1)}\right)\right] \geq \frac{1}{2} + \epsilon(n)$, wobei die Wahrscheinlichkeit über $U_{S_N(+1)}$ und die internen Münzwürfe gebildet wird. */

1. Wähle $u_0, v \in \mathbb{Z}_N$ zufällig und unabhängig. Setze $a_0 := u_0$.

2. FOR $t := 1$ TO n DO $a_t := u_t := \left[2^{-t} u_0\right]_N$

3. FOR $t := 1$ TO n /* Bestimme mit Algorithmus A^* unterste Bits */

3.1. $m_t := \min\left\{2^t, 2n\right\} \cdot \epsilon(n)^{-2}$, $M_t := \left\{ i \in \mathbb{Z} \setminus \{0\} \;:\; \left|i + \frac{1}{2}\right| \leq \frac{1}{2} m_t \right\}$

3.2. FOR all $i \in M_t$ DO

3.2.1. IF $a_t + r_{t,i} \in \mathbb{Z}_N(+1)$ THEN

$$r_{t,i} := [i u_{t-1} + v]_N \text{ und } b_{t,i} := A^*\left(N, \text{Rabin}_N^*\left((a_t + r_{t,i})x\right)\right)$$

ELSE entferne i aus M_t und dekrementiere m_t.

END for i

END for t

/* Probiere alle möglichen Approximationen und untersten Bits: */

4. FOR all $\left(k_0^{(u)}, k^{(v)}, b_0^{(u)}, b^{(v)}\right) \in \frac{1}{8}\epsilon(n)^3 \left[0, 8\epsilon(n)^{-3}\right) \times \frac{1}{4}\epsilon(n) \left[0, 4\epsilon(n)^{-1}\right) \times \{0,1\}^2$ DO

4.1. $k_0 := k_0^{(u)}$ und $b_0 := b_0^{(u)}$.

4.2. FOR $t := 1$ TO n DO

4.2.1. $k_t := \frac{1}{2}\left(k_{t-1} + b_{t-1}\right)$

4.2.2. FOR all $i \in M_t$ DO

$$\overline{w}_{t,i} := k_t + i \cdot k_{t-1}^{(u)} + k^{(v)}, \quad w_{t,i}^* := \lfloor \overline{w}_{t,i} \rfloor + \left\lceil \overline{w}_{t,i} \bmod 1 > \tfrac{1}{2} \right\rceil$$

4.2.3. $M := \left\{ i \in M_t \;\middle|\; b_{t,i} \equiv i \cdot [u_{t-1}x]_N + [vx]_N - w_{t,i}^* \pmod 2 \right\}$

4.2.4. IF $|M| \geq \frac{1}{2} m_t$ THEN $b_t := 0$ ELSE $b_t := 1$

4.2.5. $k_t^{(u)} := k_t$

END for t

4.3. IF $\text{Rabin}_N^*\left(a_n \cdot \left\lfloor k_n N + \frac{1}{2} \right\rfloor\right) = \text{Rabin}_N^*(x)$ THEN gib $\left[a_n \cdot \left\lfloor k_n N + \frac{1}{2} \right\rfloor\right]_N$ aus

END for all

ist. Das berechnete $w_{t,i}^*$ ist daher maximal mit Wahrscheinlichkeit $1 - \frac{1}{4}\epsilon(n)$ falsch.

Wir haben zu Beginn die Testpunkte r_i mit $\left(\frac{a_t + r_{t,i}}{N}\right) = -1$ ausgesondert, da wir in diesen

Fällen das Resultat von Algorithmus A^* nicht interpretieren können. Um die gleiche Anzahl von Testpunkten wie im Fall der $\mathrm{RSA}_{N,e}$-Funktion zu erhalten (und somit die gleiche Erfolgswahrscheinlichkeit), muß man zu Beginn mehr Testpunkte erzeugen. Da die Hälfte der Zahlen aus $\mathbb{Z}_N^*$ das Jacobi-Symbol -1 hat (denn $\mathbb{Z}_N(+1)$ ist eine Untergruppe mit Index 2), sollte intuitiv die doppelte Zahl von erzeugten Testpunkten ausreichen. Nach [Peralta92] ist für festes t die Verteilung der quadratischen Reste modulo einer Primzahl in der Folge $a_t + r_{t,i}$ statistisch ununterscheidbar von der Gleichverteilung, so daß wir davon ausgehen können, daß bis auf vernachlässigbare Wahrscheinlichkeit die Hälfte der Wert $a_t + r_{t,i}$ das Jacobi-Symbol $+1$ hat.

Um weiterhin die gleiche Güte bei der Approximation durch $w_{t,i}^*$ zu erhalten, verbessern wir die Approximation von $k^{(u)}$ in Schritt 4 des Algorithmus' 3.4.1 im Vergleich zum Verfahren 2.3.2 von Seite 87 um den Faktor 2. Für die j-te Bitposition wählen wir in Schritt 2:

$$\left(k_0^{(u)}, k^{(v)}, B_0^{(u)}, B^{(v)}\right) \in \tfrac{1}{8}\epsilon(n)^3 \left[0, 8\epsilon(n)^{-3}\right) \times \tfrac{1}{4}\epsilon(n) \left[0, 4\epsilon(n)^{-1}\right) \times \{0,1\}^j \times \{0,1\}^j$$

