4 Perfekte Zufallsgeneratoren

Anfang der 1980-er Jahre entstand im Umkreis der Kryptologie eine Vorstellung davon, wie man die Unvorhersagbarkeit eines Zufallsgenerators modellieren könnte, nämlich komplexitätstheoretisch: Die Vorhersage soll hart sein, d. h., auf ein bekanntes hartes Problem zurückgeführt werden können. Dadurch wurde ein neuer Qualitätsstandard gesetzt, der allerdings auf der mathematisch völlig unbewiesenen Grundlage aufbaut, dass es für gewisse zahlentheoretische Probleme wie die Primzerlegung keine effizienten Algorithmen gibt. – Die Situation ist also die gleiche wie bei der asymmetrischen Verschlüsselung.

Interessanterweise stellte sich bald heraus, dass die scheinbar viel stärkere Forderung, die erzeugte Zufallsfolge solle sich durch *keinen* effizienten Algorithmus von einer echten Zufallsfolge unterscheiden lassen, zur Unvorhersagbarkeit äquivalent ist (Satz von Yao). Auf der theoretischen Seite ist damit ein sehr gutes Modell für Zufallsgeneratoren vorhanden, die statistisch absolut einwandfrei und kryptologisch unangreifbar sind.

Die ersten konkreten Ansätze, von denen hier der BBS- (= Blum/Blum/Shub-) Generator behandelt wird, lieferten Generatoren, die für den praktischen Einsatz noch viel zu langsam waren. Es werden auch verschiedene neuere Ansätze vorgestellt, zu einigermaßen schnellen kryptographisch sicheren Zufallsgeneratoren zu kommen.

4.1 Der Blum-Blum-Shub-Generator

Der Blum-Blum-Shub-**Generator** oder **BBS-Generator** setzt bei der in Kapitel III vorgestellten Quadratrest-Vermutung an und funktioniert so: Als ersten Schritt wählt man eine große zufällige Blum-Zahl m; weitere Einschränkungen werden – wie auch beim RSA-Verfahren – bei einer wirklich zufälligen Wahl nicht mehr für nötig gehalten, schaden aber auch nicht.

Als zweites wählt man dann einen (zufälligen) Startwert $x_0 \in [\lceil\sqrt{m}\,\rceil \ldots m - \lceil\sqrt{m}\,\rceil]$, der zu m teilerfremd ist. [Falls x_0 nicht zu m teilerfremd ist, hat man m per Zufall faktorisiert. Dass das vorkommt, ist äußerst unwahrscheinlich. Auch die Einschränkung $\sqrt{m} < x_0 < m - \sqrt{m}$ ist bei wirklich zufälliger Wahl unnötig und wird für die folgende Theorie nicht verwendet.]

Nun kann man an die Erzeugung einer Zufallsfolge gehen: Man bildet die Folge $x_i = x_{i-1}^2 \bmod m$. Ausgegeben wird aber *nur das letzte Bit* $b_i = \mathrm{lsb}(x_i)$. Außer eventuell dem Startwert x_0 sind alle x_i Quadratreste.

Die Zahlen p und q werden nur zur Bildung von m gebraucht und dann vernichtet; insbesondere sind sie als Geheimnis des Generators zu behandeln. Ebenso bleiben alle nicht ausgegebenen Bits der Folgenglieder x_i geheim.

Das Programm zur Parameter-Erzeugung für den BBS-Generator besteht aus folgenden Prozeduren:

Prozedur BlumPrime

[Erzeugt die kleinste Primzahl $p \geq x$, für die auch $\frac{p-1}{2}$ prim ist.]
Eingabeparameter:
x = Ausgangswert.
Ausgabeparameter:
p = kleinste Primzahl $\geq x$ mit $\frac{p-1}{2}$ prim.
Anweisungen:
Setze $p = x$.
Falls p gerade, erhöhe p um 1.
Falls $\frac{p-1}{2}$ gerade, erhöhe p um 2.
Solange ($\frac{p-1}{2}$ nicht prim) und (p nicht prim)
    erhöhe p um 4.

Aus dem Abschnitt über den Primzahlsatz wissen wir, dass es wohl sogar sehr viele der gesuchten Zahlen gibt. Der Algorithmus ist somit „empirisch“ auch ohne eine künstliche Abbruchbedingung korrekt. Im übrigen wird für ‘BlumPrime’ ein Primzahltest benötigt, wie sie auch schon behandelt wurden.

Prozedur BlumRandomPrime

[Erzeugt eine zufällige Primzahl mit n Bits, für die auch $\frac{p-1}{2}$ prim ist.]
Eingabeparameter:
n = Zahl der gewünschten Bits.
Ausgabeparameter:
p = eine Primzahl mit $2^{n-1} < p < 2^n$ und $\frac{p-1}{2}$ prim.
Anweisungen:
Bilde eine Zufallszahl x mit $2^{n-1} \leq x < 2^n$.
Setze $p = \text{BlumPrime}(x)$.
Falls $p \geq 2^n$, setze $x = 2^{n-1}$ und $p = \text{BlumPrime}(x)$.

Die Korrektheit dieses Algorithmus ist ebenfalls nur empirisch gesichert; der Fall, dass keine Primzahl gefunden wird, kommt in der Praxis aber nicht vor. Ein ernstzunehmender Einwand ist allerdings, dass der Algorithmus die verschiedenen Primzahlen mit ungleicher Wahrscheinlichkeit ausspuckt. Die Wahrscheinlichkeit für eine bestimmte Primzahl ist nämlich proportional zur Differenz zur nächstkleineren derartigen Primzahl (wobei die Differenz $\bmod 2^{n-1}$ zu bilden ist, wenn man den Übergang am Ende von 2^n zu 2^{n-1} auch noch angemessen berücksichtigen will). Auch ändert sich die Dichte der Primzahlen zwischen 2^{n-1} und 2^n fast um den Faktor 2, wie die Überlegungen zum Primzahlsatz gezeigt haben. Da aber kein Verfahren bekannt ist, diese Ungleichverteilung zuungunsten des BBS-Generators oder ähnlicher Verfahren auszunutzen, soll der Algorithmus hier nicht verkompliziert werden.

Für den Algorithmus 'BlumRandomPrime' wird außerdem ein Zufallsgenerator zur Bildung des Ausgangswerts x benötigt; hierfür sollte man auf einen willkürlichen Wert, einen „echten" Zufallswert zurückgreifen, z. B. einen aus einer genügend langen, vom Benutzer eingegeben Passphrase gebildeten.

Prozedur BlumNumber

[Erzeugt eine zufällige Blum-Zahl mit n oder $n+1$ Bits,
die schwer zu faktorisieren ist.]
Eingabeparameter:
n = Zahl der gewünschten Bits.
Ausgabeparameter:
m = eine Blum-Zahl.
Anweisungen:
Setze $k = \lfloor \frac{n}{2} \rfloor$.
Setze $l = n - k + 1$.
Setze $p = \text{BlumRandomPrime}(k)$.
Setze $q = \text{BlumRandomPrime}(l)$.
Setze $m = p \cdot q$.

