7 Komplexitätstheorie in der Kryptologie

Es gibt (mindestens) drei Gründe, warum die „gewöhnliche“ Komplexitätstheorie (TURING-Maschinen) für die Kryptologie nicht ausreicht:

1. Die Komplexitätstheorie behandelt die Frage, ob der *schlechteste* Fall ('worst case') hart ist (d. h., nicht effizient berechenbar). Um eine effiziente Kryptoanalyse auszuschließen, muss der *Normalfall* hart sein.

   Insbesondere reicht die Aussage $\mathbf{P} \stackrel{?}{\neq} \mathbf{NP}$ *nicht* für die Existenz starker kryptographischer Basisfunktionen aus.

   Beispiele für ein solches Phänomen aus anderen Bereichen sind der NEWTON-Algorithmus zur Bestimmung von Polynom-Nullstellen und die Simplex-Methode zur linearen Optimierung, die im schlechtesten Fall hart, im Normalfall aber sehr effizient sind.

2. Man muss auch probabilistische Algorithmen zulassen, wie bei den zahlentheoretischen Problemen schon einigemale gesehen („Monte-Carlo“-Algorithmen, die sehr effizient sind, aber nicht immer das richtige Ergebnis liefern).

   Die exakte mathematische Behandlung verwendet stochastische Begriffe: Teile des Inputs enstammen einem Wahrscheinlichkeitsraum Ω; es werden Aussagen über die Verteilung des Outputs gemacht.

3. Der Kryptoanalytiker kann seine Methode an das konkrete Problem adaptieren; er benötigt nicht unbedingt einen universellen Algorithmus, der für *alle* Instanzen seines Problems effizient ist. Z. B. kann er je nach Länge des Schlüssels einen anderen Algorithmus zum Brechen der Chiffre wählen.

TURING-Maschinen reichen zur Formalisierung der in der Kryptologie nötigen Komplexitätstheorie also nicht. Man kann das beheben, indem man Familien von TURING-Maschinen zulässt – eine für jede Input-Länge – und auch probabilistischen Input einbezieht. Einfacher zu beschreiben und eleganter anzuwenden ist allerdings ein Maschinenmodell, das auf BOOLEschen Schaltnetzen (englisch: circuits) beruht: probabilistische polynomiale Schaltnetzfamilien (PPS). – Die Umsetzung der gängigen Algorithmen in Schaltnetze ist deutlich einfacher und intuitiver als die Programmierung einer TURING-Maschine.

7.1 Schaltnetze

Eine formale Beschreibung von „Algorithmen“ kann man wie gesehen mit Hilfe von TURING-Maschinen geben. Ein einfacher zu definierendes, verstehendes und handzuhabendes Konzept ist das BOOLEsche **Schaltnetz**, das die in einem Algorithmus auszuführenden Bitoperationen in Form eines Ablaufdiagramms darstellt. Es wird auf zwei Weisen verallgemeinert:

probabilistisches Schaltnetz – damit werden probabilistische Algorithmen formalisiert,

polynomiale Schaltnetzfamilie – mit deren Hilfe kann die Komplexität eines Algorithmus bei wachsender Größe der Eingabedaten ausgedrückt werden.

Man kommt so zum Begriff der Schaltnetzkomplexität, der 1949 von SHANNON eingeführt wurde und für die Anwendung in der Kryptologie besonders gut geeignet ist.

Ein **Schaltnetz** ist ein azyklischer gerichteter markierter Graph, dessen Knoten sämtlich den Innengrad 0 oder 2 haben.

> Das heisst, Knoten sind mit gerichteten Kanten (Pfeilen) verbunden, wobei kein geschlossener Weg entsteht, alle Knoten sind mit Markierungen (Attributen) versehen, in jedem Knoten enden 0 oder 2 Pfeile.

Ein **Eingang** ist ein Knoten mit Innengrad 0. Ein **Ausgang** ist ein Knoten mit Außengrad 0; von ihm gehen also keine weiteren Pfeile aus. Alle Knoten sind mit einem der Symbole $\oplus$ oder $\otimes$ markiert – das entspricht der Addition und Multiplikation zweier Bits im Körper $\mathbb{F}_2$. Einige Eingänge sind als konstant gekennzeichnet; an ihnen liegt stets ein festes Bit 0 oder 1 an. (Da die Außengrade unbeschränkt sind, würden im Prinzip sogar zwei konstante Eingänge reichen, aber dann braucht man unelegant viele zusätzliche Pfeile.) Eine **Eingabe** ist eine Belegung der r nichtkonstanten Eingänge mit einer Bitfolge $x = (x_1, \ldots, x_r) \in \mathbb{F}_2^r$. An den inneren Knoten („Gattern“) werden die anliegenden Bits dann jeweils $\bmod 2$ addiert oder multipliziert, je nach Markierung des Knotens. Am Ausgang ensteht nach Durchlaufen des gesamten Schaltnetzes die **Ausgabe** $y \in \mathbb{F}_2^s$. Das Schaltnetz definiert also eine Funktion

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

und gibt deren algorithmische Zerlegung in Bitoperationen wieder. Durch ein Schaltnetz ausdrücken lässt sich jede solche Funktion, die nur durch Addition und Multiplikation gebildet werden kann, also jedes Polynom, also nach II (Einschub über BOOLEsche Abbildungen) jede Abbildung $\mathbb{F}_2^r \longrightarrow \mathbb{F}_2^s$. Im Fall $r = 2$, $s = 1$ gibt es

$$(\#\mathbb{F}_2)^{\#(\mathbb{F}_2 \times \mathbb{F}_2)} = 2^4 = 16$$

derartige Funktionen. Sie sind alle in Tabelle 1 aufgezählt, aus der man auch leicht die entsprechenden Schaltnetze entnehmen kann; sie werden als Polynome $a+bX+cY+dXY \in \mathbb{F}_2[X,Y]$ beschrieben (algebraische Normalform). Allgemein werden Schaltnetz und zugehörige Funktion, da Verwirrung nicht zu befürchten ist, mit dem gleichen Buchstaben bezeichnet.

