Bitstrom-Verschlüsselung

Klaus Pommerening
Fachbereich Mathematik
der Johannes-Gutenberg-Universität
Saarstraße 21
D-55099 Mainz

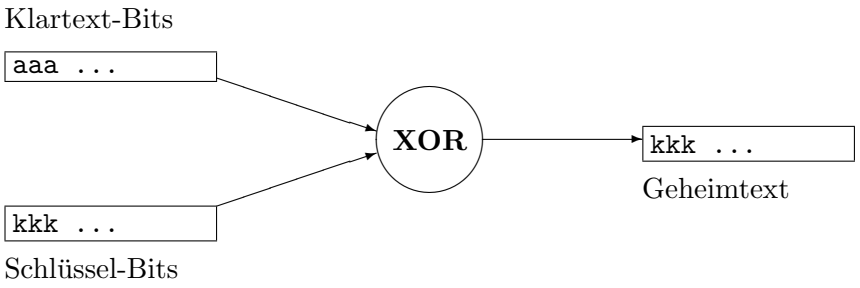
27. November 2000, letzte Revision 25. Juli 2006

Die Bitstrom-Verschlüsselung wurde bereits ganz am Anfang der Vorlesung als XOR vorgestellt. Als VERNAM-Chiffre (oder One Time Pad) über dem Alphabet $\mathbb{F}_2$ lieferte sie später ein Beispiel für perfekte Sicherheit im Sinne von SHANNON. Als Algorithmus A5 bzw. E_0 wirkt sie mit, die Mobil-Telefonie bzw. das Bluetooth-Protokoll für Datenübertragung per Funk scheinbar sicher zu machen. Sie kommt als RC4 im SSL-Protokoll vor, das die Client-Server-Kommunikation im WWW (gelegentlich) verschlüsselt, und in der PKZIP-Verschlüsselung. Viele weitere aktuelle Anwendungen kann man finden.

In diesem Kapitel werden nun Bitstrom-Chiffren systematisch untersucht. Methodisch gibt es zwei Hauptrichtungen:

- Schieberegister als Anwendung der BOOLEschen Algebra,
- perfekte Zufallsgeneratoren als Anwendung harter zahlentheoretischer Probleme.

Das Prinzip der Bitstrom-Verschlüsselung



XOR = binäre Addition
= exklusives Oder

```
  0 0 1 1
  0 1 0 1
---------
  0 1 1 0
```

Beispiel

```
       S         e         h         r
   --------- --------- --------- --------- ---------
a: 0101 0011 0110 0101 0110 1000 0111 0010 0010 0000 ...
k: 1001 1000 1101 1101 0010 1111 1000 1001 1001 0101 ...
-------------------------------------------------------------
c: 1100 1011 1011 1000 0100 0111 1111 1011 1011 0101 ...
```

1 Klassische Zufallsgeneratoren: Kongruenzgeneratoren und Schieberegister

1.1 Verschiedene Stufen der Bitstrom-Verschlüsselung

Stufe 1: Periodischer Schlüssel

Hier wird eine mehr oder weniger lange Bitfolge als Schlüssel periodisch wiederholt. Technisch handelt es sich um eine BELASO-Chiffre über dem Alphabet $\mathbb{F}_2$, und gebrochen wird sie wie andere periodische polyalphabetische Chiffren auch durch Periodenanalyse oder Finden eines wahrscheinlichen Wortes.

Stufe 2: Lauftext

Hier wird eine vorhandene Bitfolge als Schlüssel verwendet, z. B. der Inhalt einer CD ab einer bestimmten Stelle. Die Analyse verläuft nach den Methoden aus Kapitel I.5. Außerdem ist, sobald die Quelle der Bits, etwa die CD, dem Gegner bekannt ist, der Schlüsselraum viel zu klein – das lineare Durchprobieren von 700 MB Daten ist wenig aufwendig.

Stufe 4: One Time Pad

Das Extrem auf der sicheren Seite. Wegen der aufwendigen Schlüsselverteilung ist es allerdings für eine Massenanwendung nicht geeignet.

Stufe 3: Pseudozufallsfolgen

Der realistische Mittelweg. Hier wird versucht, die idealen Eigenschaften des One Time Pad zu approximieren, indem man statt einer „echten“ Zufallsfolge eine von einem Algorithmus („Zufallsgenerator“) aus einem „effektiven Schlüssel“ (= kurzen Startwert) erzeugte „pseudozufällige“ Bitfolge verwendet. Sogar bei mäßiger Qualität des Zufallsgenerators ist der Geheimtext dann resistent gegen statistische Analysen. Es bleibt das Problem, die Sicherheit gegen einen Angriff mit bekanntem Klartext in den Griff zu bekommen. Diesem Problem ist der Rest des Kapitels gewidmet.

1.2 Allgemeine Diskussion der Bitstrom-Verschlüsselung

Vorteile

- Der Verschlüsselungsalgorithmus und der Entschlüsselungsalgorithmus sind identisch, ...
- ... extrem einfach ...
- ... und sehr schnell – vorausgesetzt die Schlüsselfolge ist schon vorhanden. Für hohe Datenübertragungsraten kann man evtl. den Schlüsselbitstrom auf beiden Seiten vorherberechnen.
- Bei gut gewählter Schlüsselerzeugung ist sehr hohe Sicherheit möglich.

Nachteile

- Das Verfahren ist anfällig gegen Klartextraten; jedes erratene Klartextbit ergibt ein Schlüsselbit.
- Die Qualität der Schlüsselfolge ist sehr kritisch.
- Es gibt keine Diffusion – bei Blockchiffren war das ein sehr wichtiges Kriterium.
- Der Angreifer kann bei bekanntem Klartextstück das entsprechende Schlüsselstück ermitteln und dann den Klartext beliebig austauschen – z. B. „ich liebe dich“ durch „ich hasse dich“ ersetzen oder einen Geldbetrag von 1000 auf 9999 ändern.

Im Zusammenhang mit dem ersten Punkt hat der gewöhnliche Zeichensatz für Texte eine systematische Schwachstelle: Die Kleinbuchstaben `a..z` beginnen im 8-Bit-Code alle mit `011`, die Großbuchstaben `A..Z` alle mit `010`. Eine vermutete Folge von sechs Kleinbuchstaben enthüllt 6*3 = 18 Schlüsselbits.

[Das Auftreten vieler Nullen in den Leitbits der Bytes ist übrigens ein sehr wichtiges Erkennungsmerkmal für natürlichsprachigen Text in europäischen Sprachen.]

Kryptographische Sicherheit von Zufallsgeneratoren

Die entscheidende Frage an eine Pseudozufallsfolge bzw. an den sie erzeugenden Zufallsgenerator ist:

> Kann man aus einem bekannten (auch fragmentierten) Stück der Folge weitere Bits – vorwärts oder rückwärts – bestimmen?

Die Antwort für die „klassischen“, in statistischen Anwendungen und Simulationen verwendeten Zufallsgeneratoren wird JA sein. Wir werden aber auch Zufallsgeneratoren kennen lernen, die in diesem Sinne – vermutlich – kryptographisch sicher sind.

1.3 Lineare Kongruenzgeneratoren

Die erste wichtige Klasse von elementaren – „klassischen“ – Zufallsgeneratoren sind diejenigen einstufig rekurrenten, die lineare Kongruenzen verwenden. Sie haben zunächst den Vorteil, dass sie sehr schnell sind. Sie erzeugen aber auch lange Perioden, und ihre Zufallsqualitäten lassen sich wegen ihrer einfachen Bauart leicht theoretisch absichern.

Die Zufallserzeugung mit linearen Kongruenzen geht so:

$$x_n = s(x_{n-1}) \text{ mit}$$

$$s : \mathbb{Z}/m\mathbb{Z} \longrightarrow \mathbb{Z}/m\mathbb{Z}, \quad s(x) = ax + b \bmod m.$$

Die Folge hängt also von vier ganzzahligen Parametern ab; diese sind

- der **Modul** m mit $m \geq 2$,
- der **Multiplikator** $a \in [0 \ldots m-1]$,
- das **Inkrement** $b \in [0 \ldots m-1]$,
- der **Startwert** $x_0 \in [0 \ldots m-1]$.

Ein solcher Zufallsgenerator heißt **linearer Kongruenzgenerator**, wobei man im Fall $b = 0$ auch von einem **multiplikativen Generator**, im Falle $b \neq 0$ von einem **gemischten Kongruenzgenerator** spricht. Ein solcher Generator ist sehr einfach zu programmieren, selbst in Assembler, und ist sehr schnell. Gut ist er, *wenn die Parameter m, a, b geeignet gewählt sind.* Der Startwert ist dagegen völlig problemlos frei wählbar. Auch das ist wichtig, um bei Bedarf die erzeugten Zufallszahlen genügend variieren zu können.

Bei der Anwendung für die Bitstrom-Verschlüsselung wird der Startwert x_0 oder aber der ganze Parametersatz (m, a, b, x_0) als effektiver Schlüssel betrachtet, d. h., geheim gehalten.

Bemerkungen

1. Da nur endlich viele Werte x_n möglich sind, ist die Folge periodisch mit einer Periodenlänge $\leq m$; dabei kann auch am Anfang eine Vorperiode auftreten.

2. Die Wahl von $a = 0$ ist offensichtlich unsinnig. Aber auch $a = 1$ erzeugt eine Folge, nämlich $x_0, x_0 + b, x_0 + 2b, x_0 + 3b, \ldots$, die nicht brauchbar ist, weil sie auch mod$m$ immer wieder lange regelmäßige Stücke enthält.

3. Für $m = 13$, $a = 6$, $b = 0$, $x_0 = 1$ wird die Folge

$$6, 10, 8, 9, 2, 12, 7, 3, 5, 4, 11, 1$$

der Periodenlänge 12 erzeugt, die der Vorstellung einer zufälligen Permutation der Zahlen 1 bis 12 schon recht nahe kommt (trotz des sehr kleinen Moduls).

4. Nimmt man statt dessen den Multiplikator 7, so entsteht die deutlich weniger sympathische Folge

$$7, 10, 5, 9, 11, 12, 6, 3, 8, 4, 2, 1.$$

5. Ist a zu m teilerfremd, so ist die Folge rein-periodisch (d. h., es gibt keine Vorperiode). Es ist nämlich $a \bmod m$ invertierbar, also $ac \equiv 1 \pmod{m}$ für ein c. Daher ist stets $x_{n-1} = cx_n - cb \bmod m$. Ist nun $x_{\mu+\lambda} = x_\mu$ mit $\mu \geq 1$, so auch $x_{\mu+\lambda-1} = x_{\mu-1}$ usw., schließlich $x_\lambda = x_0$.

6. Durch Induktion beweist man sofort

$$x_k = a^k x_0 + (1 + a + \cdots + a^{k-1}) \cdot b \bmod m$$

für alle k – ein krasser Hinweis darauf, wie wenig zufällig die Folge in Wirklichkeit ist: Sogar der direkte Zugriff auf ein beliebiges Folgenglied ist möglich, denn der Koeffizient von b ist $(a^k - 1)/(a - 1)$, wobei die Division mod m vorzunehmen ist.

7. Sei $m = 2^e$ und a gerade. Dann ist

$$x_k = (1 + a + \cdots + a^{e-1}) \cdot b \bmod m$$

für alle $k \geq e$, die Periode also 1. Allgemein verkürzen gemeinsame Teiler von a und m die Periode und sind daher zu vermeiden.

8. Sei d ein Teiler des Moduls m. Die Folge $y_n = x_n \bmod d$ ist dann die entsprechende Kongruenzfolge zum Modul d, also $y_n = ay_{n-1} + b \bmod d$. Die Folge (x_n) hat $\bmod d$ also eine Periode $\leq d$, die eventuell sehr kurz ist.

9. Besonders drastisch ist dieser Effekt im Fall einer Zweierpotenz, $m = 2^e$, zu sehen: Das niedrigste Bit von x_n hat dann bestenfalls die Periode 2, ist also abwechselnd 0 und 1, wenn es nicht überhaupt konstant ist. Die k niedrigsten Bits zusammen haben höchstens die Periode 2^k.

10. Ein Modul m mit vielen Teilern, insbesondere eine Zweierpotenz, ist also gegenüber einem Primzahlmodul bei der Zufallserzeugung gehandikapt. Die Qualität ist aber oft doch noch ausreichend, wenn man die erzeugten Zahlen durch m dividiert, also als Zufallszahlen im reellen Intervall $[0, 1[$ ansieht, und am rechten Ende großzügig rundet. Für kryptographische Anwendungen sind solche Moduln aber sicher nicht geeignet.

11. Im Beispiel $m = 2^{32}$, $a = 4095 = 2^{12} - 1$, $b = 12794$ sind die Parameter nicht geeignet gewählt: Aus $x_0 = 253$ ergibt sich $x_1 = 1048829$ und $x_2 = 253 = x_0$.

    Beliebte Moduln sind

    - $m = 2^{32}$, weil er den 32-Bit-Bereich ausschöpft und außerdem sehr effizient handhabbar ist,
    - $m = 2^{31} - 1$, weil dies oft die maximale darstellbare Ganzzahl ist und weil man damit fast so effizient rechnen kann wie mit einer Zweierpotenz. Ein weiterer Vorteil: Diese Zahl ist eine Primzahl (von MERSENNE 1644 behauptet, von EULER 1772 bewiesen), und das hat gute Auswirkungen auf die Qualität der erzeugten Zufallsfolge. Allgemeiner sind FERMAT-Primzahlen $2^k + 1$ und MERSENNE-Primzahlen $2^k - 1$ ähnlich gut geeignet; die nächste solche Zahl ist $2^{61} - 1$.

Die ersten 100 Glieder einer Folge, die mit dem Modul $m = 2^{31} - 1 = 2147483647$, dem Multiplikator $a = 397204094$, dem Inkrement $b = 0$ und dem Startwert $x_0 = 58854338$ erzeugt wurde, zeigt Tabelle 1. In Abbildung 1 ist diese Information visuell umgesetzt. Man sieht daran schon eine deutliche Regellosigkeit der Folge. Es wird aber auch klar, dass ein solcher visueller Eindruck wohl kaum ausreicht, um die Qualität einer Zufallsfolge zu beurteilen.

Abbildung 1: Eine lineare Kongruenzfolge. Waagerechte Achse: Zähler von 0 bis 100, senkrechte Achse: Größe des Folgenglieds von 0 bis $2^{31} - 1$.

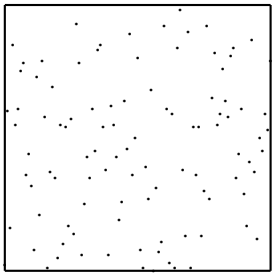


Tabelle 1: 100 Glieder einer linearen Kongruenzfolge

1292048469	319941267	173739233	1992841820
345565651	2011011872	31344917	592918912
1827933824	1691830787	857231706	1416540893
1184833417	145217588	589958351	1776690121
1330128247	558009026	1479515830	1197548384
1627901332	929586843	19840670	1268974074
1682548197	760357405	666131673	1642023821
787305132	1314353697	167412640	1377012759
963849348	971229179	247170576	1250747100
703109068	1791051358	1978610456	1746992541
177131972	1844679385	1328403386	1811091691
1586500120	1175539757	74957396	753264023
468643347	821920620	1269873360	963348259
1698955999	139484430	30476960	1327705603
1266305157	1337811914	1808105128	640050202
37935526	1185470453	2111728842	380228478
808553600	934194915	824017077	881361640
1492263703	414709486	298916786	1883338449
771128019	558671080	1935988732	798347213
120356246	1378842534	37149011	272238278
1190345324	1006355270	1161592162	1079789655
220609946	1918105148	791775291	979447727
1160648370	779600833	1170336930	1271974642
375813045	1089009771	280197098	1144249742
1236647368	1729816359	650188387	1714906064

1.4 Die maximale Periode

Wann hat ein linearer Kongruenzgenerator zum Modul m die maximal mögliche Periode m? Für einen multiplikativen Generator ist das nicht möglich, weil man vom Folgenglied 0 nie mehr wegkommt. Für diese Frage sind also nur gemischte Kongruenzgeneratoren von Interesse. Der triviale Generator mit erzeugender Funktion $s(x) = x + 1 \bmod m$ zeigt, dass dann die Periodenlänge m möglich ist; er zeigt natürlich auch, dass die maximale Periodenlänge noch lange nicht hinreicht, um die Qualität eines Zufallsgenerators nachzuweisen. Das allgemeine Ergebnis ist leicht formuliert:

Satz 1 (Hull/Dobell **1962,** Knuth**)** *Der lineare Kongruenzgenerator mit erzeugender Funktion $s(x) = ax + b \bmod m$ hat genau dann die Periode m, wenn folgende drei Bedingungen erfüllt sind:*

(i) *b und m sind teilerfremd.*

(ii) *Jeder Primteiler p von m teilt auch $a - 1$.*

(iii) *Ist m durch 4 teilbar, so auch $a - 1$.*

Die erste Bedingung bedeutet insbesondere $b \neq 0$, so dass also wirklich ein gemischter Kongruenzgenerator vorliegt. Dem Beweis wird ein Hilfssatz vorangestellt (und es werden zwei weitere Hilfssätze aus Kapitel III, Anhang A.1 verwendet).

Hilfssatz 1 *Sei $m = m_1 m_2$ mit teilerfremden natürlichen Zahlen m_1 und m_2. Seien λ, λ_1 und λ_2 die Perioden der Kongruenzgeneratoren $x_n = s(x_{n-1}) \bmod m$ bzw. $\bmod m_1$ bzw. $\bmod m_2$ zum Startwert x_0. Dann ist λ das kleinste gemeinsame Vielfache von λ_1 und λ_2.*

Beweis. Seien $x_n^{(1)}$ und $x_n^{(2)}$ die entsprechenden Folgeglieder für m_1 bzw. m_2. Dann ist $x_n^{(i)} = x_n \bmod m_i$. Da $x_{n+\lambda} = x_n$ für alle genügend großen n, folgt sofort, dass λ ein Vielfaches von λ_1 und λ_2 ist. Umgekehrt folgt aus $m \mid t \iff m_1, m_2 \mid t$, dass

$$x_n = x_k \iff x_n^{(i)} = x_k^{(i)} \quad \text{für } k = 1 \text{ und } 2.$$

Also ist λ höchstens gleich dem kleinsten gemeinsamen Vielfachen von λ_1 und λ_2. $\diamond$

Beweis des Satzes. Für beide Beweisrichtungen kann man nach dem Hilfssatz 1 o. B. d. A. $m = p^e$ mit einer Primzahl p annehmen.

„$\Longrightarrow$“: Da jede Zahl in $[0 \ldots m-1]$ genau einmal vorkommt, darf man o. B. d. A. $x_0 = 0$ annehmen. Dann ist

$$x_n = (1 + a + \cdots + a^{n-1}) \cdot b \bmod m \quad \text{für alle } n.$$

Da x_n auch den Wert 1 annimmt, muss schon mal b zu m teilerfremd sein. Da $x_m = 0$, folgt nun $m \mid 1 + a + \cdots + a^{m-1}$, also

$$p \mid m \mid a^m - 1 = (a-1)(1 + a + \cdots + a^{m-1}).$$

Nach dem kleinen Satz von FERMAT ist $a^p \equiv a \pmod{p}$, also $a^m = a^{p^e} \equiv a^{p^{e-1}} \equiv \ldots \equiv a \pmod{p}$, also $p \mid a - 1$. Die Aussage (iii) ist der Fall $p = 2$ mit $e \geq 2$. Wegen der Aussage (ii) muss a schon mal ungerade sein. Wäre nun $a \equiv 3 \pmod{4}$, so nach Hilfssatz 1 in III.A.1 bereits $x_{m/2} = 0$. Also muss $a \equiv 1 \pmod{4}$ sein.

„$\Longleftarrow$“: Auch hier kann man wieder o. B. d. A. $x_0 = 0$ annehmen. Dann ist

$$x_n = 0 \Longleftrightarrow m \mid 1 + a + \cdots + a^{n-1}.$$

Insbesondere ist der Fall $a = 1$ trivial. Sei also o. B. d. A. $a \geq 2$. Dann ist weiter

$$x_n = 0 \Longleftrightarrow m \mid \frac{a^n - 1}{a - 1}.$$

Zu zeigen ist:

- $m \mid \frac{a^m-1}{a-1}$ – dann ist $\lambda | m$;
- m kein Teiler von $\frac{a^{m/p}-1}{a-1}$ – da m eine p-Potenz ist, folgt dann $\lambda \geq m$.

Sei p^h die maximale Potenz, die in $a-1$ aufgeht. Nach Hilfssatz 2 in III.A.1 ist dann

$$a^p \equiv 1 \pmod{p^{h+1}},\ a^p \not\equiv 1 \pmod{p^{h+2}}$$

und sukzessive

$$a^{p^k} \equiv 1 \pmod{p^{h+k}},\ a^{p^k} \not\equiv 1 \pmod{p^{h+k+1}}$$

für alle k. Insbesondere folgt $p^{h+e} \mid a^m - 1$. Da in $a-1$ höchstens p^h aufgeht, folgt $m = p^e \mid \frac{a^m-1}{a-1}$. Wäre $p^e \mid \frac{a^{m/p}-1}{a-1}$, so $p^{e+h} \mid a^{p^{e-1}} - 1$, Widerspruch. $\diamondsuit$

Dieser Satz ist vor allem für Zweierpotenz-Moduln von Interesse; für Primzahl-Moduln dagegen ergibt er kein brauchbares Ergebnis.

Korollar 1 (GREENBERGER 1961) *Ist $m = 2^e$ mit $e \geq 2$, so wird die Periode m genau dann erreicht, wenn gilt:*

(i) *b ist ungerade.*

(ii) *$a \equiv 1 \pmod{4}$.*

Korollar 2 *Ist m eine Primzahl, so wird die Periode m genau dann erreicht, wenn b zu m teilerfremd und $a = 1$ ist.*

Dieses (traurige) Ergebnis lässt sich etwas allgemeiner fassen – auch für beliebige quadratfreie Moduln m gibt es keine brauchbaren linearen Kongruenzgeneratoren der Periode m:

Korollar 3 *Ist m quadratfrei, so wird die Periode m genau dann erreicht, wenn b zu m teilerfremd und $a = 1$ ist.*

Wir haben nun mit Satz 1 die überhaupt größtmögliche Periode erreicht und mit Korollar 1 auch einen brauchbaren Spezialfall gefunden.

1.5 Die maximale Periode multiplikativer Generatoren

Multiplikative Generatoren $x_n = ax_{n-1} \bmod m$ können nie die Periode m erreichen, da das Folgenglied 0 nie mehr verlassen wird. Was können sie bestenfalls? – λ ist im folgenden Satz die CARMICHAEL-Funktion und wurde genau in diesem Zusammenhang erstmals eingeführt.

Satz 2 (CARMICHAEL 1910) *Die maximale Periode eines multiplikativen Generators mit erzeugender Funktion $s(x) = ax \bmod m$ ist $\lambda(m)$. Sie wird insbesondere dann erreicht, wenn gilt:*

(i) *a ist primitiv $\bmod m$.*

(ii) *x_0 ist teilerfremd zu m.*

Beweis. Es ist $x_n = a^n x_0 \bmod m$. Ist $k = \mathrm{Ord}_m a$ die Ordnung von a, so $x_k = x_0$, also die Periode $\leq k \leq \lambda(m)$. Sei nun a primitiv $\bmod m$, also $1, a, \ldots, a^{\lambda(m)-1} \bmod m$ verschieden. Da x_0 zu m teilerfremd ist, folgt, dass die Periode $\lambda(m)$ ist. $\diamond$

Korollar 1 *Ist $m = p$ eine Primzahl, so wird die maximale Periode $\lambda(p) = p - 1$ genau dann erreicht, wenn gilt:*
(i) *a ist primitiv $\bmod p$.*
(ii) *$x_0 \neq 0$.*

Für Primzahlmoduln ist die Situation bei den multiplikativen Generatoren also sehr gut: Die Periode ist nur um 1 kleiner als überhaupt mit einstufiger Rekursion möglich und jeder Startwert außer 0 ist geeignet.