Damit sind die Prozeduren zur Parameter-Erzeugung komplett. Eine BLUM-Zahl mit $n = 1025$, die mit diesem Verfahren erzeugt wurde, ist in Tabelle 3 abgedruckt (sie hat 309 Dezimalstellen). Im Hinblick auf den Fortschritt der Faktorisierungsalgorithmen solte man allerdings lieber BLUM-Zahlen in der Größenordnung ab 2048 Bit verwenden.

```
 4506 15286 74466 50249 26225 14044 26383 22616 74480 10227
69340 10344 80414 96318 08671 21639 63710 30387 17602 25696
53909 02080 09976 45161 76261 91025 59480 62175 49124 86394
40823 70452 14981 62658 94574 67753 74945 83135 16199 61782
07594 51105 16833 44889 30109 66289 10763 64987 90309 41852
27681 66632 02722 32988 57145 85172 07427 89442 30004 31819
83739 34537
```

Tabelle 3: Eine Blum-Zahl mit 1025 Bits

Die eigentliche Zufallserzeugung geht nun so: Man setzt den Modul m als globale Konstante und den Startwert x als globale Variable, die mit einem „echt“ zufälligen Wert im Intervall $[\lceil\sqrt{m}\,\rceil \ldots m - \lceil\sqrt{m}\,\rceil]$ vorbesetzt wird. Mit der folgenden Prozedur erzeugt man dann eine Bitfolge der gewünschten Länge:

Prozedur BBSrandomBit

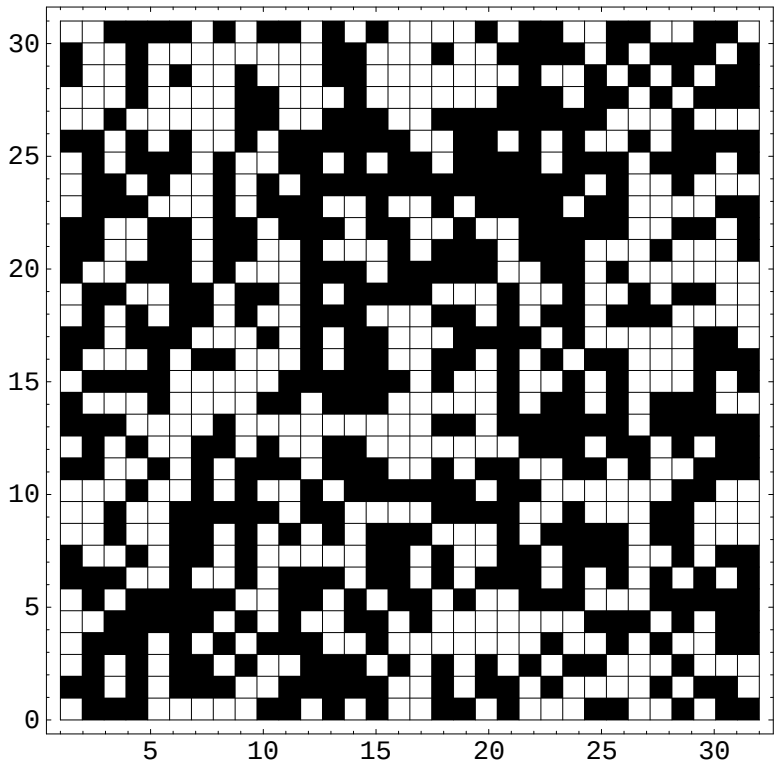
[Erzeugt eine Folge von n Pseudozufalls-Bits.]
Eingabeparameter:
n = Zahl der gewünschten Bits.
Ausgabeparameter:
blist = eine Liste von Bits.
Anweisungen:
Für $i = 1, \ldots, n$
ersetze x durch $x^2 \bmod m$,
setze $b = x \bmod 2$,
hänge b an blist an.

Mit dem oben erzeugten Modul und einem (gemäß der Konvention geheimgehaltenen) geeigneten Startwert wurde mit dieser Prozedur die Folge von 1024 Bits erzeugt, die in Tabelle 4 steht.

Einen optischen Eindruck von der Zufälligkeit der Folge vermittelt das folgende Bild.

1000 1111 1001 0101 1001 0111 0011 0100 0010 1000 1100 0001
1010 0101 1110 1001 1010 1001 0110 0010 1010 1010 0111 0111
1000 1010 1000 1101 1111 1101 1010 1100 1100 0001 0101 1001
0111 1111 0001 0100 1010 0000 1100 1010 0101 1000 1110 0000
0001 1011 0100 0100 1010 0010 1010 1010 0110 1001 0111 1100
1011 0010 0011 0100 1101 1001 0101 0100 0111 0100 0010 0111
1101 1000 0010 0111 1000 0110 1110 0111 1110 1101 0110 1000
0001 0011 1111 0011 0011 0101 0001 0001 1010 0110 0101 1000
1010 1100 1011 0011 1111 1000 1001 0100 0001 1110 1111 1111
1001 0000 0010 0000 0111 0111 1001 0001 1111 0100 1010 0011
1000 0111 1100 0000 1011 0110 1011 1010 0111 0100 1110 1001
1001 0101 0011 1000 0010 0011 1010 1001 1100 0010 1111 1001
1010 1001 0110 0011 1001 0100 1000 1111 1001 1001 0010 1000
0111 0110 1101 0011 0110 0010 1110 0010 0000 1100 1011 1111
0011 0010 0110 1110 1000 1000 1110 1110 0011 0010 0100 0100
1101 1000 0011 0010 1000 1110 1000 1101 1010 0001 0011 1100
1001 0110 1010 0000 0000 0000 1011 0111 1010 0010 1100 1010
0100 0010 0010 0010 0010 1011 0100 0000 1100 1010 1101 0000
1101 1111 0011 0001 1000 0000 0111 0111 1110 1111 0011 1011
1111 0001 0010 1000 0110 1011 0111 0011 1111 1011 0101 0100
0110 1111 1111 0011 1011 0000 1010 0010 1100 0010 1001 0101
1110 1001 1001 1001

Tabelle 4: 1024 „perfekte“ Zufallsbits


30
25
20
15
10
5
0
5
10
15
20
25
30

4.2 BBS-Generator und Quadratrest-Eigenschaft

Für einen Startwert $x \in \mathbb{M}_m$ sei $(b_1(x), \ldots, b_r(x))$ die vom BBS-Generator erzeugte Bitfolge. Ein probabilistisches Schaltnetz

$$C\colon \mathbb{F}_2^r \times \Omega \longrightarrow \mathbb{F}_2$$

hat einen ε-Vorteil bei der **BBS-Extrapolation** für m, wenn

$$P(\{(x,\omega) \in Q_m \times \Omega \mid C(b_1(x), \ldots, b_r(x), \omega) = \mathrm{lsb}(x)\}) \geq \frac{1}{2} + \varepsilon.$$

Das bedeutet: Der durch C gegebene Algorithmus sagt jeweils das Vorgängerbit zu einer Teilfolge mit ε-Vorteil „voraus“.