a	b	c	d	Polynom (ANF)	logische Operation	
0	0	0	0	0	FALSE	Konstante
1	0	0	0	1	TRUE	Konstante
0	1	0	0	X	X	Projektion
1	1	0	0	$1+X$	$\neg X$	negierte Projektion
0	0	1	0	Y	Y	Projektion
1	0	1	0	$1+Y$	$\neg Y$	negierte Projektion
0	1	1	0	$X+Y$	$X \oplus Y$	XOR
1	1	1	0	$1+X+Y$	$X \Longleftrightarrow Y$	Äquivalenz
0	0	0	1	XY	$X \wedge Y$	AND
1	0	0	1	$1+XY$	$\neg(X \wedge Y)$	NAND
0	1	0	1	$X+XY$	$X \wedge (\neg Y)$	
1	1	0	1	$1+X+XY$	$X \Longrightarrow Y$	Implikation
0	0	1	1	$Y+XY$	$(\neg X) \wedge Y$	
1	0	1	1	$1+Y+XY$	$X \Longleftarrow Y$	Implikation
0	1	1	1	$X+Y+XY$	$X \vee Y$	OR
1	1	1	1	$1+X+Y+XY$	$\neg(X \vee Y)$	NOR

Tabelle 1: Die 16 zweistelligen Bitoperationen

7.2 Schaltnetze für grundlegende Operationen

Natürlich wird bei Komplexitätsbestimmungen nicht jeder Algorithmus bis auf Schaltnetzebene heruntergebrochen. Statt dessen bestimmt man ein-für allemal die Schaltnetz-Komplexität von (z. B. arithmetischen) Grundoperationen und Basisalgorithmen und führt kompliziertere Algorithmen darauf zurück.

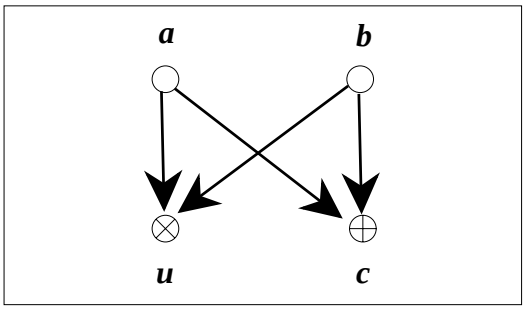

Abbildung 1: Schaltnetz zur Addition zweier Einbit-Zahlen

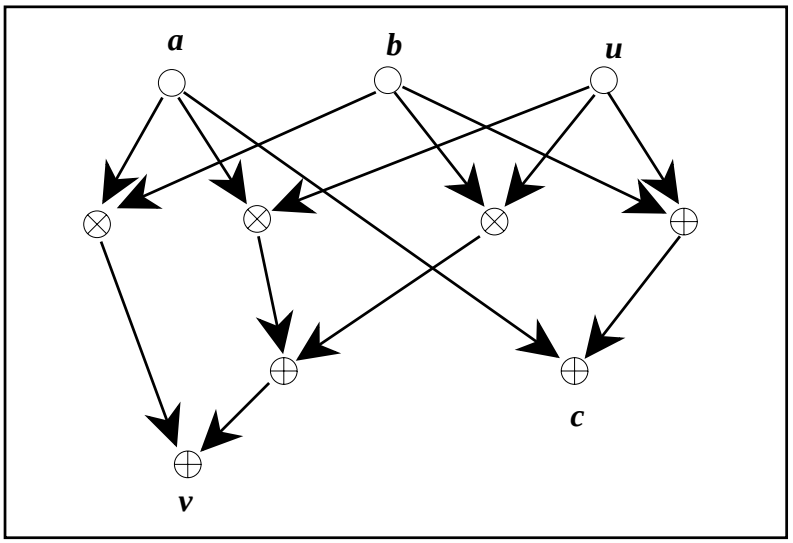

Abbildung 2: Schaltnetz zur Addition dreier Einbit-Zahlen

1. Die Addition zweier Bits a, b mit mod 2-Summe c und Übertrag u, also die „primitive“ Addition in $\mathbb{N}$ von Ziffern zur Basis 2, ist eine Funktion $C : \mathbb{F}_2^2 \longrightarrow \mathbb{F}_2^2$. Ein passendes Schaltnetz ist in Abbildung 1 wiedergegeben.

2. Die analoge Addition dreier Bits a, b, u mit mod 2-Summe c und Übertrag v ist der Grundbaustein, mit dem die Addition beliebiger Ganzzahlen zur Basis 2 aufgebaut wird. Sie wird durch eine Funktion $C : \mathbb{F}_2^3 \longrightarrow \mathbb{F}_2^2$ mit dem Schaltnetz aus Abbildung 2 beschrieben. Eine Formel für v ist nämlich:

$$v = (a \otimes b) \oplus (a \otimes u) \oplus (b \otimes u) = \begin{cases} 1, & \text{wenn mindestens 2 Eingaben 1 sind,} \\ 0 & \text{sonst.} \end{cases}$$

3. Die Addition einer Einbit-Zahl zu einer s-Bit-Zahl verläuft nach einem ähnlichen, umfangreicheren Schema und lässt sich durch ein Schaltnetz mit $3s + 1$ Knoten, davon $s + 1$ Ausgängen, erledigen.

4. Die Multiplikation zweier Bits ist trivial. Interessanter ist es, die Multiplikation zweier Zweibit-Zahlen $2a_1 + a_0$ und $2b_1 + b_0$ mit Vierbit-Ergebnis $8c_3 + 4c_2 + 2c_1 + c_0$ durch ein Schaltnetz zu beschreiben. Der klassische Algorithmus ergibt die Formeln

$$\begin{array}{llll} c_0 = a_0 \otimes b_0, & & & \\ t_1 = a_0 \otimes b_1, & t_2 = a_1 \otimes b_0, & c_1 = t_2 \oplus t_1, & u_1 = t_2 \otimes t_1, \\ t_3 = a_1 \otimes b_1, & c_2 = t_3 \oplus u_1, & c_3 = t_3 \otimes u_1 & \end{array}$$

   mit Hilfsbits t_1, t_2, t_3 und u_1. Sie lassen sich in das Schaltnetz aus Abbildung 3 übersetzen.

