6 Kryptographische Basisfunktionen und ihre Äquivalenz

Für Anwendungen der Kryptographie im praktischen Leben – also für die Konstruktion kryptographischer Protokolle – werden die folgenden Basisfunktionen benötigt:

1. Symmetrische Chiffren:
   (a) Bitblock-Chiffren,
   (b) Bitstrom-Chiffren.
2. Asymmetrische Chiffren.
3. Schlüsselfreie Chiffren:
   (a) Einweg-Funktionen,
   (b) Hash-Funktionen.
4. Zufall:
   (a) Physikalische Zufallsgeneratoren,
   (b) (Algorithmische) Pseudozufallsgeneratoren.
5. Steganographie.

Es wird sich zeigen, dass die Existenz der meisten davon – nämlich 1a, 1b, 3a, 3b, 4b in geeigneten Varianten – jeweils zum Grundproblem der theoretischen Informatik $\mathbf{P} \stackrel{?}{\neq} \mathbf{NP}$ äquivalent und damit bisher unbewiesen ist.

In diesem Abschnitt werden Komplexitätsaussagen noch naiv formuliert. Der Ansatz zur Formalisierung mittels TURING-Maschinen wird skizziert. Er ist allerdings für die Kryptologie nicht ausreichend. Die mathematisch strenge Version von Komplexitätsaussagen für kryptologische Verfahren folgt im nächsten Abschnitt 7.

6.1 Einweg-Funktionen

Es wird weiterhin die informelle Definition aus 4.1 verwendet. Eine exakte Definition wird in 7.5 nachgereicht.

Anwendung: Einweg-Funktionen können direkt zur Einweg-Verschlüsselung verwendet werden. Das bedeutet:

- *Jeder* kann verschlüsseln.
- *Niemand* kann entschlüsseln.

Wozu soll das gut sein, wenn niemand entschlüsseln kann? Dafür gibt es durchaus eine Reihe von Anwendungen (die z. T. den Spezialfall der Hash-Funktionen, siehe 6.2, einsetzen):

- Die Passwort-Verwaltung, z. B. unter Unix oder MS-Windows. Hier soll niemand das verschlüsselt abgespeicherte Passwort ermitteln können, wohl aber muss das Betriebssystem die Möglichkeit haben, das neu eingebene Passwort nach Verschlüsselung mit dem verschlüsselt abgelegten zu vergleichen.
- Ähnlich sieht die Anwendung bei Pseudonymisierung aus: Die Daten eines Falls sollen mit anderswo abgelegten Daten zusammengeführt werden, ohne dass jemand die zum Fall gehörenden Identitätsdaten sehen oder ermitteln kann.
- Eine weitere Anwendung betrifft die digitale Signatur, siehe 6.2.
- Schliesslich ist auch der entscheidende Aspekt der asymmetrischen Verschlüsselung, dass niemand den privaten Schlüssel aus dem öffentlichen ableiten kann. Hierzu sind allerdings Einweg-Funktionen nicht ohne weiteres direkt einsetzbar, wie schon das Beispiel der ElGamal-Chiffre in 4.5 gezeigt hat.

Beispiele für mutmaßliche Einweg-Funktionen:

1. Die diskrete Exponential-Funktion, siehe 4.1.
2. Eine Standard-Methode, aus einer Bitblock-Chiffre

   $$F\colon M \times K \longrightarrow C,$$

   die resistent gegen einen Angriff mit bekanntem Klartext ist, eine Einweg-Funktion $f\colon K \longrightarrow C$ abzuleiten, geht so:

   $$f(x) := F(m_0, x);$$

   es wird also ein fester Klartext m_0 – etwa der Bitblock, der aus lauter Nullen besteht – mit einem Schlüssel, der genau aus dem Einweg-umzuwandelnden Block x besteht, verschlüsselt. Die Umkehrung dieser Einweg-Funktion würde genau dem Angriff mit bekanntem Klartext m_0 auf die Chiffre F entsprechen.