Im folgenden Satz sei τ_n der Aufwand für die Operation $xy \bmod m$, wo m eine n-Bit-Zahl und $0 \leq x, y < m$ ist. Bekanntlich ist $\tau_n = \mathrm{O}(n^2)$.

Hilfssatz 1 *Sei m eine* Blum*-Zahl $< 2^n$. Das probabilistische Schaltnetz $C\colon \mathbb{F}_2^r \times \Omega \longrightarrow \mathbb{F}_2$ habe einen ε-Vorteil bei der BBS-Extrapolation für m. Dann gibt es ein probabilistisches Schaltnetz $C'\colon \mathbb{F}_2^n \times \Omega \longrightarrow \mathbb{F}_2$ der Größe $\#C' \leq \#C + r\tau_n + 4$, das einen ε-Vorteil bei der Bestimmung der Quadratrest-Eigenschaft auf $\mathbb{M}_m^+$ hat.*

Beweis. Zunächst wird mit Aufwand $r\tau_n$ die BBS-Folge $(b_1, \ldots, b_r)$ zum Startwert $x \in \mathbb{M}_m^+$ berechnet. Dann sagt C das Bit $\mathrm{lsb}(\sqrt{x}^2 \bmod m)$ mit Vorteil ε voraus. Setzt man also

$$C'(x,\omega) := \begin{cases} 1, & \text{wenn } C(b_1, \ldots, b_r, \omega) = \mathrm{lsb}(x), \\ 0 & \text{sonst,} \end{cases}$$

so hat man nach dem Abschnitt über Blum-Zahlen in Kapitel III die Quadratrest-Eigenschaft von x mit ε-Vorteil bestimmt. Der zusätzliche Aufwand für den Bitvergleich sind maximal 4 weitere Knoten im Schaltnetz. $\diamond$

Sei nun $C : \mathbb{F}_2^n \times \Omega \longrightarrow \mathbb{F}_2$ ein beliebiges probabilistisches Schaltnetz. Dann ist für $r \geq 1$ das **r-fache Schaltnetz** definiert durch

$$C^{(r)}\colon \mathbb{F}_2^n \times \Omega^r \longrightarrow \mathbb{F}_2,$$

$$C^{(r)}(x, \omega_1, \ldots, \omega_r) := \begin{cases} 1, & \text{wenn } \#\{i \mid C(x,\omega_i) = 1\} \geq \frac{r}{2}, \\ 0 & \text{sonst.} \end{cases}$$

Dieses Schaltnetz repräsentiert also die „Mehrheitsentscheidung“; es wird realisiert durch r-fache Parallelschaltung von C, eine Ganzzahl-Addition von r Bits und einen Größenvergleich von $\lceil {}^2\log r \rceil$-Bit-Zahlen, hat also eine Größe

$$\#C^{(r)} \leq r \cdot \#C + 2r^2.$$

Hilfssatz 2 (Verdichtung eines Vorteils) *Sei $A \subseteq \mathbb{F}_2^n$, und C berechne die* Boolesche *Funktion f auf A mit ε-Vorteil. Sei $r = 2s+1$ ungerade.*

Dann berechnet $C^{(r)}$ die Funktion f mit einer Irrtumswahrscheinlichkeit

$$\leq \frac{(1-4\varepsilon^2)^s}{2}.$$

Ist $\delta > 0$ beliebig, so gibt es ein

$$r \leq 3 + \frac{1}{2\delta\varepsilon^2},$$

so dass $C^{(r)}$ die Funktion f mit einer Irrtumswahrscheinlichkeit $\leq \delta$ berechnet.

Beweis. Die Wahrscheinlichkeit, bei einer Anwendung von C die korrekte Antwort zu erhalten, ist

$$p := P(\{(x,\omega) \in A \times \Omega \mid C(x,\omega) = f(x)\}) \geq \frac{1}{2} + \varepsilon.$$

Da bei Vergrößerung von ε die Behauptung verschärft wird, kann man o. B. d. A. $p = \frac{1}{2} + \varepsilon$ annehmen. Der komplementäre Wert $q := 1 - p = \frac{1}{2} - \varepsilon$ ist die Wahrscheinlichkeit dafür, bei einer Anwendung von C die falsche Antwort zu erhalten. Also ist die Wahrscheinlichkeit dafür, bei r unabhängigen Anwendungen von C genau k richtige Antworten zu erhalten, $\binom{r}{k} p^k q^{r-k}$. Die gesuchte Irrtumswahrscheinlichkeit ist also

$$P(\{(x,\omega_1,\ldots,\omega_r) \in A \times \Omega^r \mid C^{(r)}(x,\omega_1,\ldots,\omega_r) = f(x)\})$$

$$\begin{aligned}
&= \sum_{k=0}^{s} \binom{r}{k} (\frac{1}{2}+\varepsilon)^k (\frac{1}{2}-\varepsilon)^{r-k} \\
&= (\frac{1}{2}+\varepsilon)^s (\frac{1}{2}-\varepsilon)^{s+1} \cdot \sum_{k=0}^{s} \binom{r}{k} (\frac{1}{2}+\varepsilon)^{k-s} (\frac{1}{2}-\varepsilon)^{s-k} \\
&= (\frac{1}{4}-\varepsilon^2)^s \cdot (\frac{1}{2}-\varepsilon) \cdot \underbrace{\sum_{k=0}^{s} \binom{r}{k} \underbrace{\left(\frac{\frac{1}{2}-\varepsilon}{\frac{1}{2}+\varepsilon}\right)^{s-k}}_{\leq 1}}_{\leq 2^{r-1} = 4^s} \\
&\leq (1-4\varepsilon^2)^s \cdot \frac{1}{2},
\end{aligned}$$

und die erste Aussage somit bewiesen.