Die Mehrheitsentscheidung lautet

$$A\left(N, \mathrm{Rabin}_N^*\left((a_t + r_{t,i})x\right)\right) \equiv \left\lfloor \frac{i \cdot [u_{t-1}x]_N + [vx]_N - w_{t,i}^* N}{2^{j-1}} \right\rfloor \pmod 2,$$

wobei die Werte $A\left(N, \mathrm{Rabin}_N^*\left((a_t + r_i)x\right)\right)$ zu Beginn berechnen werden. Wir übertragen Satz 2.3.9 von Seite 92 von der $\mathrm{RSA}_{N,e}$- auf die Rabin_N^*-Funktion:

Satz 3.4.1
Sei $N \in \mathcal{B}_n$ eine Blumzahl und $j := j(n) \geq 1$. Sei A ein Algorithmus zur Bestimmung des j-ten Bits des Urbildes in $S_N(+1)$ mit

$$\mathrm{Ws}\left[A\left(N, \mathrm{Rabin}_N^*\left(U_{S_N(+1)}\right)\right) = \mathrm{bit}_{N,j}\left(U_{S_N(+1)}\right)\right] \geq \tfrac{1}{2} + \epsilon(n),$$

wobei die Wahrscheinlichkeit über $U_{S_N(+1)}$ und die internen Münzwürfe gebildet wird. Falls $\epsilon(n)^{-1} = n^{\mathcal{O}(1)}$ ist, bestimmt für hinreichend große N bzw. n der durch die Mehrheitsentscheidung über eine Stichprobe modifizierte und auf die j-te Bitposition verallgemeinerte Algorithmus 3.4.1 in Zeit

$$2^{7+2j} n \epsilon(n)^{-6} \log_2 n + 2^2 \epsilon(n)^{-2} n \log_2 n \cdot \left(|A| + \mathcal{O}\left(n^3\right)\right)$$

zu $\mathrm{Rabin}_N^*(x)$ ein Urbild mit Wahrscheinlichkeit mindestens $\frac{2}{9}$, wobei die Wahrscheinlichkeit nur über die internen Münzwürfe gebildet wird.

Beweis. Wie Satz 2.3.9 von Seite 92, man beachte nur, daß wir die Approximation von $k^{(u)}$ um den Faktor 2 genauer wählen. Das Berechnen des Jacobi-Symbols gelingt nach Satz 3.3.1 in Zeit $\mathcal{O}\left(n^3\right)$, die Rabin_N^*-Funktion kann in $\mathcal{O}\left(n^2\right)$ Schritten ausgewertet werden. ■

Dieses Resultat kann man auch mit der Methode des binären ggT-Algorithmus' aus Kapitel 2.2 herleiten. Wir raten beide Approximationen um den Faktor 4 besser und fordern, daß die beiden Werte ax, bx jeweils $\frac{1}{8}\epsilon(n)$-klein statt $\frac{1}{2}\epsilon(n)$-klein modulo N sind (vergleiche [ACGS88, Abschnitt 6]). Mit Hilfe der Methode des binären ggT-Algorithmus' erhalten wir in Zeit

$$2^{21,2+2j} n^3 \epsilon(n)^{-6} \left(|A| + \mathcal{O}\left(n^3\right)\right)$$

zu $\mathrm{Rabin}_N^*(x)$ ein Urbild mit Wahrscheinlichkeit für große N bzw. n etwa $2^{-2(2+j)}\epsilon(n)^2$ (vergleiche Satz 2.2.14 auf Seite 77).

Fassen wir unser Resultat über den modifizierten x^2-mod-N-Generator für die Methode der sukzessiven Approximation zusammen:

Korollar 3.4.2
Sei $j := j(n) \geq 1$ mit $j = \mathcal{O}(\log_2 n)$ und $\ell(n) \geq n+1$ durch ein Polynom beschränkt. Falls das vom modifizierten x^2-mod-N-Generator 3.3.1 auf Seite 113 mit Hardcore-Prädikat $\mathrm{bit}_{N,j}(x)$ erzeugte Ensemble des Typs $\ell(n)$ einen statistischen Test T mit nicht-vernachlässigbarer Toleranz $\delta(n)$ nicht besteht, kann man die Rabin_N^*-Funktion für unendlich viele n mit zufälliger Blumzahl $N \in_{\mathrm{R}} \mathcal{B}_n$ in Zeit

$$2^{13+2j} n\delta(n)^{-6}\ell(n)^6 \log_2 n + 2^4\delta(n)^{-2}\ell(n)^2 n \log_2 n \cdot \left(|A| + \ell(n)\cdot\mathcal{O}\left(n^3\right)\right)$$

mit Wahrscheinlichkeit mindestens $\frac{2\cdot\delta(n)}{9\cdot\ell(n)}$ invertieren.

Mit der simultanen Behandlung aller Approximationen wie in Kapitel 2.3.3 können wir für das untersten Bit $j = 1$ die eigentliche Laufzeit für den Fall $\epsilon(n)^{-1} = \Omega(n)$ verbessern. Wie in Korollar 2.3.12 auf Seite 98 folgt: Falls das vom modifizierten x^2-mod-N-Generator erzeugte Ensemble des Typs $\ell(n)$ einen statistischen Test T mit Toleranz $\delta(n) = n^{\mathcal{O}(1)}$ nicht besteht, kann man die Rabin_N^*-Funktion für unendlich viele n und zufällige Blumzahl $N \in_{\mathrm{R}} \mathcal{B}_n$ in Zeit

$$\mathcal{O}\left(n^2\ell(n)^4\delta(n)^{-4}\log_2\left(n\ell(n)\delta(n)^{-1}\right)\right) + 2^3\delta(n)^{-2}\ell(n)^2 n\log_2 n\cdot\left(|T| + \ell(n)\cdot\mathcal{O}\left(n^3\right)\right)$$

mit Wahrscheinlichkeit mindestens $\frac{2\cdot\delta(n)}{3\cdot\ell(n)}$ invertieren.

Wie im Fall von Korollar 2.3.6 auf Seite 90 ist für die simultane Sicherheit der untersten Bits zu beachten, daß wir Algorithmus A zur Vorhersage des j-ten Bits nicht mehr zu Beginn aufrufen, da er zusätzlich die untersten $j-1$ Bits als Eingabe erhält, die man aus den Approximationen erhält:

Korollar 3.4.3
Sei $N \in \mathcal{B}_n$ eine Blumzahl und $j := j(n) \geq 1$ mit $j(n) = \mathcal{O}(\log_2 n)$. Sei A ein Algorithmus zur Vorhersage des j-ten Bits des Urbildes mit

$$\mathrm{Ws}\left[A\left(N, \mathrm{Rabin}_N^*\left(U_{S_N(+1)}\right), \mathrm{Bits}_{N,j-1}\left(U_{S_N(+1)}\right)\right) = \mathrm{bit}_{N,j}\left(U_{S_N(+1)}\right)\right] \geq \tfrac{1}{2} + \epsilon(n),$$

wobei die Wahrscheinlichkeit über $U_{S_N(+1)}$ und die internen Münzwürfe gebildet wird. Falls $\epsilon(n)^{-1} = n^{\mathcal{O}(1)}$ ist, kann man für hinreichend große N bzw. n in Zeit

$$2^{7+2j} n\epsilon(n)^{-6}\log_2 n\cdot\left(|A| + \mathcal{O}\left(n^3\right)\right)$$

zu $\mathrm{Rabin}_N^*(x)$ ein Urbild mit Wahrscheinlichkeit mindestens $\frac{2}{9}$ bestimmen, wobei die Wahrscheinlichkeit nur über die internen Münzwürfe gebildet wird.

3.5 Sicherheitsbeweis

In diesem Kapitel führen wir den Sicherheitsbeweis für den ursprünglichen x^2-mod-N-Generator 3.1.1 auf Seite 107, d.h. man wählt einen zufälligen Startwert $x \in \mathrm{QR}_N$ und gibt iterativ das unterste Bit aus. Wir betrachten den Fall $j = 1$, also das unterste Bit. Sei A ein Algorithmus, der zu gegebenem $\mathrm{Rabin}_N(z)$ mit $z \in \mathrm{QR}_N$ einen $\epsilon(n)$-Vorteil zur Vorhersage von $\mathrm{bit}_{N,1}(z)$ hat. Zur Vereinfachung sei zunächst für die betrachteten n die Erfolgswahrscheinlichkeit von Algorithmus A unabhängig von N sei. Mit Hilfe von A wollen wir bezüglich n und $\epsilon(n)^{-1}$ zu gegebenem Funktionswert $\mathrm{Rabin}_N(x)$ eine Wurzel modulo N von x^2 berechnen. Wir setzen $\epsilon(n)^{-1} = n^{\mathcal{O}(1)}$ voraus, um für hinreichend große N bzw. n die Wahrscheinlichkeit, daß ein modulo N gleichverteilter Wert nicht teilerfremd zu N ist, zu vernachlässigen.

Unser Ansatz ist wie in Kapitel 3.4, allerdings kann man im Gegensatz zur Menge $S_N(+1)$ nicht mit Hilfe der Approximationen entscheiden, ob ein $a_t + r_{t,i}$ mit Jacobi-Symbol $+1$ auch in QR_N liegt, also ein quadratischer Rest modulo N ist. Diese Information benötigen wir jedoch, um für $a_t + r_{t,i}$ mit Jacobi-Symbol $+1$ das Resultat von Algorithmus A bei Eingabe von $\mathrm{Rabin}_N\left((a_t + r_{t,i})x\right)$ korrekt zu interpretieren:

- Falls $a_t + r_{t,i} \in \mathrm{QR}_N$ ist, bezieht sich das Resultat auf $(a_t + r_{t,i})x$.
- Falls $a_t + r_{t,i} \notin \mathrm{QR}_N$ ist, bezieht sich das Resultat auf $-(a_t + r_{t,i})x$.

Für den Fall $a_t + r_{t,i} \notin \mathrm{QR}_N$ haben wir in Satz 3.3.4 gezeigt, daß man das Resultat von A invertiert, damit sich das Ergebnis auf das unterste Bit von $(a_t + r_{t,i})x$ bezieht. Im Satz 3.3.4 auf Seite 114 haben wir gesehen, daß mit Algorithmus A zu bestimmen ist, ob es sich bei einem gegebenen $z \in \mathbb{Z}_N(+1)$ um einen quadratischen Rest modulo N handelt. Durch eine Mehrheitsentscheidung, wie in Kapitel 3.3 beschrieben, entscheiden wir mit je $2\epsilon(n)^{-2} \log_2 \epsilon(n)^{-1}$ Aufrufen von A mit Fehlerwahrscheinlichkeit $\frac{1}{8}\epsilon(n)$, ob $z \in \mathrm{QR}_N$ ist. Wir übertragen Algorithmus 3.4.1 von Seite 117 auf die Rabin_N- statt Rabin_N^*- und modifizieren ihn mit der Mehrheitsentscheidung über eine Stichprobe. Damit die Fehlerwahrscheinlichkeit jeder Entscheidung bei $\frac{1}{2} - \frac{3}{4}\epsilon(n)$ bleibt, wählen wir jede Approximation um den Faktor 2 besser. Aus Satz 3.4.1 folgt, daß man in Zeit

$$2^{11} n \epsilon(n)^{-6} \log_2 n + 2^3 \epsilon(n)^{-4} n^2 \log_2 n \cdot \log_2 \epsilon(n)^{-1} \cdot \left(|A| + \mathcal{O}\left(n^3\right)\right)$$

ein Urbild mit Wahrscheinlichkeit mindestens $\frac{2}{9}$ erhält.

Wir wollen die Anzahl der Aufrufe von Algorithmus A reduzieren, indem man nicht für jeden Wert bei der Mehrheitsentscheidung überprüft, ob $a_t + r_{t,i}$ ein quadratischer Rest ist. Sei zunächst 2 modulo N ein quadratischer Rest. Da QR_N eine Gruppe ist, sind mit u_0 auch $a_t = u_t = 2^{-t}u_0$ für $t = 1, 2, \ldots, n$ quadratische Reste modulo N. Wir wollen in jeder Iteration auch v mit 2^{-1} modulo N multiplizieren, sei v_0 das ursprüngliche v und $v_{t-1} := 2^{-(t-1)}v_0$ der Wert in der t-ten Iteration. Wir verwenden als Testpunkte:

$$r_{t,i} := iu_{t-1} + v_{t-1} = 2^{-(t-1)}\left(iu_0 + v_0\right) = 2^{-(t-1)} r_{1,i}$$

Zu Beginn bestimmt man, für welche i die Summe $a_0 + iu_0 + v_0$ ein quadratischer Rest modulo N ist und übernimmt diese Information für die späteren Iterationen. Für die Mehrheitsentscheidung benötigen wir das unterste Bit von $v_{t-1}x$, kennen aber nur das unterste Bit von

$v_0 x$. Nachdem wir in Iteration t das unterste Bit von $a_t x = u_t x$ bestimmt haben, ermitteln wir anschließend das unterste Bit von $v_t x$ ebenfalls mit einer Mehrheitsentscheidung über die Testpunkte

$$\widehat{r}_{t,i} := iv_{t-1} + u_t = 2^{-(t-1)} \left(iv_0 + u_0\right),$$

d.h. man „vertauscht die Rollen“ von v_{t-1} und u_{t-1}. Um die gleiche Fehlerwahrscheinlichkeit bei der Berechnung $\widehat{w}_{t,i}$ zu erreichen, müssen wir $k_0^{(u)}$ aus dem gleichen Intervall wie $k_0^{(v)}$ wählen, so daß die eigentliche Laufzeit um den Faktor $\epsilon(n)^{-2}$ steigt.

Wir verfolgen daher einen anderen Ansatz. Nach Bestimmung des untersten Bits von $u_t x$ verwenden wir Testpunkte $\widehat{r}_{t,i} := iu_t$, um das unterste Bit von $v_t x$ zu ermitteln, d.h. wir rufen Algorithmus A mit $\mathrm{Rabin}_N\left((v_t + \widehat{r}_{t,i})x\right)$ auf. Die Werte $v_t + \widehat{r}_{t,i}$ sind paarweise unabhängig, da die Transformationsmatrix

$$\begin{bmatrix} v_t + \widehat{r}_{t,i} \\ v_t + \widehat{r}_{t,j} \end{bmatrix} \equiv \begin{bmatrix} i & 1 \\ j & 1 \end{bmatrix} \cdot \begin{bmatrix} u_t \\ v_t \end{bmatrix} \pmod{N}$$

die Determinante $i - j \neq 0$ hat. Mit Wahrscheinlichkeit mindestens $1 - \frac{1}{4}\epsilon(n)$ erfüllt

$$\widehat{w}_{t,i} := \left\lfloor k_t^{(v)} + i \cdot k_t^{(u)} \right\rfloor$$

die Kongruenz:

$$\left[(v_t + \widehat{r}_{t,i})x\right]_N \equiv \left[(iu_t + v_t)x\right]_N \equiv \left[v_t x\right]_N + i \cdot \left[u_t x\right]_N - \widehat{w}_{t,i} \pmod 2$$

Wenn wir mit $\widehat{b}_{t,i}$ das Resultat von Algorithmus A zum Argument $\mathrm{Rabin}_N\left((v_t + \widehat{r}_{t,i})x\right)$ bezeichnen, lautet unsere Mehrheitsentscheidung, daß wenn für mehr als die Hälfte aller i

$$\widehat{b}_{t,i} \equiv i \cdot \left[u_t x\right]_N - \widehat{w}_{t,i} \pmod 2$$

gilt, das unterste Bit von $v_t x$ modulo N Null ist.

Da wir in jeder Iteration zwei Mehrheitsentscheidungen treffen, steigt die Laufzeit und die Fehlerwahrscheinlichkeit der Iterationen im Vergleich zur Rabin_N^*-Funktion um den Faktor 2. Aus Satz 2.3.9, insbesondere aus Abschätzung (2.27) auf Seite 93, folgt für hinreichend großes n, daß die Laufzeit für jede Möglichkeit maximal $8\epsilon(n)^{-2} n \log_2 n$ ist.

Zu Beginn wählen wir aus zweimal $2 \cdot 2n\epsilon(n)^{-2}$ Testpunkten jeweils die Hälfte geeigneter Punkte aus und bestimmen durch je $2\epsilon(n)^{-2} \log_2\left(n\epsilon(n)^{-1}\right)$ Aufrufe von Algorithmus A, wie dann in den Iterationen die Resultate von Algorithmus A zu interpretieren sind (es würde genügen, nur die Stichprobe zu betrachten). Aus der additiven Chernoff-Schranke (siehe Fakt 2.3.7 auf Seite 91) erhalten wir als obere Schranke für die Fehlerwahrscheinlichkeit der Interpretationen:

$$2n\epsilon(n)^{-2} \cdot e^{-4 \cdot \log_2\left(n\epsilon(n)^{-1}\right)} \leq \frac{2n\epsilon(n)^{-2}}{n^4 \epsilon(n)^{-4}} \leq \frac{1}{n^3}$$

Unsere Fehlerabschätzung für das gesamte Verfahren bleibt mit diesem zusätzlichen Summanden gültig. Es folgt als Gesamtlaufzeit des Verfahrens für hinreichend großes n:

$$\begin{aligned} &8\epsilon(n)^{-2} n \log_2 n \cdot 2^7 \epsilon(n)^{-4} + \left[2^3 n\epsilon(n)^{-4} \log_2\left(n\epsilon(n)^{-1}\right) + 2^3 \epsilon(n)^{-2} n \log_2 n\right] \cdot \left(|A| + \mathcal{O}\left(n^3\right)\right) \\ &\leq 2^{10} \epsilon(n)^{-6} n \log_2 n + 2^3 \epsilon(n)^{-4} n \log_2\left(n\epsilon(n)^{-1}\right) \cdot \left(|A| + \mathcal{O}\left(n^3\right)\right) \end{aligned}$$

Wir haben bisher vorausgesetzt, daß 2 ein quadratischer Rest modulo N ist. Dies gilt nach dem zweiten quadratischen Ergänzungsgesetz nur für etwa ein Viertel der Blumzahlen. In jedem Fall ist $4 = 2^2$ modulo N ein quadratischer Rest, so daß wir für gerade und ungerade Iterationen t jeweils getrennt geeignete Testpunkte wählen. Dazu wählen wir aus viermal $2 \cdot 2n\epsilon(n)^{-2}$ Testpunkten jeweils die geeigneten Punkte aus. Ob 2 ein quadratischer Rest modulo N ist, bestimmt man zu Beginn mit Hilfe von Algorithmus A. Wir haben gezeigt:

Satz 3.5.1
Sei $N \in \mathcal{B}_n$ eine Blumzahl und A ein Algorithmus zur Bestimmung des untersten Bits des Urbildes in QR_N mit

$$\mathrm{Ws}\left[\, A\left(N, \mathrm{Rabin}_N\left(U_{\mathrm{QR}_N}\right)\right) = \mathrm{lsb}_N\left(U_{\mathrm{QR}_N}\right)\right] \geq \tfrac{1}{2} + \epsilon(n),$$

wobei die Wahrscheinlichkeit über U_{QR_N} und die internen Münzwürfe gebildet wird. Falls $\epsilon(n)^{-1} = n^{\mathcal{O}(1)}$ ist, kann man für hinreichend große N bzw. n in Zeit

$$2^{10}\epsilon(n)^{-6} n \log_2 n + 2^3 \epsilon(n)^{-4} n \log_2\left(n\epsilon(n)^{-1}\right) \cdot \left(|A| + \mathcal{O}\left(n^2\right)\right)$$

zu $\mathrm{Rabin}_N(x)$ ein Urbild mit Wahrscheinlichkeit mindestens $\frac{1}{9}$ bestimmen, wobei die Wahrscheinlichkeit nur über die internen Münzwürfe gebildet wird.

Mit der simultanen Behandlung aller Approximationen wie in Kapitel 2.3.3 können wir für das unterste Bit $j = 1$ die eigentliche Laufzeit für den Fall $\epsilon(n)^{-1} = \Omega(n)$ verbessern. Wie in Korollar 2.3.12 auf Seite 98 folgt: Man kann die Rabin_N-Funktion für unendlich viele n und zufällige Blumzahl $N \in_{\mathrm{R}} \mathcal{B}_n$ in Zeit

$$\mathcal{O}\left(n^2\epsilon(n)^{-4}\log_2\left(n\epsilon(n)^{-1}\right)\right) + 2^3\epsilon(n)^{-4} n \log_2\left(n\epsilon(n)^{-1}\right) \cdot \left(|A| + \mathcal{O}\left(n^3\right)\right)$$

bzw. für $|A| = \Omega\left(n^3\right)$ in Zeit $\mathcal{O}\left(n\epsilon(n)^{-4}\log_2\left(n\epsilon(n)^{-1}\right) \cdot |A|\right)$ mit Wahrscheinlichkeit mindestens $\frac{2}{3}$ invertieren. Der bisher bekannte Sicherheitsbeweis, der die Sicherheit des x^2-mod-N-Pseudozufallsgenerator auf Faktorisierung des Moduls zurückführt, stammt von U.V. Vazirani und V.V. Vazirani [VV84]. Er verwendet die Methode des binären ggT-Algorithmus' aus Kapitel 2.2 bzw. aus der Arbeit [ACGS88] und beschränkt sich bei der Mehrheitsentscheidung auf geeignete Testpunkte. Nach [Knuth96] liefert dieses Verfahren in $\mathcal{O}\left(n^3\epsilon(n)^{-9}|A|\right)$ Schritten mit Wahrscheinlichkeit etwa $\frac{\epsilon(n)^2}{64}$ ein Urbild. Aus Satz 3.5.1 folgt für den x^2-mod-N-Generator:

Korollar 3.5.2
Sei $\ell(n) \geq n+1$ durch ein Polynom beschränkt. Falls das vom x^2-mod-N-Generator 3.1.1 auf Seite 107 erzeugte Ensemble des Typs $\ell(n)$ einen statistischen Test T mit nicht-vernachlässigbarer Toleranz $\delta(n)$ nicht besteht, kann man die Rabin_N-Funktion für unendlich viele n mit zufälliger Blumzahl $N \in_{\mathrm{R}} \mathcal{B}_n$ in Zeit

$$2^{10}\delta(n)^{-6}\ell(n)^6 n \log_2 n + 2^3 n\delta(n)^{-4}\ell(n)^4 \log_2\left(n\delta(n)^{-1}\ell(n)\right) \cdot \left(|A| + \mathcal{O}\left(n^3\right)\right)$$

mit Wahrscheinlichkeit mindestens $\frac{\delta(n)}{9\cdot\ell(n)}$ invertieren.

Diese Resultate kann man auf das j-te unterste Bit mit $j := j(n) = \mathcal{O}(\log_2 n)$ übertragen. Zu beachten ist, daß Satz 3.3.4 auf Seite 114 für das unterste Bit gilt. Diesen Satz können wir nicht unmittelbar auf die j-te Bitposition übertragen, da für $j > 1$

$$\text{bit}_{j,N}(z) \neq \text{bit}_{j,N}(N - z) \tag{3.4}$$

im allgemeinen nicht gilt. Anhand des Kriteriums (3.4) teilt man $\mathbb{Z}_N(+1)$ in zwei Mengen auf. Zu Beginn bestimmen wir mit vernachlässigbarer Abweichung durch polynomiell viele Tests, welche Vorteile $\epsilon_1(N)$ und $\epsilon_2(N)$ Algorithmus A auf den beiden Mengen hat. Sei $c(N)$ der Anteil der ersten Menge an $\mathbb{Z}_N(+1)$. Es gilt:

$$\epsilon(n) \leq c(N) \cdot \epsilon_1(N) + (1 - c(N)) \cdot \epsilon_2(N)$$

Entweder ist $c(N) = 0$ oder $c(N) \geq 2^{-j}$, so daß mindestens einer der beiden Vorteile gleich $2^{-j}\epsilon(n)$ ist, d.h. insbesondere für $j = \mathcal{O}(\log_2 n)$ nicht-vernachlässigbar. Wir beschränken uns bei der Mehrheitsentscheidung auf die Menge mit maximalem Vorteil. Falls für diese Werte (3.4) gilt, verfahren wir wie beim untersten Bit. Im anderen Fall spielt es bei einem Argument $z \in \mathbb{Z}_N(+1)$ keine Rolle, ob es ein quadratischer Rest ist, sondern nur, ob $\text{bit}_{j,N}(z) = \text{bit}_{j,N}(N - z)$ gilt.

Die simultane Sicherheit kann man mit Hilfe der XOR-Bedingung bzw. Korollar 1.5.12 auf Seite 36 genauso beweisen. Statt des j-ten Bits betrachtet man die XOR-Verknüpfung der betreffenden, unteren Bitpositionen:

Korollar 3.5.3
Sei $N \in \mathcal{B}_n$ eine Blumzahl und $j := j(n) \geq 1$ mit $j(n) = \mathcal{O}(\log_2 n)$. Sei A ein Algorithmus zur Vorhersage des j-ten Bits des Urbildes mit

$$\text{Ws}\left[A\left(N, \text{Rabin}_N\left(U_{\text{QR}_N}\right), \text{Bits}_{N,j-1}\left(U_{\text{QR}_N}\right)\right) = \text{bit}_{N,j}\left(U_{\text{QR}_N}\right)\right] \geq \tfrac{1}{2} + \epsilon(n),$$

wobei die Wahrscheinlichkeit über U_{QR_N} und die internen Münzwürfe gebildet wird. Falls $\epsilon(n)^{-1} = n^{\mathcal{O}(1)}$ ist, kann man für hinreichend große N bzw. n in Zeit $(n\epsilon(n))^{\mathcal{O}(1)}$ zu $\text{Rabin}_N(x)$ ein Urbild mit nicht-vernachlässigbarer Wahrscheinlichkeit bestimmen, wobei die Wahrscheinlichkeit nur über die internen Münzwürfe gebildet wird.

Algorithmenverzeichnis

Index

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Literaturverzeichnis

[ACGS88] W. Alexi, B. Chor, O. Goldreich und C.P. Schnorr (1988): **RSA and Rabin Functions: Certain Parts are as hard as the whole**, SIAM Journal on Computing, Band 17, Nr. 2 (April), Seiten 194–209.

[BCS83] M. Ben-Or, B. Chor und A. Shamir (1983): **On the cryptographic Security of single RSA Bits**, Proceedings of the 15.th Symposium on Theory of Computing (STOC), ACM Press, New York, Seiten 421–430.

[Blum81] M. Blum (1981): **Coin-Flipping by Telephone — A Protocol for Solving impossible Problems**, IEEE Proceedings Spring Compcon, IEEE Computer Society Press, Washington D.C., Seiten 133–137.

[BBS86] L. Blum, M. Blum und M. Shub (1986): **A simple unpredictable Random Number Generator**, SIAM Journal on Computing, Band 15, Nr. 2 (Mai), Seiten 364–383.

[BM84] M. Blum und S. Micali (1984): **How to generate cryptographically strong Sequences of Pseudo-Random Bits**, SIAM Journal on Computing, Band 13, Nr. 4 (November), Seiten 850–864.

[Boyar89] J. Boyar (1989): **Inferring Sequences produced by Pseudo-random Number Generators**, Journal of the ACM, Band 36, Nr. 1, Seiten 129–141.

[CG89] B. Chor und O. Goldreich (1989): **On the Power of two Point based Sampling**, Journal of Complexity, Band 5, Seiten 96–106.

[Cusick95] T.W. Cusick (1995): **Properties of the x^2 mod N Pseudorandom Number Generator**, IEEE Transaction on Information Theory, Band IT-41, Nr. 4 (Juli), Seiten 1155–1159.

[DH76] W. Diffie und M.E. Hellman (1976): **New Direction in Cryptography**, IEEE Transaction Information Theory, Band 22, Seiten 644–654.

[FSt96] J.B. Fischer und J. Stern (1996): **An efficient Pseudo-Random Generator provably as secure as Syndrom Decoding**, Advances in Cryptography — Proceedings Eurocrypt '96, Lecture Notes in Computer Science (LNCS), Band 1070, Springer-Verlag, Berlin/Heidelberg, Seiten 245–255.

[FSch97] R. Fischlin und C.P. Schnorr (1997): **Stronger Security Proofs for RSA and Rabin Bits**, Advances in Cryptography — Proceedings Eurocrypt '97, Lecture Notes in Computer Science (LNCS), Band 1233, Springer-Verlag, Berlin/Heidelberg, Seiten 267–279.

[GGM86] O. Goldreich, S. Goldwasser und S. Micali (1986): **How to construct Random Functions**, Journal of the ACM, Band 33, Nr. 4 (Oktober), Seiten 692–807.

[GL89] O. Goldreich und L.A. Levin (1989): **A Hard Core Predicate for any One Way Function**, Proceedings of the 21.st Symposium on Theory of Computing (STOC), ACM Press, New York, Seiten 25–32.

[Goldreich95a] O. Goldreich (1995): **Foundations of Cryptography (Fragments of a Book)**, Weizmann Institute of Science, Rehovot, Israel.

[Goldreich95b] O. Goldreich (1995): **Three XOR-Lemmas — An Exposition**, Electronic Colloquium on Computer Complexity, ECCC TR95-056.

[GNW95] O. Goldreich, N. Nisan und A. Wigderson (1995): **On Yao's XOR-Lemma**, Electronic Colloquium on Computer Complexity, ECCC TR95-050 mit Comment 01.

[GM96] O. Goldreich und B. Meyer (1996): **Computational Indistinguishability: Algorithms vs. Circuits**, Electronic Colloquium on Computer Complexity, ECCC TR96-067.

[GMT82] S. Goldwasser, S. Micali und P. Tong (1982): **Why and how to establish a Private Code on a Public Network**, 23.rd Annual IEEE Symposium on Foundation of Computer Science (FOCS), IEEE Computer Society Press, Los Alamitos, Seiten 134–144.

[IN96] R. Impagliazzo und M. Noar (1996): **Efficient cryptographic Schemes provably as secure as Subset Sum**, Journal of Cryptology, Band 9, Nr. 4 (Herbst), Seiten 199–216.

[HILL91] J. Håstad, R. Impagliazzo, L.A. Levin und M. Luby (1991): **Construction of a Pseudo-Random Generator from any One-Way Function**, Technischer Report Nr. 91-068, International Computer Science Institute (ICSI), Berkeley (USA). Eingereicht beim SIAM Journal on Computing — neue Version im WWW auf R. Impagliazzos Homepage `http://www.cse.ucsd.edu/users/russell/` erhältlich.

[HSS93] J. Håstad, A.W. Schrift und A. Shamir (1993): **The discrete Logarithm modulo a Composite hildes $\mathcal{O}(n)$ Bits**, Journal of Computer and System Science, Band 47, Seiten 376–404.

[KR95] B. Kaliski und M. Robshaw (1995): **The Secure Use of RSA**, CryptoBytes, RSA Laboratories, Band 1, Nr. 2 (Herbst), Seite 7–13 — im WWW auf `http://www.rsa.com/rsalabs/pub/cryptobytes.html` erhältlich.

[Knuth81] D.E. Knuth (1981): **The Art of Computer Programming**, Band II: Seminumercial Algorithms, 2. Auflage, Addison Wesley, Reading, neue Auflage in Vorbereitung.

[Knuth96] D.E. Knuth (1996): **Pseudorandom Numbers**, Änderungen zu „The Art of Computer Programming, Band II: Seminumercial Algorithms“, Seiten 31–38 (22. April 1996) — aktuelle Version im WWW auf D.E. Knuths Homepage `http://www-cs-staff.Stanford.EDU/~uno/taocp.html` erhältlich.

[Kranakis86] E. Kranakis (1986): **Primality and Cryptography**, John Wiley & Sons, New York.

[Lenstra87] H.W. Lenstra, Jr. (1987): **Factoring Integers with Ellipics Curves**, Annals of Mathematics, Band 126, Seiten 649–673.

[LenLen90] A.K. Lenstra und H.W. Lenstra, Jr. (1990): **Algorithms in Number Theory**, in Handbook of Theoretical Computer Science (Hrsg. J. van Leeuven), Band A, Elsevier Science Publishers B.V. Amsterdam, Seiten 673–715.

[LenLen93] A.K. Lenstra und H.W. Lenstra, Jr. (1993): **The Development of the Number Field Sieve**, Lecture Notes in Mathematics (LNM), Band 1554, Springer-Verlag, Berlin/Heidelberg.

[Levin87] L.A. Levin (1987): **One Way Functions and Pseudorandom Generators**, Combinatorica, Band 7, Nr. 4, Seiten 357–363.

[Levin93] L.A. Levin (1993): **Randomness and Nondeterminism**, Journal on Symbolic Logic, Band 58, Seiten 1102–1103.

[Levin96] L.A. Levin (1996): **Fundamentals of Computing**, Skript — im WWW auf L.A. Levins Homepage `http://www.cs.bu.edu/faculty/lnd/` erhältlich.

[LW88] D.L. Long und A. Widgerson (1988): **The discrete Logarithms hides $\mathcal{O}(\log_2 n)$ Bits**, SIAM Journal on Computing, Band 17, Nr. 2 (April), Seiten 363–372.

[Luby96] M. Luby (1996): **Pseudorandomness and cryptographic Applications**, Princeton University Press, Princeton.

[Maurer95] U. Maurer (1995): **Fast Generation of Prime Numbers and secure Public-Key cryptographic Parameters**, Journal of Cryptology, Band 8, Nr. 3 (Sommer), Seiten 123–155.

[Naor91] M. Naor (1991): **Bit Commitment Using Pseudorandomness**, Journal of Cryptology, Band 4, Nr. 3 (Herbst), Seiten 151–158.

[NR95] M. Naor und O. Reingold (1995): **Synthesizer and their Application to the Parallel Construction of Pseudo-Random Functions**, 36.th Annual IEEE Symposium on Foundation of Computer Science (FOCS), IEEE Computer Society Press, Los Alamitos, Seiten 170–181.

[Odlyzko95] A.M. Odlyzko (1995): **The Future of Integer Factorization**, CryptoBytes, RSA Laboratories, Band 1, Nr. 2 (Sommer), Seite 5–12 — im WWW auf `http://www.rsa.com/rsalabs/pub/cryptobytes.html` erhältlich.

[Peralta85] R. Peralta (1985): **Simultaneous Security of Bits in the Discret Log.,**, Advances in Cryptography — Proceedings Eurocrypt '85, Lecture Notes in Computer Science (LNCS), Band 219 (1986), Springer-Verlag, Berlin/Heidelberg, Seiten 62–72.

[Peralta92] R. Peralta (1992): **On the Distribution of quadratic Residues and Nonresidues modulo a Prime Number**, Mathematics of Computation, Volume 58, Nr. 172, Seiten 433–440.

[Pom84] C. Pomerance (1984): **The Quadratic Sieve Factoring Algorithm**, Advances in Cryptography — Proceedings Eurocrypt '84, Lecture Notes in Computer Science (LNCS), Band 209 (1985), Springer-Verlag, Berlin/Heidelberg, Seiten 169–182.

[Rabin79] M.O. Rabin (1979): **Digital Signatures and Public-Key Functions as intractable as Factorization**, MIT Laboratory for Computer Science Technical Report LCS-TR 212, Massachusetts Institute of Technology.

[RSA78] R.L. Rivest, A. Shamir und L. Adleman (1978): **A Method for Obtaining Digital Signatures and Public Key Cryptosystems**, Communication of the ACM, Band 21, Nr. 2 (Februar), Seiten 120–126.

[Scheid94] H. Scheid (1994): **Zahlentheorie**, 2. Auflage, BI Wissenschaftsverlag, Mannheim/Leipzig/Wien/Zürich.

[Schnorr97] C.P. Schnorr (1997): private Korrespondenz (3 Seiten).

[Shamir81] A. Shamir (1981): **On the Generation of cryptographically strong Pseudo-Random Number Sequences**, 8.th International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Computer Science (LNCS), Band 62, Springer-Verlag, Berlin/Heidelberg, Seiten 544–550.

[Shamir83] A. Shamir (1983): **On the Generation of cryptographically strong Pseudo-Random Sequences**, ACM Transaction on Computer Systems, Band 1, Nr. 1 (Februar), Seiten 38–44.

[Sil97] R.D Silverman (1997): **Fast Generation of Random, Strong RSA Primes**, CryptoBytes, RSA Laboratories, Band 3, Nr. 1 (Frühjahr), Seite 9–13 — im WWW auf `http://www.rsa.com/rsalabs/pub/cryptobytes.html` erhältlich.

[SoSt77] R. Solovay und V. Strassen (1977): **A fast Monte-Carlo Test for Primality**, SIAM Journal on Computing, Band 6, Nr. 1 (März), Seiten 84–85.

[SoSt78] R. Solovay und V. Strassen (1978): **Erratum: A fast Monte-Carlo Test for Primality**, SIAM Journal on Computing, Band 6, Nr. 1 (Februar), Seite 118.

[Wiener90] M.J. Wiener (1990): **Cryptanalysis of Short RSA Secret Exponents**, IEEE Transaction Information Theory, Band IT-36, Nr. 3 (Mai), Seiten 553–558.

[VV84] U.V. Vazirani und V.V. Vazirani (1984): **Efficient and Secure Pseudo-Random Number Generator**, 25.th Annual IEEE Symposium on Foundation of Computer Science (FOCS), IEEE Computer Society Press, Los Alamitos, Seiten 458–463.

[Yao82] A. Yao (1982): **Theory and Application of Trapdoor Functions**, 23.rd Annual IEEE Symposium on Foundation of Computer Science (FOCS), IEEE Computer Society Press, Los Alamitos, Seiten 80–91.