3. Sei $n \in \mathbb{N}$ ein zusammengesetzter Modul. Wir wissen aus 5.2, dass zumindest im Fall, dass n aus zwei großen ungeraden Primzahlen zusammengesetzt ist, das Ziehen der Quadratwurzel $\bmod n$ nicht effizient möglich ist. Damit ist die Quadratabbildung $x \mapsto x^2 \bmod n$ im Restklassenring $\mathbb{Z}/n\mathbb{Z}$ eine Einweg-Funktion – immer unter der Annahme, das die Faktorisierung nicht effizient möglich ist. Allerdings ist die Umkehrung möglich, wenn eine Zusatzinformation vorliegt, nämlich die Primfaktoren von n. Eine solche Zusatzinformation heißt 'trapdoor' (Falltür), und man spricht dann auch von einer „Trapdoor-Einweg-Funktion".
4. Das gleiche gilt für die RSA-Funktion $x \mapsto x^e \bmod n$ mit einem zu $\lambda(n)$ (oder $\varphi(n)$) teilerfremden Exponenten e.

6.2 Hash-Funktionen

Ein besonders wichtiger Spezialfall der Einweg-Funktionen sind die Hash-Funktionen, auch „Message Digest“ oder „kryptographische Prüfsumme“ genannt.

Definition 1. Sei Σ ein Alphabet und $n \in \mathbb{N}$ eine feste natürliche Zahl ≥ 1. Dann heißt eine Einweg-Funktion

$$h\colon \Sigma^* \longrightarrow \Sigma^n$$

schwache Hash-Funktion über Σ.

Zeichenketten *beliebiger* Länge werden dabei also auf Zeichenketten vorgegebener *fester* Länge abgebildet. Da Σ^* unendlich ist, ist gemeint, dass die Einschränkung von h auf Σ^r für alle genügend großen r Einweg-Funktion ist.

Definition 2. Eine Einweg-Funktion $f : M \longrightarrow N$ heißt **kollisionsfrei**, wenn es nicht effizient möglich ist, $x_1, x_2 \in M$ zu finden mit $x_1 \neq x_2$, aber $f(x_1) = f(x_2)$.

Man könnte das auch als „praktisch injektiv“ bezeichnen; injektive Einweg-Funktionen sind natürlich kollisionsfrei. Ist $\#M > \#N$, so kann f nicht injektiv, könnte aber durchaus kollisionsfrei sein.

Definition 3. Eine **(starke) Hash-Funktion** ist eine kollisionsfreie schwache Hash-Funktion.

Für die praktische Anwendung (meistens mit $\Sigma = \mathbb{F}_2$) soll die Länge n der Hashwerte klein sein. Da die Umkehrbarkeit aber nicht effizient sein darf, will man kryptographische Sicherheit erreichen, muss n andererseits auch genügend groß sein. Geht man davon aus, dass die Hash-Funktion statistisch zufällig aussehende, gleichverteilte Werte liefert, so muss man bei einer schwachen Hash-Funktion also sicher vor Exhaustion (vollständiger Suche) sein. Das bedeutet, dass $n = 80$ als Untergrenze gerade nicht mehr ausreichend ist, man also besser 128-Bit-Hashwerte verwenden sollte. Das ist gerade die Länge bei den bekannten Verfahren MD2, MD4, MD5.

Bei praktisch allen Anwendungen ist aber auch die Kollisionsfreiheit wichtig. Hier ist das Geburtstagsphänomen, siehe I.2.6, zu berücksichtigen: Um Kollisionen mit hinlänglicher Sicherheit unauffindbar zu machen, ist etwa die doppelte Bitlänge nötig. Hashwerte von 160 Bit sind also gerade nicht mehr lang genug. Die noch gültigen Standard-Hashverfahren SHA-1 und RIPEMD verwenden genau diese Länge, sollten also schleunigst ausgemustert werden. Im Kontext der AES-Standardisierung wurde das Hash-Verfahren SHA-2 mit mindestens 256-Bit-Hashwerten eingeführt,

das dann auch passenderweise als SHA-256 usw. bezeichnet wird [siehe `http://csrc.nist.gov/publications/`]. Für die MDx-Verfahren wurden tatsächlich schon systematisch Kollisionen gefunden [Dobbertin 1996ff.].