Um eine Irrtumswahrscheinlichkeit $\leq \delta$ zu erreichen, ist hinreichend:

$$\begin{aligned}(1-4\varepsilon^2)^s &\leq 2\delta,\\ s\cdot\ln(1-4\varepsilon^2) &\leq \ln 2+\ln\delta,\\ s &\geq \frac{\ln 2+\ln\delta}{\ln(1-4\varepsilon^2)}.\end{aligned}$$

Wählt man also

$$s := \left\lceil \frac{\ln 2+\ln\delta}{\ln(1-4\varepsilon^2)} \right\rceil,$$

so ist die Irrtumswahrscheinlichkeit von $C^{(r)}$ höchstens δ, ferner

$$\begin{aligned}s &\leq 1+\frac{\ln 2+\ln\delta}{\ln(1-4\varepsilon^2)} = 1+\frac{\ln\frac{1}{\delta}-\ln 2}{\ln\frac{1}{1-4\varepsilon^2}}\\ &\leq 1+\frac{\frac{1}{\delta}-1-\ln 2}{4\varepsilon^2} \leq 1+\frac{1}{4\delta\varepsilon^2}\end{aligned}$$

und somit die zweite Aussage bewiesen. $\diamond$

$C^{(r)}$ hat dann übrigens die Größe

$$\#C^{(r)} \leq \left[3+\frac{1}{2\delta\varepsilon^2}\right]\cdot\#C + 2\cdot\left[3+\frac{1}{2\delta\varepsilon^2}\right]^2.$$

Die Zusammenfassung der beiden Hilfssätze ergibt:

Satz 1 *Sei m eine* Blum*-Zahl $< 2^n$. Das probabilistische Schaltnetz $C : \mathbb{F}_2^r \times \Omega \longrightarrow \mathbb{F}_2$ habe einen ε-Vorteil bei der BBS-Extrapolation für m. Dann gibt es für jedes $\delta > 0$ ein probabilistisches Schaltnetz $C' : \mathbb{F}_2^n \times \Omega' \longrightarrow \mathbb{F}_2$, das die Quadratrest-Eigenschaft auf $\mathbb{M}_m^+$ mit Irrtumswahrscheinlichkeit $\leq \delta$ bestimmt, mit*

$$\#C' \leq \left[3+\frac{1}{2\delta\varepsilon^2}\right]\cdot[\#C + r\tau_n + 4] + 2\cdot\left[3+\frac{1}{2\delta\varepsilon^2}\right]^2.$$

Aus einer effizienten BBS-Extrapolation ließe sich also ein effizienter Entscheidungsalgorithmus für die Quadratrest-Eigenschaft konstruieren. Diese Aussage wird im folgenden Abschnitt präzisiert.

4.3 Perfekte Pseudozufallsgeneratoren

Es ist jetzt an der Zeit, den Begriff „Zufallsgenerator“ – bzw. genauer gesagt, den Begriff „Pseudozufallsgenerator“) formal zu definieren. Dazu braucht man eine unendliche Parametermenge $M \subseteq \mathbb{N}$; für jeden Parameter $m \in M$ soll eine Instanz des Pseudozufallsgenerators definiert sein. Man denke sich etwa M als eine Menge von BLUM-Zahlen. Es sei $M_n = M \cap [2^{n-1} \ldots 2^n[$ die Menge der n-Bit-Zahlen in M und $I = \{n \in \mathbb{N} \mid M_n \neq \emptyset\}$ der Träger von M. Ferner braucht man ein nichtkonstantes Polynom $g \in \mathbb{N}[X]$.

Ein **Pseudozufallsgenerator mit Parametermenge** M **und Streckungspolynom** g ist eine Familie $G = (G_m)_{m \in M}$ von Funktionen

$$G_m \colon X_m \longrightarrow \mathbb{F}_2^{g(n)} \quad \text{mit } X_m \subseteq \mathbb{F}_2^{k(n)},$$

wobei n die Bitzahl von m ist, so dass es eine (deterministische) polynomiale Schaltnetzfamilie $\tilde{G}$ mit $\tilde{G}_n(m, x) = G_m(x)$ gibt. (Mit anderen Worten: Die Zufallsbits sind effizient berechenbar. Insbesondere ist die Funktion k durch ein Polynom beschränkt.) X_m heißt die Menge der Startwerte zum Parameter m. Jedes G_m streckt also eine $k(n)$-Bit-Folge $x \in X_m$ zu einer $g(n)$-Bit-Folge $G_m(x) \in \mathbb{F}_2^{g(n)}$.

Der BBS-Generator passt als Beispiel so zu dieser Definition: M ist die Menge der BLUM-Zahlen oder eine unendliche Teilmenge davon, $X_m = \mathbb{M}_m$, und $G_m(x) = (b_1(x), \ldots, b_{g(n)}(x))$ mit $b_i(x) = \mathrm{lsb}(x_i)$, wobei $x_0 = x$, $x_i = x_{i-1}^2 \bmod m$.

Ein **polynomialer Test** für den Pseudozufallsgenerator G ist eine (probabilistische) polynomiale Schaltnetzfamilie $C = (C_n)_{n \in \mathbb{N}}$,

$$C_n : \mathbb{F}_2^n \times \mathbb{F}_2^{g(n)} \times \Omega_n \longrightarrow \mathbb{F}_2$$

über einem Wahrscheinlichkeitsraum $\Omega_n \subseteq \mathbb{F}_2^{s(n)}$, wobei $s(n)$ die Anzahl der probabilistischen Eingänge von C_n ist. Die Wahrscheinlichkeit, dass der Test für eine von G erzeugte Folge den Wert 1 errechnet, ist

$$p(G, C, m) = P\{(x, \omega) \in X_m \times \Omega_n \mid C_n(m, G_m(x), \omega) = 1\};$$

die Wahrscheinlichkeit, dass der Test für eine beliebige („echt zufällige“) Folge der gleichen Länge den Wert 1 errechnet, ist

$$\bar{p}(C, m) = P\{(u, \omega) \in \mathbb{F}_2^{g(n)} \times \Omega_n \mid C_n(m, u, \omega) = 1\}.$$

Diese beiden Werte sollten im Idealfall gleich sein. Man sagt, der Pseudozufallsgenerator G **besteht den Test** C, wenn für alle nichtkonstanten Polynome $h \in \mathbb{N}[X]$ die Menge der $m \in M$ mit

$$|p(G, C, m) - \bar{p}(C, m)| \geq \frac{1}{h(n)}$$

dünn in M ist. Der Pseudozufallsgenerator G heißt **perfekt**, wenn er alle polynomialen Tests besteht. D. h., es gibt keinen effizienten Algorithmus, der die vom Pseudozufallsgenerator erzeugte Bitfolge von einer „echt zufälligen“ Bitfolge unterscheiden kann.

Zum Nachweis der Perfektheit eines Pseudozufallsgenerators G reicht ein scheinbar schwächerer Test. Sei $G_m(x) = (b_1^{(m)}(x), \ldots, b_{g(n)}^{(m)}(x))$ die von G_m aus dem Startwert x erzeugte Bitfolge. Sei $C = (C_n)_{n\in\mathbb{N}}$ eine polynomiale Schaltnetzfamilie,

$$C_n : \mathbb{F}_2^n \times \mathbb{F}_2^{i_n} \times \Omega_n \longrightarrow \mathbb{F}_2$$

mit $0 \leq i_n \leq g(n)-1$, und sei $h \in \mathbb{N}[X]$ ein nichtkonstantes Polynom. Dann sagt man, C habe einen $\frac{1}{h}$-Vorteil bei der Extrapolation von G, wenn die Menge der Parameter $m \in M$ mit

$$P\{(x,\omega) \in X_m \times \Omega_n \mid C_n(m, b_{j_m+1}^{(m)}(x), \ldots, b_{j_m+i_n}^{(m)}(x), \omega) = b_{j_m}^{(m)}(x)\}$$

$$\geq \frac{1}{2} + \frac{1}{h(n)} \tag{2}$$

für einen Index j_m, $1 \leq j_m \leq g(n) - i_n$ nicht dünn in M ist; das heißt, C kann in genügend vielen Fällen aus einer Teilfolge das vorhergehende Bit mit einem kleinen Vorteil extrapolieren. Man sagt, G besteht den **Extrapolationstest**, wenn es keine solche polynomiale Schaltnetzfamilie gibt, die für irgendein Polynom $h \in \mathbb{N}[X]$ einen $\frac{1}{h}$-Vorteil bei der Extrapolation von G hat.