5. Auch logische Operationen und Verzweigungen kann man auf Additionen und Multiplikationen in $\mathbb{F}_2$ reduzieren. Für die logischen Operationen sieht man das an der Tabelle 1. Eine Verzweigung wird etwa so beschrieben: Gegeben seien drei Schaltnetze $C_0, C_1 : \mathbb{F}_2^r \longrightarrow \mathbb{F}_2^s$ und $E : \mathbb{F}_2^r \longrightarrow \mathbb{F}_2$. Gesucht ist ein Schaltnetz $C : \mathbb{F}_2^r \longrightarrow \mathbb{F}_2^s$ mit

$$C(x) = \begin{cases} C_0(x), & \text{falls } E(x) = 0, \\ C_1(x), & \text{falls } E(x) = 1. \end{cases}$$

   Dieses konstruiert man mit Hilfe der Formeln

$$\begin{aligned} y \otimes v &= \begin{cases} 0, & \text{falls } y = 0, \\ v, & \text{falls } y = 1. \end{cases} \\ (\neg y) \otimes u &= \begin{cases} u, & \text{falls } y = 0, \\ 0, & \text{falls } y = 1. \end{cases} \end{aligned}$$

$$[y \otimes v] \oplus [(1 \oplus y) \otimes u] = \begin{cases} u, & \text{falls } y = 0, \\ v, & \text{falls } y = 1. \end{cases}$$

Setzt man hier C_0, C_1 und E ein, so erhält man als Lösung

$$C(x) = [E(x) \otimes C_1(x)] \oplus [(1 \oplus E(x)) \otimes C_0(x)].$$

Dieses Schaltnetz ist in Abbildung 4 dargestellt. Es hat auch einen konstanten Eingang.

6. Der Vergleich $x \geq y$ zweier Bits ist eine der 16 zweistelligen Bitoperationen, nämlich $1 \oplus y \oplus x \otimes y$. Allgemeiner lässt sich ein Vergleich von $n+1$-Bit-Zahlen $x = (x_n \dots x_0)$ und $y = (y_n \dots y_0)$ so zusammenbasteln:

$$C(x,y) = \begin{cases} 1, & \text{wenn } x_n > y_n \\ & \text{oder } x_n = y_n \text{ und } x' \geq y', \\ 0, & \text{wenn } x_n = y_n \text{ und } x' < y' \\ & \text{oder } x_n < y_n \end{cases}$$

mit $x' = (x_{n-1} \dots x_0)$ und $y' = (y_{n-1} \dots y_0)$.

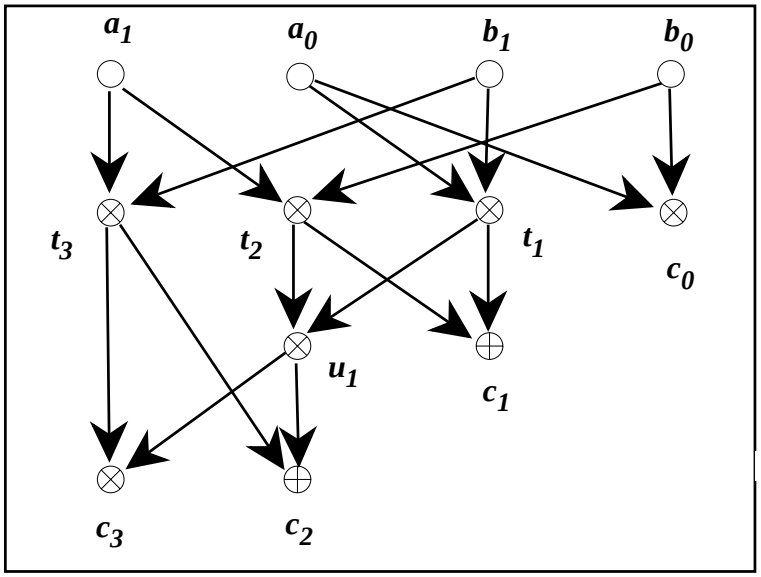

Abbildung 3: Schaltnetz zur Multiplikation zweier Zweibit-Zahlen

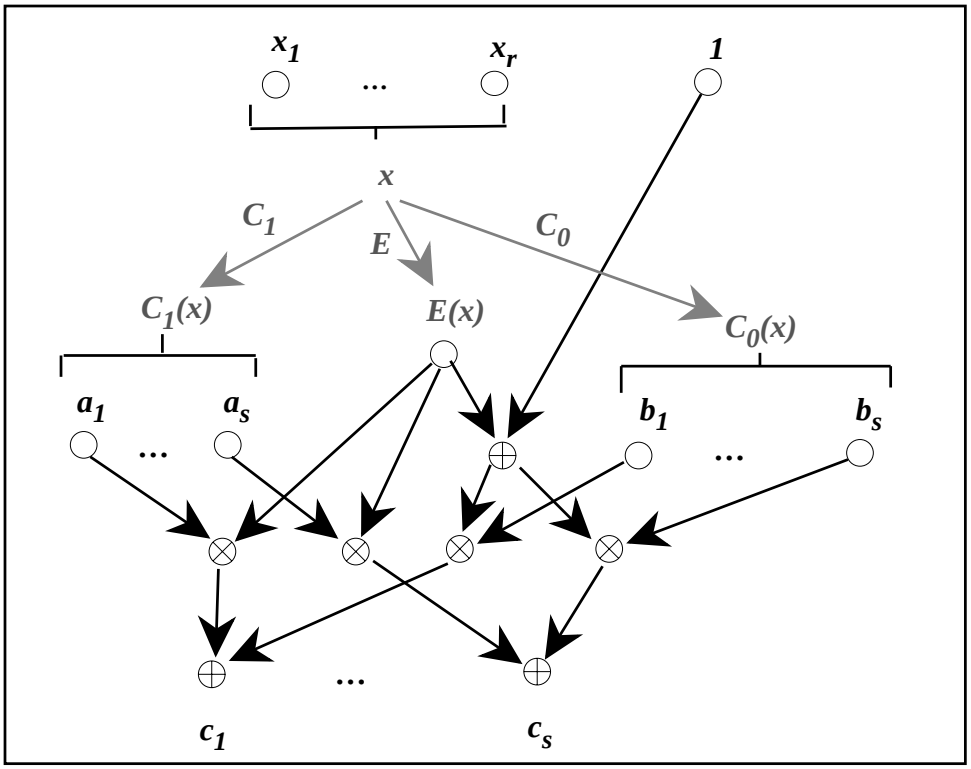

Abbildung 4: Schaltnetz für eine Verzweigung

7.3 Komplexitätsmaße für Schaltnetze

Es ist intuitiv klar und auch leicht zu zeigen, dass sich jeder Algorithmus durch ein Schaltnetz beschreiben lässt; genau nach diesem Prinzip der bitweisen Operationen arbeiten ja Computer auf der untersten Ebene – das Schaltnetz ist also ein universelles Maschinenmodell. Klar ist auch, dass man mit einem solchen simplen Modell nicht direkt genaue Aufwandsberechnungen für reale Computer durchführen kann; wohl aber ist die Unterscheidung, ob ein Algorithmus effizient, also mit polynomialem Zeitaufwand, arbeitet oder nicht, damit sehr gut möglich. Dazu betrachtet man die beiden Werte $\#C$, die **Größe** des Schaltnetzes, also die Anzahl der Knoten, und $t(C)$, die **Tiefe**, also die größte Weglänge innerhalb des Schaltnetzes. Die Größe ist ein Komplexitätsmaß für die serielle Abarbeitung, die Tiefe eines für die unbeschränkt parallele Abarbeitung des Algorithmus.

Beispiele

1. Das Schaltnetz für die Addition zweier Bits, Abbildung 1 hat die Größe 4 und die Tiefe 1.

2. Das Schaltnetz für die Addition dreier Bits, Abbildung 2 hat die Größe 10 und die Tiefe 3.

3. Aus dem obigen Beispiel 3 erhält man ein einfaches (längst nicht optimales) Schaltnetz zur Addition von s Einbit-Zahlen mit $A(s)$ Knoten, darunter $a(s)$ Ausgängen. Nach Beispiel 1 ist $A(2) = 4$, $a(2) = 2$, nach Beispiel 3 gilt die Rekursion $a(s) = a(s-1) + 1$ und $A(s) = A(s-1) + 2a(s-1) + 1$. Daraus folgt $a(s) = s$ und $A(s) = s^2$ durch Induktion.

4. Das Schaltnetz für die Multiplikation zweier Zweibitzahlen, Abbildung 3 hat die Größe 12 und die Tiefe 3.

5. Beim Schaltnetz für die Verzweigung, Abbildung 4 gelten die Formeln

$$\begin{aligned} \#C &= \#C_1 + \#C_2 + \#E - 2r + 3s + 2, \\ t(C) &= \max\{t(C_0) + 2, t(C_1) + 2, t(E) + 3\}. \end{aligned}$$

6. Für die Größe $V(n)$ des Schaltnetzes zum Vergleich in Beispiel 6 gilt die Rekursionsformel $V(n) = V(n-1)+11$ mit V(2) = 6, also $V(n) = 11n - 5$.

7. Die Aufwandsabschätzungen für die Ganzzahl-Operationen mit der Basis $B = 2$ des Zahlensystems besagen, dass Paare von n-Bitzahlen mit Schaltnetzen der Größe $\mathrm{O}(n)$ addiert und subtrahiert und mit Schaltnetzen der Größe $\mathrm{O}(n^2)$ multipliziert und dividiert werden können.

Anmerkungen

Bei der Definition von Schaltnetzen wird manchmal der Innengrad nicht durch 2 beschränkt. Diese Beschränkung ist aber durchaus sinnvoll, da reale Maschinen in jedem Schritt nur eine bestimmte Anzahl von Bits verarbeiten können. Ob man die Schranke auf 2 festsetzt oder höher, spielt für die folgenden Komplexitätsüberlegungen allerdings keine Rolle.

Zwischenergebnisse können in einem Schaltnetz wegen des unbeschränkten Außengrades beliebig oft verwendet werden; das entspricht einem unbeschränktem Speicher auf einem realen Computer. Ein engerer Begriff, bei dem dies vermieden wird, ist die BOOLEsche Formel: Eine solche ist ein Schaltnetz, dessen Graph ein Baum ist. Das bedeutet, dass der Außengrad jedes Knotens 1 ist (außer den Ausgängen mit Außengrad 0). Ein solches Schaltnetz entspricht genau dem Aufbau einer ausgeschriebenen Formel aus zweistelligen Operationen, wo ja auch jedes Zwischenergebnis nur einmal verwendet werden kann. Insbesondere muss jede Eingabe so oft wiederholt werden, wie sie gebraucht wird.

Bei einer weiteren allgemeineren Definition von Schaltnetzen werden auch beliebige zweistellige Bitoperationen statt nur $\oplus$ (XOR) und $\otimes$ (AND) zugelassen. Tabelle 1 zeigt auch, wie sich alle solchen durch $\oplus$ und $\otimes$ ausdrücken lassen. Diese allgemeinere Definition gestattet in der Regel etwas kürzere Schaltnetze, hat aber keine lohnenden Auswirkungen auf die Komplexitätsüberlegungen.

Schaltnetze (oder Algorithmen) dienen nicht nur zur Berechnung von Funktionen, sondern auch zum Finden von Lösungen, formalisiert durch das Erfüllen einer Relation. Seien $X \subseteq \mathbb{F}_2^r$ und $Y \subseteq \mathbb{F}_2^s$ zwei Mengen und E eine Relation auf $X \times Y$, beschrieben durch eine Funktion

$$E : X \times Y \longrightarrow \mathbb{F}_2.$$

Gefragt sind Algorithmen, die zu gegebenem $x \in X$ ein $y \in Y$ finden mit $E(x, y) = 1$. Den bisher behandelten Spezialfall einer auszuwertenden Funktion $f : X \longrightarrow Y$ findet man als rechtseindeutige Relation wieder – $E(x, y) = 1 \Longleftrightarrow y = f(x)$. Man erfasst aber auch das Lösen von Gleichungen, etwa von linearen Gleichungssystemen: $X = \mathbf{M}_{m,n}(\mathbb{F}_2)$ = die Menge der $m \times n$-Matrizen, $Y = \mathbb{F}_2^m \times \mathbb{F}_2^n$, $E(x, y) = 1 \Longleftrightarrow xy_1 = y_2$ (als Produkt Matrix $\times$ Vektor).

Ein Schaltnetz $C : \mathbb{F}_2^r \longrightarrow \mathbb{F}_2^s$ erfüllt E, wenn $C(X) \subseteq Y$ und $E(x, C(x)) = 1$ für alle $x \in X$.

Das O in der obigen Komplexitätsabschätzung (Beispiel 7) ist übrigens nur ein „Faulheits-O“; mit etwas mehr Mühe könnte man auch echte Abschätzungen angeben.

7.4 Probabilistische Schaltnetze

Nun zur Formalisierung probabilistischer Algorithmen. Das sieht im Ansatz etwa so aus: Gegeben sei eine Funktion

$$f\colon A \longrightarrow \mathbb{F}_2^s$$

auf einer Menge A. Ein probabilistischer Algorithmus über dem Wahrscheinlichkeitsraum Ω soll eine Funktion

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

definieren; er dient zur (probabilistischen) Berechnung von $f(x)$ bzw. f, wenn die Wahrscheinlickeit

$$P(\{\omega \mid C(x,\omega) = f(x)\}) \quad (\text{„lokal“}) \text{ bzw.}$$

$$P(\{(x,\omega) \mid C(x,\omega) = f(x)\}) \quad (\text{„global“})$$

genügend groß ist (signifikant $> \frac{1}{2^s}$). Im lokalen Fall wird bei festem x über Ω gemittelt, im globalen Fall auch über A. Dabei sollen im allgemeinen die Wahrscheinlichkeitsräume Ω und $A \times \Omega$ endlich und mit der Gleichverteilung versehen sein.

Um probabilistische Algorithmen beschreiben zu können, muss man Schaltnetze mit *drei* verschiedenen Typen von Eingängen betrachten:

- r **deterministische Eingänge**, die mit einer Eingabe $x \in \mathbb{F}_2^r$ – oder aus einer Teilmenge $A \subseteq \mathbb{F}_2^r$ – belegt werden,
- einige konstante Eingänge
- und k **probabilistische Eingänge**, die mit einem Element des Laplaceschen Wahrscheinlichkeitsraums $\Omega = \mathbb{F}_2^k$ belegt werden (k „Münzenwürfe“), oder mit einem Element eines Teilraums $\Omega \subseteq \mathbb{F}_2^k$. – Natürlich sind auch andere Wahrscheinlichkeitsverteilungen als die Gleichverteilung auf Ω möglich und manchmal sinnvoll.

Über die Ausgabe $y \in \mathbb{F}_2^s$ werden dann Wahrscheinlichkeitsaussagen der oben beschriebenen Art gemacht.

Beispiele

1. Die Suche nach einem Nicht-Quadratrest für einen Primzahlmodul p: Dabei sei p eine n-Bitzahl. Dazu wurde $b \in [1 \ldots p-1]$ zufällig gewählt und $(\frac{b}{p})$ (das Legendre-Symbol, das für Quadratreste 1, für Nichtquadratreste -1 ist) berechnet. Die Wahrscheinlichkeit für einen Erfolg ist dabei $\frac{1}{2}$, der Aufwand $O(n^2)$ (siehe Anhang A.9). Betrachten wir allgemeiner die Frage, ob in einem unabhängig gewählten h-Tupel

$(b_1, \ldots, b_h) \in \Omega = [1 \ldots p-1]^h$ ein Nicht-Quadratrest vorkommt. Es gibt (für das fest gewählte p) ein Schaltnetz ohne deterministische Eingänge (aber mit konstanten Eingängen zur Einspeisung von p),

$$C : \mathbb{F}_2^{hn} \longrightarrow \mathbb{F}_2^n,$$

$$C(\omega) = \begin{cases} b_i, & \text{das erste } b_i, \text{ das Nicht-Quadratrest ist,} \\ 0, & \text{falls kein Nicht-Quadratrest gefunden wurde,} \end{cases}$$

das die Größe $\mathrm{O}(hn^2)$ hat und mit der Wahrscheinlichkeit $1 - \frac{1}{2^h}$ einen Nichtquadratrest ausgibt.

2. Der strenge Pseudoprimzahltest: Hier entstammen die Eingaben der Menge A der ungeraden Zahlen in $[3 \ldots 2^n - 1]$. Berechnet werden soll die Funktion

$$f : A \longrightarrow \mathbb{F}_2, \quad f(m) = \begin{cases} 1, & \text{falls } m \text{ zusammengesetzt,} \\ 0, & \text{falls } m \text{ prim.} \end{cases}$$

Die zufälligen Eingänge werden durch die Wahl eines Basiselements $a \in \Omega = [2 \ldots 2^n - 1]$ besetzt. Der strenge Pseudoprimzahltest liefert ein Schaltnetz

$$C : \mathbb{F}_2^n \times \mathbb{F}_2^n \longrightarrow \mathbb{F}_2$$

der Größe $\mathrm{O}(n^3)$ mit dem Ergebnis 1, wenn m durchfällt (dann ist m sicher zusammengesetzt), oder 0, wenn m besteht (dann ist m möglicherweise prim).

Die Eigenschaft eines (probabilistischen) Schaltnetzes C mit r deterministischen Eingängen, eine Entscheidung mit einer Wahrscheinlichkeit richtig zu treffen, die signifikant vom zufälligen Erraten des Wertes $f(x) \in \mathbb{F}_2^s$ abweicht, wird so formalisiert: Ein Schaltnetz

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

hat einen **ε-Vorteil** mit $\varepsilon \geq 0$ bei der Berechnung von $f(x)$ bzw. f, wenn

$$P(\{\omega \in \Omega \mid C(x, \omega) = f(x)\}) \geq \frac{1}{2^s} + \varepsilon \quad (\text{„lokal“}) \text{ bzw.}$$

$$P(\{(x, \omega) \in A \times \Omega \mid C(x, \omega) = f(x)\}) \geq \frac{1}{2^s} + \varepsilon \quad (\text{„global“}).$$

Die Wahrscheinlichkeit bezüglich ω für das richtige Ergebnis wird also im globalen Fall noch über $x \in A$ gemittelt. Der Vorteil 0, also die Wahrscheinlichkeit $\frac{1}{2^s}$, entspricht dem puren Raten des Ergebnisses.

C hat eine **Irrtumswahrscheinlichkeit** ε bei der Berechnung von $f(x)$ bzw. f, wenn

$$P(\{\omega \in \Omega \mid C(x, \omega) = f(x)\}) \geq 1 - \varepsilon \quad \text{bzw.}$$

$$P(\{(x,\omega) \in A \times \Omega \mid C(x,\omega) = f(x)\}) \geq 1 - \varepsilon.$$

Beispiele

1. Bei der Suche nach einem Nicht-Quadratrest $\bmod p$ ist

   $$P(\{\omega \in \Omega \mid C(\omega) \text{ ist Nichtquadratrest}\}) = 1 - \frac{1}{2^h}.$$

   Das Schaltnetz hat also einen $(\frac{1}{2} - \frac{1}{2^h})$-Vorteil und eine Irrtumswahrscheinlichkeit von $\frac{1}{2^h}$.

2. Beim strengen Pseudoprimzahltest ist für festes m

   $$P(\{\omega \in \Omega \mid C(m,\omega) = f(m)\}) \begin{cases} \geq \frac{3}{4}, & \text{wenn } m \text{ zusammengestzt,} \\ = 1, & \text{wenn } m \text{ prim.} \end{cases}$$

   Das ergibt über alle m gemittelt

   $$P(\{(m,\omega) \in A \times \Omega \mid C(m,\omega) = f(m\}) \geq \frac{3}{4},$$

   also einen $\frac{1}{4}$-Vorteil und eine Irrtumswahrscheinlichkeit $\frac{1}{4}$. (Da es wesentlich mehr zusammengesetzte Zahlen als Primzahlen gibt, wird bei der Mittelung über m der Wert $\frac{1}{4}$ nicht wesentlich erhöht.)

7.5 Polynomiale Schaltnetzfamilien

Ein Schaltnetz hat eine festgelegte Zahl von Eingängen, kann also (im Gegensatz zu einer TURING-Maschine) nur Eingaben bestimmter Länge verarbeiten. Bei Effizienzuntersuchungen will man aber meist das Wachstum des Aufwandes bei immer weiterer Vergrößerung der Eingabelänge abschätzen. Dazu betrachtet man eine ganze Familie $(C_n)_{n\in\mathbb{N}}$ von Schaltnetzen mit wachsender Zahl deterministischer Eingänge, deren Größe $\#C_n$ kontrolliert wächst; damit kann man dann das Wachstum des Aufwandes als Funktion der Länge der Eingabe ausdrücken. Genauer wird definiert: Eine **polynomiale (probabilistische) Schaltnetzfamilie (PPS)** ist eine Familie $C = (C_n)_{n\in\mathbb{N}}$,

$$C_n \colon \mathbb{F}_2^{r(n)} \times \mathbb{F}_2^{k(n)} \longrightarrow \mathbb{F}_2^{s(n)}, \tag{1}$$

von (probabilistischen) Schaltnetzen mit $r(n)$ deterministischen und $k(n)$ probabilistischen Eingängen, so dass es ein Polynom $\alpha \in \mathbb{N}[X]$ (nichtnegative ganzzahlige Koeffizienten) gibt mit $\#C_n \leq \alpha(n)$ für alle $n \in \mathbb{N}$. Insbesondere ist die Zahl der Eingänge aller Arten und die Zahl $s(n)$ der Ausgänge polynomial beschränkt.

Bemerkung. Das bedeutet nicht notwendig, dass die Funktionen r, k, s selbst Polynome sind.

Sind alle $k(n) = 0$, so spricht man selbstverständlich von einer deterministischen polynomialen Schaltnetzfamilie.

Durch dieses Berechnungsmodell (im deterministischen Fall) könnten mehr Probleme berechenbar sein als durch das gebräuchliche Modell der TURING-Maschinen (und sind es tatsächlich), da für jede Eingabelänge ein anderer Algorithmus gewählt werden kann. Man spricht daher auch von einem „nicht-gleichmäßigen Berechnungsmodell“. Das erscheint auf den ersten Blick vielleicht als Nachteil dieses Berechnungsmodells, ist aber für die Kryptoanalyse sogar besonders realistisch: Nach Wahl der Inputlänge n hat der Kryptoanalytiker die Möglichkeit, einen passenden Algorithmus zu wählen. Das heisst, Aussagen über Nichteffizienz, die unter diesem Berechnungsmodell bewiesen werden, erlauben, die Inputlänge als öffentlich bekannt anzunehmen.

Ist eine TURING-Berechnung in polynomialer Zeit möglich, so gibt es für das gleiche Problem auch eine polynomiale Schaltnetzfamilie. Die Umkehrung davon gilt nicht; es sind allerdings nur „künstliche“ Beispiele bekannt. Gäbe es für irgendein NP-vollständiges Problem eine polynomiale Schaltnetzfamilie, so für alle. Nichtgleichmäßige Komplexität kann man allerdings auch mit TURING-Maschinen modellieren, indem man für jede Eingabelänge eine andere TURING-Maschine zulässt. Analog kann man natürlich auch probabilistische TURING-Maschinen zulassen.

Ein Problem soll **hart** heissen, wenn es kein PPS gibt, das es mit signifikantem Vorteil löst. Die früher behandelten „harten zahlentheoretischen

Probleme“ sind vermutlich in diesem Sinne hart, z. B. die Primzerlegung; präzisiert wird das später.

Wir wissen bereits, dass die Grundoperationen für ganze Zahlen mit polynomialen Schaltnetzfamilien berechenbar sind (sogar deterministisch).

7.6 Effiziente Algorithmen

Um die Ergebnisse aus 7.4 übertragen zu können, müssen auch die Begriffe des Vorteils und der Irrtumswahrscheinlichkeit auf Familien verallgemeinert werden. Gegeben sei dazu eine Menge $L \subseteq \mathbb{F}_2^*$, also eine Sprache über dem binären Alphabet, wobei wie üblich $L_n := L \cap \mathbb{F}_2^n$ gesetzt wird, sowie eine Abbildung

$$f\colon L \longrightarrow \mathbb{F}_2^* \quad \text{mit } f(L_{r(n)}) \subseteq \mathbb{F}_2^{s(n)}, \tag{2}$$

wobei $r(n)$ die monoton wachsende Folge der Indizes i mit $L_i \neq \emptyset$ ist. Diese Abbildung soll mit einer PPS wie in (1) berechnet werden.

Beispiele

1. Die Funktion $f(x, y, z) := xy \bmod z$ für n-Bit-Zahlen x, y, z lässt sich mit einem (deterministischen) Schaltnetz

   $$C_n\colon \mathbb{F}_2^{3n} \longrightarrow \mathbb{F}_2^n$$

   der Größe $\#C_n = \mathrm{O}(n^3)$ (und Irrtumswahrscheinlichkeit 0) berechnen; hier ist $r(n) = 3n$ und $s(n) = n$.

2. Sei L die Menge der (binären Codierungen von) ungeraden Zahlen ≥ 3 und $f : L \longrightarrow \mathbb{F}_2$ der „Primzahlanzeiger“ wie in Abschnitt 7.4. Die dort vorgestellte PPS für den strengen Pseudoprimzahltest hat die Größe $\mathrm{O}(n^3)$ und konstanten Vorteil $\frac{1}{4}$ sowie konstante Irrtumswahrscheinlichkeit $\frac{1}{4}$. Mit t Basen kommt man auf die Größe $\mathrm{O}(tn^3)$ und die Irrtumswahrscheinlichkeit $\frac{1}{4^t}$.

Definition 1. Eine Funktion $\varphi\colon \mathbb{N} \longrightarrow \mathbb{R}_+$ heißt **vernachlässigbar**, wenn für jedes nichtkonstante Polynom $\eta \in \mathbb{N}[X]$ gilt

$$\varphi(n) \leq \frac{1}{\eta(n)} \quad \text{für fast alle } n \in \mathbb{N}.$$

Beispiel. Ein offensichtliches Beispiel ist $\varphi(n) = 2^{-n}$.

Definition 2. Sei $f : L \longrightarrow \mathbb{F}_2^*$ wie in (2). Sei C eine PPS, die bei der Berechnung von f auf $L_{r(n)}$ eine Irrtumswahrscheinlichkeit von ε_n hat, die als Funktion von n vernachlässigbar ist. Dann heißt C **effizienter probabilistischer Algorithmus** für f.

f heißt **(probabilistisch) effizient berechenbar**, wenn es einen effizienten Algorithmus für f gibt.

Für den Primzahltest von Rabin, also die wiederholte Ausführung des strengen Pseudoprimzahltests, kann man diese Forderung erfüllen, wenn man die Zahl t der Basen mit n wachsen lässt; damit die Familie polynomial bleibt, nimmt man t als Polynom $\tau \in \mathbb{N}[X]$. Dann hat C_n n deterministische und $n\tau(n)$ probabilistische Eingänge, die Größe $\mathrm{O}(n^3\tau(n))$ und die Irrtumswahrscheinlichkeit $\frac{1}{4^{\tau(n)}}$. Damit ist gezeigt:

Satz 1 *Der Primzahltest von* Rabin *ist ein effizienter probabilistischer Algorithmus für die Bestimmung der Primzahleigenschaft.*

7.7 Harte Probleme

Etwas kniffliger ist die exakte Definition eines harten Problems. Es ist klar, dass die Forderung, das Problem solle für fast alle Eingaben nicht effizient lösbar sein, durch die schlichte Negierung der Eigenschaft „effizient“ nicht erfüllt wird. Näher kommt dem schon die Forderung, der Vorteil des Algorithmus solle mit wachsendem n gegen 0 gehen; aber auch das ist keine geeignete Definition, da der Vorteil nur eine untere Schranke ist. Der beste Ansatz ist, zu verlangen, dass es keinen Vorteil gibt, der „zu langsam“ gegen 0 geht. „Zu langsam“ soll bedeuten

$$\frac{1}{\eta(n)} \quad \text{mit einem beliebigen Polynom } \eta \in \mathbb{N}[X],$$

und es soll „fast keine“ Eingaben geben, die Ausnahmen sind – die Ausnahmemenge soll „dünn“ sein. Diese Vorstellung wird jetzt in eine exakte Definition umgesetzt.

Für $x \in L_{r(n)}$ betrachten wir die Wahrscheinlichkeit

$$p_x := P(\{\omega \in \Omega_{k(n)} \mid C_n(x,\omega) = f(x)\}),$$

ferner die Menge der Eingaben x, für die C_n einen ε-Vorteil hat:

$$L_{r(n)}(\varepsilon) := \{x \in L_{r(n)} \mid p_x \geq \frac{1}{2^{s(n)}} + \varepsilon\}.$$

Für ein Polynom $\eta \in \mathbb{N}[X]$ ist dann $L_{r(n)}(\frac{1}{\eta(n)})$ die Menge von Eingaben x, für die $f(x)$ von C mit Vorteil $\frac{1}{\eta(n)}$ berechnet wird. Die Ausnahmemenge für η ist damit

$$L^{[f,C,\eta]} := \bigcup_{n\in\mathbb{N}} L_{r(n)}(\frac{1}{\eta(n)}).$$

Wir bezeichnen sie als „**Vorteilsmenge für f, C und η**“. Ihre Bestandteile sollen mit wachsendem n immer unbedeutender werden, und das wird so definiert:

Definition 3. Eine Teilmenge $A \subseteq L$ heisst **dünn**, wenn

$$\frac{\#A_n}{\#L_n}$$

vernachlässigbar ist

Bemerkungen und Beispiele

1. Ist $\#A_n = c$ konstant und $L_n = \mathbb{F}_2^n$, so ist A dünn in L, denn die verlangte Ungleichung ist $c \cdot \varphi(n) \leq 2^n$.

2. Wächst $\#A_n$ höchstens wie ein Polynom, aber $\#L_n$ schneller als jedes Polynom, so ist A dünn in L.

3. Ist $\#A_n = c \cdot \#L_n$ ein fester Anteil, so ist A nicht dünn in L.

4. Ist $L = \mathbb{N}$ und A die Menge der Primzahlen (beides binär codiert), so ist nach dem Primzahlsatz

$$\#A_n \approx \frac{2^{n-1}}{n \cdot \ln(2)} = \frac{\#L_n}{n \cdot \ln(2)}.$$

   Die Menge der Primzahlen ist also nicht dünn in $\mathbb{N}$.

5. Es ist kein effizienter Algorithmus bekannt, der mehr als eine dünne Teilmenge der Menge M aller Produkte von zwei Primzahlen, die sich in der Länge um höchstens ein Bit unterscheiden, faktorisieren kann.

Definition 4. Sei f wie in (2). Dann heißt f **hart**, wenn für jede PPS wie in (1) und für jedes Polynom $\eta \in \mathbb{N}[X]$ die Vorteilsmenge $L^{[f,C,\eta]}$ dünne Teilmenge von L ist.

Beispiele

1. Nach Bemerkung 5 ist die Primzerlegung vermutlich hart.

2. **Quadratrest-Vermutung:** Sei B die Menge von Produkten zweier Primzahlen $\equiv 3 \pmod 4$ (BLUM-Zahlen),

$$L = \{(m, a) \mid m \in B, 1 \leq a \leq m-1\},$$

$$f : L \longrightarrow \mathbb{F}_2$$

   die Funktion

$$f(m,a) = \begin{cases} 1, & \text{wenn } a \text{ Quadratrest mod } m, \\ 0 & \text{sonst.} \end{cases}$$

   Dann ist f hart.

7.8 Kryptographische Basisfunktionen

Damit haben wir auch die theoretische Grundlage, um Einwegfunktionen und starke symmetrische Chiffren exakt zu definieren:

Definition 5. Gegeben sei $f : L \longrightarrow \mathbb{F}_2^*$ wie in (2). Eine Rechtsinverse zu f ist eine Abbildung $g : f(L) \longrightarrow L \subseteq \mathbb{F}_2^*$ mit $f(g(y)) = y$ für alle $y \in f(L)$ – d. h., g findet Urbilder für f. f heißt **Einwegfunktion**, wenn jede Rechtinverse von f hart ist.

Nach dieser Definition ist die diskrete Exponentialfunktion in endlichen Primkörpern (bei entsprechender Interpretation) vermutlich Einwegfunktion.

Zur Definition einer starken Chiffre sehen wir uns erst nochmal eine „gewöhnliche“ Blockchiffre

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

an. Die zugehörige Entschlüsselungsfunktion ist ein

$$G \colon \mathbb{F}_2^r \times \mathbb{F}_2^q \longrightarrow \mathbb{F}_2^r$$

mit $G(F(x,k),k) = x$ für alle $x \in F_2^r$ und $k \in F_2^q$.

Ein Angriff mit bekanntem Klartext findet zu gegebenen $x, y \in \mathbb{F}_2^r$ einen passenden Schlüssel $k \in F_2^q$ mit $F(x,k) = y$. Formalisieren lässt sich das als Abbildung

$$H \colon \mathbb{F}_2^r \times \mathbb{F}_2^r \longrightarrow \mathbb{F}_2^q$$

mit $F(x, H(x,y)) = y$ für alle $x, y \in \mathbb{F}_2^r$ mit $y \in F(x, \mathbb{F}_2^q)$ („mögliche Paare“ (x,y)).

Allgemeiner benutzt ein solcher Angriff ja mehrere bekannte Klartextblöcke, sagen wir s Stück. Er verwendet also eine Abbildung

$$H \colon \mathbb{F}_2^{rs} \times \mathbb{F}_2^{rs} \longrightarrow \mathbb{F}_2^q$$

mit $F(x_i, H(x_i, y_i)) = y_i$ für $i = 1, \ldots s$ für alle möglichen Paare $x, y \in \mathbb{F}_2^{rs}$.

Übungsaufgabe. Formuliere, was ein mögliches Paar ist.

Daraus soll jetzt die komplexitätstheoretische Definition abgeleitet werden.

Definition 6. Eine **symmetrische Chiffre** ist eine Familie $F = (F_n)_{n \in \mathbb{N}}$ von Blockchiffren

$$F_n \colon \mathbb{F}_2^{r(n)} \times \mathbb{F}_2^{q(n)} \longrightarrow \mathbb{F}_2^{r(n)}$$

mit streng monoton wachsenden r und q, so dass $F_n(\bullet, k)$ für jedes $k \in \mathbb{F}_2^{q(n)}$ bijektiv ist und

- F effizient berechenbar ist und
- es eine effizient berechenbare Familie $G = (G_n)_{n\in\mathbb{N}}$ von zugehörigen Entschlüsselungsfunktionen gibt.

Definition 7. Ein **Angriff** auf eine symmetrische Chiffre F **mit bekanntem Klartext** ist eine Familie $H = (H_n)_{n\in\mathbb{N}}$ von Abbildungen

$$H_n \colon \mathbb{F}_2^{r(n)s(n)} \times \mathbb{F}_2^{r(n)s(n)} \longrightarrow \mathbb{F}_2^{q(n)}$$

mit

$$F_n(x_i, H_n(x_i, y_i)) = y_i \quad \text{für } i = 1, \ldots, s(n)$$

für alle möglichen $x, y \in \mathbb{F}_2^{r(n)s(n)}$.

F heißt **starke symmetrische Chiffre**, wenn jeder Angriff auf F mit bekanntem Klartext hart ist.

Die Definition einer Hash-Funktion ist etwas kniffliger und wird hier nicht ausgeführt.