Anwendungen: (Starke) Hash-Funktionen verwendet man

- bei der digitalen Signatur. Eine lange Nachricht direkt mit dem privaten Schlüssel zu verschlüsseln würde bei der Langsamkeit der asymmetrischen Verfahren zu lange dauern. Daher signiert man einen Hashwert der Nachricht. Damit das sicher ist, braucht man eine kollisionsfreie Hash-Funktion. Sonst nämlich könnte ein Angreifer sich auf folgende Weise ein beliebiges Dokument a von Alice signieren lassen: Er fertigt ein unverdächtiges Dokument b an, das Alice gerne unterschreibt. Von beiden Dokumenten stellt er $m = 2^k$ Varianten $a_1, \ldots, a_m$ und $b_1, \ldots, b_m$ her, indem er an k verschiedenen Stellen jeweils ein Leerzeichen einfügt oder nicht. Falls er eine Kollision findet, etwa $h(a_i) = h(b_j)$, lässt er b_j von Alice signieren und hat dann eine gültige Signatur für a_i.
- für die Umwandlung einer langen, für einen Menschen merkbaren Passphrase in einen (schwer merk- und eingebbaren) n-Bit-Schlüssel für eine symmetrische Chiffre.

6.3 Umwandlungstricks

Die Äquivalenz der folgenden Aussagen (A) bis (D) und ihre Implikation von (E) wird plausibel hergeleitet; für einen richtigen mathematischen Beweis fehlen ja noch die exakten Definitionen. Die Implikationen haben durchaus auch praktische Bedeutung für die Konstruktion von Basisfunktionen aus anderen. Diese kann man stark vereinfacht in Hinblick auf die immer wieder aufkommende politische Debatte über eine Kryptographie-Regulierung so zusammenfassen:

- Wer Kryptographie verbieten will, muss auch Hash-Funktionen und Pseudozufallsgeneratoren verbieten.
- Wer Kryptographie unmöglich machen will, muss $\mathbf{P} = \mathbf{NP}$ beweisen.

(A) Es gibt eine Einweg-Funktion $f\colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$.

($\tilde{\mathbf{A}}$) Es gibt eine Einweg-Funktion $\tilde{f}\colon \mathbb{F}_2^{2n} \longrightarrow \mathbb{F}_2^n$.

(B) Es gibt eine schwache Hash-Funktion $h\colon \mathbb{F}_2^* \longrightarrow \mathbb{F}_2^n$.

(C) Es gibt eine starke symmetrische Chiffre $F\colon \mathbb{F}_2^n \times \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ (d. h. eine, die sicher vor einem Angriff mit bekanntem Klartext ist).

(D) Es gibt einen perfekten Zufallsgenerator $\sigma\colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^{p(n)}$.

(E) $\mathbf{P} \neq \mathbf{NP}$.

Anmerkung 1: Bei der komplexitätstheoretischen Präzisierung steht in den Aussagen (A) – (D) stets eine mit n parametrisierte Familie von Funktionen.

Anmerkung 2: Die Perfektheit des Zufallsgenerators besagt, dass bei unbekanntem Urbild $x \in \mathbb{F}_2^n$ aus einigen bekannten Bits des Bildes $\sigma(x)$ keine weiteren Bits effizient bestimmt werden können; insbesondere auch nicht das Urbild x. In der Definition ist p ein ganzzahliges Polynom, das „Streckungspolynom“.

Die Implikation „(D) $\Longrightarrow$ (E)“ wird hier nicht bewiesen.

„(C) $\Longrightarrow$ (D)“: Man setzt $\sigma(x) = (s_1, \ldots, s_{p(n)/n})$ mit $s_0 := x$ und $s_i := F(s_{i-1}, z)$ für $i \geq 1$, wobei als Schlüssel z ein geheim gehaltener konstanter Parameter verwendet wird; man erkennt den OFB-Modus für Bitblock-Chiffren wieder. Dann kann aus jedem Block s_i der Folge der Vorgängerblock s_{i-1} nicht bestimmt werden – sonst wäre die Chiffre nicht sicher. – Dass das schon für die Perfektheit reicht, wird in Kapitel IV gezeigt.

„(D) $\Longrightarrow$ (C)“: Die Bitstrom-Chiffre mit $\sigma(x)$ als Bitstrom zum Schlüssel x ist sicher.