Zum Beispiel besteht der lineare Kongruenzgenerator den Extrapolationstest nicht.

Hauptsatz 1 [YAOs Kriterium] *Für einen Pseudozufallsgenerator G sind folgende Aussagen äquivalent:*

(i) *G ist perfekt.*

(ii) *G besteht den Extrapolationstest.*

Beweis. „(i) $\Longrightarrow$ (ii)“: Wenn G den Extrapolationstest nicht besteht, gibt es eine polynomiale Schaltnetzfamilie C mit $\frac{1}{h}$-Vorteil bei der Extrapolation von G. Sei $A \subseteq M$ die nicht dünne Menge von Parametern, für die die Ungleichung (2) gilt. Daraus wird ein polynomialer Test $C' = (C'_n)_{n\in\mathbb{N}}$ konstruiert:

$$C'_n(m,u,\omega) = C_n(m, u_{j_m+1}, \ldots, u_{j_m+i_n}, \omega) + u_{j_m} + 1;$$

für $m \in \mathbb{F}_2^n - A$ sei dabei $j_m = 1$ gesetzt (auf diesen Wert kommt es nicht an). Es ist also

$$C'_n(m,u,\omega) = 1 \Longleftrightarrow C_n(m, u_{j_m+1}, \ldots, u_{j_m+i_n}, \omega) = u_{j_m}.$$

Für $m \in A$ folgt

$$p(G, C', m) = P\{C_n(m, b^{(m)}_{j_m+1}(x), \ldots, b^{(m)}_{j_m+i_n}(x), \omega) = b^{(m)}_{j_m}(x)\} \geq \frac{1}{2} + \frac{1}{h(n)}.$$

Dieser Wert ist zu vergleichen mit

$$\bar{p}(C', m) = P\{C_n(m, u_{j_m+1}, \ldots, u_{j_m+i_n}, \omega) = u_{j_m}\}$$

$$= P\{C_n(\ldots) = 0 \text{ und } u_{j_m} = 0\} + P\{C_n(\ldots) = 1 \text{ und } u_{j_m} = 1\}.$$

(Die Summe entspricht einer Zerlegung in zwei disjunkte Teilmengen.) Da hier jeweils die Wahrscheinlichkeit des Zusammentreffens zweier unabhängiger Ereignisse steht, ist

$$\bar{p}(C', m) = \frac{1}{2} P\{C_n(\ldots) = 0\} + \frac{1}{2} P\{C_n(\ldots) = 1\} = \frac{1}{2}.$$

Für $m \in A$ gilt also

$$p(G, C', m) - \bar{p}(C', m) \geq \frac{1}{h(n)}.$$

Daher besteht G den Test C' nicht und ist nicht perfekt.

„(ii) $\Longrightarrow$ (i)“: Sei G nicht perfekt. Dann gibt es einen polynomialen Test C, den G nicht besteht, also ein nichtkonstantes Polynom $h \in \mathbb{N}[X]$ und ein $t \in \mathbb{N}$ mit

$$|p(G, C, m) - \bar{p}(C, m)| \geq \frac{1}{h(n)}$$

für m aus einer nicht dünnen Teilmenge $A \subseteq M$ mit $\#A_n \geq \#M_n/n^t$ für unendlich viele $n \in I$. Für mindestens die Hälfte aller $m \in A_n$ gilt $p(G, C, m) > \bar{p}(C, m)$ oder die umgekehrte Ungleichung; zuerst wird der erste dieser Fälle durchgezogen (bei festem n).

Für $k = 0, \ldots, g(n)$ sei

$$p^k_m = P\{C_n(m, t_1, \ldots, t_k, b^{(m)}_{k+1}(x), \ldots, b^{(m)}_{g(n)}(x), \omega) = 1\},$$

wobei $t_1, \ldots, t_k \in \mathbb{F}_2$ zufällige Bits sind; die Wahrscheinlichkeit wird also in $X_m \times (\mathbb{F}_2^k \times \Omega_n)$ gebildet. Es ist

$$p^0_m = p(G, C, m), \quad p^{g(n)}_m = \bar{p}(C, m),$$

$$\frac{1}{h(n)} \leq p^0_m - p^{g(n)}_m = \sum_{k=1}^{g(n)} (p^{k-1}_m - p^k_m)$$

für die betrachteten $m \in A_n$. Es gibt also ein r_m mit $1 \leq r_m \leq g(n)$, so dass

$$p^{r_m-1}_m - p^{r_m}_m \geq \frac{1}{g(n)h(n)}.$$

Einer dieser Werte r_m kommt mindestens $(\#M_n/2n^t g(n))$-mal vor; er wird k_n genannt.

Sei $\Omega'_n = \mathbb{F}_2^{k_n} \times \Omega_n$. Die polynomiale Schaltnetzfamilie C', deren deterministische Eingänge aus $A_n \times \mathbb{F}_2^{g(n)-k_n}$ und deren probabilistische Eingänge aus Ω'_n besetzt werden, wird für dieses n so definiert:

$$C'_n(m, u_1, \ldots, u_{g(n)-k_n}, t_1, \ldots, t_{k_n}, \omega) = C_n(m, t, u, \omega) + t_{k_n} + 1.$$

Es ist also

$$C'_n(m, u, t, \omega) = t_{k_n} \Longleftrightarrow C_n(m, t, u, \omega) = 1.$$

Nun ist

$$C'_n(m, b^{(m)}_{k_n+1}(x), \ldots, b^{(m)}_{g(n)}(x), t, \omega) = b^{(m)}_{k_n}(x)$$

$$\Longleftrightarrow \begin{cases} C_n(m, t, b^{(m)}_{k_n+1}(x), \ldots, b^{(m)}_{g(n)}(x), \omega) = 1 \quad \text{und } t_{k_n} = b^{(m)}_{k_n}(x) \\ \text{oder} \\ C_n(m, t, b^{(m)}_{k_n+1}(x), \ldots, b^{(m)}_{g(n)}(x), \omega) = 0 \quad \text{und } t_{k_n} \neq b^{(m)}_{k_n}(x) \end{cases}$$