Dieses Ergebnis wird in Abschnitt 1.9 weitgehend verallgemeinert.

1.6 Lineare Schieberegister

Neben den bisher behandelten linearen Kongruenzgeneratoren gibt es eine andere klassische und weitverbreitete Methode zur Erzeugung von Pseudozufallsfolgen: die Schieberegister-Methode. Diese Methode wurde von GOLOMB 1955 erstmals vorgeschlagen, wird aber meist nach TAUSWORTHE benannt, der die Idee 1965 in einer Arbeit aufgriff. Sie ist besonders leicht in Hardware zu realisieren. Für die theoretische Beschreibung fasst man Blöcke von jeweils l Bits als Elemente des Vektorraums $\mathbb{F}_2^l$ über dem Körper $\mathbb{F}_2$ aus zwei Elementen auf.

Eine lineare Abbildung

$$A\colon \mathbb{F}_2^l \longrightarrow \mathbb{F}_2$$

ist nichts anderes als eine Vorschrift, aus einem l-Bit-Block eine Teilsumme zu bilden:

$$Au = \sum_{i=1}^{l} a_i u_i,$$

wobei alle Koeffizienten a_i ja 0 oder 1 sind. Als potenzielle Zufallsfolge wird die Folge von Bits betrachtet, die nach der Vorschrift

$$u_n = a_1 u_{n-1} + \cdots + a_l u_{n-l}$$

entsteht. Man braucht als Parameter des Verfahrens

- die **Registerlänge** l mit $l \geq 2$,
- eine **Rückkopplungsvorschrift** A, die eine Folge $(a_1, \ldots, a_l) \in \mathbb{F}_2^l$ ist, und daher auch durch eine Teilmenge $I \subseteq \{1, \ldots, l\}$ beschrieben werden kann.
- einen **Startwert** $u = (u_{l-1} \ldots u_0)$ aus l Bits.

Die Iterationsformal lässt sich damit auch in der Form

$$u_n = \sum_{j \in I} u_{n-j}$$

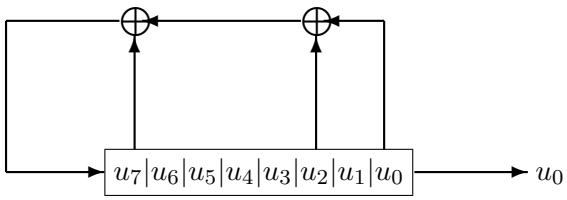
schreiben.

Die Hardware-Realisierung stellt man sich so vor, dass das rechte Bit des Schieberegisters ausgegeben wird, die übrigen $l - 1$ Bits nach rechts nachrücken und auf der linken Seite als „Rückkopplung“ die Summe der durch I angegebenen Bits nachgeschoben wird, siehe Abbildung 2.

Bei der Anwendung für die Bitstrom-Verschlüsselung wird der Startwert u oder aber alle drei Parameter l, I, u als Schlüssel betrachtet, d. h., geheim gehalten.

Bei geschickter Wahl der Parameter, die hier nicht weiter behandelt wird, hat die Folge eine Periode nahe 2^l und ist durch statistische Tests praktisch nicht von einer gleichverteilten Zufallsfolge zu unterscheiden, siehe 1.9 und 1.10.

Abbildung 2: Ein lineares Schieberegister



1.7 Mehrstufige Generatoren

Die gemeinsame Verallgemeinerung von linearen Kongruenzgeneratoren und linearen Schieberegister-Generatoren sind die **mehrstufigen linearen Rekurrenzgeneratoren**. Sie lassen sich bequem im Rahmen eines endlichen Rings R (kommutativ mit 1) behandeln; damit sind nicht nur die Ringe $\mathbb{Z}/m\mathbb{Z}$ erfasst, sondern auch die endlichen Körper zusätzlich zu den Primkörpern $\mathbb{F}_p$, die ebenfalls zur Zufallserzeugung benützt werden können. Bei einem r-stufigen linearen Rekurrenzgenerator wird eine Folge (x_n) in R nach der Vorschrift

$$x_n = a_1 x_{n-1} + \cdots + a_r x_{n-r} + b$$

erzeugt. Als Parameter braucht man

- die **Rekursionstiefe** r (o. B. d. A. $a_r \neq 0$),
- die **Koeffizientenfolge** $a = (a_1, \ldots, a_r) \in R^r$,
- das **Inkrement** $b \in R$,
- einen **Startvektor** $(x_0, \ldots, x_{r-1}) \in R^r$.

Der lineare Rekurrenzgenerator heißt **homogen** oder **inhomogen**, je nachdem, ob $b = 0$ ist oder nicht.

Die Funktionsweise eines linearen Rekurrenzgenerators kann man ähnlich einem Schieberegister veranschaulichen, siehe Abbildung 3.

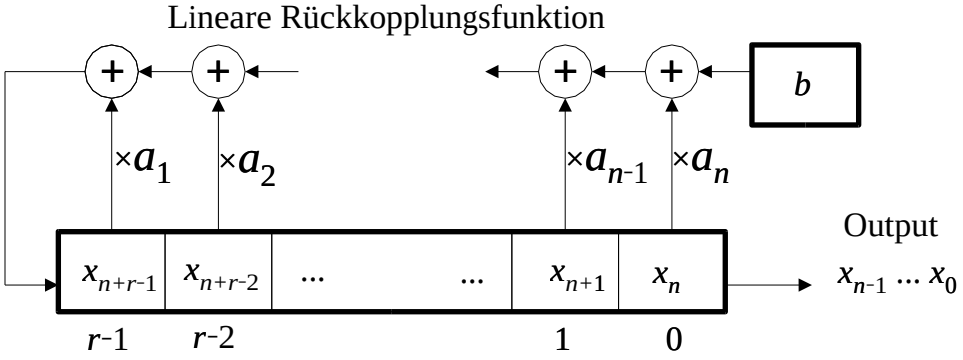


Abbildung 3: Ein linearer Rekurrenzgenerator

Inhomogene lineare Rekurrenzgeneratoren kann man leicht auf homogene reduzieren, wobei man allerdings eine Rekursionsstufe zusätzlich in Kauf nehmen muss: Aus den beiden Gleichungen

$$\begin{aligned} x_{n+1} &= a_1 x_n + \cdots + a_r x_{n-r+1} + b, \\ x_n &= a_1 x_{n-1} + \cdots + a_r x_{n-r} + b, \end{aligned}$$

folgt nämlich durch Subtraktion

$$x_{n+1} = (a_1 + 1)x_n + (a_2 - a_1)x_{n-1} \cdots + (-a_r)x_{n-r}.$$

Im Falle $r = 1$, $x_n = ax_{n-1} + b$, wird diese Formel zu

$$x_n = (a + 1)x_{n-1} - ax_{n-2}.$$

Daher wird der inhomogene Fall im folgenden vernachlässigt.

Im homogenen Fall kann man unter Verwendung der **Zustandsvektoren** $x_{(n)} = (x_n, \ldots, x_{n+r-1})^t$ schreiben

$$x_{(n)} = Ax_{(n-1)} \quad \text{für } n \geq 1,$$

mit der **Begleitmatrix**

$$A = \begin{pmatrix} 0 & 1 & \ldots & 0 \\ & \ddots & \ddots & \\ & & & 1 \\ a_r & a_{r-1} & \ldots & a_1 \end{pmatrix}.$$

Die nächste Stufe der Verallgemeinerung ist also ein **Matrixgenerator**. Parameter sind:

- eine $r \times r$-Matrix $A \in M_r(R)$,
- ein Startvektor $x_0 \in R^r$.

Die Folge wird gebildet nach der Formel

$$x_n = Ax_{n-1} \in R^r.$$

1.8 Allgemeine lineare Generatoren

Noch allgemeiner (und begrifflich einfacher) ist die abstrakt-algebraische Version, der **allgemeine lineare Generator**. Gegeben sind:

- ein Ring R (kommutativ und mit Einselement),
- ein R-Modul M,
- eine R-lineare Abbildung $A : M \longrightarrow M$,
- ein Startwert $x_0 \in M$.

Daraus wird eine Folge $(x_n)_{n\in\mathbb{N}}$ gebildet nach der Formel

$$x_n = Ax_{n-1} \quad \text{für } n \geq 1.$$

Beispiele

1. Für einen homogenen linearen Kongruenzgenerator ist

$$R = \mathbb{Z}/m\mathbb{Z}, \quad M = R \quad (r = 1), \quad A = (a).$$

2. Für einen inhomogenen linearen Kongruenzgenerator ist

$$R = \mathbb{Z}/m\mathbb{Z}, \quad M = R^2 \quad (r = 2), \quad A = \begin{pmatrix} 0 & 1 \\ -a & a+1 \end{pmatrix}.$$

3. Für ein lineares Schieberegister ist

$$R = \mathbb{F}_2, \quad M = \mathbb{F}_2^l \quad (r = l), \quad A = \text{die Begleitmatrix},$$

   die nur aus Nullen und Einsen besteht.

Falls M endlich ist, kann die Rekursion nur endlich viele verschiedene Werte annehmen, muss also nach einer eventuellen Vorperiode periodisch werden.

Satz 3 *Sei M ein endlicher R-Modul und $A : M \longrightarrow M$ linear. Genau dann, wenn A bijektiv ist, sind alle vom zugehörigen allgemeinen linearen Generator erzeugten Folgen rein-periodisch.*

Beweis. Sei A bijektiv und x_0 ein Startvektor. Sei t der kleinste Index, so dass x_t einen bereits vorher durchlaufenen Wert annimmt, und sei s der kleinste Index mit $x_t = x_s$. Wäre $s \geq 1$, so $x_s = Ax_{s-1}$ und $x_t = Ax_{t-1}$, also

$$x_{t-1} = A^{-1}x_t = A^{-1}x_s = x_{s-1},$$

im Widerspruch zur Minimalität von t.

Sei umgekehrt A nicht bijektiv; da M endlich ist, ist A dann auch nicht surjektiv. Man kann also $x_0 \in M - A(M)$ wählen. Dann kann niemals $x_0 = Ax_t$ sein, die Folge ist also nicht reinperiodisch. $\diamond$

Dieses Ergebnis lässt sich über die Begleitmatrix auf homogene mehrstufige Kongruenzgeneratoren, insbesondere auf lineare Schieberegister anwenden:

Korollar 1 *Ein homogener linearer Kongruenzgerator der Rekursionstiefe r erzeugt stets rein-periodische Folgen, wenn der Koeffizient a_r in $\mathbb{Z}/m\mathbb{Z}$ invertierbar ist. Ein lineares Schieberegister der Länge l erzeugt rein-periodische Folgen, wenn der Rückkopplungskoeffizient $a_l \neq 0$ ist.*

Die erste Aussage gilt auch im nicht-homogenen Fall, da die Formel

$$x_{n-r} = a_r^{-1}(x_n - a_1 x_{n-1} - \cdots - a_{r-1} x_{n-r+1} - b)$$

für die Rückwärtsberechnung der Folge sorgt.

1.9 Matrixgeneratoren über endlichen Körpern

Ein Matrix-Generator über einem Körper K wird durch eine $r \times r$-Matrix

$$A \in M_r(K)$$

vollständig beschrieben (bis auf die Wahl des Startvektors $x_0 \in K^r$). Das Ziel dieses Abschnitts ist die Charakterisierung der Folgen mit maximaler Periodenlänge.

Im Polynomring $K[T]$ in einer Unbestimmten T bildet die Menge

$$\{\rho \in k[T] \mid \rho(A) = 0\}$$

ein Ideal. Da $k[T]$ Hauptidealring ist (sogar euklidischer Ring), wird dieses Ideal von einem eindeutig bestimmten normierten Polynom μ erzeugt; dieses heißt das **Minimalpolynom** von A. Da A auch Nullstelle seines charakteristischen Polynoms χ ist, gilt also $\mu|\chi$. Ist A invertierbar, so ist das absolute Glied von μ nicht 0; denn sonst hätte μ die Nullstelle 0 und A den Eigenwert 0.

Hilfssatz 2 *Sei K ein Körper, $A \in GL_r(K)$ von endlicher Ordnung t, μ das Minimalpolynom von A, $s = \operatorname{Grad} \mu$, $R := K[T]/\mu K[T]$ und $a \in R$ die Restklasse von T. Dann gilt:*

$$a^k = 1 \Longleftrightarrow \mu | T^k - 1 \Longleftrightarrow A^k = \mathbf{1}.$$

Insbesondere ist $a \in R^\times$, t auch die Ordnung von a und $\mu | T^t - 1$.

Beweis. R ist eine K-Algebra der Dimension s. Ist $\mu = b_s T^s + \cdots + b_0$, so

$$\mu - b_0 = T \cdot (b_s T^{s-1} + \cdots + b_1);$$

da $b_0 \neq 0$, ist also T mod μ invertierbar, also $a \in R^\times$. Da a^k die Restklasse von T^k ist, folgt die behauptete Äquivalenzkette. $\diamond$

Korollar 1 *Ist K ein endlicher Körper mit q Elementen, so ist*

$$t \leq \#R^\times \leq q^s - 1 \leq q^r - 1.$$

Sei von jetzt an K ein endlicher Körper mit q Elementen. Dann ist auch die Gruppe $GL_r(K)$ der invertierbaren $r \times r$-Matrizen endlich. Der Vektorraum K^r besteht aus q^r Vektoren. Wir wissen bereits, dass jede Folge, die von dem Matrixgenerator zu A erzeugt wird, rein-periodisch ist. Eine volle Periode wird immer vom Nullvektor $0 \in K^r$ alleine gebildet. Alle übrigen Vektoren werden im allgemeinen auf mehrere Perioden aufgeteilt sein. Ist s die Länge einer solchen Periode und x_0 der entsprechende Startvektor, so

ist $x_0 = x_s = A^s x_0$. Also hat A^s den Eigenwert 1 und folglich A eine s-te Einheitswurzel als Eigenwert.

Denkbar ist aber auch, dass alle Vektoren $\neq 0$ zusammen eine Periode der maximal möglichen Länge $q^r - 1$ bilden. In diesem Fall gilt $A^s x = x$ für alle Vektoren $x \in K^r$ mit $s = q^r - 1$, aber für keinen kleineren Exponenten > 0. Also ist $t = q^r - 1$ die Ordnung von A. Damit ist gezeigt:

Korollar 2 *Ist K endlich mit q Elementen, so gilt:*

(i) *Erzeugt der Matrixgenerator zu A für einen Startvektor $\neq 0$ eine Folge der Periode s, so hat A eine s-te Einheitswurzel als Eigenwert.*
(ii) *Gibt es eine Periode der Länge $q^r - 1$, so ist $t = q^r - 1$ die Ordnung von A.*

Hilfssatz 3 *Sei K ein endlicher Körper mit q Elementen und $\varphi \in K[T]$ ein irreduzibles Polynom vom Grad d. Dann gilt $\varphi | T^{q^d - 1} - 1$.*

Beweis. Der Restklassenring $R = k[T]/\varphi K[T]$ ist ein Erweiterungskörper vom Grad $d = \mathrm{Dim}_K R$, hat also $h := q^d$ Elemente und enthält mindestens eine Nullstelle a von φ, nämlich die Restklasse von T. Da jedes $x \in R^\times$ die Gleichung $x^{h-1} = 1$ erfüllt, ist insbesondere a auch Nullstelle von $T^{h-1} - 1$. Also ist $\mathrm{ggT}(\varphi, T^{h-1} - 1)$ nicht konstant. Da φ irreduzibel ist, folgt $\varphi | T^{h-1} - 1$. $\diamond$

Definition. Ein Polynom $\varphi \in K[T]$ vom Grad d über dem endlichen Körper K mit q Elementen heißt **primitiv**, wenn φ irreduzibel und kein Teiler von $T^k - 1$ ist für $1 \leq k < q^d - 1$.

Hauptsatz 1 *Sei K ein endlicher Körper mit q Elementen und $A \in GL_r(K)$. Dann sind folgende Aussagen äquivalent:*

(i) *Der Matrixgenerator zu A erzeugt eine Folge der Periode $q^r - 1$.*
(ii) *A hat die Ordnung $q^r - 1$.*
(iii) *Das charakteristische Polynom χ von A ist primitiv.*

Beweis. „(i) $\Longrightarrow$ (ii)“: Siehe Korollar 2 (ii).

„(ii) $\Longrightarrow$ (iii)“: In Korollar 1 ist $t = q^r - 1$. Also ist $\#R^\times = q^s - 1$, also R ein Körper und daher μ irreduzibel. Ferner ist $s = r$, also $\mu = \chi$, und μ nach Hilfssatz 2 kein Teiler von $T^k - 1$ für $1 \leq k < q^r - 1$, also μ primitiv.

„(iii) $\Longrightarrow$ (i)“: Da χ irreduzibel ist, ist $\chi = \mu$. Die Restklasse a von T ist Nullstelle von μ und hat nach der Definition von „primitiv“ die multiplikative Ordnung $q^r - 1$. Da das Potenzieren mit q ein Automorphismus des Körpers R ist, der K elementweise festlässt, sind auch die r Potenzen a^{q^k} für $0 \leq k <$

r Nullstellen von μ, und zwar alle verschieden. Dies müssen daher sämtliche Nullstellen sein, und alle haben die multiplikative Ordnung $q^r - 1$. Daher hat A keinen Eigenwert von geringerer Ordnung und daher gibt es nach Korollar 2 (i) auch keine kürzere Periode. $\diamond$

Für ein lineares Schieberegister ist A die Begleitmatrix wie in 1.7. Das charakterische Polynom ist also $T^l - a_1 T^{l-1} - \cdots - a_l$.

Korollar 1 *Ein lineares Schieberegister der Länge l erzeugt genau dann eine Folge der maximal möglichen Periode $2^l - 1$, wenn sein charakteristisches Polynom primitiv und der Startwert $\neq 0$ ist.*

Die Konstruktion von linearen Schieberegistern, die Folgen maximaler Periode erzeugen, ist also auf die Konstruktion primitiver Polynome über dem Körper $\mathbb{F}_2$ zurückgeführt.

Im eindimensionalen Fall $r = 1$ erhalten wir speziell den multiplikativen Generator mit der Rekursionsvorschrift $x_n = ax_{n-1}$ über dem endlichen Körper K mit q Elementen. Die zugehörige Matrix $A = (a)$ bewirkt die Multiplikation mit a, also ist a der einzige Eigenwert und $\chi = T - a \in K[T]$ das charakteristische Polynom. Dieses ist, da linear, in jedem Fall irreduzibel. Da

$$\chi | T^k - 1 \Longleftrightarrow a \text{ Nullstelle von } T^k - 1 \Longleftrightarrow a^k = 1,$$

ist χ also genau dann primitiv, wenn a erzeugendes Element der multiplikativen Gruppe $K^\times$, also primitives Element ist. Damit ist die folgende leichte Verallgemeinerung des Korollars zu Satz 2 gezeigt:

Korollar 2 *Ein multiplikativer Generator über K mit Multiplikator a erzeugt genau dann eine Folge der Periode $q - 1$, wenn a primitives Element und der Startwert $x_0 \neq 0$ ist.*

1.10 Statistische Eigenschaften von linearen Schieberegistern

Die statistischen Eigenschaften von Schieberegisterfolgen der maximalen Periode $2^l - 1$, wobei l die Länge des Schieberegisters ist, wurden bereits von GOLOMB ausführlich untersucht.

Referenz:

Solomon E. GOLOMB: **Shift Register Sequences.** Revised Edition, Aegean Park Press, Laguna Hills 1982. ISBN 0-89412-048-4

Hier einige Aussagen dazu:

Bemerkungen

1. In jeder vollen Periode kommen genau 2^{l-1} Einsen und $2^{l-1} - 1$ Nullen vor.

   Beweis. Es werden alle 2^l Zustandsvektoren $\in \mathbb{F}_2^l$ außer 0 jeweils genau einmal angenommen; das entspricht den ganzen Zahlen im Intervall $[1 \dots 2^l - 1]$. Davon sind 2^{l-1} ungerade, der Rest gerade, und ihre Paritäten bilden genau die Output-Folge des Schieberegisters.

2. Ein **Run** in einer Folge ist ein maximales konstantes Stück.

   Beispiel: $\dots 0111110 \dots$ ist ein Einser-Run der Länge 5.

   Bedenkt man, dass die Stücke der Länge l der Schieberegister-Folge genau die verschiedenen Zustandsvektoren $\neq 0$ sind, so ist klar, dass in der vollen Periode folgendes vorkommt:

   - Kein Run der Länge $> l$.
   - Genau ein Einser-Run und kein Nuller-Run der Länge l – denn sonst käme der Nuller-Zustand vor bzw. der Einser-Zustand öfter als einmal vor.
   - Jeweils genau ein Einser- und Nuller-Run der Länge $l - 1$.
   - Allgemein jeweils genau 2^{k-1} Einser- und Nuller-Runs der Länge $l - k$ für $1 \leq k \leq l - 1$.
   - Insbesondere genau 2^{l-1} Runs der Länge 1, davon jeweils genau die Hälfte Nullen und Einsen.

3. Für eine periodische Folge $x = (x_n)_{n \in \mathbb{N}}$ in $\mathbb{F}_2$ der Periode s ist die **Autokorrelation** zur Verschiebung um t definiert durch

$$\begin{aligned} \kappa_x(t) &= \frac{1}{s} \cdot [\#\{n \mid x_{n+t} = x_n\} - \#\{n \mid x_{n+t} \neq x_n\}] \\ &= \frac{1}{s} \cdot \sum_{n=0}^{s-1} (-1)^{x_{n+t} + x_n} \end{aligned}$$

(analog wie in Kapitel II für BOOLEsche Funktionen). Wird nun x von einem Schieberegister der Länge l erzeugt,

$$x_n = a_1 x_{n-1} + \cdots + a_l x_{n-l} \quad \text{für } n \geq l,$$

so kann man die Differenzenfolge $y_n = x_{n+t} - x_n$ bilden. Diese wird offensichtlich von dem gleichen Schieberegister erzeugt. Sind die Startwerte $y_0, \ldots, y_{l-1}$ sämtlich 0, so ist der Zustandsvektor $x_{(t)} = x_{(0)}$, also t ein Vielfaches der Periode und $\kappa_x(t) = 1$. Andernfalls – und falls x die maximal mögliche Periode $s = 2^l - 1$ hat – durchläuft y in einer Periode nach Bemerkung 1 genau 2^{l-1} Einsen und $2^{l-1} - 1$ Nullen. Daher ist

$$\kappa_x(t) = \begin{cases} 1, & \text{wenn } s|t, \\ -\frac{1}{s}, & \text{sonst.} \end{cases}$$

Die Autokorrelation ist also – außer bei Verschiebungen um Vielfache der Periode – *gleichmäßig klein.*

Diese Aussagen bedeuten, dass die Folge sehr gleichmäßig verteilt ist, und wurden von GOLOMB als die drei Zufälligkeits-Postulate bezeichnet. Wegen dieser Eigenschaften werden solche Folgen, also insbesondere Schieberegisterfolgen maximaler Periode, in der Elektrotechnik auch als „Rauschen“ bezeichnet (PN-sequences = Pseudo Noise Sequences).

Hier eine Implementation von Schieberegistern in der leicht verständlichen Sprache von Mathematica – für eine Anwendung mit hohem Effizienzbedarf würde man natürlich eine Implementation in C vorziehen.

```
linShRep[n_Integer] :=
  Module[{y, outlist = {}},
    For[i = 0, i < n, i++,
      outlist = Append[outlist, Last[x]];
      y = Mod[a.x, 2];
      x = RotateRight[x];
      x[[1]] = y
      ];
    Return[outlist]
    ]
linShRep::usage =
"Generate a linear feedback shift register sequence.\n
1. Set up the coefficient array a consisting of 0s and 1s.\n
2. Set up the initial state of the shift register as an array
   x of the same length consisting of 0s and 1s.\n
3. Call linShRep with the desired number n of output Bits."
```

Ein exemplarischer Aufruf dieser Funktion mit einem Schieberegister der Länge 16, aus dem 1024 Bits erzeugt werden sollen, sieht so aus:

```
a = {0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1}
x = {0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1}
z = linShRep[1024]
```

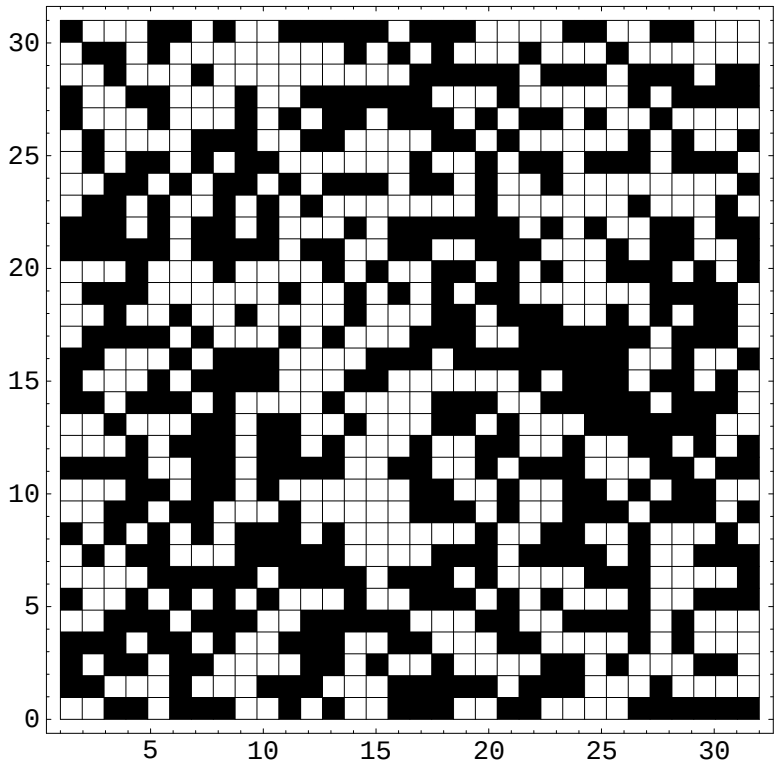
und ergibt den Output (ohne Klammern und Kommata wiedergegeben):

```
11001000110101100011001111000000
00111011100011100000100011101111
01001001111001011011110010111001
00010010110001100111001111010111
11000100011000001110011000010111
01101010101110110001010111011000
11110000010000100010111100011110
10100111000001111000100001011000
01010101000101111110110011011101
11001001110111110001011000100010
11100100101111110011011001010011
00001100100001100110100011100100
11101000100101110110011011001010
11011100100110111001011100000011
00100010111101111000110000010001
01110100001110011111101000100101
00111010001111000100000000110110
10000101110101110001100000010001
11011011011110111001000110101001
10001111110110101010011111100001
11101110111101011001010110001010
00000100001001100110001110100110
00010100101110100000010101100100
10010110101011111110111111011101
11001010010100010010110111111110
10100101001111110110100100010001
10111100011001111001011111010110
01110111010100100010100101101111
01100111011000000111011111010000
11011101111111110000010001000100
10010111111110101011101110111111
01110010110000010001111001100111
```

Eine Visualiserung, die mit dem Mathematica-Kommando

```
DensityPlot[z[[32*i + j]], {j, 1, 32}, {i, 0, 31},
   PlotPoints -> 32]
```

erzeugt wurde, zeigt, dass zumindest der äußere Eindruck der einer ziemlich zufälligen Bitfolge ist:



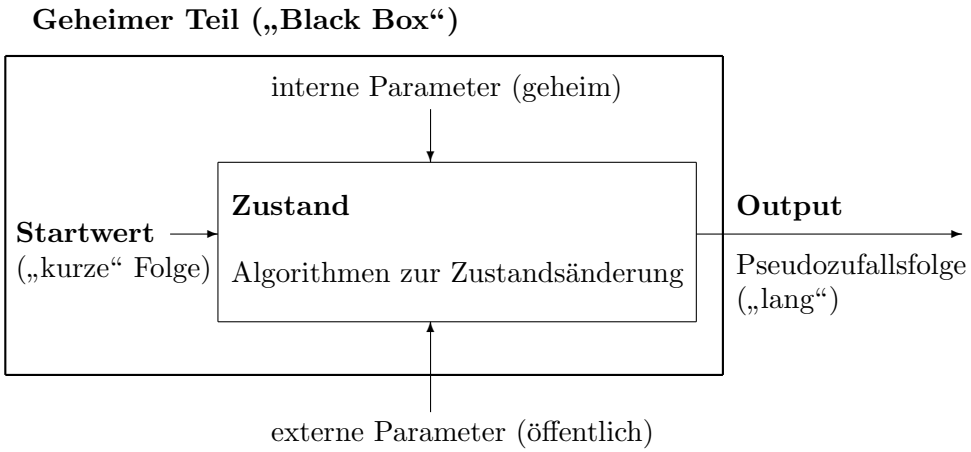
Das im Beispiel verwendete Schieberegister erzeugt übrigens eine Folge der maximalen Periode $2^{16} - 1 = 65535$, da sein charakteristisches Polynom

$$T^{16} + T^{14} + T^{13} + T^{11} + 1 \in \mathbb{F}_2$$

primitiv ist.

2 Kryptoanalyse von Zufallsgeneratoren

Schematisch sieht die Funktionsweise eines Zufallsgenerators so aus:



„Kryptoanalyse“ bedeutet für Zufallsgeneratoren, dass aus einem Teil ihres Outputs eine der folgenden Informationen bestimmt werden kann:

- die geheimen Parameter,
- der Startwert,
- weitere Teile des Outputs („Vorhersageproblem“).

2.1 Der allgemeine lineare Generator

Erinnern wir uns an die Beschreibung des allgemeinen linearen Generators: Gegeben sind

- als externe Parameter ein Ring R und ein R-Modul M,
- als interner Parameter eine lineare Abbildung $A\colon M \longrightarrow M$,
- als Zustand der Vektor $x_n \in M$,
- als Startwert der Vektor $x_0 \in M$,
- als Zustandsänderung die Rekursion $x_n = Ax_{n-1}$ für $n \geq 1$.

Bemerkung (Trivialfall): Falls A bekannt ist, ist aus jedem Folgeglied x_k die weitere Folge $(x_n)_{n\geq k}$ komplett vorhersagbar. Dieser Fall ist also kryptologisch völlig uninteressant. Die Rückwärtsberechnung von x_n mit $0 \leq n < k$ ist allerdings im allgemeinen nur möglich, wenn A injektiv ist. Das reicht natürlich nicht, um kryptologische Brauchbarkeit zu erreichen. Daher wird im folgenden meist nur das Problem der Vorwärtsberechnung behandelt und angenommen, dass ein Anfangsstück der Folge $x_0, \ldots, x_k$ bekannt ist. Trotzdem sollte man das Problem der Rückwärtsberechnung auch immer im Auge behalten.

Annahme also jetzt: R und M sind bekannt, A ist unbekannt, ein Anfangsstück $x_0, \ldots, x_k$ ist bekannt (o. B. d. A. $x_0 \neq 0$). Das *Vorhersageproblem* ist: Kann man daraus $x_{k+1}, x_{k+2}, \ldots$ bestimmen?

Man kann, wenn es einem gelingt, eine Linearkombination

$$x_k = c_1 x_{k-1} + \cdots + c_k x_0$$

zu bestimmen – also mit bekannten Koeffizienten $c_1, \ldots, c_k$. Dann ist nämlich

$$\begin{aligned} x_{k+1} &= Ax_k = c_1 Ax_{k-1} + \cdots + c_k Ax_0 \\ &= c_1 x_k + \cdots + c_k x_1 \\ &\vdots \\ x_n &= c_1 x_{n-1} + \cdots + c_k x_{n-k} \quad \text{für alle } n \geq k, \end{aligned}$$

die weitere Folge also komplett bestimmt – ohne dass man A kennt(!). Wie findet man eine solche Linearkombination?

Die Antwort liegt – natürlich – in der linearen Algebra. Im gegenwärtigen abstrakten Rahmen setzt man voraus, dass M noethersch ist (das ist die „richtige“ Verallgemeinerung von endlich-dimensionalen Vektorräumen); dann ist die aufsteigende Folge von Untermoduln

$$Rx_0 \subseteq Rx_0 + Rx_1 \subseteq \ldots \subseteq M$$

stationär, d. h., es gibt ein k mit $x_k \in Rx_0 + \cdots + Rx_{k-1}$: das ist die gesuchte lineare Relation. – Falls M endlich ist, wie wir es bei der Zufallserzeugung ja meist einrichten, ist die noethersche Eigenschaft selbstverständlich trivial. – Das erste solche k reicht, alle übrigen x_n, $n \geq k$, liegen dann auch in diesem Untermodul.

Wir haben also gezeigt:

Satz 1 (Noethersches Prinzip für lineare Generatoren) *Sei R ein Ring, M ein noetherscher R-Modul, $A\colon M \longrightarrow M$ linear und $(x_n)_{n\in\mathbb{N}}$ eine Folge in M mit $x_n = Ax_{n-1}$ für $n \geq 1$. Dann gibt es ein $k \geq 1$ und $c_1, \ldots, c_k \in R$ mit*

$$x_n = c_1 x_{n-1} + \cdots + c_k x_{n-k} \quad \textit{für alle } n \geq k.$$

Jedes k mit $x_k \in Rx_0 + \cdots + Rx_{k-1}$ ist geeignet.

Wie bestimmt man aber den Index k und die Koeffizienten $c_1, \ldots, c_k$ praktisch? Dazu muss man natürlich in R und M rechnen können. Wir betrachten im folgenden zwei Beispiele: $R = K$ ein Körper oder $R = \mathbb{Z}/m\mathbb{Z}$ ein Restklassenring von ganzen Zahlen.

In beiden Fällen kann man von vornherein etwas darüber sagen, wie oft eine echte Zunahme in der Kette der Untermoduln vorkommen kann. Ist z. B. R ein Körper, so ist die Anzahl der echten Schritte durch die Vektorraum-Dimension $\operatorname{Dim} M$ beschränkt. Allgemein gilt:

Satz 2 (Krawczyk) *Sei M ein R-Modul und $0 \subset M_1 \subset \ldots \subset M_l \subseteq M$ eine echt aufsteigende Kette von Untermoduln. Dann ist $2^l \leq \#M$.*

Dieser Satz ist natürlich nur dann nützlich, wenn M endlich ist. Aber das ist ja derjenige Fall, der für die Vorhersage von Kongruenzgeneratoren am meisten interessiert. Man kann dann auch $l \leq {}^2\log(\#M)$ schreiben. Das ist nicht so viel schlechter als die Abschätzung im Fall Körper/Vektorraum, beides endlich: $l \leq \operatorname{Dim}(M) \leq {}^2\log(\#M)/\,{}^2\log(\#R)$.

Beweis. Sei $b_i \in M_i - M_{i-1}$ für $i = 1, \ldots, l$ (mit $M_0 = 0$). Dann besteht die Menge

$$U = \{c_1 b_1 + \cdots + c_l b_l \mid \text{alle } c_i = 0 \text{ oder } 1\} \subseteq M$$

aus 2^l verschiedenen Elementen. Wären nämlich zwei davon gleich, so wäre ihre Differenz (für ein t mit $1 \leq t \leq l$) von der Form

$$e_1 b_1 + \cdots + e_t b_t = 0 \text{ mit } e_i \in \{0, \pm 1\}, e_t \neq 0.$$

Da $e_t = \pm 1 \in R^\times$, folgte $b_t = -e_t^{-1}(e_1 b_1 + \cdots + e_{t-1} b_{t-1}) \in M_{t-1}$, Widerspruch. Also ist $\#M \geq \#U = 2^l$. $\diamond$

2.2 Lineare Generatoren über Körpern

Hier wird der Spezialfall betrachtet, dass $R = K$ ein Körper und M ein endlich-dimensionaler K-Vektorraum ist (also ein noetherscher K-Modul).

Dann muss man nur das minimale k finden mit

$$\mathrm{Dim}(Kx_0 + \cdots + Kx_k) = \mathrm{Dim}(Kx_0 + \cdots + Kx_{k-1})$$

– diese Zahl ist dann notwendig $= k$ – und dann die Linearkombination

$$x_k = c_1 x_{k-1} + \cdots + c_k x_0.$$

Das ist eine Standard-Aufgabe der linearen Algebra.

Zur konkreten Berechnung wählt man eine feste Basis $(e_1, \ldots, e_r)$ von M. Sei

$$x_n = \sum_{i=1}^{r} x_{in} e_i$$

die jeweilige Basis-Darstellung. Da $\mathrm{Rang}(x_0, \ldots, x_{k-1}) = k$, gibt es eine Indexmenge $I = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, r\}$ mit $\#I = k$, so dass die Matrix

$$X = (x_{ij})_{i \in I, 0 \le j < k} = \begin{pmatrix} x_{i_1 0} & \ldots & x_{i_1 k-1} \\ \vdots & & \vdots \\ x_{i_k 0} & \ldots & x_{i_k k-1} \end{pmatrix}$$

invertierbar ist. Die bisher noch unbekannten c_j gewinnt man aus dem Ansatz

$$x_k = \sum_{j=0}^{k-1} c_j x_j,$$

also

$$\sum_{i=1}^{r} x_{ik} e_i = \sum_{j=0}^{k-1} \sum_{i=1}^{r} c_j x_{ij} e_i,$$

also

$$x_{ik} = \sum_{j=0}^{k-1} x_{ij} c_j \quad \text{für alle } i \in I,$$

oder in Matrixschreibweise:

$$\bar{x} = (x_{ik})_{i \in I} = X \cdot c.$$

Die Lösung ist

$$c = X^{-1} \cdot \bar{x}.$$

Damit sind auch schon die ersten beiden Aussagen des folgenden Zusatzes zu Satz 1 bewiesen:

Satz 3 *Zusätzlich zu den Voraussetzungen von Satz 1 sei $R = K$ ein Körper. Dann gilt:*

(i) *Das minimale geeignete k ist das kleinste mit* $\mathrm{Dim}(Kx_0+\cdots+Kx_k) = k$*; es ist* $k \leq \mathrm{Dim}\, M =: r$.

(ii) *Die Koeffizienten $c_1, \ldots, c_k$ werden durch Lösung eines linearen Gleichungsystems mit invertierbarer quadratischer Koeffizientenmatrix bestimmt, deren Einträge aus Basiskoeffizienten von $x_0, \ldots, x_{k-1}$ bestehen.*

(iii) *Ist $k = r$, so ist A eindeutig aus den Basiskoeffizienten von $x_0, \ldots, x_k$ bestimmbar.*

Beweis. (iii) Seien

$$X_1 = (x_r, \ldots, x_1),\ X_0 = (x_{r-1}, \ldots, x_0) \in M_r(K).$$

Dann ist $X_1 = AX_0$ in Matrix-Darstellung bezüglich der Basis $(e_1, \ldots, e_r)$ von M. Da $\mathrm{Rang}\, X_0 = r$, ist X_0 invertierbar und

$$A = X_1 X_0^{-1},$$

wie behauptet. $\diamond$

Ist A invertierbar, so kann man analog die Folge (x_n) auch rückwärts berechnen, sobald man ein Stück $x_t, \ldots, x_{t+r}$ der Länge $r + 1$ mit $\mathrm{Rang}(x_t, \ldots, x_{t+r-1}) = r$ gefunden hat.

Beispiel.

Im Fall eines r-stufigen homogenen linearen Kongruenzgenerators $x_n = a_1 x_{n-1} + \cdots a_r x_{n-r}$ über $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ mit p prim ist

$$A = \begin{pmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & 0 & 1 \\ a_r & \ldots & a_2 & a_1 \end{pmatrix}, \quad \mathrm{Det}\, A = (-1)^r a_r.$$

Hier ist A also genau dann invertierbar, wenn $a_r \neq 0$, und das kann man o. B. d. A. annehmen – sonst ist die Rekursionstiefe $< r$.

Für die Vorhersage der Folge benötigt man dann höchstens $r + 1$ Zustandsvektoren, also $2r$ Folgenglieder:

Korollar 1 *Ein r-stufiger homogener linearer Kongruenzgenerator mit Primzahlmodul ist aus den $2r$ Folgegliedern $x_0, \ldots, x_{2r-1}$ vorhersagbar.*

Korollar 2 *Ein lineares Schieberegister der Länge l ist aus den ersten $2l$ Bits vorhersagbar.*

Korollar 3 *Ein homogener linearer Kongruenzgenerator mit Primzahlmodul ist aus x_0, x_1, ein inhomogener aus x_0, x_1, x_2, x_3 vorhersagbar.*

Im nächsten Abschnitt wird u. a. gezeigt, dass bereits x_0, x_1, x_2 genügen.

Damit sind lineare Schieberegister als Quelle von Schlüsselbits für eine Bitstrom-Chiffre ein- für allemal kryptologisch erledigt. – Sollte die Länge zusätzlich geheim sein, kann der Kryptoanalytiker sie durch sukzessives Probieren bestimmen; das erhöht die Schwierigkeit des Angriffs nur unwesentlich.

Bei linearen Kongruenzgeneratoren könnte allerdings noch der Fall interessant sein, dass der Modul m geheimgehalten wird (und eventuell nicht prim ist). Dieser Fall wird im folgenden ebenfalls erledigt.

2.3 Lineare Kongruenzgeneratoren mit bekanntem Modul

Die Behandlung hier ist elementar ohne Benutzung der allgemeinen Theorie der vorhergehenden Abschnitte.

Die Parameter a und b des linearen Kongruenzgenerators $x_n = ax_{n-1} + b \bmod m$ seien unbekannt, bekannt hingegen sei zunächst der Modul m.

Für die Vorhersage reichen, auch wenn m nicht prim ist, 3 aufeinanderfolgende Folgenglieder x_0, x_1, x_2, wie im folgenden gezeigt wird. Aus der Relation

$$x_2 - x_1 \equiv a(x_1 - x_0) \pmod{m}$$

erhält man sofort (falls $x_1 - x_0$ zu m teilerfremd ist – das wird zunächst angenommen)

$$a = \frac{x_2 - x_1}{x_1 - x_0} \bmod m,$$

wobei die Division $\bmod m$ vorzunehmen ist (mit dem erweiterten Euklidischen Algorithmus). Das Inkrement b ergibt sich aus

$$b = x_1 - ax_0 \bmod m.$$

Damit ist das Bildungsgesetz bekannt und die Folge total vorhersagbar.

Typisch war schon in diesem einfachen Fall die Verwendung der **Differenzenfolge**

$$y_i = x_i - x_{i-1} \quad \text{für } i \geq 1.$$

Sie gehorcht dem Bildungsgesetz

$$y_{i+1} \equiv ay_i \pmod{m}.$$

Zu beachten ist, dass die y_i auch negativ sein können; sie liegen im Bereich $-m < y_i < m$. Da m bekannt ist, könnte man sie durch $y_i \bmod m$ ersetzen, aber das spielte, wie gesehen, keine Rolle, und bei unbekanntem m – später – geht es sowieso nicht.

Hilfssatz 1 (von der Differenzenfolge) *Die Folge (x_i) sei von dem linearen Kongruenzgenerator mit Modul m, Multiplikator a und Inkrement b erzeugt. Sei (y_i) ihre Differenzenfolge, $c = \mathrm{ggT}(m, a)$ und $d = \mathrm{ggT}(m, y_1)$. Dann gilt:*

(i) *Folgende Aussagen sind äquivalent:*
  (a) *Die Folge (x_i) ist konstant.*
  (b) $y_1 = 0$.
  (c) *Für alle i ist $y_i = 0$.*

(ii) $\mathrm{ggT}(m, y_i) | \mathrm{ggT}(m, y_{i+1})$ *für alle i.*

(iii) $d | y_i$ *für alle i.*

(iv) *Ist* $\mathrm{ggT}(y_1, \ldots, y_t) = 1$ *für ein $t \geq 1$, so $d = 1$.*

(v) $c | y_i$ *für alle $i \geq 2$.*

(vi) *Ist* $\mathrm{ggT}(y_2, \ldots, y_t) = 1$ *für ein $t \geq 2$, so $c = 1$.*

(vii) $m|y_iy_{i+2} - y_{i+1}^2$ *für alle* i.
(viii) *Sind* $\tilde{a}, \tilde{m}$ *ganze Zahlen,* $\tilde{m} \geq 1$, *mit* $y_i \equiv \tilde{a}y_{i-1} \pmod{\tilde{m}}$ *für* $i = 2, \ldots, r$, *so gilt* $x_i = \tilde{a}x_{i-1} + \tilde{b} \bmod \tilde{m}$ *für alle* $i = 1, \ldots, r$ *mit* $\tilde{b} = x_1 - \tilde{a}x_0 \bmod \tilde{m}$.

Beweis. (i) Es ist nur zu bemerken, dass mit einem y_i auch alle folgenden 0 sind.

(ii) Ist e Teiler von y_i und m, so wegen $y_{i+1} = ay_i + k_im$ auch Teiler von y_{i+1}.

(iii) ist ein Spezialfall von (ii).

(iv) gilt, weil $d|\operatorname{ggT}(y_1, \ldots, y_t)$ nach (iii).

(v) Sei $m = c\tilde{m}$ und $a = c\tilde{a}$. Dann ist $y_{i+1} = c\tilde{a}y_i + k_ic\tilde{m}$, also $c|y_{i+1}$ für $i \geq 1$.

(vi) gilt, weil $c|\operatorname{ggT}(y_2, \ldots, y_t)$ nach (v).

(vii) $y_iy_{i+2} - y_{i+1}^2 \equiv a^2y_i - a^2y_i \pmod{m}$.

(viii) durch Induktion: Für $i = 1$ ist die Behauptung die Definition von $\tilde{b}$. Für $i \geq 2$ folgt

$$x_i - \tilde{a}x_{i-1} - \tilde{b} \equiv x_i - \tilde{a}x_{i-1} - x_{i-1} + \tilde{a}x_{i-2} \equiv y_i - \tilde{a}y_{i-1} \equiv 0 \pmod{\tilde{m}},$$

wie behauptet. $\diamond$

Der triviale Fall der konstanten Folge braucht nicht weiter untersucht zu werden. Man erkennt an ihm aber, dass die Parameter eines linearen Kongruenzgenerators oft nicht eindeutig durch die erzeugte Folge bestimmt sind. Zum Beispiel kann man die konstante Folge mit einem beliebigen Modul m und einem beliebigen Multiplikator a erzeugen, wenn man nur das Inkrement $b = -(a-1)x_0 \bmod m$ setzt. Auch bei gegebenem m ist a dabei noch nicht eindeutig festgelegt, nicht einmal $a \bmod m$.

Im oben behandelten Fall war y_1 zu m teilerfremd und somit $a = y_2/y_1 \bmod m$. Im allgemeinen kann es allerdings passieren, dass die Division $\bmod m$ gar nicht eindeutig ist; genau dann trifft das zu, wenn m und y_1 nicht teilerfremd sind, also $d = \operatorname{ggT}(m, y_1) > 1$ ist. Die **reduzierte Differenzenfolge** $\bar{y}_i = y_i/d$ (vgl. (iii) in Hilfssatz 1) folgt dann der Rekursionsformel

$$\bar{y}_{i+1} \equiv \bar{a}\bar{y}_i \pmod{\bar{m}}$$

mit dem reduzierten Modul $\bar{m} = m/d$ und reduzierten Multiplikator $\bar{a} = a \bmod \bar{m}$, aus der sich $\bar{a} = \bar{y}_2/\bar{y}_1$ eindeutig bestimmen lässt. Setzt man $\tilde{a} = \bar{a} + k\bar{m}$ mit einer beliebigen ganzen Zahl k und $\tilde{b} = x_1 - \tilde{a}x_0 \bmod m$, so folgt nach Hilfssatz 1 (viii) auch $x_i = \tilde{a}x_{i-1} + \tilde{b} \bmod m$ für alle $i \geq 1$. Damit ist der folgende Satz gezeigt:

Satz 4 *Die Folge* (x_i) *sei von einem linearen Kongruenzgenerator mit bekanntem Modul* m, *aber unbekanntem Multiplikator* a *und Inkrement* b *erzeugt. Dann ist die gesamte Folge aus* x_0, x_1 *und* x_2 *bestimmbar. Falls die*

Folge (x_i) nicht konstant ist, ist der Multiplikator a genau bis auf ein Vielfaches des reduzierten Moduls $\bar{m}$ bestimmt.

Man muss sich also auch hier unter Umständen damit begnügen, die Folge vorherzusagen, ohne letzte Gewissheit über die wirklich verwendeten Parameter erlangen zu können. Wer ein ganz einfaches Zahlenbeispiel möchte: Für $m = 24$, $a = 2k + 1$ mit $k \in [0 \ldots 11]$ und $b = 12 - 2k \bmod 24$ wird aus dem Startwert $x_0 = 1$ stets die Folge $(1, 13, 1, 13, \ldots)$ erzeugt.

2.4 Lineare Kongruenzgeneratoren mit unbekanntem Modul

Schwieriger wird es natürlich, wenn der Modul m ebenfalls unbekannt ist und sich auch nicht leicht erraten lässt. Es wird angenommen, dass man nur ein Stück $x_0, x_1, \ldots$ der Folge zur Verfügung hat. Überraschenderweise ist es leichter, zuerst den Multiplikator zu bestimmen. Aus dem folgenden Satz erhält man in wenigen Schritten einen geeigneten Wert dafür, der dann später bei der Suche nach dem Modul hilft. Man erkennt den noetherschen Ansatz in der Form $y_{t+1} \in \mathbb{Z}y_1 + \cdots + \mathbb{Z}y_t$ wieder.

Satz 5 (Plumstead) *Sei (y_i) die Differenzenfolge des linearen Kongruenzgenerators mit erzeugender Funktion $s(x) = ax + b \bmod m$, $m \geq 2$, und Startwert x_0. Sei $y_1 \neq 0$ und t die kleinste Zahl mit $e = \mathrm{ggT}(y_1, \ldots, y_t) \,|\, y_{t+1}$. Dann gilt:*

(i) $t < 1 + {}^2\log m$.

(ii) *Ist $e = c_1y_1 + \cdots + c_ty_t$ mit $c_i \in \mathbb{Z}$ und $a' = (c_1y_2 + \cdots + c_ty_{t+1})/e$, so ($a' \in \mathbb{Z}$ und)*

$$y_{i+1} \equiv a'y_i \pmod{m} \quad \textit{für alle } i.$$

(iii) *Mit $b' = x_1 - a'x_0$ gilt*

$$x_i = a'x_{i-1} + b' \bmod m \quad \textit{für alle } i.$$

Beweis. (i) Ist $e_j = \mathrm{ggT}(y_1, \ldots, y_j)$ kein Teiler von y_{j+1}, so $e_{j+1} \leq e_j/2$. Da $e_1 = |y_1| < m$, folgt $e = e_t < m/2^{t-1}$, also $t - 1 < {}^2\log m$.

(ii) Es ist

$$ae = c_1ay_1 + \cdots + c_tay_t \equiv c_1y_2 + \cdots + c_ty_{t+1} = a'e \pmod{m}.$$

Der größte gemeinsame Teiler d von m und y_1 teilt e nach Hilfssatz 1, also ist auch $d = \mathrm{ggT}(m, e)$. Die Kongruenz wird zuerst durch d geteilt:

$$a\frac{e}{d} \equiv a'\frac{e}{d} \pmod{\bar{m}}$$

mit dem reduzierten Modul $\bar{m} = m/d$. Da e/d zu $\bar{m}$ teilerfremd ist, kann man es wegdividieren:

$$a \equiv a' \pmod{\bar{m}}, \;\; a = a' + k\bar{m}.$$

Also ist $y_{i+1} \equiv ay_i = a'y_i + ky_i\bar{m} \pmod{m}$. Da $d \,|\, y_i$, folgt $y_i \equiv 0$, also $y_{i+1} \equiv a'y_i \pmod{m}$.

(iii) folgt aus Hilfssatz 1 (viii). $\diamond$

Bemerkungen und Beispiele

1. Sei $m = 8397$, $a = 4381$ und $b = 7364$ [Reeds 1977]. Damit wird erzeugt

$$\begin{array}{lll} x_0 = 2134 & & \\ x_1 = 2160 & y_1 = 26 & e_1 = 26 \\ x_2 = 6905 & y_2 = 4745 & e_2 = 13 \\ x_3 = 3778 & y_3 = -3127 & e_3 = 1 \\ x_4 = 8295 & y_4 = 4517 & \end{array}$$

   Es folgt $c_1 = 87542$, $c_2 = -481$, $c_3 = -1$ und $a' = 416881843$.

2. Sei $m = 2^q + 1$, $a = 2^{q-1}$, $b = 2^q$ und $x_0 = 0$. Nach dem Korollar zum folgenden Hilfssatz 2 ist $y_i = (-1)^{i-1} \cdot 2^{q-i+1}$ für $i = 1, \ldots, q+1$ und daher $e_i = 2^{q-i+1}$. Damit ist $t = q+1$. Die Abschätzung für t im Satz 5 ist also scharf, und man braucht tatsächlich $q+3$ der Folgeglieder x_i, also x_0 bis x_{q+2}, um a' zu ermitteln.

Hilfssatz 2 *Die Folge (c_i) in $\mathbb{Z}$ sei durch $c_0 = 0$, $c_i = 2^{i-1} - c_{i-1}$ für $i \geq 1$, definiert. Dann ist*
(i) $c_i = \frac{1}{3} \cdot [2^i - (-1)^i]$ *für alle i,*
(ii) $c_i - 2c_{i-1} = (-1)^{i-1}$ *für alle $i \geq 1$.*

Beweis. (i) zeigt man durch Induktion und (ii) durch direkte Rechnung. ◇

Korollar 1 *Die Folge (x_i) sei von dem linearen Kongruenzgenerator mit Modul $m = 2^q + 1$, Multiplikator $a = 2^{q-1}$, Inkrement $b = 2^q$ und Startwert $x_0 = 0$ erzeugt; (y_i) sei ihre Differenzenfolge. Dann gilt:*
(i) $x_i = c_i \cdot 2^{q-i+1}$ *für $i = 0, \ldots, q+1$,*
(ii) $y_i = (-1)^{i-1} \cdot 2^{q-i+1}$ *für $i = 1, \ldots, q+1$.*

Ein „Ersatzmultiplikator“ a' läßt sich mit Hilfe von Satz 5 also effizient ermitteln. Nun fehlt noch ein Verfahren zur Ermittlung des Moduls m. Dieser wird durch „sukzessive Korrektur“ eingekesselt; im j-ten Schritt wird ein „Ersatzmodul“ m_j und ein „Ersatzmultiplikator“ a_j bestimmt:

- Im ersten Schritt setzt man $m_1 = \infty$ und $a_1 = a'$. [Rechnen mod ∞ soll einfach Rechnen mit ganzen Zahlen bedeuten, und $\mathrm{ggT}(c, \infty)$ soll c sein, wenn $c \neq 0$, und ∞, wenn $c = 0$.]

- Im j-ten Schritt, $j \geq 2$, sei $y'_j := a_{j-1} y_{j-1} \bmod m_{j-1}$. Dann setzt man $m_j = \mathrm{ggT}(m_{j-1}, y'_j - y_j)$ und $a_j = a_{j-1} \bmod m_j$.

Es wird also stets mit den aktuellen Ersatzwerten m_{j-1} und a_{j-1} für m und a eine Voraussage y'_j für y_j gemacht und diese mit dem tatsächlichen Wert y_j verglichen. Stimmen diese beiden Zahlen nicht überein, so unterscheiden sie sich um ein Vielfaches von m; dann werden die Ersatzwerte

korrigiert. Stets gilt $m \mid m_j$. Die j-te Korrektur ändert an der bisherigen Rechnung nichts, denn $y_i \equiv a_j y_{i-1} \pmod{m_j}$ für $i = 2, \ldots, j$, und auch $y_i \equiv a_j y_{i-1} \pmod{m}$ für alle $i \geq 2$. Auch die eigentliche Folge (x_i) erfüllt stets $x_i \equiv a_j x_{i-1} + b_j \pmod{m_j}$ für $i = 1, \ldots, j$ mit $b_j = x_1 - a_j x_0$ nach Hilfssatz 1 (viii).

Im oben gerechneten Beispiel 1 ist

$$\begin{array}{lll} & m_1 = \infty & a_1 = 416881843 \\ y_2' = 10838927918 & m_2 = 10838923173 & a_2 = 416881843 \\ y_3' = 5420327549 & m_3 = 8397 & a_3 = 4381 \end{array}$$

Der Wert m_3 errechnet sich als

$$\mathrm{ggT}(10838923173, 5420330676) = 8397.$$

Da $m_3 \leq 2x_2$, ist $m = m_3$, $a = a_3$ und $b = x_1 - ax_0 \bmod m = 7364$. Wir haben also die korrekten Werte mit zwei Korrekturen gefunden und dabei keine weiteren Folgenglieder gebraucht als die fünf, die schon zur Bestimmung von a' nötig waren. Auffallend sind die großen Zwischenergebnisse, so dass man mit der gewöhnlichen Ganzzahlarithmetik nicht mehr auskommt, sondern eine Arithmetik mit erweiterter Stellenzahl braucht.

Kommt das Verfahren stets zum Ziel? Spätestens wenn die Periode der Folge erreicht ist, also nach höchstens m Schritten, ist die gesamte Folge korrekt voraussagbar. Diese Schranke hat allerdings keinen praktischen Wert. Leider ist sie schon scharf: Bei beliebigem m sei $a = 1$, $b = 1$ und $x_0 = 0$. Dann ist $x_i = i$ und $y_i = 1$ für $i = 0, \ldots, m-1$. Der Startwert für den Ersatzmultiplikator ist $a' = 1$. Die erste falsche Voraussage ist $y_m' = 1$ statt des korrekten Werts $y_m = 1 - m$. Erst nach Auswertung von x_m ist das Verfahren beendet. Nun ist dieser schlechteste Fall leicht erkennbar und separat zu behandeln. Er erschwert aber das Auffinden guter allgemeingültiger Ergebnisse, und in der Tat sind keine solchen bekannt.

Ein etwas anderer Gesichtspunkt ergibt sich, wenn man die Anzahl der notwendigen Korrekturschritte zählt, also die Schritte, in denen der Ersatzmodul sich ändert. Ist nämlich $m_j \neq m_{j-1}$, so $m_j \leq m_{j-1}/2$. Sei $m^{(0)} = \infty > m^{(1)} > \ldots$ die Folge der *verschiedenen* Ersatzmoduln. Dann gilt

$$m^{(1)} = m_{j_1} = |y_{j_1}' - y_{j_1}| < a'|y_{j_1-1}| + m < m(a'+1),$$

$$m \leq m^{(j)} < \frac{m(a'+1)}{2^{j-1}},$$

also stets $j < 1 + {}^2\log(a'+1)$. Damit ist eine obere Schranke für die Anzahl der nötigen Korrekturen gefunden. Joan Plumstead-Boyar gab auch einen Algorithmus an, der zu einem eventuell kleineren Wert von a' und zu der oberen Schranke $2 + {}^2\log m$ für die Anzahl der Korrekturschritte führt. Allerdings wird diese Anzahl von Korrekturschritten in der Regel gar nicht erreicht, so dass die Schranke als Abbruchkriterium nichts nützt.

Hier scheint noch ein lohnendes Betätigungsfeld für die Suche nach theoretischen Ergebnissen offenzustehen. Lässt sich eine kleine Klasse von (vielleicht sowieso schlechten) linearen Kongruenzgeneratoren ausgrenzen, so dass für den großen Rest ein praktisch brauchbares Abbruchkriterium herleitbar ist? Das ist eigentlich zu erwarten. Lässt sich die Verteilung der nötigen Schrittzahl in den Griff bekommen? Wenigstens der Mittelwert? Jedenfalls reichen die vorliegenden Ergebnisse schon, um lineare Kongruenzgeneratoren endgültig als kryptologisch ungeeignet einzustufen.

2.5 Eine allgemeine Vorhersagemethode

Das Verfahren von PLUMSTEAD (die später unter dem Namen BOYAR publiziert hat) ist weitgehend durch das Verfahren von BOYAR/KRAWCZYK verallgemeinert worden: auf Rekursionsvorschriften, die sich durch eine Linearkombination *irgendwelcher* bekannten Funktionen ausdrücken lassen. Man beschreibt es zunächst wieder besonders passend in der Sprache der kommutativen Algebra, also durch Ringe und Moduln.

Sei also R ein kommutativer Ring (mit $1 \neq 0$), und X, Z seien R-Moduln. Gegeben sei eine Familie von Abbildungen

$$\Phi^{(i)} : X^i \longrightarrow Z \text{ für } i \geq h,$$

die wir uns als bekannt denken, und eine lineare Abbildung

$$\alpha : Z \longrightarrow X,$$

die als geheim angesehen wird (also als interner Parameter des zu beschreibenden Zufallsgenerators). Damit wird eine Folge $(x_n)_{n\in\mathbb{N}}$ in X erzeugt:

- $x_0, \ldots, x_{h-1} \in X$ werden als Startwerte gesetzt.
- Sind $x_0, \ldots, x_{n-1}$ schon erzeugt für $n \geq h$, so sei

$$\begin{aligned} z_n &:= \Phi^{(n)}(x_0, \ldots, x_{n-1}) \in Z, \\ x_n &:= \alpha(z_n) \in X. \end{aligned}$$

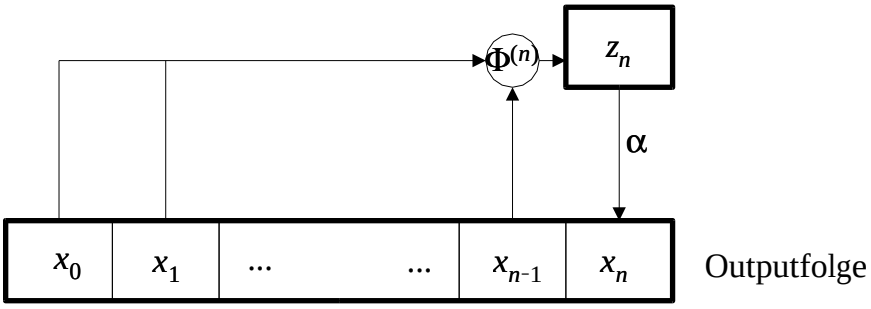


Abbildung 4: Ein allgemeiner Generator

Hier kann also, allgemeiner als bisher, jedes Folgenglied von *allen* vorhergehenden, also von der gesamten „Vergangenheit“ abhängen. Damit ein solches Verfahren sinnvoll zur Zufallserzeugung eingesetzt werden kann, müssen die $\Phi^{(i)}$ natürlich effizient berechenbar sein – im Fall $R = \mathbb{Z}/m\mathbb{Z}$ und $X = R^k$ etwa soll der Aufwand höchstens polynomial mit $\log(m)$ und k wachsen.

Beispiele

1. Der lineare Kongruenzgenerator: $R = \mathbb{Z}/m\mathbb{Z} = X$, $Z = R^2$, $h = 1$,

$$\Phi^{(i)}(x_0, \ldots, x_{i-1}) = \begin{pmatrix} x_{i-1} \\ 1 \end{pmatrix},$$

$$\alpha \begin{pmatrix} s \\ t \end{pmatrix} = as + bt.$$

2. Der linear-inversive Kongruenzgenerator: R, X, Z, h, α wie oben,

$$\Phi^{(i)}(x_0, \ldots, x_{i-1}) = \begin{pmatrix} x_{i-1}^{-1} \bmod m \\ 1 \end{pmatrix}.$$

3. Kongruenzgeneratoren höheren Grades: $R = \mathbb{Z}/m\mathbb{Z} = X$, $Z = R^{d+1}$, $h = 1$, $x_n = a_d x_{n-1}^d + \cdots + a_0$,

$$\Phi^{(i)}(x_0, \ldots, x_{i-1}) = \begin{pmatrix} x_{i-1}^d \\ \vdots \\ x_{i-1} \\ 1 \end{pmatrix},$$

$$\alpha \begin{pmatrix} t_0 \\ \vdots \\ t_d \end{pmatrix} = a_d t_0 + \cdots + a_0 t_d.$$

4. Beliebige Kongruenzgeneratoren: $R = \mathbb{Z}/m\mathbb{Z}$, $x_n = s(x_{n-1})$, $h = 1$. Ist m prim, so lässt sich jede Funktion $s : R \longrightarrow R$ als Polynom vom Grad $< m$ schreiben. Ist m zusammengesetzt, so verwendet man eben statt der Basis aus den Monomen die Basis $\{e_0, \ldots, e_{m-1}\}$ mit $e_i(j) = \delta_{ij}$ von R^R. Die Basis-Darstellung ist $s = \sum_{i=0}^{m-1} s(i)e_i$. Man nimmt $X = R$, $Z = R^m$ und

$$\Phi^{(i)}(x_0, \ldots, x_{i-1}) = \begin{pmatrix} e_0(x_{i-1}) \\ \vdots \\ e_{m-1}(x_{i-1}) \end{pmatrix},$$

$$\alpha \begin{pmatrix} t_0 \\ \vdots \\ t_{m-1} \end{pmatrix} = s(0)t_0 + \cdots + s(m-1)t_{m-1}.$$

Dass die $\Phi^{(i)}$ effizient berechenbar sein sollen, kann hier, egal welche Basis verwendet wird, nur bedeuten, dass eine Familie s_m von Funktionen auf $\mathbb{Z}/m\mathbb{Z}$ gegeben ist, die sich einheitlich als Linearkombination einer Teilmenge der Basis beschreiben lassen, die höchstens polynomial mit $\log(m)$ wächst.

5. Mehrstufige Kongruenzgeneratoren werden natürlich auch erfasst, wenn man h gleich der Rekursionstiefe nimmt.

6. Auch nichtlineare Schieberegister sind Beispiele, siehe den nächsten Abschnitt 2.6.

Für die Kryptoanalyse nimmt man wie gesagt an, dass die $\Phi^{(i)}$ bekannt sind, aber α unbekannt ist. (Später wird im Fall $R = \mathbb{Z}/m\mathbb{Z}$ auch noch m als unbekannt angenommen.) Die Frage ist: Kann man aus einem Anfangsstück $x_0, \ldots, x_{n-1}$ $(n \geq h)$ der Folge das nächste Glied x_n bestimmen?

Dazu betrachtet man die aufsteigende Kette $Z_h \subseteq Z_{h+1} \subseteq \ldots \subseteq Z$ von Untermoduln mit

$$Z_n = Rz_h + \cdots + Rz_n.$$

Falls $Z_n = Z_{n-1}$, ist $z_n = t_h z_h + \cdots + t_{n-1} z_{n-1}$ mit $t_h, \ldots, t_{n-1} \in R$ und daher

$$x_n = t_h x_h + \cdots + t_{n-1} x_{n-1}$$

bestimmbar ohne Verwendung von α. Ist Z ein noetherscher R-Modul, so wird nach endlich vielen Schritten der stationäre Zustand erreicht: $Z_n = Z_l$ für $n \geq l$. Ab dieser Stelle ist die Folge der x_n komplett vorhersagbar nach folgendem „Algorithmus“:

- Bilde $z_n = \Phi^{(n)}(x_0, \ldots, x_{n-1})$.
- Finde eine Linearkombination $z_n = t_h z_h + \cdots + t_{n-1} z_{n-1}$.
- Setze $x_n = t_h x_h + \cdots + t_{n-1} x_{n-1}$.

Damit aus dem „Algorithmus“ ein Algorithmus wird, muss das Verfahren im zweiten Schritt zum Finden einer Linearkombination algorithmisch durchführbar sein.

In unserem Standard-Beispiel mit (bekanntem) Modul $m = 8397$, $x_0 = 2134$, $x_1 = 2160$, $x_2 = 6905$ ist

$$z_1 = \begin{pmatrix} 2134 \\ 1 \end{pmatrix}, z_2 = \begin{pmatrix} 2160 \\ 1 \end{pmatrix}, z_3 = \begin{pmatrix} 6905 \\ 1 \end{pmatrix}.$$

Der Versuch, z_3 als Linearkombination $t_1 z_1 + t_2 z_2$ zu schreiben, führt auf das Gleichungssystem (in $R = \mathbb{Z}/8397\mathbb{Z}$)

$$\begin{aligned} 2134t_1 + 2160t_2 &= 6905, \\ t_1 + t_2 &= 1. \end{aligned} \tag{1}$$

Durch Elimination kommt man auf $26t_1 = -4745 = 3652$. Das Inverse von 26 mod 8397 ist 323, und daraus ergibt sich $t_1 = 4016$, $t_2 = 4382$. Damit wird $x_3 = 3778$ korrekt vorhergesagt.

Auch der Rest der Folge wird so korrekt vorhergesagt, denn es ist schon $Z_2 = Z$: Da $z_2 - z_1 = \binom{26}{0}$, ist $e_1 = \binom{1}{0} \in Z_2$ und $e_2 = \binom{0}{1} = z_1 - 2134 \cdot e_1 \in Z_2$.

Das Beispiel liefert auch eine Teilantwort auf die Frage, wann die Kette der Z_n stationär wird: Spätestens bei $Z_l = Z$, wenn das überhaupt vorkommt. Im allgemeinen kann man das nicht erwarten. Es folgt im allgemeinen auch nicht notwendig aus $Z_l = Z_{l+1}$, dass die Kette bei Z_l schon stationär ist – sie könnte später wieder ansteigen. Eine Schranke dafür, wie oft ein echter Anstieg möglich ist, gibt Satz 2.

In einem Schleifendurchlauf des Vorhersage-Algorithmus sind zwei Ereignisse möglich:

- $z_n \notin Z_{n-1}$. Dann ist keine Vorhersage für x_n möglich, aber Z_{n-1} wird zu $Z_n = Z_{n-1} + Rz_n$ erweitert, und zwar echt.
- $z_n \in Z_{n-1}$. Dann wird x_n korrekt vorhergesagt.

Der Satz besagt, dass das erste dieser Ereignisse höchstens ${}^2\log(\#Z)$-mal vorkommen kann. Bei jedem dieser Vorkommnisse braucht man dann den Zugriff auf das Folgeglied x_n, um weiter zu kommen. Das befriedigt nicht ganz, entspricht bei genauem Hinsehen aber der Situation des Kryptoanalytikers, der beim Brechen einer Verschlüsselung mit einem vermuteten Schlüssel weiterarbeitet, bis sinnloser Text entsteht, dann die nächsten Zeichen zu erraten versucht, seinen vermuteten Schlüssel korrigiert und damit weiter entschlüsselt. Im übrigen kennen wir diese Situation ja schon aus dem vorigen Abschnitt. Bemerkenswert ist, dass der neue Algorithmus recht einfach ist, aber sich auch ganz auf die Vorhersage konzentriert und nicht versucht, die unbekannten Parameter zu bestimmen.

2.6 Nichtlineare Schieberegister

Als weiteres Beispiel für die allgemeine Vorhersagemethode werden hier beliebige, nicht notwendig lineare, Schieberegister behandelt. Ein solches wird durch Abbildung 5 beschrieben.

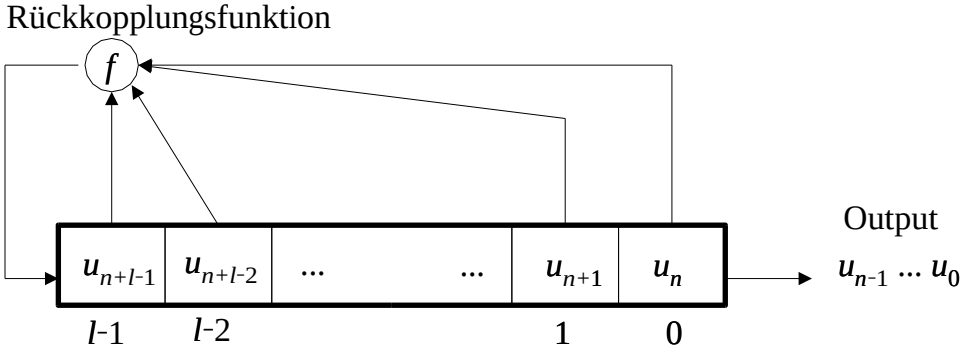


Abbildung 5: Ein Schieberegister der Länge l

Hierbei ist die Rückkopplungsfunktion $f : \mathbb{F}_2^l \longrightarrow \mathbb{F}_2$ eine beliebige Boolesche Funktion, die sich in algebraischer Normalform als Polynom

$$f(y_1, \ldots, y_l) = \sum_{I \subseteq \{1,\ldots,l\}} a_I y^I \qquad \text{mit } y^I = \prod_{j \in I} y_j$$

schreiben lässt.

Die Funktion f ist genau dann effizient (etwa durch ein Boolesches Schaltnetz) berechenbar, wenn „fast alle“ Koeffizienten $a_I = 0$ sind; d. h., es gibt ein Polynom $p \in \mathbb{N}[X]$ mit

$$\#\{I \mid a_I \neq 0\} \leq p(l).$$

Es ist dem Kryptoanalytiker allerdings nicht bekannt, welche $a_I \neq 0$ sind – vielmehr ist es eins seiner Ziele, das herauszubekommen.

Für die Anwendung der Vorhersagemethode wird $R = X = \mathbb{F}_2$, $h = l$, $Z = \mathbb{F}_2^{2^l}$ gesetzt. Für $i \geq l$ ist

$$\Phi^{(i)} : \mathbb{F}_2^i \longrightarrow Z$$

gegeben durch

$$z_i := \Phi^{(i)}(x_1, \ldots, x_i) = (y^I)_{I \subseteq \{1,\ldots,l\}} \qquad \text{mit } y = (x_{i-l+1}, \ldots, x_i).$$

Und schließlich ist

$$\alpha : Z \longrightarrow X, \quad \alpha((t^I)_{I \subseteq \{1,\ldots,l\}}) = \sum a_I t^I.$$

Zunächst zwei konkrete Beispiele für die Vorhersage:

Beispiele

1. $l = 2$, $f = T_1T_2 + T_1$. Aus den Startwerten $u_0 = 1$, $u_1 = 0$ wird die Folge
$$u_0 = 1,\ u_1 = 0,\ u_2 = 1,\ u_3 = 0,\ \ldots$$
erzeugt. Es ist
$$Z = \mathbb{F}_2^4, \qquad z_n = \begin{pmatrix} u_{n-1}u_{n-2} \\ u_{n-1} \\ u_{n-2} \\ 1 \end{pmatrix},$$
$$z_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}, \quad z_3 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}, \quad z_4 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = z_2, \quad \ldots$$
Also vermutet der Kryptoanalytiker die lineare Rekursion
$$z_n = z_{n-2} = 0 \cdot z_{n-1} + 1 \cdot z_{n-2} \quad \text{für } n \geq 4$$
und sagt korrekt voraus
$$u_n = 0 \cdot u_{n-1} + 1 \cdot u_{n-2} = u_{n-2} \quad \text{für } n \geq 4.$$
Die Folge kann also auch durch ein *lineares* Schieberegister der Länge 2 erzeugt werden. Benötigt wurden u_0 bis u_3.

2. $l = 3$, $f = T_1T_3 + T_2$. Aus den Startwerten $u_0 = 0$, $u_1 = 1$, $u_2 = 1$ wird die weitere Folge
$$u_3 = 1,\ u_4 = 0,\ u_5 = 1,\ u_6 = 1,\ u_7 = 1,\ u_8 = 0,\ u_9 = 1,\ \ldots$$
erzeugt. Es ist
$$Z = \mathbb{F}_2^8, \qquad z_n = \begin{pmatrix} u_{n-1}u_{n-2}u_{n-3} \\ u_{n-1}u_{n-2} \\ u_{n-1}u_{n-3} \\ u_{n-2}u_{n-3} \\ u_{n-1} \\ u_{n-2} \\ u_{n-3} \\ 1 \end{pmatrix},$$
$$z_3 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}, \quad z_4 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \quad z_5 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \quad z_6 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}, \quad z_7 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} = z_3, \quad \ldots$$

Also ist die vermutete lineare Rekursion hier

$$z_n = z_{n-4} \quad \text{für } n \geq 4,$$

$$u_n = u_{n-4} \quad \text{für } n \geq 4,$$

und auch das ist wieder korrekt. Benötigt wurden u_0 bis u_6; und gefunden wurde ein „äquivalentes“ lineares Schieberegister der Länge 4.

Wegen der exponentiellen Zunahme der Dimension von Z sieht es zunächst so aus, als ob das Vorhersageverfahren bald an seine Grenzen stößt; der stationäre Zustand der aufsteigenden Unterräume, d. h., die gesuchte lineare Relation, wird womöglich erst nach 2^l Schritten erreicht; die Bitfolge kann also auch durch eine lineares Schieberegister der Länge $\leq 2^l$ erzeugt werden, das durch das Verfahren von Boyar/Krawczyk explizit gefunden wird. Immerhin ist dabei noch ein Schieberegister der Länge 32 mit linearer Algebra im 2^{32}-dimensionalen binären Vektorraum mit realistischem Aufwand vorhersagbar.

Im allgemeinen Fall kommt aber ein anderer Gesichtspunkt zum Tragen: Die Rückkopplungsfunktion f hängt ja von 2^l Parametern ab. Um zu einem handhabbaren Schlüsselraum zu kommen, muss man die möglichen Koeffizienten $\neq 0$ *von vorneherein* auf eine polynomiale Anzahl beschränken. Diese Auswahl ist aber Teil des Algorithmus – etwa des in Hardware realisierten Schieberegisters – und nicht Bestandteil des Schlüssels, wird also nach dem Kerckhoffs-Prinzip früher oder später dem Gegner bekannt sein. Die Notwendigkeit, eine effizient berechenbare Rückkopplungsfunktion zu wählen, führt also dazu, dass die Vorhersagemethode ebenfalls effizient wird. Daher kann man sagen:

Satz 6 *Jede durch ein Schieberegister mit effizient berechenbarer Rückkopplungsfunktion erzeugte Bitfolge ist vorhersagbar.*

Schieberegister, ob linear oder nichtlinear, sind somit zur Erzeugung kryptographisch brauchbarer Zufallsfolgen nicht geeignet – jedenfalls nicht bei direkter Verwendung. Das bedeutet nicht, dass das Verfahren von Boyar/Krawczyk eine Erfolgsgarantie für den Kryptoanalytiker liefert; allerdings kann der Kryptograph sich auch mit nichtlinearen Schieberegistern nicht sicher fühlen.

2.7 Der allgemeine Kongruenzgenerator

Etwas komplizierter, aber nicht entmutigend, wird das Vorhersageverfahren für Kongruenzgeneratoren, bei denen auch der Modul unbekannt ist. Hier bringt die allgemeine Sprache der kommutativen Algebra nicht mehr viel, da sehr spezielle Eigenschaften der Ringe $\mathbb{Z}$ und $\mathbb{Z}/m\mathbb{Z}$ verwendet werden, insbesondere das „kanonische“ Repräsentantensystem $\{0, \ldots, m-1\} \subseteq \mathbb{Z}$ von $\mathbb{Z}/m\mathbb{Z}$.

Sei $X = \mathbb{Z}^r$, $\bar{X} = (\mathbb{Z}/m\mathbb{Z})^r$, $Z = \mathbb{Z}^k$, $\bar{Z} = (\mathbb{Z}/m\mathbb{Z})^k$. Gegeben seien die Abbildungen

$$\Phi^{(i)} : X^i \longrightarrow Z \text{ für } i \geq h,$$

$$\alpha : \bar{Z} \longrightarrow \bar{X} \text{ linear,}$$

wobei α und m für die Kryptoanalyse als unbekannt behandelt werden. Mit Hilfe des kanonischen Repräsentantensystems wird $\bar{X}$ als Teilmenge $\{0, \ldots, m-1\}^r$ von X aufgefasst. Dann funktioniert die Erzeugung der Folge wie gehabt, und wir nennen das Verfahren einen **allgemeinen Kongruenzgenerator**, wenn die Berechnung aller $\Phi^{(i)}$ effizient möglich ist, d. h., mit einem Aufwand, der polynomial von r, k und $\log(m)$ abhängt. Insbesondere gibt es eine Schranke M für die Werte der $\Phi^{(i)}$ auf $\{0, \ldots, m-1\}^i$, die polynomial in r, k und $\log(m)$ ist.

Die Kryptoanalyse wird in zwei Phasen unterteilt. In der ersten Phase wird über dem Ring $\mathbb{Z}$ bzw. seinem Quotientenkörper $\mathbb{Q}$ gearbeitet und ein Vielfaches $\hat{m}$ des Moduls m bestimmt. In der zweiten Phase arbeitet man über dem Ring $\mathbb{Z}/\hat{m}\mathbb{Z}$. Bei der Voraussage von x_n sind jetzt drei Ereignisse möglich:

- $z_n \notin Z_{n-1}$; der ($\mathbb{Q}$- oder $\mathbb{Z}/m\mathbb{Z}$-) Modul Z_{n-1} muss zu Z_n erweitert werden, für x_n ist keine Vorhersage möglich. (D. h., es muss ein weiteres Folgenglied anderswie beschafft werden; für die Kryptoanalyse bedeutet das: Man braucht weiteren bekannten Klartext.)

- x_n wird korrekt vorhergesagt.

- x_n wird falsch vorhergesagt. Dann wird der Modul $\hat{m}$ korrigiert.

In der ersten Phase ist Z_{n-1} der $\mathbb{Q}$-Vektorraum, der von $z_h, \ldots, z_{n-1}$ aufgespannt wird, wobei man natürlich redundante z_i einfach weglässt.

1. Fall: $z_n \notin Z_{n-1}$. Dann wird $Z_n = Z_{n-1} + \mathbb{Q}z_n$ gesetzt und x_n nicht vorhergesagt. Dieser Fall kann höchstens k-mal auftreten.

2. Fall: [Die lineare Voraussageformel] $z_n = t_h z_h + \cdots + t_{n-1} z_{n-1}$. Dann wird $x_n = t_h x_h + \cdots + t_{n-1} x_{n-1}$ vorhergesagt (als Element von $\mathbb{Q}^r$). (Es treten höchstens k der z_i in der konstruierten Basis von Z_{n-1} auf, also auch höchstens k von 0 verschiedene Koeffizienten t_i.)

3. Fall: Genauso, aber $\hat{x}_n = t_h x_h + \cdots + t_{n-1} x_{n-1}$ stimmt nicht mit x_n überein. Sei dann $d \in \mathbb{N}$ der Hauptnenner von $t_h, \ldots, t_{n-1}$. Dann ist

$$d\hat{x}_n = \alpha(dt_h z_h + \cdots + dt_{n-1} z_{n-1}) = \alpha(dz_n) = dx_n$$

in $\bar{X}$, also mod m gerechnet. Damit ist gezeigt:

Hilfssatz 3 *(*BOYAR*) Der größte gemeinsame Teiler $\hat{m}$ der Komponenten von $d\hat{x}_n - dx_n$ im 3. Fall ist ein Vielfaches des Moduls m.*

Die erste Phase liefert also ein Vielfaches $\hat{m} \neq 0$ des Moduls m. Der Aufwand dafür beträgt:

- höchstens $k+1$ Versuche, ein lineares Gleichungssystem mit höchstens k Unbekannten über $\mathbb{Q}$ zu lösen,
- eine Bestimmung des größten gemeinsamen Teilers von r Zahlen.

Daneben wird eine unbestimmte Anzahl von Folgegliedern x_n korrekt vorhergesagt, was jeweils ebenfalls mit der Lösung eines solchen linearen Gleichungssystems bezahlt wird.

Wie groß kann $\hat{m}$ sein? Zur Abschätzung braucht man eine obere Schranke M für alle Komponenten aller $\Phi^{(i)}$ auf $\{0, \ldots, m-1\}^i \subseteq X^i$. Zur Herleitung wird die Ungleichung von HADAMARD verwendet: Für beliebige Vektoren $x_1, \ldots, x_k \in \mathbb{R}^k$ gilt

$$|\operatorname{Det}(x_1, \ldots, x_k)| \leq \|x_1\|_2 \cdots \|x_k\|_2$$

mit der euklidischen Norm $\| \bullet \|_2$.

Hilfssatz 4 *$\hat{m} \leq (k+1) \cdot m \cdot \sqrt{k^k} \cdot M^k$, insbesondere wächst $\log(\hat{m})$ höchstens polynomial mit k, $\log(m)$ und $\log(M)$.*

Beweis. Der Koeffizientenvektor t ist Lösung eines linearen Gleichungssystems aus höchstens k Gleichungen mit ebensovielen Unbekannten. Die Koeffizienten z_i dieses Gleichungssystems sind durch M beschränkt. Nach der Ungleichung von HADAMARD für die Determinante und der CRAMERschen Regel sind Zähler dt_i und Nenner d der Lösung durch

$$\prod_{i=1}^{k} \sqrt{\sum_{j=1}^{k} M^2} = \prod_{i=1}^{k} \sqrt{kM^2} = \sqrt{k^k} \cdot M^k$$

beschränkt. Die Komponenten von $d\hat{x}_n$ sind also durch

$$\|d\hat{x}_n\|_\infty = \|\sum dt_i x_i\|_\infty \leq \sqrt{k^k} \cdot M^k \cdot \sum \|x_i\|_\infty \leq km \cdot \sqrt{k^k} \cdot M^k$$

beschränkt, weil m eine Schranke für die Komponenten der x_i ist. Daraus folgt

$$\|d\hat{x}_n - dx_n\|_\infty \leq km \cdot \sqrt{k^k} \cdot M^k + \sqrt{k^k} \cdot M^k \cdot m = (k+1) \cdot m \cdot \sqrt{k^k} \cdot M^k,$$

wie behauptet. $\diamond$

Wie sieht das im Beispiel des gewöhnlichen linearen Kongruenzgenerators aus? Hier ist

$$z_1 = \begin{pmatrix} x_0 \\ 1 \end{pmatrix}, z_2 = \begin{pmatrix} x_1 \\ 1 \end{pmatrix}, z_3 = \begin{pmatrix} x_2 \\ 1 \end{pmatrix}, \ldots$$

Falls $x_1 = x_0$, sind wir im trivialen Fall der konstanten Folge. Andernfalls ist z_3 rationale Linearkombination $t_1 z_1 + t_2 z_2$: Die Lösung des Gleichungssystems

$$\begin{aligned} x_0 t_1 + x_1 t_2 &= x_2, \\ t_1 + t_2 &= 1 \end{aligned}$$

ist

$$t = \frac{1}{d} \cdot \begin{pmatrix} -x_2 + x_1 \\ x_2 - x_0 \end{pmatrix} \quad \text{mit } d = x_1 - x_0.$$

Vorhergesagt wird dann

$$\hat{x}_3 = t_1 x_1 + t_2 x_2 = \frac{-x_2 x_1 + x_1^2 + x_2^2 - x_2 x_0}{x_1 - x_0} = \frac{(x_2 - x_1)^2}{x_1 - x_0} + x_2.$$

Also ist $d(\hat{x}_3 - x_3) = (x_2 - x_1)^2 - (x_1 - x_0)(x_3 - x_2) = y_2^2 - y_1 y_3$ mit der Differenzenfolge (y_i). Falls $\hat{x}_3 = x_3$, müssen wir weiter machen. Sonst erhalten wir, wie aus Hilfssatz 1, $m|\hat{m} = |y_1 y_3 - y_2^2|$.

Im Standard-Beispiel $x_0 = 2134$, $x_1 = 2160$, $x_2 = 6905$, $x_3 = 3778$, also mit $y_1 = 26$, $y_2 = 4745$, $y_3 = -3127$, erhalten wir mit diesem allgemeinen Ansatz

$$\hat{m} = 4745^2 + 26 \cdot 3127 = 22596327.$$

Sieht man aber im konkreten Fall genauer hin und wendet Hilfssatz 3 direkt an, erhält man

$$t_1 = -\frac{365}{2},\ t_2 = \frac{367}{2},\ \hat{x}_3 = \frac{1745735}{2},\ \hat{m} = 2 \cdot (\hat{x}_3 - x_3) = 1738179.$$

In der zweiten Phase des Algorithmus wird das gleiche Verfahren, aber über dem Ring $\hat{R} = \mathbb{Z}/\hat{m}\mathbb{Z}$ durchgeführt. Da man die rationalen Ergebnisse aus der ersten Phase nicht einfach $\bmod \hat{m}$ reduzieren kann, startet man wieder neu bei z_h. Es gibt wieder drei Fälle für jeden Einzelschritt:

1. Fall: $z_n \notin \hat{Z}_{n-1} = \hat{R}z_h + \cdots + \hat{R}z_{n-1}$. Dann wird $\hat{Z}_n = \hat{Z}_{n-1} + \hat{R}z_n$ gesetzt (und dieser $\hat{R}$-Modul durch ein nicht-redundantes Erzeugendensystem

$\{z_{j_1}, \ldots, z_{j_l}\}$ repräsentiert, wobei $z_{j_l} = z_n$). Hier wird x_n nicht vorhergesagt (sondern muss anderswie beschafft werden).

2. Fall: $z_n = t_h z_h + \cdots + t_{n-1} z_{n-1}$. Dann wird $x_n = t_h x_h + \cdots + t_{n-1} x_{n-1}$ vorhergesagt (als Element von $\hat{X} = (\mathbb{Z}/\hat{m}\mathbb{Z})^r$). Die Voraussage sei korrekt.

3. Fall: Genauso, aber die Voraussage $\hat{x}_n = t_h x_h + \cdots + t_{n-1} x_{n-1}$ stimmt in $\hat{X}$ nicht mit x_n überein. Dann wird $\hat{x}_n - x_n$ als Element von $\mathbb{Z}^r$ betrachtet:

Hilfssatz 5 *Der größte gemeinsame Teiler der Koeffizienten von $\hat{x}_n - x_n$ im 3. Fall ist ein Vielfaches von m, aber kein Vielfaches von $\hat{m}$.*

Beweis. Er ist ein Vielfaches von m, weil $\hat{x}_n \bmod m = x_n$ sein muss. Er ist kein Vielfaches von $\hat{m}$, weil sonst ja $\hat{x}_n = x_n$ in $\hat{X}$ wäre. $\diamond$

Im 3. Fall wird $\hat{m}$ durch den ggT dieses größten gemeinsamen Teilers mit $\hat{m}$ ersetzt und die ganze Kette der bisherigen z_j (soweit sie nicht schon redundant waren) $\bmod \hat{m}$ reduziert. Wegen der zweiten Aussage im Hilfssatz ist dabei $\hat{m}$ echt kleiner geworden.

Wegen Hilfssatz 4 kann der dritte Fall insgesamt nicht zu oft auftreten; die Anzahl der Vorkommnisse ist polynomial in k, $\log(m)$ und $\log(M)$. Ist das richtige m erreicht, kann dieser Fall gar nicht mehr vorkommen. Der erste Fall kann in der zweiten Phase insgesamt wegen Satz 2 höchstens ${}^2\log(\#(\mathbb{Z}/\hat{m}\mathbb{Z})^k) = k \cdot {}^2\log(\hat{m})$ Mal vorkommen, und diese Schranke ist polynomial in k, $\log(m)$ und $\log(M)$.

> **Anmerkung.** Die Gemeinsamkeit von erster und zweiter Phase besteht darin, dass beide Male über dem vollen Quotientenring gerechnet wird: Der volle Quotientenring von $\mathbb{Z}$ ist der Quotientenkörper $\mathbb{Q}$. In einem Restklassenring $\mathbb{Z}/m\mathbb{Z}$ dagegen sind die Nicht-Nullteiler genau die zu m teilerfremden Elemente, also die Einheiten. Daher ist $\mathbb{Z}/m\mathbb{Z}$ sein eigener voller Quotientenring.

Im Standard-Beispiel haben wir nach der ersten Phase $\hat{m} = 1738179$ und müssen nun das lineare Gleichungssystem (1) $\bmod \hat{m}$ lösen. Da dessen Determinante -26 zu $\hat{m}$ teilerfremd sind, ist bereits $Z_2 = \hat{R}^2$, und Fall1 wird nicht mehr auftreten. Es ist $-26t_1 = 4745$, also $t_1 = 868907$, da das Inverse von -26 gleich 66853 ist (alles in $\mathbb{Z}/\hat{m}\mathbb{Z}$). Damit folgt $t_2 = 1 - t_1 = 869273$, und $\hat{x}_3 = 1_1 x_1 + t_2 x_2 = 3778$ wird korrekt vorausgesagt.

Im nächsten Schritt werden die neuen Koeffizienten t_1 und t_2 in der Linearkombination $z_4 = t_1 z_1 + t_2 z_2$ bestimmt. Zu lösen ist also (in $\mathbb{Z}/\hat{m}\mathbb{Z}$)

$$\begin{aligned} 2134t_1 + 2160t_2 &= 3778, \\ t_1 + t_2 &= 1. \end{aligned}$$

Elimination von t_2 ergibt $-26t_1 = 1618$, also $t_1 = 401056$, und daraus $t_2 = 1337124$ sowie $\hat{x}_4 = 1_1 x_1 + t_2 x_2 = 302190$. Da $x_4 = 8295$, sind wir im

3. Fall und haben $\hat{m}$ zu korrigieren:

$$\mathrm{ggT}(\hat{x}_4 - x_4, \hat{m}) = \mathrm{ggT}(293895, 1738179) = 8397.$$

Weil $\hat{m} < 2x_2$, wird von jetzt an nur noch der zweite Fall auftreten, d. h., der Rest der Folge wird korrekt vorhergesagt.

Ein **Vorhersageverfahren** für einen allgemeinen Kongruenzgenerator ist ein Algorithmus, der als Eingabe die Startwerte $x_0, \ldots, x_{h-1}$ erhält, dann Schätzungen für $x_h, x_{h+1}, \ldots$ auswirft und diese anschließend mit dem jeweiligen wahren Wert vergleicht; bei einer Fehlvorhersage werden unter Verwendung des wahren Werts die Parameter des Verfahrens adjustiert. Das Vorhersageverfahren ist **effizient**, wenn

(a) der Aufwand für die Vorhersage jedes x_n polynomial in r, k und $\log(m)$ ist,

(b) die Zahl der Fehlvorhersagen durch ein Polynom in r, k und $\log(m)$ beschränkt ist, ebenso der Aufwand für die Parameteradjustierung im Fall einer Fehlvorhersage.

Der Algorithmus von Boyar/Krawczyk, den wir in diesem Abschnitt behandelt haben, erfüllt (b). Er erfüllt auch (a), da das Lösen linearer Gleichungssysteme über Restklassenringen $\mathbb{Z}/m\mathbb{Z}$ effizient möglich ist, wie schon früher gezeigt. Damit ist bewiesen:

Hauptsatz 1 *Für einen beliebigen effizienten Kongruenzgenerator ist der Algorithmus von* Boyar/Krawczyk *ein effizientes Vorhersageverfahren.*

Die Anwendung auf nichtlineare Generatoren wird an einem weiteren einfachen Zahlenbeispiel gezeigt. Von einem quadratischen Generator der Form

$$x_n = ax_{n-1}^2 + bx_{n-1} + c \bmod m$$

sei die Zahlenfolge

$$x_0 = 63,\ x_1 = 96,\ x_2 = 17,\ x_3 = 32,\ x_4 = 37,\ x_5 = 72$$

erzeugt worden. Wir verwenden also $X = \mathbb{Z}$, $Z = \mathbb{Z}^3$, $h = 1$. In der ersten Phase spannen

$$z_1 = \begin{pmatrix} 3969 \\ 63 \\ 1 \end{pmatrix} z_2 = \begin{pmatrix} 9216 \\ 96 \\ 1 \end{pmatrix} z_3 = \begin{pmatrix} 289 \\ 17 \\ 1 \end{pmatrix}$$

schon ganz $\mathbb{Q}^3$ auf, denn die Koeffizientenmatrix ist die Vandermonde-Matrix mit Determinante 119922. Die Auflösung von

$$z_4 = \begin{pmatrix} 1024 \\ 32 \\ 1 \end{pmatrix} = t_1 z_1 + t_2 z_2 + t_3 z_3$$

ergibt $t_1 = \frac{160}{253}, t_2 = -\frac{155}{869}, t_3 = \frac{992}{1817}$ mit Hauptnenner $d = 11{\cdot}23{\cdot}79 = 19987$. Die Voraussage ist $\hat{x}_4 = \frac{1502019}{19987} \neq x_4$. Der erste geschätzte Modul ist also

$$\hat{m} = d\hat{x}_4 - dx_4 = 762500.$$

Damit ist die erste Phase abgeschlossen. Das gleiche lineare Gleichungssystem soll jetzt über $\mathbb{Z}/\hat{m}\mathbb{Z}$ aufgelöst werden, wo die Determinante allerdings ein Nullteiler ist, und ergibt zwei Lösungen, darunter

$$t_1 = 156720, t_2 = 719505, t_3 = 648776.$$

Vorausgesagt wird also der korrekte Wert

$$\hat{x}_4 = 156720 \cdot 96 + 719505 \cdot 17 + 648776 \cdot 32 \bmod 763500 = 37.$$

Daher sind wir im Fall 2 und versuchen, x_5 vorauszusagen:

$$z_5 = \begin{pmatrix} 1369 \\ 37 \\ 1 \end{pmatrix} = t_1 z_1 + t_2 z_2 + t_3 z_3$$

ergibt zwei Lösungen, darunter

$$t_1 = 2010, t_2 = 558640, t_3 = 201851,$$

also

$$\hat{x}_5 = 136572, \quad \hat{x}_5 - x_5 = 136500.$$

Also wird $\hat{m}$ korrigiert zu

$$\mathrm{ggT}(762500, 136500) = 500.$$

Damit sind die bekannten Werte erschöpft. Da alle z_i in $\hat{Z}_3 = \hat{R}z_1 + \hat{R}z_2 + \hat{R}z_3 \neq \hat{R}^3$ liegen, bleibt der erste Fall bei der weiteren Vorhersage möglich. Da $x_0, \ldots, x_5$ kleiner als die Hälfte des aktuellen Moduls $\hat{m}$ sind, bleibt auch der dritte Fall möglich, der Modul muss also eventuell noch weiter korrigiert werden.

Für die Vorhersage von x_6 ergibt sich (mod500)

$$t_1 = 240, t_2 = 285, t_3 = 476, x_6 = 1117.$$

Übungsaufgabe. Was passiert, wenn man im Standardbeispiel nach der ersten Phase mit dem Wert $\hat{m} = 22596327$ weiterrechnet?

2.8 Analyse bei gestutztem Output

Schwieriger wird die Kryptoanalyse, wenn der Zufallsgenerator nicht alle Bits der erzeugten Zahlen ausgibt. Das kann auf Absicht beruhen, aber auch als Nebeneffekt dadurch entstehen, dass die Zahlen ins reelle Intervall $[0, 1]$ oder sonstwie transformiert und dabei gerundet werden; besonders wenn der Modul m keine Zweierpotenz ist, muss man den Verlust einiger gering signifikanter Bits in Erwägung ziehen. Die Differenzenfolge ist dann natürlich auch nur ungefähr bekannt, die größten gemeinsamen Teiler sind nicht mehr zu ermitteln, und die Algorithmen von PLUMSTEAD-BOYAR und BOYAR/KRAWCZYK brechen zusammen.

Sind die Parameter des Zufallsgenerators bekannt, kann man es mit systematischem Probieren versuchen. Für die folgende Überlegung (die im übrigen nicht streng durchgeführt wird) muss der Zufallsgenerator nicht einmal notwendig linear sein. Nehmen wir an, es werden n-Bit-Zahlen erzeugt, aber jeweils nur q Bits ausgegeben und $n - q$ Bits zurückgehalten. Die ausgegebenen Bits stammen jeweils von festen, bekannten Positionen. Dann gibt es zu jedem ausgegebenen q-Bit-Fragment 2^{n-q} mögliche vollständige Werte; anders ausgedrückt, enthält eine zufällig gewählte n-Bit-Zahl mit der Wahrscheinlichkeit $1/2^q$ die vorgegebenen Bits an den richtigen Stellen.

Die weitere Überlegung wird hier nur exemplarisch für den Fall durchgeführt, dass die q ausgegebenen Bits die Leitbits sind. Man zerlegt also den Startwert x in $x = x_0 2^{n-q} + x_1$ mit $0 \le x_1 < 2^{n-q}$. Der Wert x_0, die ersten q Bits, ist bekannt. Der Angreifer startet eine Exhaustion über die 2^{n-q} verschiedenen möglichen Werte für x_1. Zu jeder Wahl von x_1 bildet er $x = x_0 2^{n-q} + x_1$ und $y = s(x)$ mit der erzeugenden Funktion s des betrachteten Zufallsgenerators. Diesen Wert y vergleicht er mit den ihm bekannten führenden q Bits des wahren Wertes. Ist der Zufallsgenerator statistisch gut, so ist die Wahrscheinlichkeit eines Treffers hierbei $1/2^q$. Das bedeutet, dass von den 2^{n-q} Werten für x_0 noch ungefähr 2^{n-2q} übrig bleiben. Falls $q \ge \frac{n}{2}$, können wir also genau einen Treffer erwarten. Ansonsten wird weitergemacht. Nach k Schritten ist die Trefferzahl ungefähr 2^{n-kq}. Die zu erwartende nötige Schrittzahl ist also $\ge k$ nur, wenn $kq \le n$, also $q \ge \frac{n}{k}$. Bei $q = \frac{1}{4}$ (zum Beispiel $n = 32$, $q = 8$, d. h., Ausgabe von 8 Bit einer 32-Bit-Zahl) reichen also vier q-Bit-Fragmente (wobei man im Beispiel allerdings schon 2^{24} Zahlen durchprobieren muss). Dieses Probierverfahren ist bei kleinem Modul m durchführbar; der Aufwand wächst aber exponentiell mit m (wenn der Anteil $r = \frac{q}{n}$ der ausgegebenen Bits gegen 1 beschränkt bleibt).

Für lineare Kongruenzgeneratoren haben FRIEZE/KENNAN/LAGARIAS, HÅSTAD/SHAMIR und J. STERN ein besseres (probabilistisches) Verfahren entwickelt, dessen erster Schritt im folgenden Satz resümiert wird (ohne Beweis).

Satz 7 (FRIEZE, KANNAN, LAGARIAS) *Sei* $0 \le r \le 1$ *und* p_n *die Wahr-*

scheinlichkeit, dass das Verfahren nicht den Modul m eines linearen Kongruenzgenerators bestimmt. Dann gilt für beliebiges $\varepsilon > 0$

$$p_n = \mathrm{O}(n^{\frac{5r-3}{2}+\varepsilon}).$$

Insbesondere $p_n \to 0$ mit $n \to \infty$, wenn $r > \frac{2}{5}$. *Es gelingt also mit großer Wahrscheinlichkeit, m zu finden, wenn mehr als $2/5$ der Leitbits ausgegeben werden.*

Im zweiten Schritt geht es darum, den Multiplikator a unter der Annahme zu bestimmen, dass der Modul m schon bekannt ist. Im dritten Schritt sind noch die vollen Zahlen x_i oder die Differenzen y_i zu bestimmen. Auch dies gelingt außer für eine vernachlässigbare Menge von Multiplikatoren, die mit wachsendem m immer kleiner gewählt werden kann, und für die „guten“ Multiplikatoren benötigt man nur noch etwas mehr als ein Drittel der Leitbits von x_0, x_1, x_2 und x_3, um die volle Bitinformation herzuleiten. Ähnliche, etwas schwächere Ergebnisse hat J. Stern auch für den Fall gefunden, dass statt der Leitbits „innere Bits“ der erzeugten Zahlen ausgegeben werden.

Die Kryptoanalyse der linearen Kongruenzgeneratoren hat also grundsätzliche Schwächen aufgedeckt, und zwar unabhängig davon, ob ein solcher Generator statistisch gute oder schlechte Eigenschaften hat.

Trotzdem sind die linearen Kongruenzgeneratoren für statistische Anwendungen durchaus brauchbar, denn es scheint extrem unwahrscheinlich, dass ein Anwendungsprogramm „aus Versehen“ die nötigen Schritte enthält, um einen linearen Kongruenzgenerator zu knacken und so seinen Determinismus aufzudecken. Für die kryptographische Anwendung sind die linearen Kongruenzgeneratoren auch bei gestutztem Output aber ein für allemal disqualifiziert. Offen ist allerdings, ob die Einwände auch zutreffen, wenn man nur „ganz wenige“ Bits ausgibt (etwa nur ein Viertel oder gar nur $\log\log(m)$ Bits).

Literaturverweise

- J. Stern: Secret linear congruential generators are not cryptographically secure. FOCS 28 (1987), 421–426.
- Frieze/Håstad/Kannan/Lagarias/Shamir: Reconstructing truncated integer variables satisfying linear congruences. SIAM J. Comput. 17 (1988), 262–280.
- J. Boyar: Inferring sequences produced by a linear congruential generator missing low-order bits. J. Cryptology 1 (1989), 177–184.

2.9 Resümee

Die Abschnitte 2.1 bis 2.7 ergaben ein Vorhersageverfahren, das so abläuft:

1. Der Kryptoanalytiker findet durch Klartextraten ein Stück der Schlüssel-Bitfolge, so lange, bis sich eine geeignete lineare Relation aufstellen lässt (NOETHERsches Prinzip).

2. Er sagt mit Hilfe dieser linearen Relation weitere Schlüsselbits voraus.

3. Erweisen sich vorausgesagte Bits als falsch (weil der Klartext an dieser Stelle aufhört, sinnvoll zu sein), muss der Kryptoanalytiker wieder etwas Klartext raten und damit die Parameter adjustieren; dann kann er weiter vorhersagen.

Dieses Verfahren ist für die „klassischen“ Zufallsgeneratoren effizient, also für Kongruenzgeneratoren – auch bei unbekanntem Modul – und für Schieberegister – auch nichtlineare. „Effizient“ bedeutet hier auch, dass die benötigte Menge von bekanntem oder erratenem Klartext klein ist.

Das Fazit daraus ist, dass für kryptographisch sichere Zufallserzeugung niemals der Zustand des Zufallsgenerators direkt als Output verwendet werden sollte; vielmehr ist eine Transformation dazwischen zu schalten. Abschnitt 2.8 zeigt exemplarisch, dass das schlichte Unterdrücken einiger Bits, das „Stutzen“ oder die „Dezimierung“, als Output-Transformation aber auch nicht ohne weiteres ausreicht. Bessere Output-Transformationen werden in den folgenden Abschnitten behandelt.

Die Grauzone zwischen dem, was dem Kryptoanalytiker Erfolg garantiert, und dem, was den Kryptologen ruhig schlafen lässt, ist freilich sehr breit. Auf jeden Fall sollten besser beide Prozesse

- Zustandsänderung,

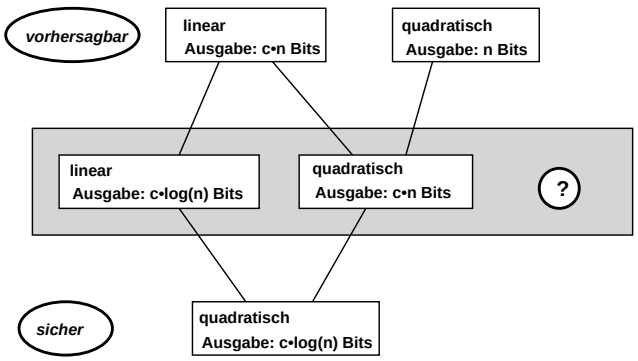
- Output-Transformation,

nichtlinear sein. In der Grauzone, wo keine nützlichen Aussagen über die Sicherheit bekannt sind, liegen unter anderem quadratische Kongruenzgeneratoren mit mäßig gestutztem Output.

Im Folgenden werden zwei Ansätze behandelt, zu sicheren Zufallsgeneratoren zu kommen:

- Kombination linearer Schieberegister mit nichtlinearer Output-Transformation,

- nichtlineare Kongruenzgeneratoren mit stark gestutztem Output.


vorhersagbar
linear
Ausgabe: c•n Bits
quadratisch
Ausgabe: n Bits
linear
Ausgabe: c•log(n) Bits
quadratisch
Ausgabe: c•n Bits
?
sicher
quadratisch
Ausgabe: c•log(n) Bits

3 Schieberegister

3.1 Die lineare Komplexität von Bitfolgen

Wir betrachten – zunächst unendliche – Bitfolgen $u = (u_i)_{i\in\mathbb{N}} \in \mathbb{F}_2^{\mathbb{N}}$. Gesucht ist ein lineares Schieberegister möglichst geringer Länge, das die Folge produziert.

Gibt es ein solches Schieberegister, muss die Folge periodisch sein. Umgekehrt wird jede periodische Folge stets durch ein lineares Schieberegister erzeugt, dessen Länge die Summe der Längen von Vorperiode und Periode ist – nämlich durch das **zirkuläre Schieberegister**, das als Rückkopplung nur dasjenige Bit wieder einspeist, bei dem die Periode beginnt; ist $u_{l+i} = u_{k+i}$ für $i \geq 0$, so sind die Koeffizienten $a_{l-k} = 1$, $a_i = 0$ sonst. Damit ist gezeigt:

Hilfssatz 1 *Eine Bitfolge $u \in \mathbb{F}_2^{\mathbb{N}}$ lässt sich genau dann von einem linearen Schieberegister erzeugen, wenn sie periodisch ist.*

Definition. Die **lineare Komplexität** $\lambda(u)$ einer Bitfolge $u \in \mathbb{F}_2^{\mathbb{N}}$ ist die minimale Länge eines linearen Schieberegisters, das u erzeugt.

Ist u konstant 0, wird $\lambda(u) = 0$, ist u nicht periodisch, wird $\lambda(u) = \infty$ gesetzt.

Es handelt sich hierbei also um einen Komplexitätsbegriff, der auf dem sehr speziellen Maschinen-Modell der linearen Schieberegister beruht.

Bemerkungen und Beispiele

1. Falls $\tau(u)$ die Summe aus der Länge der Periode und der Vorperiode von u ist und u von einem linearen Schieberegister der Länge l erzeugt wird, gilt
$$\lambda(u) \leq \tau(u) \leq 2^l - 1 \quad \text{und} \quad \lambda(u) \leq l.$$

2. Die periodisch wiederholte Folge $0, \ldots, 0, 1$ ($l-1$ Nullen) hat die Periode l und die lineare Komplexität l. Ein lineares Schieberegister der Länge $< l$ würde nämlich mit dem Nullvektor als Startwert gefüttert und könnte dann nur noch weitere Nullen produzieren.

Für endliche Bitfolgen $u = (u_0, \ldots, u_{N-1}) \in \mathbb{F}_2^N$ definiert man die lineare Komplexität analog. Insbesondere ist $\lambda(u)$ die minimale Zahl l, so dass es $a_1, \ldots, a_l \in \mathbb{F}_2$ gibt mit

$$u_i = a_1 u_{i-1} + \cdots + a_l u_{i-l} \quad \text{für } i = l, \ldots, N-1.$$

Bemerkungen und Beispiele

3. Für $u \in \mathbb{F}_2^n$ gilt $0 \leq \lambda(u) \leq N$.

4. $\lambda(u) = 0 \Longleftrightarrow u_0 = \cdots = u_{N-1} = 0$.

5. $\lambda(u) = N \iff u = (0, \ldots, 0, 1)$. Die Implikation „$\Longleftarrow$“ folgt wie in Bemerkung 2. Für die Umkehrung kann u_{N-1} nicht 0 sein, denn sonst könnte man das lineare Schieberegister der Länge $N-1$ mit Rückkopplung konstant 0 nehmen. Also muss $u_{N-1} = 1$ sein. Gäbe es vorher in der Folge schon eine 1, könnte man das Schieberegister der Länge $N-1$ nehmen, das genau diese Bitposition rückkoppelt.

6. Sind die ersten $2\lambda(u)$ Bits der Bitfolge u bekannt, so lässt sich der Rest von u daraus vorhersagen.

3.2 Synthese linearer Schieberegister

In diesem Abschnitt geht es darum, zu einer gegebenen endlichen Bitfolge ein lineares Schieberegister kürzester Länge zu finden. Der Ansatz aus Abschnitt 2 zur Vorhersage von Zufallsgeneratoren lieferte ein lineares Schieberegister unter der Annahme, dass man schon ein evtl. nichtlineares hat, lässt aber kaum erkennen, ob es vielleicht ein kürzeres gäbe. Mit einem etwas anderen Ansatz ist die gestellte Aufgabe aber überraschend einfach zu lösen: mit einem Algorithmus von Massey (1969), der in einem anderen Kontext vorher schon von Berlekamp (1968) angegeben worden war.

Da keine speziellen Eigenschaften des Körpers $\mathbb{F}_2$ benützt werden, wird hier ein beliebiger Körper K zu Grunde gelegt. Gesucht wird ein homogener Kongruenzgenerator möglichst geringer Rekursionstiefe l, der eine gegebene endliche Folge $u \in K^N$ erzeugt.

Für einen solchen Kongruenzgenerator mit Bildungsgestz

$$u_k = a_1 u_{k-1} + \cdots + a_l u_{k-l} \quad \text{für } k = l, \ldots, N-1$$

ist $(a_1, \ldots, a_l) \in K^l$ der Koeffizientenvektor; das Polynom

$$\varphi = 1 - a_1 T - \cdots - a_l T^l \in K[T]$$

heißt **Rückkopplungspolynom**. Es ist das reziproke Polynom zum charakteristischen Polynom der Begleitmatrix A: Ist dieses $\chi = \mathrm{Det}(T \cdot 1 - A)$, so ist

$$\chi = T^l - a_1 T^{l-1} - \cdots - a_l, \quad \text{also} \quad \varphi = T^l \cdot \chi(\frac{1}{T}).$$

Hilfssatz 2 *Der homogene lineare Kongruenzgenerator mit Koeffizienten $(a_1, \ldots, a_l)$ erzeuge die Folge $u = (u_0, \ldots, u_{n-1}) \in K^n$, aber nicht die Folge $\hat{u} = (u_0, \ldots, u_n) \in K^{n+1}$. Dann hat jeder homogene lineare Kongruenzgenerator, der $\hat{u}$ erzeugt, eine Länge $m \geq n + 1 - l$.*

Beweis. **Fall 1:** $l \geq n$. Dann ist $l + m \geq n + 1$, außer wenn $l = n$, $m = 0$. In diesem Fall müsste aber wegen $m = 0$ notwendig $u_0 = \ldots = u_n$ sein, und der Generator würde auch $\hat{u}$ erzeugen, Widerspruch.

Fall 2: $l \leq n - 1$. *Annahme*: $m \leq n - l$. Es ist

$$u_j = a_1 u_{j-1} + \cdots + a_l u_{j-l} \quad \text{für } l \leq j \leq n-1.$$

Sei $(b_1, \ldots, b_m)$ der Koeffizientenvektor eines homogenen linearen Kongruenzgenerators, der $\hat{u}$ erzeugt; dann ist

$$u_j = b_1 u_{j-1} + \cdots + b_m u_{j-m} \quad \text{für } m \leq j \leq n.$$

Insgesamt folgt

$$\begin{aligned} u_n &\neq a_1 u_{n-1} + \cdots + a_l u_{n-l} \\ &= \sum_{i=1}^{l} a_i \cdot \underbrace{\sum_{k=1}^{m} b_k u_{n-i-k}}_{u_{n-i}} \quad [\text{da } n-l \geq m] \\ &= \sum_{k=1}^{m} b_k \cdot \underbrace{\sum_{i=1}^{l} a_i u_{n-k-i}}_{u_{n-k}} = u_n, \end{aligned}$$

Widerspruch. $\diamond$

Sei nun $u \in K^N$ eine Folge. Für $0 \leq n \leq N$ sei $\lambda_n(u) = \lambda_n$ die kleinste Rekursionstiefe eines Generators, der $(u_0, \ldots, u_{n-1})$ erzeugt.

Hilfssatz 3 *Für jede Folge $u \in K^N$ gilt:*

(i) $\lambda_{n+1} \geq \lambda_n$.
(ii) *Genau dann, wenn es einen Generator der Rekursionstiefe λ_n gibt, der $(u_0, \ldots, u_n)$ erzeugt, gilt $\lambda_{n+1} = \lambda_n$.*
(iii) *Gibt es einen solchen nicht, ist*

$$\lambda_{n+1} \geq n + 1 - \lambda_n.$$

Beweis. (i) Jeder Generator, der $(u_0, \ldots, u_n)$ erzeugt, erzeugt erst recht $(u_0, \ldots, u_{n-1})$.

(ii) folgt aus (i).

(iii) Die Voraussetzung von Hilfssatz 2 gilt für jeden Generator von $(u_0, \ldots, u_{n-1})$. $\diamond$

Satz 1 [Massey] *Sei $u \in K^N$ und $0 \leq n \leq N-1$. Sei ferner $\lambda_{n+1}(u) \neq \lambda_n(u)$. Dann ist*

$$\lambda_n(u) \leq \frac{n}{2} \quad \textit{und} \quad \lambda_{n+1}(u) = n + 1 - \lambda_n(u).$$

Beweis. Der Fall $\lambda_n = 0$ ist besonders leicht: Es ist $u_0 = \ldots = u_{n-1} = 0$. Falls $u_n = 0$, ist $\lambda_{n+1} = \lambda_n = 0$, also nichts zu beweisen. Falls $u_n \neq 0$, ist $\lambda_{n+1} = n + 1 = n + 1 - \lambda_n$ nach Bemerkung 5 in 3.1.

Allgemein folgt die erste Aussage aus der zweiten, denn wegen $\lambda_n < \lambda_{n+1}$ ist $2\lambda_n < n + 1$.

Die zweite Aussage wird nun durch Induktion über n bewiesen. Im Fall $n = 0$ ist $\lambda_0 = 0$ – dieser Fall ist schon erledigt.

Sei jetzt $n \geq 1$. Sei o. B. d. A. $l := \lambda_n \geq 1$. Sei

$$u_j = a_1 u_{j-1} + \cdots + a_l u_{j-l} \quad \text{für } j = l, \ldots, n-1;$$

das zugehörige Rückkopplungspolynom ist also

$$\varphi := 1 - a_1 T - \cdots - a_l T^l \in K[T].$$

Die „n-te Diskrepanz“ sei

$$d_n := u_n - a_1 u_{n-1} - \cdots - a_l u_{n-l}.$$

Ist $d_n = 0$, so erzeugt der Generator auch u_n, und es ist nichts zu beweisen. Sei also $d_n \neq 0$. Sei r die Länge der Folge vor der letzten Zunahme der linearen Komplexität, also

$$t := \lambda_r < l, \quad \lambda_{r+1} = l.$$

Wegen der Induktionsannahme ist $l = r + 1 - t$. Ist

$$u_j = b_1 u_{j-1} + \cdots + b_t u_{j-t} \quad \text{für } j = t, \ldots, r-1,$$

so ist das zugehörige Rückkopplungspolynom

$$\psi := 1 - b_1 T - \cdots - b_t T^t \in K[T],$$

und für die analog gebildete r-te Diskrepanz gilt

$$d_r := u_r - b_1 u_{r-1} - \cdots - b_t u_{r-t} \neq 0,$$

Ist $t = 0$, so $\psi = 1$ und $d_r = u_r$. Nun wird das Polynom

$$\eta := \varphi - \frac{d_n}{d_r} \cdot T^{n-r} \cdot \psi = 1 - c_1 T - \cdots - c_m T^m \in K[T]$$

gebildet mit $m = \operatorname{Grad} \eta$. Was macht der zugehörige homogene lineare Kongruenzgenerator? Es ist

$$\begin{aligned} u_j - \sum_{i=1}^{m} c_i u_{j-i} &= u_j - \sum_{i=1}^{l} a_i u_{j-i} - \frac{d_n}{d_r} \cdot \left[u_{j-n+r} - \sum_{i=1}^{t} b_i u_{j-n+r-i} \right] \\ &= 0 \quad \text{für } j = m, \ldots, n; \end{aligned}$$

für $j = 0, \ldots, n-1$ folgt das direkt, für $j = n$ kommt zunächst $d_n - [d_n/d_r] \cdot d_r$ heraus. Er erzeugt also $(u_0, \ldots, u_n)$. Nun ist

$$\lambda_{n+1} \leq m \leq \max\{l, n - r + t\} = \max\{l, n + 1 - l\}.$$

Wegen der Monotonie der linearen Komplexität ist $m > l$, nach Hilfssatz 2 ist $m \geq n + 1 - l$. Also folgt $m = n + 1 - l$ und $\lambda_{n+1} = m$. Damit ist die Behauptung bewiesen. $\diamond$

Korollar 1 *Ist $d_n \neq 0$ und $\lambda_n \leq \frac{n}{2}$, so ist*

$$\lambda_{n+1} = n + 1 - \lambda_n > \lambda_n.$$

Beweis. Nach Hilfssatz 2 ist $\lambda_{n+1} \geq n + 1 - \lambda_n$, also $\lambda_{n+1} \geq \frac{n}{2} + 1 > \lambda_n$. Nach Satz 1 folgt daraus $\lambda_{n+1} = n + 1 - \lambda_n$. $\diamond$

Bei dem sukzessiven Aufbau eines linearen Rekurrenzgenerators im Beweis des Satzes tritt also in jedem Iterationsschritt einer der folgenden Fälle ein:

- $d_n = 0$: Dann ist $\lambda_{n+1} = \lambda_n$.
- $d_n \neq 0$: Dann ist
  - $\lambda_{n+1} = \lambda_n$, falls $\lambda_n > \frac{n}{2}$,
  - $\lambda_{n+1} = n + 1 - \lambda_n$, falls $\lambda_n \leq \frac{n}{2}$.

  Insbesondere gilt stets:

- Ist $\lambda_n > \frac{n}{2}$, so $\lambda_{n+1} = \lambda_n$.
- Ist $\lambda_n \leq \frac{n}{2}$, so $\lambda_{n+1} = \lambda_n$ oder $\lambda_{n+1} = n + 1 - \lambda_n$.

Nebenbei haben wir eine alternative Möglichkeit gefunden, lineare Schieberegister vorherzusagen:

Korollar 2 *Wird $u \in \mathbb{F}_2^n$ von einem linearen Schieberegister der Länge $\leq l$ erzeugt, so lässt sich ein solches aus $u_0, \ldots, u_{2l-1}$ bestimmen.*

Beweis. Wäre erstmals $d_n \neq 0$ für $n \geq 2l$, so $\lambda_n \leq l \leq \frac{n}{2}$, also $\lambda_{n+1} = n + 1 - \lambda_n \geq l + 1$, Widerspruch. $\diamond$

3.3 Der Berlekamp-Massey-Algorithmus

Der Beweis von Satz 1 ist konstruktiv: Er enthält einen Algorithmus, mit dem sukzessive ein linearer Rekurrenzgenerator aufgebaut wird. Beim Schritt von der Folgenlänge n zur Folgenlänge $n+1$ gibt es drei mögliche Fälle (1, 2a, 2b):

1. Fall $d_n = 0$, d. h. der Generator zum Rückkopplungspolynom φ erzeugt auch u_n: Dann bleiben φ und l ungeändert und ebenso ψ, t, r, d_r.

2. Fall $d_n \neq 0$, d. h. der Generator zum Rückkopplungspolynom φ erzeugt u_n nicht: Dann wird ein neues Rückkopplungspolynom η gebildet, dessen zugehöriger Generator $(u_0, \ldots, u_n)$ erzeugt. Es wird weiter unterschieden:

a) $l > \frac{n}{2}$: Dann ist $\lambda_{n+1} = \lambda_n$; es wird φ durch η ersetzt, l bleibt, und ebenso bleiben ψ, t, r, d_r.

b) $l \leq \frac{n}{2}$: Dann ist $\lambda_{n+1} = n + 1 - \lambda_n$; es wird φ durch η und l durch $n + 1 - l$ ersetzt, ferner ψ durch φ, t durch l, r durch n und d_r durch d_n.

Damit ist der Berlekamp-Massey-Algorithmus semiformal beschreibbar:

Input: Eine Folge $u = (u_0, \ldots, u_{N-1}) \in K^N$.

Output: Die lineare Komplexität $\lambda_N(u)$,

das Rückkopplungspolynom φ eines linearen Rekurrenzgenerators der Länge $\lambda_N(u)$, der u erzeugt.

Hilfsvariablen: n: aktueller Index, initialisiert mit $n := 0$,

l: aktuelle lineare Komplexität, initialisiert mit $l := 0$,

φ: aktuelles Rückkopplungspolynom $= 1 - a_1 T - \cdots - a_l T^l$, initialisiert mit $\varphi := 1$,

Invarianzbedingung: $u_i = a_1 u_{i-1} + \cdots + a_l u_{i-l}$ für $l \leq i < n$,

d: aktuelle Diskrepanz $= u_n - a_1 u_{n-1} - \cdots - a_l u_{n-l}$,

r: voriger Index, initialisiert mit $r := -1$,

t: vorige lineare Komplexität,

ψ: voriges Rückkopplungspolynom $= 1 - b_1 T - \cdots - b_t T^t$, initialisiert mit $\psi := 1$,

Invarianzbedingung: $u_i = b_1 u_{i-1} + \cdots + b_t u_{i-t}$ für $t \leq i < r$,

d': vorige Diskrepanz $= u_r - b_1 u_{r-1} - \cdots - b_t u_{r-t}$, initialisiert mit $d' := 1$,

η: neues Rückkopplungspolynom,

m: neue lineare Komplexität.

Iterationsschritte: Für $n = 0, \ldots, N-1$:

$d := u_n - a_1 u_{n-1} - \cdots - a_l u_{n-l}$
Falls $d \neq 0$
  $\eta := \varphi - \frac{d}{d'} \cdot T^{n-r} \cdot \psi$
  Falls $l \leq \frac{n}{2}$ [lineare Komplexität wächst]
    $m := n + 1 - l$
    $t := l$
    $l := m$
    $\psi := \varphi$
    $r := n$
    $d' := d$
  $\varphi := \eta$

Natürlich kann man sich auch gleich die ganze Folge (λ_n) ausgeben lassen.

Dieser Algorithmus wird jetzt auf das **Beispiel** der Folge 001101110 angewendet. Der Fall $d \neq 0$, $l \leq \frac{n}{2}$ wird durch „[!]“ bezeichnet.

Eingangsbedingungen				Aktionen
$n = 0$	$u_0 = 0$	$l = 0$	$\varphi = 1$	$d := u_0 = 0$
$r = -1$	$d' = 1$	$t =$	$\psi = 1$	
$n = 1$	$u_1 = 0$	$l = 0$	$\varphi = 1$	$d := u_1 = 0$
$r = -1$	$d' = 1$	$t =$	$\psi = 1$	
$n = 2$	$u_2 = 1$	$l = 0$	$\varphi = 1$	$d := u_2 = 1$ [!]
$r = -1$	$d' = 1$	$t =$	$\psi = 1$	$\eta := 1 - T^3$
				$m := 3$
$n = 3$	$u_3 = 1$	$l = 3$	$\varphi = 1 - T^3$	$d := u_3 - u_0 = 1$
$r = 2$	$d' = 1$	$t = 0$	$\psi = 1$	$\eta := 1 - T - T^3$
$n = 4$	$u_4 = 0$	$l = 3$	$\varphi = 1 - T - T^3$	$d := u_4 - u_3 - u_1 = -1$
$r = 2$	$d' = 1$	$t = 0$	$\psi = 1$	$\eta := 1 - T + T^2 - T^3$
$n = 5$	$u_5 = 1$	$l = 3$	$\varphi = 1 - T + T^2 - T^3$	$d := u_5 - u_4 + u_3 - u_2 = 1$
$r = 2$	$d' = 1$	$t = 0$	$\psi = 1$	$\eta := 1 - T + T^2 - 2T^3$

Von jetzt an unterscheiden sich die Ergebnisse je nach Charakteristik des Grundkörpers K. Sei zuerst $\operatorname{char} K \neq 2$. Dann geht es so weiter:

Eingangsbedingungen	Aktionen
$n = 6 \quad u_6 = 1 \quad l = 3$ $\varphi = 1 - T + T^2 - 2T^3$ $r = 2 \quad d' = 1 \quad t = 0 \quad \psi = 1$	$d := u_6 - u_5 + u_4 - 2u_3 = -2$ [!] $\eta = 1 - T + T^2 - 2T^3 + 2T^4$ $m := 4$
$n = 7 \quad u_7 = 1 \quad l = 4$ $\varphi = 1 - T + T^2 - 2T^3 + 2T^4$ $r = 6 \quad d' = -2 \quad t = 3$ $\psi = 1 - T + T^2 - 2T^3$	$d := u_7 - u_6 + u_5 - 2u_4 + 2u_3 = 3$ $\eta = 1 + \frac{1}{2}T - \frac{1}{2}T^2 - \frac{1}{2}T^3 - T^4$
$n = 8 \quad u_8 = 0 \quad l = 4$ $\varphi = 1 + \frac{1}{2}T - \frac{1}{2}T^2 - \frac{1}{2}T^3 - T^4$ $r = 6 \quad d' = -2 \quad t = 3$ $\psi = 1 - T + T^2 - 2T^3$	$d := u_8 + \frac{1}{2}u_7 - \frac{1}{2}u_6 - \frac{1}{2}u_5 - u_4 = -\frac{1}{2}$ [!] $\eta := 1 + \frac{1}{2}T - \frac{3}{4}T^2 - \frac{1}{4}T^3 - \frac{5}{4}T^4 + \frac{1}{2}T^5$ $m := 5$

Als Ergebnis erhalten wir die Folge der linearen Komplexitäten

$$\lambda_0 = 0, \lambda_1 = 0, \lambda_2 = 0, \lambda_3 = 3, \lambda_4 = 3, \lambda_5 = 3, \lambda_6 = 3, \lambda_7 = 4, \lambda_8 = 4, \lambda_9 = 5$$

und die Rekursionsvorschrift

$$u_i = -\frac{1}{2}u_{i-1} + \frac{3}{4}u_{i-2} + \frac{1}{4}u_{i-3} + \frac{5}{4}u_{i-4} - \frac{1}{2}u_{i-5} \quad \text{für } i = 5, \ldots, 8.$$

Im Falle $\operatorname{char} K = 2$ sehen die letzten drei Iterationen so aus:

Eingangsbedingungen	Aktionen
$n = 6 \quad u_6 = 1 \quad l = 3$ $\varphi = 1 - T - T^2$ $r = 2 \quad d' = 1 \quad t = 0 \quad \psi = 1$	$d := u_6 - u_5 - u_4 = 0$
$n = 7 \quad u_7 = 1 \quad l = 3$ $\varphi = 1 - T - T^2$ $r = 2 \quad d' = 1 \quad t = 0 \quad \psi = 1$	$d := u_7 - u_6 - u_5 = 1$ [!] $\eta = 1 - T - T^2 - T^5$ $m := 5$
$n = 8 \quad u_8 = 0 \quad l = 5$ $\varphi = 1 - T - T^2 - T^5$ $r = 7 \quad d' = 1 \quad t = 3 \quad \psi = 1 - T - T^2$	$d := u_8 - u_7 - u_6 - u_3 = 1$ $\eta := 1 - T^3 - T^5$

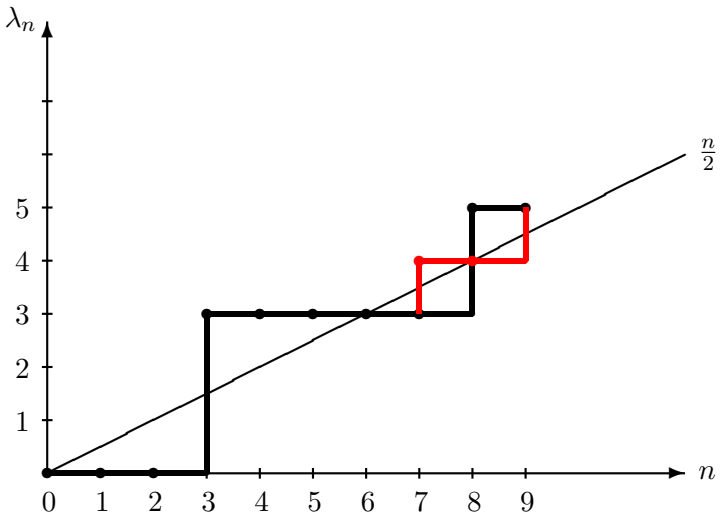
Die Folge der linearen Komplexitäten ist hier

$$\lambda_0 = 0, \lambda_1 = 0, \lambda_2 = 0, \lambda_3 = 3, \lambda_4 = 3, \lambda_5 = 3, \lambda_6 = 3, \lambda_7 = 3, \lambda_8 = 5, \lambda_9 = 5$$

und die Rekursionsvorschrift

$$u_i = u_{i-3} + u_{i-5} \quad \text{für } i = 5, \ldots, 8.$$

Die Entwicklung der linearen Komplexität wird auch noch grafisch dargestellt; die rote Linie kennzeichnet den Fall $\operatorname{char} K \neq 2$:



Der Aufwand für den Berlekamp-Massey-Algorithmus ist $O(N^2 \log N)$.

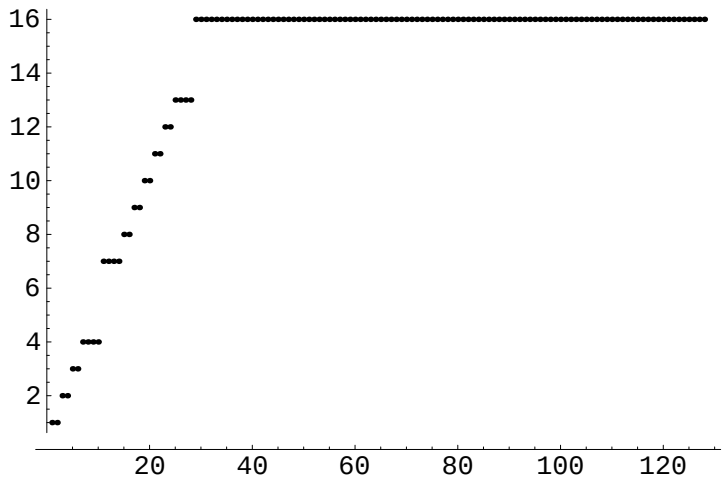
Die Folge $(\lambda_n)_{n\in\mathbb{N}}$ bzw. (für endliche Bitfolgen) $(\lambda_n)_{0\le n\le N}$ heißt das **Linearitätsprofil** der Bitfolge u.

Für die ersten 128 Bits der Folge, die in 1.10 von einem linearen Schieberegister erzeugt wurde, ist das Linearitätsprofil:

$$(0, 1, 1, 2, 2, 3, 3, 4, 4, 4, 4, 7, 7, 7, 7, 8, 8, 9, 9, 10, 10, 11, 11, 12,$$

$$12, 13, 13, 13, 13, 16, 16, 16, 16, \ldots),$$

und graphisch dargestellt sieht das so aus:

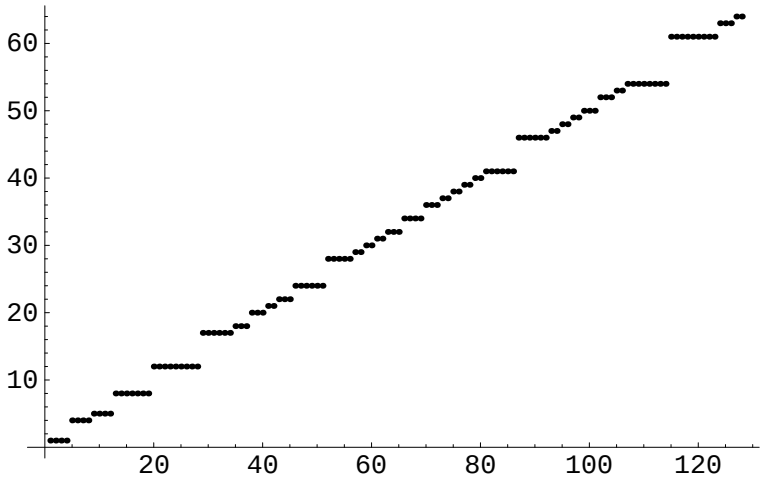


Für die ersten 128 Bits der Folge, die in 4.1 von einem „perfekten“ Zufallsgenerator erzeugt werden wird, ist das Linearitätsprofil:

$$(0, 1, 1, 1, 1, 4, 4, 4, 4, 5, 5, 5, 5, 8, 8, 8, 8, 8, 8, 8, 8, 12, 12, 12, 12,$$

$$12, 12, 12, 12, 12, 17, 17, 17, 17, 17, 17, 18, 18, 18, 20, 20, 20, 21, 21,$$
$$22, 22, 22, 24, 24, 24, 24, 24, 24, 28, 28, 28, 28, 28, 29, 29, 30, 30, 31,$$
$$31, 32, 32, 32, 34, 34, 34, 34, 36, 36, 36, 37, 37, 38, 38, 39, 39, 40, 40,$$
$$41, 41, 41, 41, 41, 41, 46, 46, 46, 46, 46, 46, 47, 47, 48, 48, 49, 49, 50,$$
$$50, 50, 52, 52, 52, 53, 53, 54, 54, 54, 54, 54, 54, 54, 54, 61, 61, 61, 61,$$
$$61, 61, 61, 61, 61, 63, 63, 63, 64, 64),$$

und das graphisch dargestellt sieht so aus:



Man sieht im zweiten Fall die unregelmäßige Schwankung um die Diagonale, wie es sich für eine „gute“ Zufallsfolge gehört. Im ersten Fall ist dieser Effekt auch vorhanden, aber nur, bis die lineare Komplexität der Folge erreicht ist.

3.4 Der Erwartungswert für die lineare Komplexität

... lässt sich exakt bestimmen (ohne Beweis):

Hauptsatz 1 (Rueppel) *Für den Mittelwert*

$$E_N = \frac{1}{2^N} \cdot \sum_{u \in \mathbb{F}_2^N} \lambda(u)$$

und die Varianz V_N der linearen Komplexität aller Bitfolgen der Länge N gilt:

$$\begin{aligned} E_N &= \frac{N}{2} + \frac{2}{9} + \frac{\varepsilon}{18} - \frac{N}{3 \cdot 2^N} - \frac{2}{9 \cdot 2^N} \approx \frac{N}{2}, \\ V_N &= \frac{86}{81} + \frac{14 - \varepsilon}{27} \cdot \frac{N}{2^N} + \frac{82 - 2\varepsilon}{81} \cdot \frac{1}{2^N} + \frac{9N^2 + 12N + 4}{81} \cdot \frac{1}{2^{2N}} \approx \frac{86}{81} \end{aligned}$$

mit $\varepsilon = 0$ für gerades, $\varepsilon = 1$ für ungerades N.

Bemerkenswert ist, dass die Varianz von N praktisch unabhängig ist.

3.5 Lineare Komplexität und TURING-Komplexität

Eine **universelle** TURING-**Maschine** kann jede andere TURING-Maschine durch ein geeignetes Programm simulieren. Sei **M** eine solche, und sei $u \in \mathbb{F}_2^n$ eine Bitfolge der Länge n. Dann ist die TURING-KOLMOGOROV-CHAITIN-**Komplexität** $\chi(u)$ gleich der Länge des kürzesten Programms von **M**, das u als Output produziert. Ein Programm der ungefähren Länge n gibt es immer: Es nimmt einfach u als Input-Folge und gibt diese wieder aus.

Anmerkung. Die Funktion $\chi : \mathbb{F}_2^* \longrightarrow \mathbb{N}$ ist selbst nicht berechenbar; d. h., es gibt keine TURING-Maschine, die χ berechnet. Daher ist die TURING-KOLMOGOROV-CHAITIN-Komplexität als Komplexitätsmaß kaum praktisch verwendbar. Sie ist allerdings in den letzten Jahren durch die Arbeiten von VITANYI und anderen in präzisierter Form wieder in Mode gekommen; siehe dazu etwa:

- MING LI, PAUL VITANYI: *An Introduction to Kolmogorov Complexity and Its Applications.* Springer-Verlag, New York 1993, 1997.

Eine wichtige Aussage der Theorie ist:

$$\frac{1}{2^n} \cdot \#\{u \in \mathbb{F}_2^n \mid \chi(u) > n \cdot (1-\varepsilon)\} > 1 - \frac{1}{2^{n\varepsilon - 1}},$$

d. h., fast alle Folgen haben eine TKC-Komplexität nahe am maximalen Wert, also keine wesentlich kürzere Beschreibung als durch vollständiges Hinschreiben. Eine gängige Interpretation dieses Sachverhaltes ist: „Fast alle Folgen sind zufällig." Dies entspricht der intuitiven Vorstellung von Zufall sehr gut. Eine Folge mit einer kurzen Beschreibung wie „eine Million Mal abwechselnd 0 und 1" wird nämlich niemand als auch nur im geringsten zufällig ansehen.

Das Komplexitätsmaß „lineare Komplexität" λ, das auf dem sehr speziellen Maschinenmodell des linearen Schieberegisters beruht, hat dagegen auf den ersten Blick gravierende Mängel. Die Folge „999999 Mal die 0, dann eine 1" hat eine sehr geringe TKC-Komplexität – und eine sehr geringe intuitive Zufälligkeit –, aber die lineare Komplexität 1 Million. Der Vorteil der linearen Komplexität ist, wie gesehen, ihre leichte explizite Bestimmbarkeit, und sie beschreibt „im allgemeinen" die Zufälligkeit ener Bitfolge doch recht gut. Diese Aussage lässt sich überraschend präzise fassen (ohne Beweis):

Satz 2 (BETH/DAI, EUROCRYPT 89)

$$\begin{aligned}
\frac{1}{2^n} \cdot \#\{u \in \mathbb{F}_2^n \mid (1-\varepsilon)\lambda(u) \le \chi(u)\} &\ge 1 - \frac{8}{3 \cdot 2^{\frac{n\varepsilon}{2-\varepsilon}}}, \\
\frac{1}{2^n} \cdot \#\{u \in \mathbb{F}_2^n \mid (1-\varepsilon)\chi(u) \le \lambda(u)\} &\ge 1 - \frac{1}{3} \cdot \frac{1}{2^{n\varepsilon - (1-\varepsilon)(1+\log n)+1}} - \frac{1}{3} \cdot \frac{1}{2^n}.
\end{aligned}$$

Interpretation: „Für fast alle Bitfolgen stimmen die lineare Komplexität und die TKC-Komplexität mit vernachlässigbarer Abweichung überein."

Damit ist klar, dass die lineare Komplexität trotz ihrer Einfachheit ein gutes Komplexitätsmaß ist, und dass Bitfolgen hoher linearer Komplexität im allgemeinen auch mit anderen Ansätzen nicht kürzer erklärbar sind. Sie sind also kryptographisch brauchbar. (Ein anderer Ansatz wäre etwa, die lineare Komplexität auf andere endliche Körper oder Restklassenringe zu verallgemeinern – das würde also dem Kryptoanalytiker kaum nützen.) Jedes effiziente Vorhersageverfahren im Sinne der Kryptoanalyse von Bitstromchiffren wäre auch eine Kurzbeschreibung im Sinne der TKC-Komplexität, so dass man als Fazit festhalten kann: *Bitfolgen hoher linearer Komplexität sind im allgemeinen nicht vorhersagbar.*

Natürlich gibt es auch Ansätze, nichtlineare Schieberegister zur Beurteilung der Komplexität heranzuziehen, siehe etwa:

- Agnes Hui Chan, Richard A. Games: On the quadratic span of periodic sequences. Crypto 89, 82–89.
- Cees J. A. Jansen, Dick E. Boekee: The shortest feedback shift register that can generate a given sequence. Crypto 89, 90–96.

3.6 Nichtlinearität für Schieberegister – Ansätze

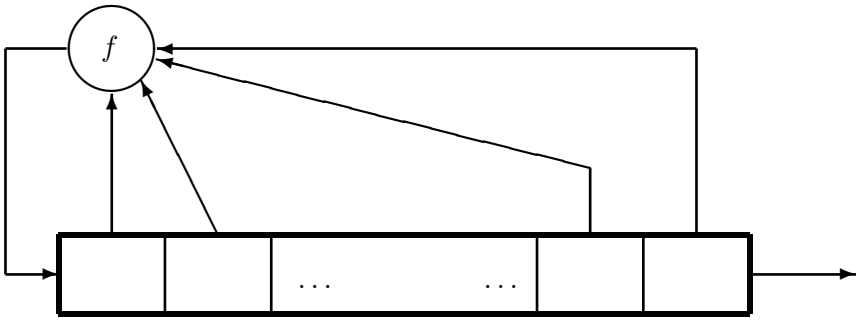
Lineare Schieberegister sind beliebt – vor allem bei Elektro-Ingenieuren und beim Militär – denn sie sind

- sehr einfach zu realisieren,
- extrem effizient, vor allem in Hardware,
- als Zufallsgeneratoren für statistische Zwecke sehr gut geeignet,
- problemlos parallel zu betreiben,
- aber leider kryptologisch völlig unsicher.

Um die positiven Eigenschaften zu nutzen und die kryptologische Schwäche zu vermeiden, gibt es verschiedene Ansätze.

Ansatz 1: Nichtlineare Rückkopplung

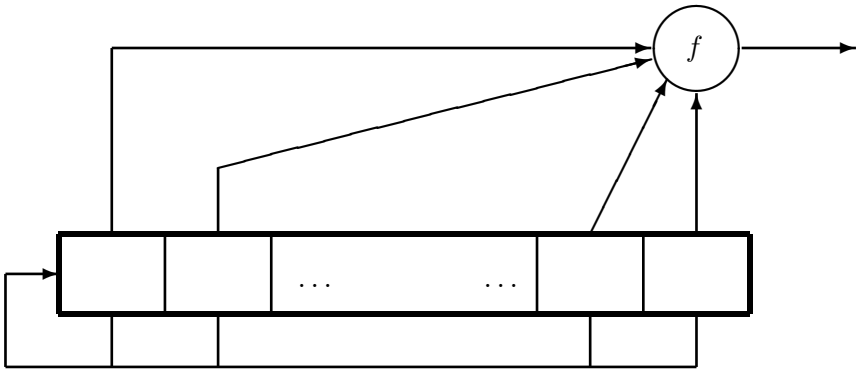
Die nichtlineare Rückkopplung (nonlinear feedback) folgt dem Schema:



Sie wurde schon in Abschnitt 2.6 behandelt und dort als kryptographisch nicht hinreichend sicher eingestuft.

Ansatz 2: Nichtlinearer Ausgabefilter

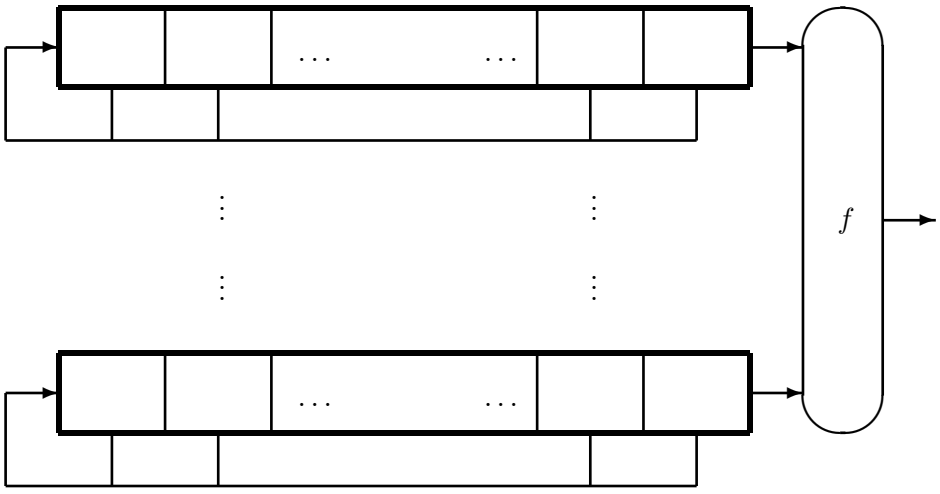
Der nichtlineare Ausgabefilter (nonlinear feed forward) folgt dem Schema:



Das Schieberegister selbst ist linear. Der nichtlineare Ausgabefilter ist ein Spezialfall des nächsten Ansatzes. (**Übungsaufgabe:** Wie?)

Ansatz 3: Nichtlinearer Kombinierer

Hier wird eine „Batterie“ aus n linearen Schieberegistern – die durchaus unterschiedliche Länge haben können und sollen – parallel betrieben. Ihre Outputfolgen werden in eine BOOLEsche Funktion $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2$ gefüttert:



Die ausführliche Diskussion dieses Verfahrens folgt in Abschnitt 3.7 ff. Natürlich kann man auch allgemeiner eine BOOLEsche Abbildung $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^q$ verwenden, die in jedem Takt q Bits ausgibt.

Ansatz 4: Auswahlsteuerung/Dezimierung/Taktung

Weitere Möglichkeiten bestehen in verschiedenen Ansätzen zur Steuerung einer Batterie von n parallel betriebenen linearen Schieberegistern

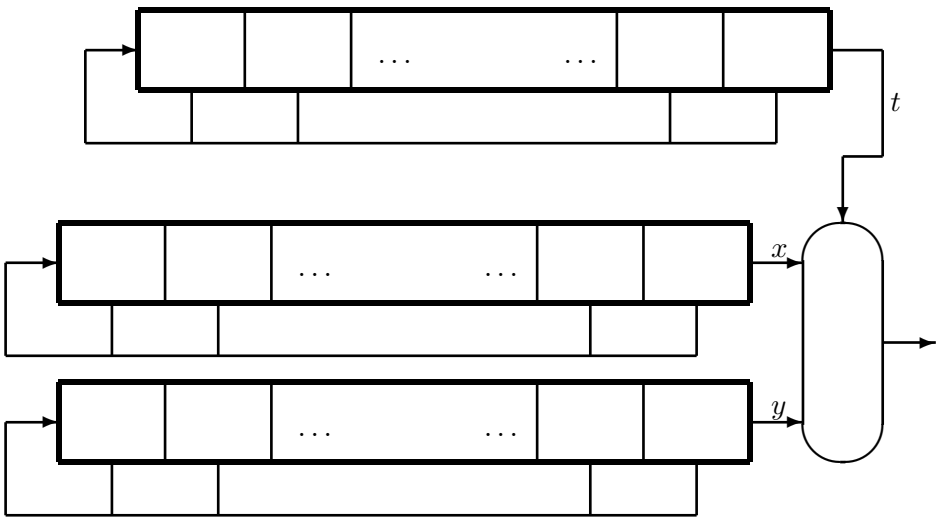
durch ein weiteres lineares Schieberegister:

- Bei der **Auswahlsteuerung** wird je nach Zustand des „Hilfsregisters“ das aktuelle Output-Bit von genau einem der „Batterie-Register“ als Output des Zufallsgenerators ausgewählt. Allgemeiner kann man auch eine Auswahl „r aus n“ treffen.
- Bei der **Dezimierung** nimmt man im allgemeinen $n = 1$ an und gibt das Output-Bit des einen Batterie-Registers nur dann aus, wenn das Hilfsregister einen bestimmten Zustand hat. Diese Art der Dezimierung kann man natürlich analog auf jede Bitfolge anwenden.
- Bei der **Taktung** gibt der Zustand des Hilfsregisters an, welche der Batterie-Register im aktuellen Taktzyklus weitergeschoben werden (und um wieviele Positionen) und welche in ihrem momentanen Zustand bleiben. Das ist vergleichbar mit der Steuerlogik von Rotor-Maschinen.

Diese Ansätze lassen sich oft bequem auch als nichtlineare Kombinierer schreiben, so dass Ansatz 3 als *der* Ansatz zur Rettung der linearen Schieberegister angesehen werden kann.

Beispiel: Der Geffe-Generator

Das einfachste Beispiel für die Auswahlsteuerung ist der Geffe-Generator, der durch das folgende Schema beschrieben wird:



Die Ausgabe ist x, wenn $t = 0$, und y, wenn $t = 1$. Das kann man so als Formel ausdrücken:

$$\begin{aligned} u &= \begin{cases} x, & \text{wenn } t = 0, \\ y, & \text{wenn } t = 1 \end{cases} \\ &= (1-t)x + ty = x + tx + ty. \end{aligned}$$

Also lässt sich der GEFFE-Generator auch durch einen nichtlinearen Kombinierer mit einer BOOLEschen Funktion $f\colon \mathbb{F}_2^3 \longrightarrow \mathbb{F}_2$ vom Grad 2 beschreiben.

3.7 Nichtlineare Kombinierer

Ein nichtlinearer Kombinierer wird beschrieben durch eine BOOLEsche Funktion $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2$ sowie eine Batterie aus n linearen Schieberegistern der Längen $l_1, \ldots, l_n$.

Bemerkungen

1. Sind die Schieberegister und f bekannt, so besteht ein Schlüssel der zugehörigen Bitstrom-Chiffre aus dem n-Tupel der Startvektoren, also hat der Schlüsselraum die Größe $2^{l_1} \cdots 2^{l_n}$.

2. Die lineare Komplexität der erzeugten Bitfolge ist „im allgemeinen“ $f(l_1, \ldots, l_n)$, wobei f in algebraischer Normalform geschrieben und als Polynom $f \in \mathbb{Z}[X]$ ausgewertet wird. Insbesondere ist ein möglichst hoher Grad von f erstrebenswert.

Beispiele

1. Der GEFFE-Generator ließ sich als nichtlinearer Kombinierer mit $f(x, t, y) = x + tx + ty$ deuten. Er hat die sehr große Periode $(2^{l_1} - 1)(2^{l_2} - 1)(2^{l_3} - 1)$, einen Schlüsselraum der Größe $2^{l_1+l_2+l_3}$, sowie die recht beachtliche lineare Komplexität $l_1 + l_1 l_2 + l_2 l_3$.

2. Nimmt man bei n beliebigen „Batterie-Registern“ als Kombinier-Funktion $f = T_1 \cdots T_n \in \mathbb{F}_2[T]$, also einfach die Multiplikation in $\mathbb{F}_2$, so hat die erzeugte Folge die lineare Komplexität $\leq l_1 \cdots l_n = f(l_1, \ldots, l_n)$. Hinreichend für die Gleichheit ist:

   (a) Alle charakteristischen Polynome der Batterie-Register sind irreduzibel,

   (b) die Längen $l_1, \ldots, l_n$ sind paarweise teilerfremd.

3.8 Korrelationsattacken – die Achillesferse der Kombinierer

Sie $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2$ die Kombinierfunktion eines nichtlinearen Kombinierers. Die Anzahl

$$K_f := \#\{x = (x_1, \ldots, x_n) \in \mathbb{F}_2^n \mid f(x) = x_1\}$$

misst, wie oft der Funktionswert mit dem ersten Argument übereinstimmt. Ist sie $> 2^{n-1}$, so ist die Wahrscheinlichkeit für diese Übereinstimmung

$$p = \frac{1}{2^n} \cdot K_f > \frac{1}{2},$$

also überdurchschnittlich. Die kombinierte Outputfolge „korreliert“ also stärker mit dem Output des ersten linearen Schieberegisters, als zufällig zu erwarten wäre.

Diesen Effekt kann sich der Kryptoanalytiker bei einem Angriff mit bekanntem Klartext zunutze machen: Die (ersten) Schlüsselbits $b_0, \ldots, b_{r-1}$ seien bekannt. Mit einer Exhaustion über die 2^{l_1} Startvektoren des ersten Registers erzeugt man jedesmal die Folge $u_0, \ldots, u_{r-1}$ und zählt die Koinzidenzen. Zu erwarten ist

$$\frac{1}{2^r} \cdot \#\{i \mid u_i = b_i\} \approx \begin{cases} p & \text{beim richtigen Startvektor,} \\ \frac{1}{2} & \text{sonst.} \end{cases}$$

Falls r groß genug ist, kann man also den echten Startvektor des ersten Registers mit einem Aufwand $\sim 2^{l_1}$ bestimmen. Macht man dann mit den anderen Registern genauso weiter, gelingt die Identifikation des gesamten Schlüssels mit einem Aufwand $\sim 2^{l_1} + \cdots + 2^{l_n}$. Das ist zwar exponentiell, aber wesentlich geringer als der Aufwand $\sim 2^{l_1} \cdots 2^{l_n}$ für die naive vollständige Schlüsselsuche.

In der Sprache von Kapitel II haben wir hier die lineare Relation (T_1, T) für f ausgenutzt. Klar ist, dass man analog jede lineare Relation ausnutzen kann, um die Komplexität der vollständigen Schlüsselsuche zu reduzieren.

Beispiel: Geffe-Generator. Hier werden die Korrelationen durch die folgende Tabelle beschrieben:

x	0	0	0	0	1	1	1	1
t	0	0	1	1	0	0	1	1
y	0	1	0	1	0	1	0	1
$f(x,t,y)$	0	0	0	1	1	1	0	1

Als Wahrscheinlichkeit für die Übereinstimmung erhält man also

$$p = \begin{cases} \frac{3}{4} & \text{für das Register 1,} \\ \frac{1}{2} & \text{für das Register 2 (Steuerung),} \\ \frac{3}{4} & \text{für das Register 3.} \end{cases}$$

Daher lassen sich bei einer Korrelationsattacke die Startwerte für die Register 1 und 3 – die Batterieregister – leicht schon aus kurzen Outputfolgen bestimmen; den Startwert für Register 2, das Steuerungsregister, findet man dann auch leicht durch Exhaustion seiner Startvektoren.

Aus der bisherigen Diskussion lassen sich als Design-Kriterien für nichtlineare Kombinierer herleiten:

- Die einzelnen Batterieregister müssen möglichst lang sein; genaueres siehe unten.
- Die Kombinierfunktion f muss balanciert sein
- ... und soll eine möglichst hohe Nichtlinearität, d. h., ein möglichst geringes lineares Potenzial haben.

Zum letzten Punkt: Wegen der nötigen Balanciertheit sind krumme Funktionen trotz ihrer „Korrelationsimmunität“ nicht geeignet. Die nächst beste Eigenschaft ist „resilient“ = unter den balancierten maximal nichtlinear.

Wie lang sollen die Batterieregister sein? Es gibt verschiedene Ansätze zu „schnellen“ Korrelationsattacken, z. B. mit Hilfe der Walsh-Transformation, besonders gegen dünn besetzte Rückkopplungspolynome. Diese reduzieren zwar nicht die Komplexitätsklasse des Angriffs, aber der Aufwand wird um einen beträchtlichen Proportionalitätsfaktor verringert. Auf diese Weise werden Register angreifbar, die bis zu 100 Koeffizienten 1 im Rückkopplungspolynom haben. Folgerung:

- Die einzelnen linearen Schieberegister sollten mindestens 200 Bits lang sein und eine „dicht besetzte“ Rückkopplung besitzen.

Ein eleganter Ausweg, der die Korrelationsattacke zusammenbrechen lässt, wurde von Rueppel vorgeschlagen: eine zeitabhängige Kombinierfunktion, also eine Familie $(f_t)_{t\in\mathbb{N}}$ zu verwenden.

Als Fazit kann man festhalten: Mit linearen Schieberegistern und nichtlinearen Kombinierern lassen sich sich ziemlich effiziente Pseudozufallsgeneratoren aufbauen. Für deren kryptologische Sicherheit gibt es zwar keine umfassende befriedigende Theorie, aber durchaus eine Absicherung, die – ähnlich wie bei Bitblock-Chiffren – auf der Theorie der Nichtlinearität Boolescher Funktionen beruht.

4 Perfekte Zufallsgeneratoren

Anfang der 80-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 2 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 2: 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

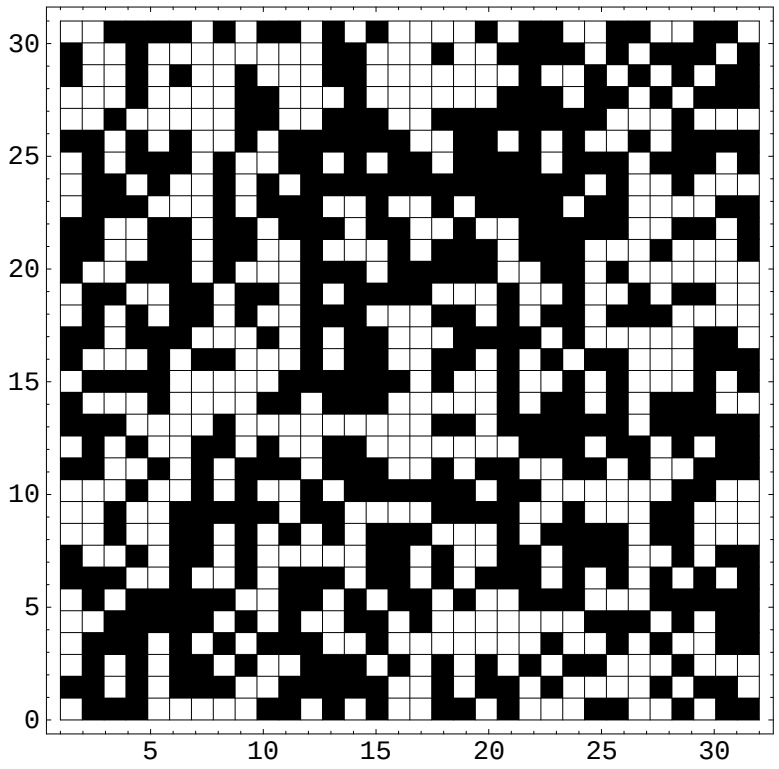
[Erzeugt eine Folge von n Pseudozufalls-Bits.]
Eingabeparameter:
n = Zahl der gewünschten Bits.
Ausgabeparameter:
blist = eine Liste von Bits.
Anweisungen:
Für $i = 1, \ldots, n$
ersetze x durch $x^2 \bmod m$,
setze $b = x \bmod 2$,
hänge b an blist an.

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 3 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 3: 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 $A \subseteq \mathbb{F}_2^n$, $r = 2s+1$ ungerade, und C berechne die* Boolesche *Funktion f auf A mit ε-Vorteil.*

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 r 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 von 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.

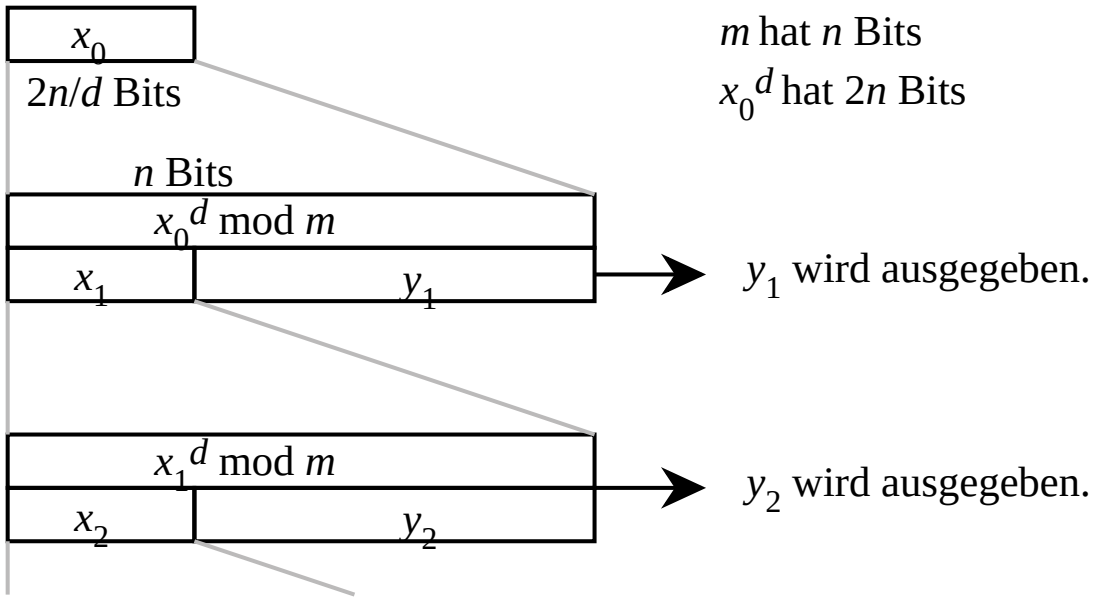
Der Index-Generator (Blum/Micali). Man wählt als Modul zufällig eine große Primzahl p und bestimmt dazu eine Primitivwurzel a. Ferner wählt man einen zufälligen Startwert $x = x_0$, teilerfremd zu $p-1$. Dann bildet man $x_i = a^{x_{i-1}} \bmod p$ und gibt das erste oder die ersten $\lfloor {}^2\log {}^2\log p \rfloor$ 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

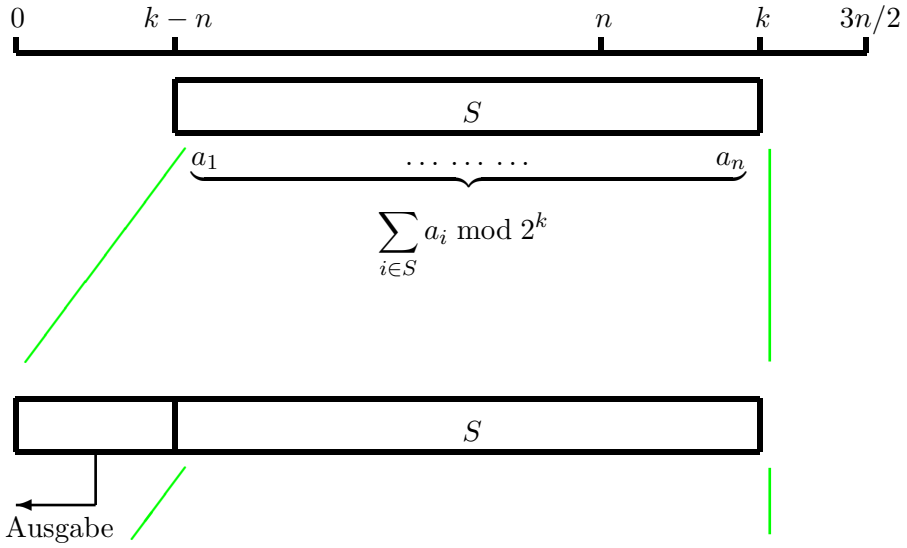
$$\sum_{i \in S} a_i = T.$$

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.