„(A) $\Longrightarrow$ (C)“: Am einfachsten ist der Ansatz von E. Backus; dabei wird $F(a,k) = f(a) + f(k)$ gesetzt. Bei einem Angriff mit bekanntem Klartext sind a und $c = F(a,k)$ bekannt; damit ist auch $f(k) = c + f(a)$ bekannt. Der Angriff ist also auf die Umkehrung von f reduziert. [Andere Ansätze sind MDC (= Message Digest Cryptography) von P. Gutmann und das Feistel-Prinzip.]

„(C) $\Longrightarrow$ (A)“: Das war als Beispiel in Abschnitt 6.1 vorgestellt worden.

„(A) $\Longrightarrow$ ($\tilde{\text{A}}$)“: Sei $\tilde{f}$ durch $\tilde{f}(x,y) := f(x+y)$ definiert. Ist dann zu c ein Urbild (x,y) unter $\tilde{f}$ bestimmbar, so hat man auch das Urbild $x+y$ unter f bestimmt.

„($\tilde{\text{A}}$) $\Longrightarrow$ (B)“: $x \in \mathbb{F}_2^*$ wird mit (höchstens $n-1$) Nullen zu $(x_1, \ldots, x_r) \in (\mathbb{F}_2^n)^r$ aufgefüllt. Dann setzt man

$$\begin{aligned} c_0 &:= 0, \\ c_i &:= \tilde{f}(c_{i-1}, x_i) \quad \text{für } 1 \le i \le r, \\ h(x) &:= c_r. \end{aligned}$$

Damit ist $h\colon \mathbb{F}_2^* \longrightarrow \mathbb{F}_2^n$ definiert. Findet man nun zu gegebenem $y \in \mathbb{F}_2^n$ ein Urbild $x \in (\mathbb{F}_2^n)^r$ mit $h(x) = y$, so auch ein $z \in (\mathbb{F}_2^n)^2$ mit $\tilde{f}(z) = y$, nämlich $z = (c_{r-1}, x_r)$ (wobei $y = c_r$ in der Konstruktion von h).

„(B) $\Longrightarrow$ (A)“: Die Einschränkung von h auf $\mathbb{F}_2^n$ ist auch Einweg-Funktion.

6.4 Physikalische Komplexität

Der unmittelbar einleuchtende Ansatz zur Beurteilung der Komplexität von Algorithmen ist die Zählung von Basisoperationen, wie sie auf einem handelsüblichen Prozessor ausgeführt werden, oder genauer von Taktzyklen. Dies führt zu konkreten Komplexitätsaussagen der Art: „Zur Berechnung von ... sind mindestens (z. B.) 10^{80} der folgenden Rechenschritte nötig: ... ". Hier würde man z. B. elementare arithmetische Operationen zählen (Additionen, Multiplikationen, ...) und berücksichtigen, wie groß die Wortlänge eines gegebenen Prozessors ist (z. B. 32 Bit) und wieviele Taktzyklen eine solche Operation auf einem gegebenen Prozessor in Anspruch nimmt. [Diese Zahl ist auf modernen Prozessoren mit Pipeline-Architektur allerdings nicht immer wohldefiniert.]

Derartige Aussagen sind natürlich für konkret gegebene Algorithmen möglich und führen oft auf interessante mathematische Probleme, wie D. Knuth in seinen Büchern immer wieder vorgeführt hat. Ergebnisse, die aussagen, wieviele Rechenschritte *jeder* Algorithmus zur Lösung eines bestimmten Problems mindestens enthalten muss, hat leider keine Art von Komplexitätstheorie zu bieten, außer für ganz einfache Probleme wie die Auswertung eines Polynoms an einer Stelle. Mit solchen Aussagen könnte man wirkliche Sicherheitsaussagen für Verschlüsselungsverfahren mathematisch beweisen, ohne auf unbewiesene Vermutungen und heuristische Argumente zurückgreifen zu müssen.