Beide Möglichkeiten sind jeweils ein Zusammentreffen unabhängiger Ereignisse. Die zweite hat daher die Wahrscheinlichkeit $\frac{1}{2}(1 - p_m^{k_n})$. Die erste ist äquivalent zu

$$C_n(m, t_1, \ldots, t_{k_n-1}, b^{(m)}_{k_n}(x), \ldots, b^{(m)}_{g(n)}(x), \omega) = 1 \text{ und } t_{k_n} = b^{(m)}_{k_n}(x);$$

ihre Wahrscheinlichkeit ist $p_m^{k_n-1}/2$. Zusammen ergibt das

$$P\{C'_n(m, b^{(m)}_{k_n+1}(x), \ldots, b^{(m)}_{g(n)}(x), t, \omega) = b^{(m)}_{k_n}(x)\}$$

$$= \frac{1}{2} + \frac{1}{2}(p_m^{k_n-1} - p_m^{k_n}) \geq \frac{1}{2} + \frac{1}{2g(n)h(n)}$$

für mindestens $\#M_n/2n^t g(n)$ der Parameter $m \in M_n$. Mit $u = t + \mathrm{Grad}(g) + 1$ ist das $\geq \#M_n/n^u$ für unendlich viele $n \in I$.

Im Falle $p(G, C, m) < \bar{p}(C, m)$ für mindestens die Hälfte aller $m \in A_n$ wird analog

$$C'_n(m, u, t, \omega) = C_n(m, t, u, \omega) + t_{k_n}$$

gesetzt; damit klappt der Schluss genauso.

Also besteht G den Extrapolationstest nicht (mit $i_n = g(n) - k_n$ und $j_m = k_n$). $\diamond$

Im Beweis wurde übrigens die Nichtgleichmäßigkeit des Berechnungsmodells verwendet: C'_n hängt von k_n ab, und es wurde kein Algorithmus zur Bestimmung von k_n angegeben.

Der Extrapolationstest wirkt etwas unnatürlich, weil er die erzeugten Bits in umgekehrter Richtung extrapoliert. Das steht im Gegensatz zu den

kryptoanalytischen Verfahren, wo man sich bemüht, Bits *vorher*zusagen. Nun denn:

Sei $C = (C_n)_{n\in\mathbb{N}}$ eine polynomiale Schaltnetzfamilie,

$$C_n : \mathbb{F}_2^n \times \mathbb{F}_2^{i_n} \times \Omega_n \longrightarrow \mathbb{F}_2$$

mit $0 \leq i_n \leq g(n) - 1$, und sei $h \in \mathbb{N}[X]$ ein nichtkonstantes Polynom. Dann hat C einen $\frac{1}{h}$-Vorteil bei der Vorhersage von G, wenn die Menge der Parameter $m \in M$ mit

$$P\{(x,\omega) \mid C_n(m, b_1^{(m)}(x), \ldots, b_{i_n}^{(m)}(x), \omega) = b_{i_n+1}^{(m)}(x)\} \geq \frac{1}{2} + \frac{1}{h(n)}$$

nicht dünn in M ist. Der Pseudozufallsgenerator G besteht den **Vorhersagetest**, wenn keine polynomiale Schaltnetzfamilie einen Vorteil bei der Vorhersage von G hat. Der Beweis von „(i) $\Longrightarrow$ (ii)“ im Hauptsatz 1 lässt sich direkt auf diese Situation adaptieren und ergibt:

Korollar 1 *Jeder perfekte Pseudozufallsgenerator besteht den Vorhersagetest.*

Korollar 2 *Wenn die Quadratrest-Vermutung richtig ist, ist der BBS-Generator perfekt.*

Beweis. Aus Satz 1 ließe sich sonst eine polynomiale Schaltnetzfamilie konstruieren, die die Quadratrest-Eigenschaft für eine nicht dünne Menge von Blum-Zahlen entscheidet. $\diamond$

4.4 Beispiele und praktische Überlegungen

Der BBS-Generator ist also perfekt *unter einer „vernünftigen", aber unbewiesenen Annahme,* nämlich der Quadratrest-Vermutung. Wir wissen aber nichts Konkretes:

- Wie groß muss man die Bitzahl des Parameters m wählen?
- Welches sind die schlechten Werte für den Modul m?
- Wie groß ist der Anteil der schlechten Startwerte bei gegebenem m?
- Wieviele Bits am Stück darf man verwenden bei gegebenem Modul und Startwert, d. h., welche Wahl für das Streckungspolynom g ist angemessen?

Die hergeleiteten Aussagen sind qualitativ, nicht quantitativ. Die tatsächliche Qualität der erzeugten Zufallsbits, sei es für statistische oder kryptographische Anwendungen, kann bis auf weiteres nur empirisch beurteilt werden. Man kann davon ausgehen, dass für Moduln, die sich den gegenwärtigen Faktorisierungsalgorithmen noch sicher entziehen, also etwa ab 2048 Bit Länge, bei zufälliger Wahl des Moduls und des Startwerts die Gefahr extrem gering, auf jeden Fall vernachlässigbar, ist, eine „schlechte" Bitfolge zu erzeugen. Die bewiesenen Ergebnisse lassen allerdings auch zu, dass man die Menge M aller Blum-Zahlen von vorneherein um bekannte schlechte Fälle verkleinert, um die Quadratrest-Vermutung zu retten.

Hier stellt sich auch die Frage, wie groß in Abhängigkeit vom Startwert die Gefahr ist, einen kurzen Zyklus zu erzeugen. Nach den Ergebnissen von Shparlinski – siehe das Literaturverzeichnis zur Vorlesung – ist diese Gefahr vernachlässigbar. Dort gibt es auch nichttriviale untere Schranken für das Linearitätsprofil des BBS-Generators.

Als weitere Frage drängt sich auf: Darf man, um die praktische Verwertbarkeit des Generators zu verbessern, in jedem Iterationsschritt mehr als nur ein Bit verwenden? Wenigstens 2? Diese Frage wurde von Vazirani/Vazirani und unabhängig von Alexi/Chor/Goldreich/Schnorr teilweise, aber auch wieder nur qualitativ, beantwortet: Wenigstens $O(^2\log {}^2\log m)$ der niedrigsten Bits sind „sicher". Je nach Wahl der Konstanten, die in dem „O" steckt, muss man die Bitzahl des Moduls genügend groß machen und auf empirische Erfahrungen vertrauen. Entscheiden wir uns für genau $^2\log {}^2\log m$ Bits. Hat dann m 2048 Bits, also etwa 600 Dezimalstellen, so kann man also in jedem Schritt 11 Bits verwenden. Um $x^2 \bmod m$ zu berechnen, wenn m eine n-Bit-Zahl ist, braucht man $(\frac{n}{32})^2$ Multiplikationen von 32-Bit-Zahlen und anschließend ebensoviele Divisionen „64 Bit durch 32 Bit". Bei $n = 2048$ sind das $2 \cdot (2^6)^2 = 8192$ solcher elementaren Operationen, um 11 Bits zu erzeugen, also etwa 800 Operationen pro Bit. Lineare Kongruenzgeneratoren im 32-Bit-Bereich brauchen pro Schritt zwei

elementare Operationen; selbst wenn man jeweils nur 20 Bits verwendet, hat man noch einen Geschwindigkeitsvorteil mit dem Faktor 8000, allerdings bei wesentlich geringerer „Zufallsqualität“ der Bits.

In der Literatur werden einige weitere Pseudozufallsgeneratoren betrachtet, die nach ähnlichen Prinzipien funktionieren wie der BBS-Generator. Stets wird von vorneherein ein nichtkonstantes Polynom $g \in \mathbb{N}[X]$ festgelegt, das die Anzahl der maximal auszugebenden Bits spezifiziert.

Der RSA-Generator (Shamir). Man wählt einen zufälligen Modul m, der ein Produkt zweier großer Primzahlen p, q ist, und einen Exponenten d, der teilerfremd zu $\varphi(m) = (p-1)(q-1)$ ist, ferner einen zufälligen Startwert $x = x_0$ im Zustandsraum $\mathbb{M}_m$. Die interne Transformation ist $T_m(x) = x^d \bmod m$. Man bildet also $x_i = x_{i-1}^d \bmod m$ und gibt das letzte Bit oder auch die letzten $\lfloor {}^2\log {}^2\log m \rfloor$ Bits aus, bis zu insgesamt $g(\lfloor {}^2\log(m) \rfloor)$ Bits. Wenn dieser Zufallsgenerator (mit m als Parameter) nicht perfekt ist, dann gibt es einen effizienten Algorithmus zum Brechen der RSA-Verschlüsselung. Der Rechenaufwand ist größer als beim BBS-Generator in dem Maße, wie das Potenzieren mit d aufwendiger als das Quadrieren ist; ist d zufällig gewählt, so ist der Zeitbedarf $\mathrm{O}(n^3 \cdot g(n))$, da das Potenzieren mit einer n-Bit-Zahl im Vergleich zum schlichten Quadrieren in jedem Schritt den Aufwand mit dem Faktor n vergrößert.

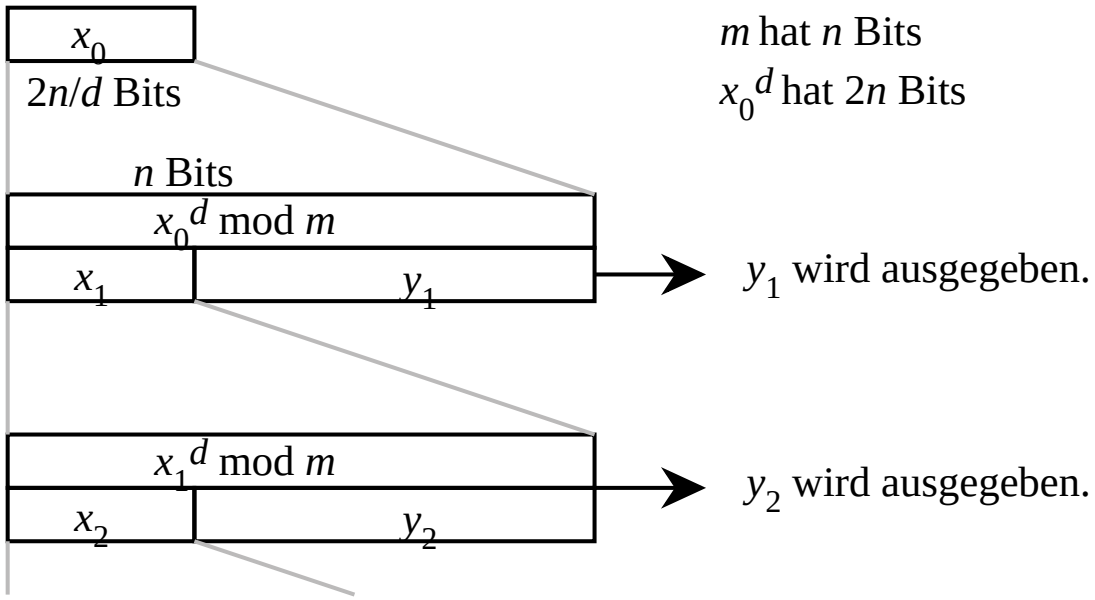
Der Index-Generator (Blum/Micali). Man wählt als Modul zufällig eine große Primzahl p und bestimmt dazu eine Primitivwurzel a. Ferner wählt man einen zufälligen Startwert $x = x_0$, teilerfremd zu $p-1$. Dann bildet man $x_i = a^{x_{i-1}} \bmod p$ und gibt das erste oder die ersten $\lfloor {}^2\log {}^2\log p \rfloor$ Bits aus, bis zu insgesamt $g(\lfloor {}^2\log(p) \rfloor)$ Bits. (Mit den letzten Bits geht's genauso.) Die Perfektheit dieses Zufallsgenerators (mit den Primzahlen als Indexmenge) beruht auf der Vermutung, dass diskrete Logarithmus mod$p$ hart ist. Auch hier ist der Zeitbedarf pro Bit $\mathrm{O}(n^3)$.

Der elliptische Index-Generator (Kaliski). Er funktioniert wie der Index-Generator, nur dass man die Gruppe $\mathbb{M}_p = \mathbb{F}_p^\times$ durch eine elliptische Kurve über dem Körper $\mathbb{F}_p$ ersetzt (eine solche Kurve ist auf kanonische Weise eine endliche Gruppe).

4.5 Der Micali-Schnorr-Generator

Der von Micali und Schnorr vorgeschlagene Zufallsgenerator ist ein Abkömmling des RSA-Generators. Sei dazu $d \geq 3$ ungerade. Als Parametermenge dient die Menge aller Produkte m von zwei Primzahlen p und q, die sich in ihrer Bitanzahl höchstens um 1 unterscheiden und für die d zu $\varphi(m) = (p-1)(q-1)$ teilerfremd ist. Wenn m eine n-Bit-Zahl ist, sei $r(n) \approx \frac{2n}{d}$; die d-te Potenz einer $r(n)$-Bit-Zahl ist dann (ungefähr) eine $2n$-Bit-Zahl.

Im i-ten Schritt wird $z_i = x_{i-1}^d \bmod m$ gebildet; davon werden die ersten $r(n)$ Bits, also $\lfloor z_i / 2^{n-r(n)} \rfloor$, als x_i genommen, die übrigen Bits, also $y_i = z_i \bmod 2^{n-r(n)}$ werden ausgegeben. Bemerkenswert ist, dass die Bits auf zwei *disjunkte* Teile verteilt werden: den Wert x_i für den nächsten Schritt und die Ausgabe y_i. Die folgende Abbildung macht das deutlich.



Ist G perfekt? Hier wird folgendes angenommen: Kein effizienter Test kann die Gleichverteilung auf $[1 \ldots m]$ von der Verteilung von $x^d \bmod m$ für gleichverteiltes $x \in [1 \ldots 2^{r(n)}]$ unterscheiden. Ist diese Annahme richtig, so ist der Micali-Schnorr-Generator perfekt.

Es scheint auf den ersten Blick fast, als ob hier nur die Perfektheit unter der Annahme der Perfektheit bewiesen wird; allerdings gibt es heuristische Überlegungen, die die Annahme in enge Beziehung zur Sicherheit des RSA-Verschlüsselungsverfahrens und zur Primzerlegung bringen.

Wie schnell purzeln nun die Zufallszahlen aus der Maschine? Als elementare Operationen gezählt werden sollen wieder die Multiplikation zweier 32-Bit-Zahlen und die Division einer 64-Bit-Zahl durch eine 32-Bit-Zahl mit

32-Bit-Quotient. Multipliziert und dividiert wird nach der klassischen Methode; das Produkt von r (32-Bit-)Wörtern mit s Wörtern kostet also rs elementare Operationen, bei der Division ist der Aufwand das Produkt der Wortzahlen von Divisor und Quotient. Die Multiplikation mit der schnellen Fourier-Transformation bringt erst bei größeren Stellenzahlen Vorteile.

Die Erfinder machen nun einen konkreten Vorschlag: $d = 7$, $n = 512$. Ausgegeben werden jeweils 384 Bits, zurückbehalten werden 128 Bits. Das binäre Potenzieren einer 128-Bit-Zahl x mit 7 kostet eine Reihe elementarer Operationen:

- x hat 128 Bits, also 4 Wörter.
- x^2 hat 256 Bits, also 8 Wörter, und kostet $4 * 4 = 16$ elementare Operationen.
- x^3 hat 384 Bits, also 12 Wörter, und kostet $4 * 8 = 32$ elementare Operationen.
- x^4 hat 512 Bits, also 16 Wörter, und kostet $8 * 8 = 64$ elementare Operationen.
- x^7 hat 896 Bits, also 28 Wörter, und kostet $12 * 16 = 192$ elementare Operationen.
- $x^7 \bmod m$ hat ≤ 512 Bits und kostet ebenfalls $12 * 16 = 192$ elementare Operationen.

Insgesamt braucht man also 496 elementare Operationen; es war nur eine Reduktion $\bmod m$ nötig. Die Belohnung besteht aus 384 Bits. Man erhält also 32 Bits mit etwa 40 elementaren Operationen. Damit hat man gegenüber den linearen Kongruenzgeneratoren, wenn man dort alle 32 Bits verwendet, nur einen Faktor 10, den heutige Computer gut verkraften.

Eine fast beliebige Geschwindigkeitssteigerung ergibt sich durch Parallelisierung: Der Micali-Schnorr-Generator ist vollständig parallelisierbar; das bedeutet, dass eine Verteilung der Arbeit auf k Prozessoren einen Gewinn um den echten Faktor k bedeutet: Die Prozessoren können unabhängig voneinander arbeiten ohne Notwendigkeit zur Kommunikation.

4.6 Der Impagliazzo-Naor-Generator

Das Rucksack-Problem (knapsack problem, subset sum problem) ist bekanntlich das folgende:

Gegeben: Natürliche Zahlen $a_1, \ldots, a_n \in \mathbb{N}$ und $T \in \mathbb{N}$.

Gesucht: Eine Teilmenge $S \subseteq \{1, \ldots, n\}$ mit

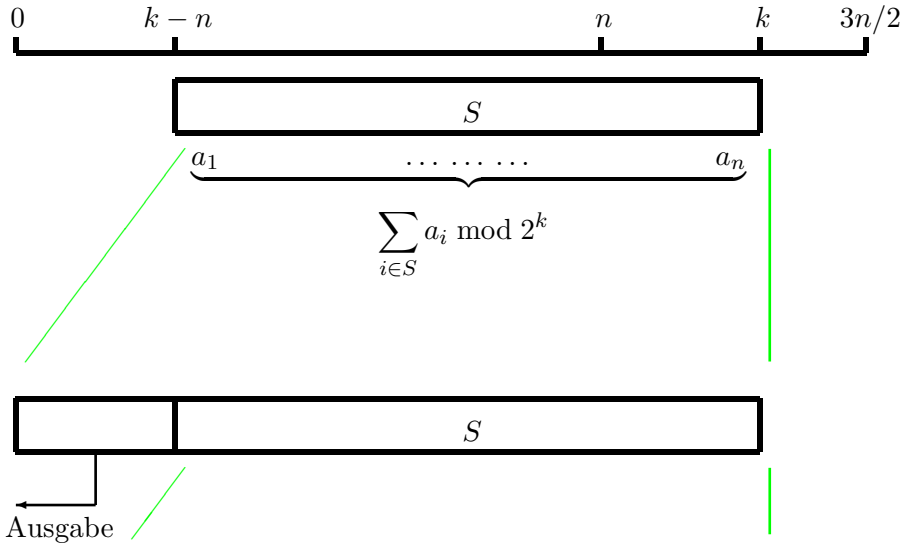
$$\sum_{i \in S} a_i = T.$$

Dieses Problem gilt als hart; es ist sogar NP-vollständig. Aufbauend darauf haben Impagliazzo und Naor den folgenden Zufallsgenerator entwickelt:

Seien k und n (genügend große) natürliche Zahlen mit $n < k < \frac{3n}{2}$; als Parameter werden $a_1, \ldots, a_n \in [1 \ldots 2^k]$ zufällig gewählt. [Achtung: Das sind viele große Zahlen.] Der Zustandsraum besteht aus der Potenzmenge von $\{1, \ldots, n\}$; die Zustände sind also Teilmengen $S \subseteq \{1, \ldots, n\}$ und werden durch Bitfolgen in $\mathbb{F}_2^n$ auf natürliche Weise repräsentiert. In jedem einzelnen Schritt wird die Summe

$$\sum_{i \in S} a_i \mod 2^k$$

gebildet. Das ist eine k-Bit-Zahl. Die ersten $k - n$ Bits werden ausgegeben, die letzten n Bits als neuer Zustand zurückbehalten, siehe die Abbildung.



Transformation und Outputfunktion sind also:

$$\begin{aligned} T(S) &= \sum_{i \in S} a_i \bmod 2^n \\ &\quad \text{(rechte } n \text{ Bits weiterverwenden)}, \\ U(S) &= \lfloor \frac{\sum_{i \in S} a_i \bmod 2^k}{2^n} \rfloor \\ &\quad \text{linke } k - n \text{ Bits ausgeben.} \end{aligned}$$

Falls dieser Zufallsgenerator nicht perfekt ist, ist das Rucksack-Problem effizient lösbar.

- R. Impagliazzo, M. Naor: Efficient cryptographic schemes provably as secure as subset sum. J. Cryptology 9 (1996), 199–216.