Dazu würde man auf physikalische Schranken zurückgreifen können, die sagen, welche Ressourcen Rechner höchstens zur Verfügung haben. Eine bekannte Abschätzung dieser Art sieht so aus (nach Louis K. Scheffer in `sci.crypt`):

- Es gibt höchstens 10^{90} Elementarteilchen im Universum – das ist eine Schranke für Zahl der möglichen CPUs –,
- Es braucht mindestens 10^{-35} Sekunden, um ein Elementarteilchen mit Lichtgeschwindigkeit zu durchqueren – das ist eine Zeitschranke für eine Operation –,
- Das Universum hat eine Lebensdauer von höchstens 10^{18} Sekunden ($\approx 30 \times 10^9$ Jahre) – das ist eine Schranke für die verfügbare Zeit.

Daraus folgt, dass höchstens $10^{143} \approx 2^{475}$ Operationen in diesem Universum möglich sind. Insbesondere sind 500-Bit-Schlüssel sicher vor vollständiger Suche ...

> ... until such time as computers are built from something other than matter, and occupy something other than space. (Paul Ciszek)

Diese Sicherheitsschranke gilt aber nur für den einen Algorithmus „vollständige Suche“; sie sagt nichts über die Sicherheit auch nur eines einzigen kryptographischen Verfahrens!

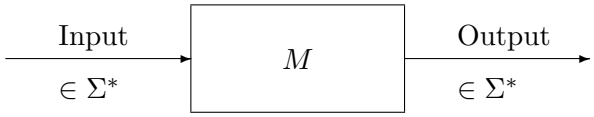
Natürlich ist eine realistische obere Schranke um viele Größenordnungen kleiner.

Zum Vergleich noch ein paar kryptologisch relevante Größen:

Sekunden/Jahr	3×10^7
CPU-Zyklen/Jahr auf 1-GHz-Rechner	3.2×10^{16}
Alter des Universums in Jahren	10^{10}
CPU-Zyklen seither (1 GHz)	3.2×10^{26}
Atome in der Erde	10^{51}
Elektronen im Universum	8.37×10^{77}
ASCII-Ketten der Länge 8 (95^8)	6.6×10^{15}
Binärketten der Länge 56 (2^{56})	7.2×10^{16}
Binärketten der Länge 80	1.2×10^{24}
Binärketten der Länge 128	3.4×10^{38}
Binärketten der Länge 256	1.2×10^{77}
75-stellige Primzahlen (ca 250 Bit)	5.2×10^{72}

6.5 TURING-Maschinen

Die mathematische Komplexitätstheorie führt fast ausschließlich zu asymptotischen Aufwandsabschätzungen, so gut wie immer Abschätzungen nach oben. Sie beruht in ihren verschiedenen Varianten auf verschiedenen Berechnungs- oder Maschinen-Modellen. Hier wird die übliche Formalisierung von Komplexitätsaussagen durch TURING-Maschinen kurz skizziert.



Dabei ist Σ wie üblich ein endliches Alphabet. Der Input ist eine endliche Zeichenkette auf einem (in beide Richtungen unendlich langen) Band. Die TURING-Maschine M besitzt sie eine endliche Zustandsmenge, die unter anderem den Zustand „halt“ enthält. In Abhängigkeit vom Zustand führt sie gewisse Operationen aus, z. B. ein Zeichen vom Band lesen oder auf das Band schreiben und den Lesekopf um eine Stelle nach links oder rechts verschieben. Kommt M in den Zustand „halt“, so ist die dann auf dem Band befindliche Zeichenkette der Output.

Sei $L \subseteq \Sigma^*$ eine Sprache. Falls M für alle $x \in L$ nach endlich vielen Schritten den Zustand „halt“ erreicht, sagt man, M **akzeptiert die Sprache** L. Ist $f : L \longrightarrow \Sigma^*$ eine Funktion und kommt M für jedes $x \in L$ nach endlich vielen Schritten zum Zustand „halt“ mit dem Output $f(x)$, so sagt man, M **berechnet** f.

Mit etwas Mühe und nicht besonders elegant lassen sich alle Algorithmen im naiven Sinne durch TURING-Maschinen beschreiben. Die Komplexität lässt sich durch Zählen der Schritte ausdrücken; für den Input x braucht M bis zum Zustand „halt“ τ_x Schritte.

Meistens betrachtet man die „Worst-Case“-Komplexität. Sei wie üblich $L_n := L \cap \Sigma^n$. Dann wird die Funktion

$$t_M \colon \mathbb{N} \longrightarrow \mathbb{N}, \quad t_M(n) := \max\{\tau_x \mid x \in L_n\},$$

als **(Zeit-) Komplexität** der TURING-Maschine M bezeichnet.

Die Teilmenge $\mathbf{P}$ („polynomiale Zeit“) der Menge aller Funktionen aus Σ^* nach Σ^* wird so definiert: Die Funktion $f : L \longrightarrow \Sigma^*$ liegt in $\mathbf{P}$, wenn es eine TURING-Maschine M und eine natürliche Zahl $k \in \mathbb{N}$ gibt mit

(i) M berechnet f,

(ii) $t_M(n) \leq n^k$ für fast alle $n \in \mathbb{N}$.

Bemerkung. Äquivalent zu (ii) ist: Es gibt ein Polynom $p \in \mathbb{N}[X]$ mit $t_M(n) \leq p(n)$ für alle $n \in \mathbb{N}$.

Gibt es nämlich solch ein Polynom $p = a_r X^r + \cdots + a_0$ mit $a_r \neq 0$, so ist

$$\begin{aligned} a_r n^r &\geq a_{r-1} n^{r-1} + \cdots + a_0 \quad \text{für } n \geq n_0, \\ p(n) &\leq 2a_r n^r \quad \text{für } n \geq n_0, \\ p(n) &\leq n^{r+1} \quad \text{für } n \geq n_1 = \max\{2a_r, n_0\}. \end{aligned}$$

Ist umgekehrt $t_M(n) \leq n^k$ für $n \geq n_0$, so kann man $c \in \mathbb{N}$ wählen mit $t_M(n) \leq c$ für die endlich vielen $n = 0, \ldots, n_0 - 1$. Dann ist $t_M(n) \leq p(n)$ für alle $n \in \mathbb{N}$ mit $p = X^k + c$.

Analog ist die Menge **EXPTIME** („exponentielle Zeit“) definiert: f liegt in **EXPTIME**, wenn es eine TURING-Maschine M, eine natürliche Zahl $k \in \mathbb{N}$ und reelle Zahlen $a, b \in \mathbb{R}$ gibt mit

(i) M berechnet f,

(ii) $t_M(n) \leq a \cdot 2^{bn^k}$ für fast alle $n \in \mathbb{N}$.

Klar ist $\mathbf{P} \subseteq \mathbf{EXPTIME}$.

Beispiele mit $\Sigma = \mathbb{F}_2$.

1. Sei
$$L := \{(p, z) \in \mathbb{N}^2 \mid p \text{ prim } \equiv 3 \pmod 4,\ z \in \mathbb{M}_n^2\},$$
durch eine geeignete Binärdarstellung als Teilmenge von Σ^* codiert. Sei $f(p, z) =$ Quadratwurzel von $z \mod p$ – ebenfalls als Element von Σ^* codiert. Dann ist $f \in \mathbf{P}$ nach 5.3.

2. Sei $L = \mathbb{N}_2$ die Menge aller natürlichen Zahlen ≥ 2 (binär codiert). Sei $f(x) =$ der kleinste Primfaktor von x. Dann ist $f \in \mathbf{EXPTIME}$ – man kann ja alle Zahlen $\leq \sqrt{x} \leq 2^{n/2}$ durchprobieren. *Vermutlich* ist aber $f \notin \mathbf{P}$.

3. **Das Rucksackproblem** (‘knapsack problem’). Hier ist
$$L = \{(m, a_1, \ldots, a_m, N) \mid m, a_1, \ldots, a_m, N \in \mathbb{N}\}$$
in geeigneter binärer Codierung,
$$f(m, a_1, \ldots, a_m, N) = \begin{cases} 1, & \text{wenn es } S \subseteq \{1, \ldots, m\} \text{ gibt} \\ & \text{mit } \sum_{i \in S} a_i = N, \\ 0 & \text{sonst.} \end{cases}$$
Dann ist $f \in \mathbf{EXPTIME}$ – man kann ja alle 2^m Teilmengen $S \subseteq \{1, \ldots, m\}$ durchprobieren. *Vermutlich* ist $f \notin \mathbf{P}$.

6.6 Die Klasse NP

Die TURING-Maschine M berechnet $f : L \longrightarrow \Sigma^*$ **nichtdeterministisch**, wenn es zu jedem $x \in L$ ein $y \in \Sigma^*$ gibt, so dass M mit der Verkettung xy von x und y als Input nach endlich vielen Schritten mit dem Output $f(x)$ anhält.

Beispiel. Sei $\Sigma = \mathbb{F}_2$ und $L = \{(n, a, x) \in \mathbb{N}^3 \mid n \geq 2, a, x \in \mathbb{M}_n\}$. Sei $f = \log_a \bmod n$ der diskrete Logarithmus.

Zu gegebenem x sei y der Logarithmus von x – woher wir ihn haben spielt in der Definition keine Rolle, er existiert jedenfalls. Dann muss die TURING-Maschine M nur noch prüfen, ob $a^y = x$.

Vorstellung. Ein Kandidat y für die Lösung wird vorgegeben, M macht nur noch die Probe.

Alternativ-Vorstellung. Unbeschränkt viele *parallele* TURING-Maschinen probieren je ein $y \in \Sigma^*$ auf Eignung aus.

Die Menge **NP** („nichtdeteministisch-polynomiale Zeit“) ist definiert als die Menge der Funktionen, für die es eine TURING-Maschine M und eine natürliche Zahl $k \in \mathbb{N}$ gibt mit

(i) M berechnet f nichtdeterministisch,

(ii) $t_M(n) \leq n^k$ für fast alle $n \in \mathbb{N}$.

Es gelten die Inklusionen

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXPTIME};$$

die erste davon ist trivial, die zweite ein Satz, der hier nicht bewiesen wird.

Das schon öfter angesprochene wichtigste ungelöste Problem der theoretischen Informatik ist die Vermutung

$$\mathbf{P} \neq \mathbf{NP}.$$

Ebenfalls unbewiesen ist die Vermutung

$$\mathbf{NP} \neq \mathbf{EXPTIME}.$$

Bewiesen ist allerdings

$$\mathbf{P} \neq \mathbf{EXPTIME},$$

wenn auch nur durch „künstliche“ Probleme; ein interessantes „natürliches“ Problem in der Differenzmenge ist nicht bekannt.

Die Kryptoanalyse schwieriger als **NP** zu machen, ist übrigens nicht möglich: Die Exhaustion – das Durchprobieren aller Schlüssel mit jeweiliger

Probeverschlüsselung bei bekanntem Klartext – ist nämlich immer möglich und die Verschlüsselungsfunktion muss effizient, also in **P** sein.

Beispiele

1. Ist f der diskrete Logarithmus wie oben, so $f \in \mathbf{NP}$.
2. Genauso ist die Faktorisierung natürlicher Zahlen in **NP**.
3. Auch das Rucksackproblem ist in **NP**.

Die Funktion f heißt **NP-vollständig**, wenn es für jede TURING-Maschine, die f berechnet (deterministisch!) und jede Funktion $g \in \mathbf{NP}$ eine TURING-Maschine, die g berechnet, und eine natürliche Zahl $k \in \mathbb{N}$ gibt, so dass

$$t_N(n) \leq t_M(n)^k \quad \text{für fast alle } n \in \mathbb{N}.$$

D. h., die Komplexität von N ist höchstens polynomial in der Komplexität von M.

Vorstellung: **NP**-vollständige Probleme sind die maximal komplexen unter denen in **NP**.

Es gibt **NP***-vollständige Probleme.* – Das ist ein Satz, der hier nicht bewiesen wird. Z. B. ist das Rucksack-Problem **NP**-vollständig, ebenso die Nullstellenbestimmung von (Polynom-) Funktionen $p\colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2$. Die Faktorisierung natürlicher Zahlen ist vermutlich nicht **NP**-vollständig.

Sollte $\mathbf{P} = \mathbf{NP}$ sein – was niemand glaubt –, so wären alle Funktionen in $\mathbf{P} = \mathbf{NP}$ auch **NP**-vollständig. Andernfalls gibt die folgende Skizze eine Vorstellung von der relativen Lage dieser Mengen: