Mark de Longueville

Vorlesung Kryptographie
Wintersemester 2004/2005
Freie Universität Berlin

Stand 26. August 2005

Vorwort

Dieses Manuskript zur Vorlesung *Kryptographie*, welche im Wintersemester 2004/05 an der Freien Universität Berlin gehalten wurde, ist von den Hörern der Vorlesung aufgezeichnet, in LaTeX-Form gebracht und vom Dozenten überarbeitet worden.

Mitgeschrieben haben Sonja Lange, Timm Rometzki, Tina Scherer, Patrik Marschalik, Maya Gussmann, Bernhard Streit, Jana John, Martin Fischer, Dana Woitas, Sonja Overesch, Daria Schymura, Jan Felix Breuer, Leo Schmidt und Wolfgang Mulzer. Vielen Dank für die viele Mühe!

Weitere Informationen zur Vorlesung befinden sich unter der Webadresse

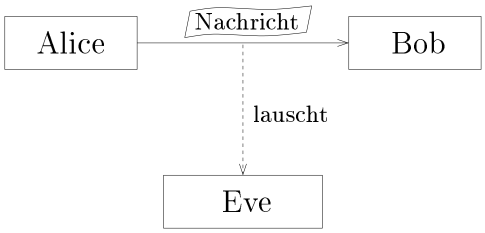
`http://www.math.fu-berlin.de/~delong/teaching/kryptographie/index.html`

Das Manuskript ist mit der SVMono–LaTeX2ε–Dokumentenklasse des Springerverlages erstellt worden.

Inhaltsverzeichnis

Einleitung

In der Kryptographie geht es um das sichere Versenden von Nachrichten. Das Grundschema ist durch folgende Situation gegeben, die traditionell so beschrieben wird: Alice will eine Nachricht an Bob über eine unsichere Leitung schicken. Der Angreifer Eve (ähnlich dem englischen *eavesdrop*, was soviel heißt wie *horchen, lauschen*) hört unberufen mit. Deshalb verschlüsselt Alice den Text, in der Hoffnung, daß Eve nichts damit anfangen kann.



Dabei machen wir die *Grundannahme,* daß in der Übermittlung der Nachrichten keine Fehler auftreten, die Übermittlung von Alices Nachricht also störungsfrei vonstatten geht. Das Erkennen und Beheben von Übermittlungsfehlern ist ebenfalls ein interessantes Gebiet der Mathematik und wird in der *Kodierungstheorie* behandelt.

Als exemplarische *Anwendungen* der Kryptographie seien folgende genannt:

- das sichere Versenden von Emails,
- Bestellungen über das Internet,
- Übermittlung personenbezogener Daten (im Gesundheitswesen, Landeseinwohneramt etc.),
- Geschäftsverhandlungen, z.B. das sichere Übersenden eines Angebots,

- militärische Nachrichten und Informationen, z.B. über Truppenstandorte etc.,
- sicheres Feststellen der Identität eines Senders.

Der Vorlesung liegt folgende *Literatur* zugrunde.

1. J. Buchmann, *Einführung in die Kryptographie*, Springer 2004 (Grundlage der Vorlesung),
2. A. Beutelspacher, *Kryptologie*, Vieweg 1987 (Lesebuch),
3. N. Koblitz, *A course in number theory and cryptography*, Springer 1987 (ernsthaftes Buch),
4. M. Miller, *Symmetrische Verschlüsselungsverfahren*, Teubner 2003 (vom Charakter her ingenieurhaft, viele interessante Anekdoten),
5. C. Papadimitriou, *Computational Complexity*, Addison-Wesley, 1994 (Hintergrund für die Komplexitätstheoretischen Fragestellungen in der Vorlesung),
6. G. Selke, *Kryptographie*, O'Reilly 2000 (Lesebuch),
7. S. Singh, *The code book*, Doubleday 1999 (Nachtlektüre, gibt's auch auf deutsch).

Historisch gesehen ist die Kryptographie gar keine ursprünglich mathematische Disziplin: die frühen kryptographischen Verfahren wurden von Staatsoberhäupten, deren Diplomaten und Universalgelehrten entwickelt, um geheime Informationen sicher zu übermitteln. Bald wurden kryptographische Verfahren vom Militär eingesetzt, und es gilt wohl bis heute, daß die Kryptographie vorrangig eine Domäne des Militärs und der Geheimdienste ist. Die Erfahrung zeigt, daß davon auszugehen ist, daß der Wissensstand über die Kryptographie in Militär– und Geheimdienstkreisen größer ist als öffentlich bekannt.

Erst die *Kryptoanalyse*, also die Theorie des Entschlüsselns als Angriff auf ein Kryptosystem, brachte im Lauf der Zeit die Mathematik ins Spiel. Generell lässt sich sagen, daß die Verschlüsselungs- und Entschlüsselungsalgorithmen vergleichsweise einfach sind. Dies begründet sich unter anderem in der Erwartung an ein Verschlüsselungsverfahren, daß es effizient ist. Schwieriger ist die daraus resultierende Theorie: zum einen für das Brechen einer Chiffre und zum anderen um zu zeigen, daß das Brechen einer bestimmten Chiffre als schweres Problem angesehen werden kann.

In der modernen Kryptographie werden vorrangig Techniken der algorithmischen Zahlentheorie benötigt!

1

Grundlagen und historische Beispiele

Kryptosysteme

In diesem Teil werden wir anhand von historischen Beispielen grundlegende Definitionen geben. Als erstes klassisches Beispiel betrachten wir die sogenannte *Caesar-Chiffre* (nach Julius Caesar: 100–44 v.Chr.). Hierbei werden alle Buchstaben des *Klartextes* um eine feste Zahl k von Buchstaben im Alphabet (mod 26) verschoben. Für $k = 3$ erhalten wir beispielsweise das folgende.

$$\text{caesar} \xrightarrow{+3} \text{fdhvdu}$$

Den verschlüsselten Klartext nennen wir *Chiffretext (bzw. Geheimtext)* und k wird der *Schlüssel* genannt. Die Schlüsselmenge hat im Beispiel der Caesar-Chiffre 26 Elemente, ist also recht klein, und lädt daher den Angreifer Eve dazu ein, alle Schlüssel einfach durchzuprobieren, was die Caesar-Chiffre nicht besonders sicher macht.

Das behandelte Beispiel gibt Anlass zu folgender Definition.

Definition 1.1 (Kryptosystem/Verschlüsselungsverfahren). *Ein* Kryptosystem *besteht aus:*

- *einer Menge P von Klartexten (engl.: plaintext),*
- *einer Menge C von Chiffretexten (engl.: ciphertext),*
- *einer Menge K von Schlüsseln (engl.: keys),*
- *einer Verschlüsselungsfunktion $E : P \times K \longrightarrow C$ (engl.: encipher),*
- *einer Entschlüsselungsfunktion: $D : C \times K \longrightarrow P$ (engl.: decipher),*

so daß für jeden Schlüssel $e \in K$ ein $d \in K$ existiert mit der Eigenschaft, daß für alle $m \in P$ die Gleichung $D(E(m, e), d) = m$ gilt.

Bemerkung 1.1. Für die Verschlüsselungs- und Entschlüsselungsfunktion werden wir auch die folgende kürzere Notation

$$E_e(\cdot) := E(\cdot, e)$$
$$D_d(\cdot) := D(\cdot, d)$$

verwenden.

In der Geschichte der Kryptographie hat sich gezeigt, daß unberufene Mithörer bzw. Mitleser oft dazu in der Lage waren das Verschlüsselungssystem zu erraten, oder es (beispielsweise durch Spionage) in Erfahrung zu bringen. In seiner Abhandlung "La cryptographie militaire" von 1883 hat der niederländer Jean Guillaume Hubert Victor François Alexandre August Kerckhoffs von Nieuwenhof das folgende Prinzip erklärt.

> **Kerckhoffs Prinzip:** Die Sicherheit eines Kryptosystems darf nicht von der Geheimhaltung des Verschlüsselungsverfahrens abhängen. Die Sicherheit gründet sich nur auf die Geheimhaltung des Schlüssels.

Dieses Prinzip wollen wir von nun an stillschweidend akzeptieren, d.h., wann immer wir die Sicherheit eines Kryptosystems diskutieren, gehen wir davon aus, daß der Angreifer Eve das Verschlüsselungsverfahren kennt.

Alphabete und Wörter

Als nächstes führen wir die Begriffe ein, um die in der Definition 1.1 eines Kryptosystems benötigten Klar- und Chiffretexte im Allgemeinen bequem angeben zu können.

Definition 1.2. *Ein* Alphabet *ist eine nichtleere endliche Menge. Die Elemente des Alphabets werden* Zeichen *beziehungsweise* Buchstaben *genannt.*

Beispiel 1.1. Typische Alphabete sind

- $\Sigma = \{A, B, C, \ldots, Z\}$,
- $\Sigma = \{0, 1\}$,
- $\Sigma =$ ASCII-Zeichensatz (American Standard Code for Information Interchange), $|\Sigma| = 128$,
- $\Sigma = \mathbb{Z}_m = \mathbb{Z}/m\mathbb{Z}$.

Bemerkung 1.2. Jedes Σ mit $|\Sigma| = m$ können und werden wir mit $\mathbb{Z}_m$ identifizieren.

Nun betrachten wir endliche Folgen $(\sigma_1, \ldots, \sigma_r)$ aus Σ und schreiben dafür auch $\sigma_1\sigma_2 \cdots \sigma_r$. Es gibt auch eine *Nullfolge bzw. leere Folge* () und schreiben dafür auch einfach nichts hin. Folgen *verknüpfen* wir auf folgende Weise:

$$(\sigma_1, \ldots, \sigma_r) \circ (\tau_1, \ldots, \tau_s) = (\sigma_1, \ldots, \sigma_r, \tau_1, \ldots, \tau_s) \quad \text{bzw.}$$
$$\sigma_1 \cdots \sigma_r \circ \tau_1 \cdots \tau_s = \sigma_1 \cdots \sigma_r \tau_1 \cdots \tau_s.$$

Definition 1.3. *Sei Σ ein Alphabet.*

- *Eine endliche Folge mit Elementen aus Σ heisst* Wort. *Die Nullfolge heisst* leeres Wort.
- *Die* Länge $|w|$ *eines Wortes w ist die Länge der entsprechenden Folge.*
- *Σ^* sei die* Menge aller Wörter *über Σ.*
- *Σ^n sei die* Menge aller Wörter der Länge *n.*

Bemerkung 1.3. Mit $\circ$ wird Σ^* zu einer Halbgruppe.

Um die neuen Begriffe etwas besser kennen zu lernen, kehren wir noch einmal zur Caesar-Chiffre zurück. In diesem Fall haben wir also $\Sigma = \mathbb{Z}_{26}, P = \Sigma^*, C = \Sigma^*$ und $k = \Sigma$. Sei nun $v = v_1 \cdots v_r \in P$ ein Klartext und $k \in K$ ein Schlüssel. Setze dann

$$E(v,k) = w_1 \cdots w_r \quad \text{mit} \quad w_i = v_i + k,$$

und entsprechend

$$D(w_1 \cdots w_r, k) = v_1 \cdots v_r \quad \text{mit} \quad v_i = w_i - k.$$

Allgemein sei nun $\pi : \Sigma \longrightarrow \Sigma$ eine Permutation und

$$E_\pi(v_1 \cdots v_r) = w_1 \cdots w_r \quad \text{mit} \quad w_i = \pi(v_i),$$

und

$$D_\pi(w_1 \cdots w_r) = v_1 \cdots v_r \quad \text{mit} \quad v_i = \pi^{-1}(w_i).$$

Die oben beschriebenen Kryptosysteme heissen *monoalphabetische Ersetzungschiffren (oder einfach nur kurz monoalphabetische Chiffren)*. Die Schlüsselmenge hat für $\Sigma = \mathbb{Z}_m$ $m!$ viele Elemente, also im Beispiel von $\Sigma = \mathbb{Z}_{26}$ bereits $26!(\approx 4 \cdot 10^{26})$ viele Elemente, eine recht beachtliche Zahl. Somit scheint die monoalphabetische Chiffre sicherer als die Caesar-Chiffre. Aber muß man im ungünstigsten Fall alle Schlüssel durchsuchen, um die Chiffre zu brechen?

Statistische Analyse (Frequenzanalyse)

Monoalphabetische Kryptosysteme lassen sich in den meisten Fällen leicht brechen, zum Beispiel mit der statistischen Analyse. Die älteste bekannte Aufzeichnung dieser Methode stammt von dem arabischen Universalgelehrten Abū Yūsūf Ya'qūb ibu Is-hāq ibu as Sabbāh ibu 'omrān ibn Ismaīl al-Kindī aus dem neunten Jahrhundert. Er beobachtete, daß Buchstaben in einer Sprache im Allgemeinen eine gewisse Häufigkeitsverteilung haben. Ist ein Text lang genug und regelmäßig kann man mit einer statistischen Analyse dieser Verteilung leicht zum Ziel kommen. Im Deutschen und Englischen sind die

Im Deutschen:	
E	18 %
N	10 %
R	8 %
I	7,5 %
S	6,3 %

Im Englischen:	
E	12,7 %
T	9 %
A	8,1 %
I	7 %
N	6,8 %

Abb. 1.1. Häufigkeitsverteilung von Buchstaben im Deutschen und Englischen

häufigsten fünf Buchstaben mit ihren Wahrscheinlichkeiten in Abbildung 1.1 gezeigt.

Es gibt allerdings auch Texte die der Annahme einer regelmäßigen Verteilung widersprechen, wie zum Beispiel der bemerkenswerte Roman "La disparition" von Georges Perec aus dem Jahr 1969, der ohne den Buchstaben "e" auskommt! Dieses Buch wurde von Gilbert Adair ins Englische übersetzt mit dem Titel "A Void", in dem der Buchstabe "e" ebenfalls nicht vorkommt. Der Anfang dieser Übersetzung liest sich so:

> Today, by radio, and also on giant hoardings, a rabbi, an admiral notorious for his links to Masonry, a trio of cardinals, a trio, too, of insignificant politicians (bought and paid for by a rich and corrupt Anglo-Canadian banking corporation), inform us all of how our country now risks dying of starvation. A rumour, that's my initial thought as I switch off my radio, a rumour or possibly a hoax. Propaganda, I murmur anxiously — as though, just by saying so, I might allay my doubts — typical politicians' propaganda. But public opinion gradually absorbs it as a fact. Individuals start strutting around with stout clubs. "Food, glorious food!" is a common cry (occasionally sung to Bart's music), with ordinary hard-working folk harassing officials, both local and national, and cursing capitalists and captains of industry. Cops shrink from going out on night shift. In Macon a mob storms a municipal building. In Rocadamour ruffians rob a hangar full of foodstuffs, pillaging tons of tuna fish, milk and cocoa, as also a vast quantity of corn — all of it, alas, totally unfit for human consumption.

Typen von Attacken

Als nächstes besprechen wir die drei bekanntesten Typen von Attacken auf ein Kryptosystem.

1. *Angriff mit der Chiffre (ciphertext only attack):* Der Angreifer Eve hat nur den Chiffretext zur Verfügung.
2. *Angriff mit bekanntem Klartext (known plaintext attack):* Der Angreifer Eve hat Klar- und Chiffretext für einige Teile der Nachricht zur Hand und kann diese auch einander zuordnen.

3. *Angriff mit gewähltem Klartext (chosen plaintext attack):* Eve kann sich zunächst Chiffretexte zu gewählten Nachrichten erzeugen lassen.

Beispiel 1.2. Diese drei Attacken werden wir im folgenden an den zwei Beispielen, denen wir bislang begegnet sind, der Caesar-Chiffre und allgemeiner der monoalphabetischen Ersetzungschiffre durchdenken. Für die *Caesarchiffre* ergibt sich:

1. Ein Angriff mit der Chiffre ist durch das vollständige Durchsuchen der 26-elementigen Schlüsselmenge schnell bewerkstelligt.
2. Bei einem Angriff mit bekanntem Klartext ist man sofort fertig, da bereits die Chiffrierung eines Buchstabens ausreicht, um den Schlüssel in Erfahrung zu bringen.
3. Bei einem Angriff mit gewähltem Klartext reicht es also, sich einen einzelnen Buchstaben verschlüsseln zu lassen.

Für die *monoalphabetische Ersetzungschiffre* sehen wir:

1. Ein Angriff mit der Chiffre ist mit Hilfe der statistischen Analyse meistens erfolgreich.
2. Ein Angriff mit bekanntem Klartext gibt dem Angreifer zumindest einen Teil der Permutation, und mit Glück lässt sich der gesamte Schlüssel eventuell bereits bestimmen.
3. Bei einem Angriff mit gewähltem Klartext lässt der Angreifer sich das Alphabet verschlüsseln und erhält die ganze Permutation.

Nun stellt sich die Frage, ob die genannten Angriffsformen überhaupt realistisch sind.

Für den Angriff mit der Chiffre bedarf es keinen Kommentars. Der Angriff mit bekanntem Klartext kommt öfter vor, denn man weiss bei Briefen zum Beispiel wie normalerweise adressiert wird, oder wie gängige Grußformeln lauten. Zum Beispiel weiss man, daß eine schriftliche Nachricht oft mit dem Worten "An␣" beginnt und dadurch könnte man schon einen ersten Hinweis auf einen möglichen Schlüssel haben. Im zweiten Weltkrieg haben die Deutschen Militärs beispielsweise den Wetterbericht, meist verlässlich zur selben Uhrzeit, in verschlüsselter Form an die Truppen gesendet. Der Klartext zu dieser Chiffre enthielt mit großer Wahrscheinlichkeit das Wort "Wetter".

Ein weiteres historisches Beispiel für einen Angriff mit gewähltem Klartext liefert ebenfalls der Zweite Weltkrieg. Wenn eine Wasserbombe gesichtet wurde sendeten deutsche Schiffe eine Nachricht mit Abwurfsort und Abwurfszeit. Die Alliierten warfen deshalb Wasserbomben ab, die bewußt keine Schiffe trafen, und erhielten so Chiffretexte zu vorgegebenem Ort und Zeit.

Lineare Verschlüsselungen (im Falle einer monoalphabetischen Chiffre)

Wir betrachten eine spezielle Permutationschiffre mit $P = \mathbb{Z}_n^*$, $C = \mathbb{Z}_n^*$, also jeweils die Menge aller Wörter über $\mathbb{Z}_n$. Wir betrachten zunächst Wörter der Länge eins $m \in \mathbb{Z}_n$ und Permutationen, welche von zwei Parametern abhängig sind:

$$\pi_{(a,b)}(m) = am + b = c.$$

Dies gibt Anlass zur Chiffre $E(v_1 \cdots v_r, (a, b)) = \pi_{(a,b)}(v_1) \cdots \pi_{(a,b)}(v_r)$. Zum Dechiffrieren geht man wie folgt vor: Man betrachtet $am \equiv c - b \mod n$. Um m zu erhalten benötigt man ein $u \in \mathbb{Z}_n$ mit $ua \equiv 1 \mod n$. Ein solches u existiert, falls a und n teilerfremd sind, und dann gilt

$$m \equiv u \cdot a \cdot m \equiv u \cdot c - u \cdot b \mod n.$$

Wir dechiffrieren also folgendermaßen:

$$D(v_1 \ldots v_r(\alpha, \beta)) = \pi_{(\alpha,\beta)}(v_1) \cdots \pi_{(\alpha,\beta)}(v_r) \text{ mit } \alpha = u \text{ und } \beta = -u \cdot b.$$

Die Menge der Schlüssel ist demnach

$$K = \{(a, b) : a, b \in \mathbb{Z}_n, a \text{ teilerfremd zu } n\}.$$

Und somit gilt für die Mächtigkeit der Schlüsselmenge: $|K| = \varphi(n) \cdot n$. Hierbei ist φ die Eulersche φ-Funktion (siehe Kapitel 2) und steht für die Anzahl aller zu a teilerfremden Zahlen zwischen 1 und n.

Blockchiffre

Definition 1.4. *Eine* Blockchiffre *ist eine Verschlüsselungsfunktion, die auf einer Funktion $E_t : \Sigma^t \times K \longrightarrow C$ basiert, d.h. es werden Wörter der Länge rt verschlüsselt: Für $m = m_1 \ldots m_r \in \Sigma^{rt}$ mit $m_i \in \Sigma^t$ ist $E(m, k) := E_t(m_1, k) \ldots E_t(m_r, k)$.*

Im Allgemeinen wird ein Wort $m \in \Sigma^k$ mit beliebigen Zeichen bis auf Länge rt verlängert. Die Art der Ergänzung beeinflusst dabei die Schwierigkeit des Codebrechens.

Vigenère-Chiffre (Blaise Vigenère 1586)

Ein klassisches Beispiel für die Blockchiffre ist die Vigenère-Chiffre. Als Alphabet Σ wird der Ring $\mathbb{Z}_{26}$ gewählt und mit den Buchstaben `{A,B,...}` identfiziert. Es wird ein Schlüsselwort gewählt, z.B. `BLUME`. In dem Klartext werden nun immer Blöcke in der Länge des Schlüsselworts verschlüsselt. Dabei verschiebt sich jeder Buchstabe in einem Block um den entsprechenden Buchstaben aus dem Schlüsselwort. Aus dem Klartext „`EINBEISPIELTEXT`“ wird der Chiffretext „`FTZOIJDKUIMEYJX`“.

```
EINBEISPIELTEXT
BLUMEBLUMEBLUME
---------------
FTZOJKELVJNFZKY
```

Anmerkung 1.1. Die Verallgemeinerung der Vigenère-Chiffre sieht dann wie folgt aus: $P = \mathbb{Z}_n^*$, $C = \mathbb{Z}_n^*$, $K = \mathbb{Z}_n^* \setminus \{()\}$. Für $k = (k_1, \ldots, k_t) \in K$ setzte $E_t(m_1 \ldots m_t, k) = (m_1 + k_1) \ldots (m_t + k_t), \quad m_i \in \mathbb{Z}_n$.

Die Attacke auf die Vigenère-Chiffre geht auf Charles Babbage (1854) und Friedrich Wilhelm Kasiski (1863) zurück. Ist die Schlüssellänge bekannt, so führt eine statistische Analyse weiter, indem man sich entsprechende Stellen betrachtet. Um eine Idee für die Schlüssellänge zu bekommen, sucht man im Chiffretext nach gleichen Wörtern, der ggT der Abstände ist ein guter Kandidat für die gesuchte Länge.

Hill-Chiffre (Hill 1929)

Bisher hat die Änderung eines einzelnen Buchstabens im Klartext zu einer einzigen Änderung im Chiffretext geführt. Nun soll eine einzelne Änderung weit mehr Einfluss auf den Chiffretext haben. Hat eine kleine Änderung des Klartextes oder des Schlüssels großen Einfluss auf den gesamten Chiffretext, so spricht man von großer **Diffusion**. Je geringer der Einfluss der Wahrscheinlichkeitsverteilung der Klartextbestandteile auf die Wahrscheinlichkeitsverteilung der Chiffretexte, desto größer die **Konfusion** (Shannon 1949).

Die Ausgangssituation bei der Hill-Chiffre ist die folgende: $P = \mathbb{Z}_n^*$, $C = \mathbb{Z}_n^*$, $K = \{A \in \mathbb{Z}^{t \times t} : A \text{ invertierbar}\}$, $E_t(v, A) = Av$, für $v \in \mathbb{Z}_n^t$

Anmerkung 1.2. Der Spezialfall, daß A eine Permutationsmatrix ist, mit $E_t(v, \pi) = v_{\pi(1)} \ldots v_{\pi(t)}$ mit $v = v_1 \ldots v_t$ wurde bereits von den Griechen um 400 v. Chr. genutzt. Praktisch wickelt man hierzu einen Streifen Papier um einen Stab und schreibt zeilenweise den Klartext darauf. Man nennt diese „Verschlüsselungsmaschine“ **Skytala**.

Die affin lineare Blockchiffre

Die affin lineare Blockchiffre ist eine Kombination aus den beiden vorangegangen Chiffriermethoden. $P = \mathbb{Z}_n^*$, $C = \mathbb{Z}_n^*$, $K = \{(A, b) \in \mathbb{Z}^{t \times t} \times \mathbb{Z}_n : A \text{ invertierbar}\}$, $E_t(v, A) = Av + b$.

Bemerkung 1. $A \in \mathbb{Z}_n^{t \times t}$ ist invertierbar über $\mathbb{Z}_n \Leftrightarrow \det(A) \in \mathbb{Z}_n^\times$

Beweis. In der Cramerschen Regel wird zur Bestimmung der inversen Matrix durch $\det(A)$ geteilt, daß macht über $\mathbb{Z}_n$ nur dann Sinn, wenn $\det(A)$ selbst invertierbar, also aus $\mathbb{Z}_n^\times$ ist.

Übung 1.1. Beschreibe eine Attacke mit gewähltem Klartext auf die affin lineare Chiffre, mit möglichst wenig gewählten Klartexten.

2

Komplexität und Zahlentheorie

Der folgende Abschnitt ist durch die in der Kryptographie wichtige Frage, wie groß der Berechnungsaufwand für einen kryptographischen Algorithmus ist und wieviel Speicherplatz er benötigt, motiviert. Hierzu liefern wir zunächst einige Vorarbeiten.

Komplexität

Binärentwicklung und Länge

Sei $m \in \mathbb{N}$. Dann kann man m in seiner Binärentwicklung darstellen, d.h.

$$m = \sum_{i=0}^{k} a_i 2^i \text{ mit } a_i \in \{0, 1\}.$$

Dabei ist $k = \lfloor \log_2 m \rfloor + 1$. Wir setzen $\text{size}(m) := \lfloor \log_2 m \rfloor + 1$, und ferner $\text{size}(0) := 1$. Hierbei ist $\text{size}(m)$ die *Länge der Zahl* m in der Binärdarstellung, d.h. die Anzahl der Bits, die benötigt werden, um m darzustellen.

$\mathcal{O}$-Notation

Die $\mathcal{O}$-Notation dient dazu die Aufwandsabschätzungen für einen kryptographischen Algorithmus zu vereinfachen. Seien $f, g : \mathbb{N}^k \to \mathbb{R}$ Funktionen. Wir schreiben $f = O(g)$, falls Konstanten $B, C \in \mathbb{N}$ existieren, so dass für alle Eingabeparameter $n_1, \ldots, n_k \geq B$ gilt:

$$f(n_1, \ldots, n_k) \leq C \cdot g(n_1, \ldots, n_k).$$

Diese Notation nutzen wir nun um über Rechenzeiten reden zu können.

Rechenzeiten

Seien zwei natürliche Zahlen a, b mit ihrer Binärdarstellung gegeben und wir gehen davon aus, dass die Addition von zwei Bits konstante Zeit $\mathcal{O}(1)$ benötigt. Dann zeigen Schulmethoden zum Rechnen, d.h. die gewöhnlichen Rechenregeln angewandt auf die Binärdarstellung der Zahlen folgendes.

- Die Addition von a und b benötigt $\mathcal{O}(\max\{\text{size}(a), \text{size}(b)\})$ viel Zeit.
- Die Multiplikation von a und b erfordert als Zeit $\mathcal{O}(\text{size}(a) \cdot \text{size}(b))$.
- Die Division von a durch b erfordert die Zeit $\mathcal{O}(\text{size}\left(\lfloor \frac{a}{b} \rfloor\right) \cdot \text{size}(b))$.

Der benötigte Platz dafür ist jeweils $\mathcal{O}(\text{size}(a) + \text{size}(b))$.

Polynomzeit

Hierbei soll präzisiert werden, was es bedeutet, dass ein Algorithmus effizient ist. Angenommen ein Algorithmus berechnet etwas zu den Eingabezahlen $z_1, \ldots, z_n \in \mathbb{Z}$.

Definition 2.1. *Wir sagen, dass der Algorithmus* polynomielle Laufzeit *hat, falls es Exponenten* $e_1, \ldots, e_n \geq 0$ *gibt, so dass der Algorithmus die Laufzeit von*

$$\mathcal{O}(\text{size}(|z_1|)^{e_1} \cdots \text{size}(|z_n|)^{e_n})$$

hat. Ein polynomieller Algorithmus wird auch als effizient *bezeichnet.*

Zahlentheorie

Nun kommen wir zu Erkenntnissen aus der Zahlentheorie.

Theorem 2.1. *Seien* $a, b \in \mathbb{Z}$. *Dann gilt:* $a \cdot \mathbb{Z} + b \cdot \mathbb{Z} = ggT(a, b) \cdot \mathbb{Z}$, *wobei* $ggT(a, b)$ *für den größten gemeinsamen Teiler von* a *und* b *steht.* □

Zur einfacheren Notation schreiben wir zukünftlich auch $(a, b) := \text{ggT}(a, b)$.

Korollar 2.1. *Zu zwei ganzen Zahlen* a *und* b *existieren immer* $x, y \in \mathbb{Z}$ *so, dass die Gleichung* $ax + by = (a, b)$ *gilt.*

Euklidischer Algorithmus

Um den größten gemeinsamen Teiler von zwei ganzen Zahlen a, b zu berechnen verwendet man den euklidischen Algorithmus. Dieser funktioniert wie folgt:
Initialisiere $r_0 = |a|,\ r_1 = |b|,\ k = 1$.
While $\mathbf{r_k \neq 0}$ do: setze $r_{k+1} := r_{k-1} \mod r_k$ und erhöhe k um eins.
Gib aus $\text{ggT}(a, b) = r_{k-1}$.

Bemerkung 2.1. Falls in der Ausgangssituation $|a| > |b|$ galt, dann bilden die r_i eine streng monoton fallende Folge. Falls $|a| < |b|$, dann werden die Rollen von a, b im ersten Schritt des Algorithmus vertauscht und die r_i bilden eine streng monoton fallende Folge für $i \geq 1$.

Beispiel 2.1. Man berechne $\text{ggT}(102, 36)$:

$$\begin{aligned} 102 \bmod 36 &= 30 \\ 36 \bmod 30 &= 6 \\ 30 \bmod 6 &= 0 \end{aligned}$$

Die Ausgabe hier wäre 6 und die Folge der r_i wäre gegeben durch $102, 36, 30, 6, 0$.

Der Satz, der hinter dem euklidischen Algorithmus steht, ist folgender.

Theorem 2.2. *1. Wenn $b = 0$ ist, dann ist $ggT(a, b) = |a|$.*
2. Wenn $b \neq 0$ ist, dann ist $ggT(a, b) = ggT(|b|, a \bmod |b|)$. □

Laufzeitanalyse

Wir wollen nun die Anzahl n der Iterationen, die der euklidische Algorithmus verwendet, abschätzen. Es sei o.B.d.A. $a > b > 0$ und $n \in \mathbb{N}$, so dass für die r_i aus dem euklidischen Algorithmus gilt: $r_0, \ldots, r_n \neq 0$ und $r_{n+1} = 0$. Ziel ist es eine obere Schranke für die Anzahl n der Iterationen des euklidischen Algorithmus zu finden. Definiere

$$q_k := \left\lfloor \frac{r_{k-1}}{r_k} \right\rfloor .$$

Dann gilt also: $r_{k-1} = q_k \cdot r_k + r_{k+1}$.

Lemma 2.1. *Es gilt $q_k \geq 1$ für $1 \leq k \leq n-1$ und $q_n \geq 2$.*

Beweis. Die Folge der r_i ist streng monoton fallend, d.h. $r_{k-1} > r_k > r_{k+1}$, also $q_k = \left\lfloor \frac{r_{k-1}}{r_k} \right\rfloor \geq 1$. Ferner gilt

$$r_{n-1} = q_n r_n + \underbrace{r_{n+1}}_{=0},$$

also folgt aus $r_{n-1} \neq r_n$, dass $q_n \geq 2$ gilt. □

Theorem 2.3. *Seien $a, b > 0$. Dann ist die Anzahl der Iterationen des euklidischen Algorithmus höchstens*

$$\frac{\log_2 b}{\log_2 \theta} + 1 < 1{,}441 \cdot \log_2 b + 1 = \mathcal{O}(\text{size}(b)),$$

wobei $\theta = \frac{1+\sqrt{5}}{2}$ der goldene Schnitt ist.

Beweis. O.B.d.A. sei $r_n = 1$.

Übung 2.1. Zeige, dass diese Annahme gerechtfertigt ist.

Wir zeigen per Induktion dass gilt: $\circledast \quad r_k \geq \theta^{n-k},\ 0 \leq k \leq n$
Induktionsanfang für $k = n$: $r_n = 1 = \theta^0$ ist erfüllt.
Induktionsschluss: Gelte nun $\circledast$ für alle $k' > k$ (müssen zeigen es gilt für k).

$$\begin{aligned} r_k &= q_{k+1} r_{k+1} + r_{k+2} \\ &\geq q_{k+1} \cdot \theta^{n-(k+1)} + \theta^{n-(k+2)} \\ &\geq \theta^{n-k-1} + \theta^{n-k-2} \\ &= \theta^{n-k-1} \underbrace{\left(1 + \frac{1}{\theta}\right)}_{=\theta} \\ &= \theta^{n-k} \end{aligned}$$

Also gilt $b = r \geq \theta^{n-1}$ und somit $\frac{\log_2 b}{\log_2 \theta} \geq n - 1$. □

Theorem 2.4. *Sind a und b zwei ganze Zahlen, so benötigt der euklidische Algorithmus die Zeit $\mathcal{O}(\mathrm{size}(a) \cdot \mathrm{size}(b))$.*

Beweis. O.B.d.A. sei $a > b > 0$. In jeder Iteration wird eine Division mit Rest durchgeführt. Die Bestimmung von q_k und r_{k+1} benötigt also die Zeit

$$\mathcal{O}\left(\mathrm{size}\left(\left\lfloor \frac{r_{k-1}}{r_k} \right\rfloor\right) \cdot \mathrm{size}(r_k)\right) = \mathcal{O}(\mathrm{size}(q_k) \cdot \mathrm{size}(r_k)).$$

Wir betrachten die Folge $r_1, r_2, \ldots$
Es gilt $r_k \leq b$, also $\mathrm{size}(r_k) \leq \mathrm{size}(b)$.
Wenn wir dies summieren ergibt sich eine Schranke:

$$\mathcal{O}\left(\mathrm{size}(b) \cdot \sum_{k=1}^{n} \mathrm{size}(q_k)\right) = \mathcal{O}\left(\mathrm{size}(b) \cdot (n + \sum_{k=1}^{n} \lfloor \log_2 q_k \rfloor)\right).$$

Nun ist $n = \mathcal{O}(\mathrm{size}(b))$ und wir wollen die Summe abschätzen:

$$a = r_0 = q_1 r_1 + r_2 \geq q_1 r_1 = q_1(q_2 r_2 + r_3) \geq q_1 q_2 r_2 \ldots \geq q_1 \cdots q_n.$$

Also ist $\sum_{k=1}^{n} \log_2 q_k \leq \log_2 a = \mathcal{O}(\mathrm{size}(a))$ und wir erhalten also eine Schranke (da aus $a > b$ folgt $\mathrm{size}(a) > \mathrm{size}(b)$):

$$\mathcal{O}(\mathrm{size}(b)(\mathrm{size}(b) + \mathrm{size}(a))) = \mathcal{O}(\mathrm{size}(a)\,\mathrm{size}(b)). \qquad \square$$

Mit etwas mehr Arbeit erhält man:

Theorem 2.5 (Erweiterter euklidischer Algorithmus). *Seien $a, b \in \mathbb{Z}$, so lässt sich in der Zeit $\mathcal{O}(\mathrm{size}(a) \cdot \mathrm{size}(b))$ der ggT(a, b) und $x, y \in \mathbb{Z}$ bestimmen mit $ax + by = \mathrm{ggT}(a, b)$.* □

Der Restklassenring $(\mathbb{Z}_m, +, \cdot)$

Theorem 2.6. *Die Restklasse $a + m\mathbb{Z}$ ist genau dann in $\mathbb{Z}/m\mathbb{Z}$ multplikativ invertierbar wenn $ggT(a, m) = 1$.* □

Theorem 2.7. *Angenommen Restklassen modulo m werden durch Vertreter in $\{0, \ldots, m-1\}$ repräsentiert und zusätzlich ist ihre Binärdarstellung bekannt, so erfordert die Addition und Subraktion die Zeit $\mathcal{O}(\text{size}(m))$. Die Multiplikation und Division benötigen die Zeit $\mathcal{O}((\text{size}(m))^2)$. Außerdem brauchen alle einen Speicherplatz von $\mathcal{O}(\text{size}(m))$.*

Die Restklassen $a + m\mathbb{Z}$ mit $\text{ggT}(a, m) = 1$ heißen relativ prime Restklassen.

Theorem 2.8. *Die Menge $\mathbb{Z}_m^\times$ der primen Restklassen bildet eine abelsche Gruppe bzgl. der Multiplikation.*

Definition 2.2. *Es sei*

$$\Phi(m) = \{a : 1 \leq a \leq m-1 \text{ mit } \text{ggT}(a, m) = 1\}$$

die Menge aller zu m relativ primen Elemente, und demnach ist $\varphi(m) = |\Phi(m)|$ die Eulerfunktion.

Mit dieser Notation ist $\mathbb{Z}_m^\times = \Phi(m) + m\mathbb{Z}$ und diese Gruppe hat Ordnung $\varphi(m)$.

Beispiel 2.2. Sei $p \in \mathbb{N}$ Primzahl, dann ist $\varphi(p) = p - 1$.

Theorem 2.9. *Sei $m \in \mathbb{Z}$, dann gilt*

$$\varphi(m) = m \cdot \prod_{p|m,\ p \text{ prim}} \left(1 - \frac{1}{p}\right).$$

Beispiel 2.3. $\varphi(pq) = pq - 1 - (q-1) - (p-1)$ für $p, q \in \mathbb{N}$ prim. Dabei wird folgendermaßen gerechnet: Wir haben $pq - 1$ Elemente, davon allerdings $q - 1$ Vielfache von p und $p - 1$ Vielfache von q.

Im Beweis von Theorem 2.9 werden wir das Inklusions-/Exklusionsprinzip verwenden:

Anmerkung 2.1. Zur Erinnerung eine kleine Wiederholung des Inklusions-/Exklusionsprinzips (ohne Beweis): Seien $A_1, \ldots, A_r \subseteq \{1, \ldots, m\}$. Dann gilt:

$$\left|\{1, \ldots, m\} \setminus \bigcup_{i=1}^r A_i\right| = m - \sum_{i=1}^r |A_i| + \sum_{i<j} |A_i \cap A_j| - \sum_{i<j<k} |A_i \cap A_j \cap A_k| \pm \ldots \pm |A_1 \cap \ldots \cap A_r|$$

Zum Beweis benutze man:

$$\sum_{i=0}^k (-1)^i \binom{k}{i} = 0 \tag{2.1}$$

Beweis. Sei $m = p_1^{k_1} p_2^{k_2} \cdots p_r^{k_r}$ die Primfaktorzerlegung von m, d.h. $p_i \neq p_j$ für $i \neq j$ und p_i ist prim. Sei außerdem $\{i_1, \ldots, i_s\} \subseteq \{1, \ldots, r\}$ und $N(i_1, \ldots, i_s) := \{a : 1 \leq a \leq m-1, a \text{ ist Vielfaches von } p_{i_1} \cdots p_{i_s}\}$. Dann ist:

$$
\begin{aligned}
\varphi(m) &= m - 1 - \sum_{i=1}^{r} N(i) + \sum_{i<j} N(i,j) - \sum_{i,j,k} N(i,j,k) \pm \ldots \pm |N(1,\ldots,r)| \\
&= m - 1 - \sum_{i=1}^{r} \left(\frac{m}{p_i} - 1\right) + \sum_{i<j} \left(\frac{m}{p_i p_j} - 1\right) - \sum_{i<j<k} \left(\frac{m}{p_i p_j p_k} - 1\right) \pm \\
&\quad \ldots \pm \left(\frac{m}{p_1 \cdots p_r} - 1\right) \\
&= m + \underbrace{\left(-\binom{r}{0} + \binom{r}{1} - \binom{r}{2} + \binom{r}{3} \mp \ldots \pm \binom{r}{r}\right)}_{\text{die } -1\text{en aufaddiert ergeben 0 (siehe Gleichung 2.1)}} - \sum_{i=1}^{r} \frac{m}{p_i} + \sum_{i<j} \frac{m}{p_i p_j} \mp \\
&\quad \ldots \pm \frac{m}{p_i \cdots p_r} \\
&= m + m \left(-\sum_{i=1}^{r} \frac{m}{p_i} + \sum_{i<j} \frac{m}{p_i p_j} \mp \ldots \pm \frac{m}{p_i \cdots p_r}\right) \\
&= m \left(1 - \sum_{i=i}^{r} \frac{1}{p_i} + \sum_{i<j} \frac{1}{p_i p_j} \mp \ldots \pm \frac{1}{p_1 \ldots p_r}\right) \\
&= m \prod_{\substack{p \mid m \\ p \text{ prim}}} \left(1 - \frac{1}{p}\right)
\end{aligned}
$$

□

Korollar 2.2. *Ist die Primfaktorzerlegung $m = p_1^{k_1} \cdots p_r^{k_r}$ bekannt, so lässt sich $\varphi(m)$ in Zeit $\mathcal{O}(?)$ berechnen.*

Übung 2.2. Bestimme die Laufzeit zur Berechnung von $\varphi(m)$ in Korollar 2.2

Korollar 2.3. *Sind $p_1, \ldots, p_r$ paarweise verschiedene Primzahlen, so ist für $m = p_1 \cdots p_r$: $\varphi(m) = (p_1 - 1)(p_2 - 1) \cdots (p_r - 1)$.*

Beweis.

$$
\begin{aligned}
\varphi(m) &= m \prod_{p \mid m} (1 - \frac{1}{p}) = p_1 \cdots p_r (1 - \frac{1}{p_1}) \cdots (1 - \frac{1}{p_r} = p_1 (1 - \frac{1}{p_1}) \cdots p_r (1 - \frac{1}{p_r}) \\
&= (p_1 - 1)(p_2 - 1) \cdots (p_r - 1).
\end{aligned}
$$

Korollar 2.4. *Ist $(m, n) = 1$, so ist $\varphi(m \cdot n) = \varphi(m) \cdot \varphi(n)$.*

Beweis. Folgt aus Korollar 2.3, da m und n paarweise verschiedene Primfaktoren haben. □

Theorem 2.10. *Für $m \geq 1$ gilt: $m = \sum_{a|m} \varphi(a)$*

Beweis. Sei $m = p_1^{k_1} p_2^{k_2} \cdots p_r^{k_r}$ die Primfaktorzerlegung. Nun werden wir folgendes Produkt auf zwei verschiedene Weisen berechnen und auf diese Weise die gewünschte Identität erhalten:

$$
\begin{aligned}
&\prod_{i=1}^{r} \Big(\varphi(1) + \varphi(p_i) + \varphi(p_i^2) + \ldots + \varphi(p_i^{k_i})\Big) \\
&\stackrel{1.}{=} \prod_{i=1}^{r} \Big(1 + (p_i - 1) + (p_i^2 - p_i) + (p_i^3 - p_i^2) + \ldots + (p_i^{k_i} - p_i^{k_i - 1})\Big) \\
&= \prod_{i=1}^{r} p_i^{k_i} = m \\
&\stackrel{2.}{=} \sum_{\substack{0 \leq l_i \leq k_i \\ i=1,\ldots,r}} \varphi(p_1^{l_1}) \varphi(p_2^{l_2}) \cdots \varphi(p_r^{k_r}) = \sum_{\substack{0 \leq l_i \leq k_i \\ i=1,\ldots,r}} \varphi(p_1^{l_1} \cdots p_r^{l_r}) = \sum_{a|m} \varphi(a)
\end{aligned}
$$

□

Der kleine Satz von Fermat

Theorem 2.11. *Sei $m \geq 1$ und $a \in \mathbb{Z}$ mit $(a, m) = 1$. Dann ist $a^{\varphi(m)} \equiv 1 \pmod{m}$.*

Beweis. Der kleine Satz von Fermat folgt unmittelbar aus dem Satz von Lagrange. □

3

Die Enigma

Die Enigma war eine Verschlüsselungsmaschine, die im Zweiten Weltkrieg vom deutschen Militär eingesetzt wurde. Das Patent zur Enigma wurde 1918 von Arthur Scherbius eingereicht, später aber noch weiterentwickelt. Die Niederlande, Schweden und die USA hatten ähnliche Modelle.

Grundidee und Aufbau

Die Grundelemente der Enigma sind:

- 1 Steckbrett
- 3 bewegliche Rotoren
- 1 Reflektor

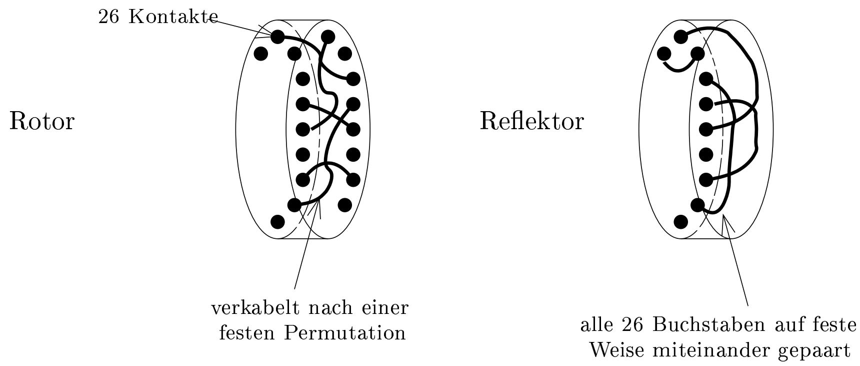


Abb. 3.1. Grundelemente der Enigma

Während die Verkabelungen in den Rotoren und im Reflektor nicht verändert werden können, gibt es auf dem Steckbrett die Möglichkeit, 6 (später

10) Buchstabentranspositionen festzulegen. Außerdem kann die Position der einzelnen Rotoren sowie ihre Anfangsstellung verändert werden. Nach jedem getippten Buchstaben drehen sich die Rotoren weiter, ähnlich einem Kilometerzähler. An den Rotoren sind dazu kleine Stifte angebracht, die den folgenden Rotor um eine Stelle weiterdrehen koennen.

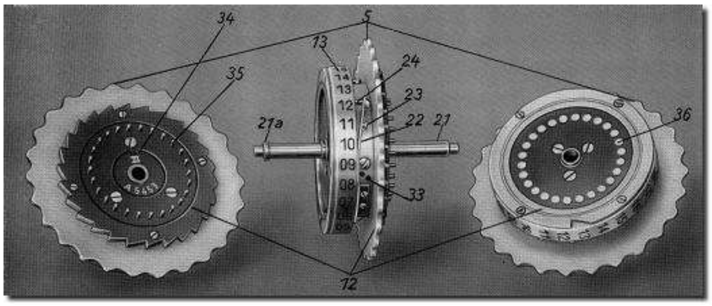


Abb. 3.2. Walzen der Rotor-Chiffriermaschine Enigma (Abbildung Deutsches Technikmuseum München).

Abb. 3.3. Eine Heeresenigma der Wehrmacht (Abbildung Deutsches Technikmuseum München).

Bedient wurde die Enigma üblicherweise von drei Personen: Einer saß an der Tastatur und schrieb die Nachricht, woraufhin für jeden getippten Buchstaben eine Lampe mit einem anderen Buchstaben aufleuchtete. Der zweite notierte die chiffrierte Botschaft und der dritte sendete diese. Wie man er-

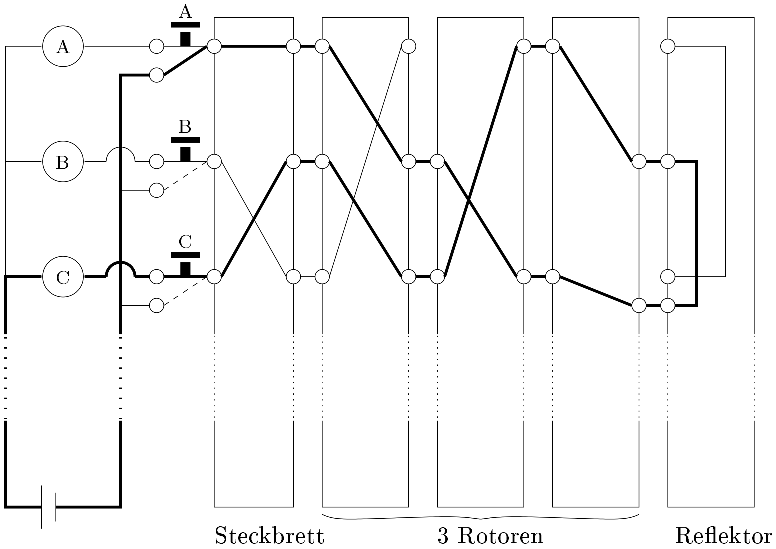


Abb. 3.4. Schematischer Aufbau der Enigma.

kennen kann, verschlüsselt die Enigma einen Buchstaben niemals durch den gleichen Buchstaben. Darüber hinaus sieht man, daß durch die Buchstabenpaarung im Reflektor ein Text genauso entschlüsselt werden kann wie er verschlüsselt wurde, sofern die gleiche Anfangsstellung verwendet wird.

Schlüsselzahl

Die Anzahl der Schlüssel $|K|$ lässt sich durch

- die Position der Rotoren ($3! = 6$),
- die Anfangsstellung der Rotoren ($26^3 = 17576$) und
- die Anzahl der Steckbretttranspositionen ($\binom{26}{12}\frac{12!}{2^6 6!} \approx 10^{11}$)

berechnen. Damit ergibt sich $|K| \approx 10^{16}$

Historisches zur Enigma

Nachdem Scherbius 1918 das Patent zur Enigma eingereicht hatte, wurde diese in Deutschland weiterentwickelt und vom Militär verwendet. Ab 1931 lieferte der deutsche Spion Hans-Thilo Schmidt Informationen über die Enigma an den französischen Geheimdienst. Zunächst teilte er diesem die Funktionsweise, die Bedienungsanleitung und die Art der Schlüsselweitergabe mit (diese wurden monatsweise in einem Heft notiert und durch Boten verteilt), später gab er unter anderem auch Tagesschlüssel weiter. Schmidt wurde 1943 gefasst und hingerichtet.

Die Franzosen erachteten den Code jedoch für unbrechbar und gaben die Informationen an den polnischen Geheimdienst weiter. Dort beschäftigte man bald Mathematiker, um den Enigma-Code zu brechen. Einer davon war Marian Rejewski, der in den 30er Jahren täglich Enigma-Nachrichten entschlüsselte, ohne die Schlüssel zu kennen, die der Geheimdienst von den Franzosen erhalten hatte, den Codebrechern aber vorenthielt.

1938 wurde die Enigma noch einmal verbessert: Statt 3 Rotoren gab es nun 5 Rotoren, von denen drei ausgewählt und verwendet werden konnten (Die Anzahl der möglichen Positionen der Rotoren erhöhte sich damit auf $5 \cdot 4 \cdot 3 = 60$.). Die Anzahl der Steckbretttranspositionen wurde ebenfalls von 6 auf 10 erhöht, die Geltungsdauer des Tagesschlüssels wurde von 24 Stunden auf 8 Stunden reduziert und es wurden weniger offensichtliche Fehler bei der Auswahl der Klartexte gemacht (siehe polnische Analysen).

Dies erschwerte die Arbeit für die polnischen Codebrecher, die außerdem an ihre finanziellen Grenzen stießen. Aus diesem Grund luden sie 1939 Franzosen und Engländer ein, um diesen ihre Fortschritte zu präsentieren und ihr Wissen weiterzugeben. Die Gäste waren von den Ergebnissen der Polen beeindruckt und als Ergebnis des Treffens wurde in England Bletchley Park errichtet, das Zentrum der alliierten Codebrecher im zweiten Weltkrieg. Die wahrscheinlich bekannteste Person, die dort wirkte, war Alan Turing.

Die meisten dieser Informationen sind erst seit den 70er Jahren öffentlich bekannt, da sie vorher als geheim eingestuft waren. Heute sind nur noch wenige Enigmas erhalten, viele wurden nach dem Krieg vernichtet.

Attacken auf die Enigma - Zwei Beispiele

Polnische Analysen

Die monatlich verteilten Tagesschlüssel wurden von den Deutschen verwendet, um Spruchschlüssel für die Anfangsstellung der Rotoren in verdoppelter Form zu übermitteln (z.B. fwtfwt, mit dem Tagesschlüssel chiffriert). Die Verdopplung diente als Sicherheit bei möglichen Übermittlungsfehlern. Vor dem Versenden der eigentlichen Nachricht wurden die Rotoren auf den Spruchschlüssel eingestellt.

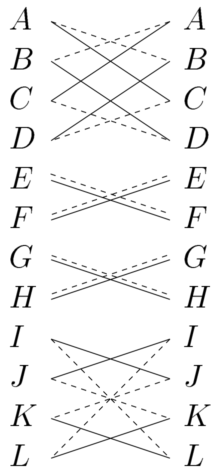
Wenn man nun bedenkt, daß die Polen eine Enigma nachgebaut hatten, die sie bei der Entschlüsselung der deutschen Nachrichten zur Hilfe nehmen konnten, war das Problem also, den jeweiligen Spruchschlüssel in Erfahrung zu bringen. Da man bei bekanntem Tagesschlüssel den Spruchschlüssel aber sofort herausfinden konnte, reduzierte sich das Problem auf die Suche nach dem Tagesschlüssel. Sobald man diesen herausgefunden hatte, konnte man sämtliche aufgefangenen Nachrichten des Tages dechiffrieren.

Vorgehensweise zur Bestimmung des Tagesschlüssels

Seien $P_1, \ldots, P_6$ die ersten sechs Permutationen, die zu einem festen Tagesschlüssel durchgeführt werden. Dann ist aufgrund der Konstruktion der Enigma $P_i x \neq x$ und $P_i^2 x = x$ für alle x.

Betrachte nun für $i \in \{1, 2, 3\}$ folgende Situation: $P_i a = x$ und $P_{i+3} a = y$. Dabei sei a ein Buchstabe des Spruchschlüssels (insbesondere ist a unbekannt), der durch P_i auf x und durch P_{i+3} auf y abgebildet wird (insbesondere sind x und y Teil des Chiffretextes und somit bekannt). Wegen obiger Betrachtung ist $P_i^2 a = a = P_i x$, also ist $P_{i+3} P_i x = y$.

Fängt man im Laufe eines Tages genügend Spruchschlüssel ab, erhält man auf diese Weise die kompletten Permutationen $P_{i+3} P_i,\ i \in \{1, 2, 3\}$. Aber was können wir mit diesem bereits erstaunlichen Wissen anfangen? Zunächst bemerken wir, dass die P_i Produkt elementefremder Transpositionen sind, und betrachten nun als Beispiel (für die Buchstaben A-L) die Permutationen $P_i = (A\,C)(B\,D)(E\,F)(G\,H)(I\,J)(K\,L)$ und $P_{i+3} = (A\,B)(\,CD)(E\,F)(G\,H)(I\,L)(J\,K)$. Der zugehörige (bipartite) Graph sieht so aus:



Die durchgezogenen Linien entsprechen den P_i, die gepunkteten den P_{i+3}. Wir betrachten nun die Zykeldarstellung von $P_{i+3} \circ P_i = (AD)(BC)(E)(F)(GIK)(HLJ)$. Offenbar treten alle Zykel in Paaren gleicher Länge auf; aus der Spiegelsymmetrie des Graphen folgt, dass dies für beliebige

Hintereinanderschaltungen $P_i \circ P_j$ gilt, wobei P_i und P_j nur aus Transpositionen bestehen. Die Steckbretttranspositionen haben übrigens keinen Einfluss auf die auftretenden Zykellängen in den $P_{i+3}P_i$, denn sie bewirken im Graph nur ein vertauschen der Labels der Ecken.

Im Folgenden wollen wir nun einige Berechnungen anstellen. Betrachten wir zuerst die drei Rotoren: Wir können diese in beliebiger Reihenfolge in die Maschine einsetzen, was uns insgesamt $3! = 6$ Möglichkeiten liefert. Dann kann jeder Rotor jeweils auf einen beliebigen Buchstaben gedreht werden; das macht für jeden Rotor 26 Möglichkeiten, insgesamt also $26^3 = 17576$ viele Möglichkeiten. Rechnet man das zusammen, erhält man $3! \cdot 26^3 = 105.456$ verschiedene Einstellmöglichkeiten für die Rotoren.

Als nächstes überlegen wir uns die Anzahl der Zahlpartitionen von 13; diese beträgt 101. Damit ergibt sich für die Anzahl der Möglichkeiten verschiedener Zykellängen für $P_4 \circ P_1$, $P_5 \circ P_2$ und $P_6 \circ P_3$ also $(101)^3 = 1.030.301$.

Dies gibt Anlass zu der Hoffnung, dass die Zykellängen bereits vollständigen Aufschluss über die Rotorreihenfolge und -anfangsstellung geben. Die polnischen Analysten haben daraufhin zu allen möglichen Rotorstellungen die Zykellängen katalogisiert. Dieser Katalog zusammen mit dem Wissen, welche Anfangsstellung der Rotoren bevorzugt waren, reichte aus, um die komplette Rotoreninformation zu erhalten. Die noch unbekannte Steckbretttransposition konnte dann meist durch Raten gefunden werden.

Britische Analysen

Einer der Schwachpunkte der Enigma ist die Fixpunktfreiheit der Permutationen, die die Enigma vollführt. Wusste man einen Teil des Klartextes (z.B. weil die um punkt sechs Uhr gesendete Nachricht mit großer Wahrscheinlichkeit der Wetterbericht war), so konnte man den Chiffretext mit der Known-Plaintext-Attacke bearbeiten. Abbildung 3.5 zeigt ein Beispiel aus Friedrich L. Bauer, *Kryptologie*, Springer, 1993.

U	L	O	E	B	**Z**	**M**	**G**	**E**	**R**	**F**	**E**	**W**	**M**	**L**	**K**	**M**	**T**	**A**	**W**	**X**	**T**	**S**	**W**	**V**	**U**	**I**	**N**	**Z**	P	R
o	b	e	r	k	o	m	m	a	n	d	o	d	e	r	w	e	h	r	m	a	c	h	t							
	o	b	e	r	k	o	m	m	a	n	d	o	d	e	r	w	e	h	r	m	a	c	h	t						
		o	b	e	r	k	o	m	m	a	n	d	o	d	e	r	w	e	h	r	m	a	c	h	t					
			o	b	e	r	k	o	m	m	a	n	d	o	d	e	r	w	e	h	r	m	a	c	h	t				
				o	b	e	r	k	o	m	m	a	n	d	o	d	e	r	w	e	h	r	m	a	c	h	t			
					o	**b**	**e**	**r**	**k**	**o**	**m**	**m**	**a**	**n**	**d**	**o**	**d**	**e**	**r**	**w**	**e**	**h**	**r**	**m**	**a**	**c**	**h**	**t**		
						o	b	e	r	k	o	m	m	a	n	d	o	d	e	r	w	e	h	r	m	a	c	h	t	
							o	b	e	r	k	o	m	m	a	n	d	o	d	e	r	w	e	h	r	m	a	c	h	t

Abb. 3.5. Positionsbestimmung eines vermuteten Klartextes.

Diese Tatsache war eine der wichtigsten Angriffspunkte für die britischen Analysten.

4

Perfekte Geheimhaltung

Alle bislang vorgestellten Kryptosysteme haben wir auf die ein oder andere Weise als unsicher erkannt. Dem Mathematiker stellt sich die Frage, was Sicherheit eigentlich ist, und ob ein sicheres Kryptosystem überhaupt existiert. Diese Fragen wurden erstmals rigoros von C. E. Shannon in den vierziger Jahren behandelt.

Die Theorie von Shannon

Seien P, C, K endliche Mengen, die zusammen mit E_k, D_k für $k \in K$ ein Kryptosystem bilden. Die Klartexte seien gemäß einer Wahrscheinlichkeitsverteilung verteilt mit Wahrscheinlichkeitsmaß Pr_P. Diese hängt von der Sprache und vom Kontext ab. Die Wahl des Schlüssels erfolge gemäß einer Verteilung mit Maß Pr_K, unabhängig von Pr_P. Definiere $Pr(p,k) = Pr_P(p) \cdot Pr_K(k)$ ein Wahrscheinlichkeitsmaß auf $P \times K$. Es gilt dann für festes $p \in P$

$$Pr(\{(p,k) \mid k \in K\}) = Pr_P(p),$$

und für $k \in K$

$$Pr(\{(p,k) \mid p \in P\}) = Pr_K(k).$$

Zur Vereinfachung führen wir folgende Notation ein: $p := \{(p,k) \mid k \in K\}$ und $k := \{(p,k) \mid p \in P\}$. Dann gilt also $Pr(p) = Pr_P(p)$ und $Pr(k) = Pr_K(k)$. Für $c \in C$ definiere das Ereignis $c = \{(p,k) \mid E_k(p) = c\}$.

Definition 4.1. *Das Kryptosystem heißt* perfekt geheim (perfekt sicher), *falls für alle $p \in P$ und $c \in C$ gilt: $Pr(p \mid c) = Pr(p)$.*

Beispiel 4.1. Sei $P = \{a, b\}$ die Klartextmenge mit der Wahrscheinlichkeitsverteilung $Pr_P(a) = \frac{1}{3}$ und $Pr_P(b) = \frac{2}{3}$. Der Schlüsselraum bestehe aus $K = \{0, 1\}$ mit den Wahrscheinlichkeiten $Pr_K(0) = \frac{1}{3}$ und $Pr_K(1) = \frac{2}{3}$. Die

Menge der Chiffretexte sei $C = \{A, B\}$. Die Verschlüssellungsfunktionen seien wie folgt definiert: $E_0(a) = A$, $E_0(b) = B$, $E_1(a) = B$, $E_1(b) = A$. Es folgt

$$\begin{aligned}
Pr(A) &= Pr(\{(a,0),(b,1)\}) = \frac{1}{3}\cdot\frac{1}{3}+\frac{2}{3}\cdot\frac{2}{3} = \frac{5}{9}\\
Pr(B) &= Pr(\{(a,1),(b,1)\}) = \frac{1}{3}\cdot\frac{2}{3}+\frac{2}{3}\cdot\frac{1}{3} = \frac{4}{9}\\
Pr(a \mid A) &= \frac{Pr(a\cap A)}{Pr(A)} = \frac{Pr(\{(a,0)\})}{\frac{5}{9}} = \frac{\frac{1}{9}}{\frac{5}{9}} = \frac{1}{5}.\\
Pr(b \mid A) &= \frac{4}{5};\quad Pr(a \mid B) = \frac{1}{2};\quad Pr(b \mid B) = \frac{1}{2}.
\end{aligned}$$

Das System ist also nicht perfekt sicher. Dies schlägt sich anschaulich darin nieder, dass wenn Eve das vermitteln von A belauscht, er mit 80%-iger Wahrscheinlichkeit davon ausgehen kann, dass b gesendet wurde.

Nun können wir uns dem Satz von Shannon zuwenden, der perfekt sichere Kryptosysteme charakterisiert.

Theorem 4.1 (Shannon (1949)). *Seien $|P| = |C| = |K| < \infty$ und $Pr_P(p) > 0$ für alle $p \in P$. Das Kryptosystem $(E_k)_{k\in K}$, $(D_k)_{k\in K}$ ist perfekt sicher genau dann, wenn die Wahrscheinlichkeitsverteilung auf K die Gleichverteilung ist und zu jedem $p \in P$, $c \in C$ genau ein $k \in K$ existiert mit $E_k(p) = c$.*

Beweis. Wir beweisen zuerst die Hinrichtung: Seien $p \in P$, $c \in C$ und angenommen, es existiert kein $k \in K$ mit $E_k(p) = c$; dann ist $Pr_P(p) = Pr(p) = Pr(p \mid c) = 0$, also Widerspruch. Für festes c ist also die Abbildung $P \to K$ mit $p \mapsto k(p)$ und $E_{k(p)}(p) = c$ eine Bijektion. Betrachte nun für $c \in C$:

$$Pr(p) = Pr(p \mid c) = \frac{Pr(c \mid p)\cdot Pr(p)}{Pr(c)} = \frac{Pr_K(k(p))\cdot Pr(p)}{Pr(c)};$$

es gilt also für alle p: $Pr_K(k(p)) = Pr(c) = \frac{1}{|K|}$.

Es bleibt noch die Rückrichtung zu zeigen: Sei $k(p,c) \in K$ der Schlüssel mit $E_{k(p,c)}(p) = c$; dann

$$Pr(p \mid c) = \frac{Pr(p)\cdot Pr(c \mid p)}{Pr(c)} = \frac{Pr(p)\cdot Pr_K(k(p,c))}{Pr(c)} = \frac{Pr(p)\cdot\frac{1}{|K|}}{Pr(c)}.$$

Nun ist $Pr(c) = Pr(\{(q,k) \mid E_k(q) = c\}) = \sum_{q\in P} Pr_P(q)\underbrace{Pr_K(k(q,c))}_{\frac{1}{|K|}} = \frac{1}{|K|}$.

□

Das Vernam-one-time-pad

Das im folgenden erklärte *One-time-pad* von Gilbert Vernam (1917) ist ein Kryptosystem mit perfekter Sicherheit. Seien $P = C = K = \mathbb{Z}_2^n$ für ein $n \in \mathbb{N}$. Zu einem Klartext $p \in P$ mit Schlüssel $k \in K$ definiere $E_k(p) := p \oplus k = (p_1 + k_1, \ldots, p_n + k_n)$ und $D_k(c) := c \oplus k$. Möchte Alice eine Nachricht verschlüsseln, so wählt sie einen Schlüssel $k \in K$ gleichverteilt aus K aus.

Korollar 4.1 (Shannon). *Ist $Pr(p) > 0$, so ist das one-time-pad perfekt sicher.*

Beweis. Es gilt $|P| = |C| = |K| < \infty$, und es ist $Pr(p) > 0$ nach Voraussetzung. Wir müssen noch zeigen, dass für alle $p \in P$ und $c \in C$ genau ein $k \in K$ existiert mit $E_k(p) = c$. Für dieses k muß $k = p \oplus (p \oplus k) = p \oplus c$ gelten. Also existiert es und ist eindeutig bestimmt. □

Pseudozufallszahlen

Das One-time-pad verlangt also, dass Alice in der Lage ist aus der Menge der Schlüssel einen mit Gleichverteilung herauszuziehen, beziehungsweise einen Schlüssel gleichverteilt zu erzeugen. Wie macht sie das? Dies rührt an die philosophische Frage, ob es Zufall überhaupt gibt. Anstatt uns auf diese Diskussion einzulassen, werden wir einige Eigenschaften, die wir von einer zufällig gewählten Zahlenfolge intuitiv erwarten, in den Rang einer Definition erheben. Wir stützen uns hier auf das folgende Werk von Donald Knuth, welches als Referenz empfohlen sei.

- D.E. Knuth, *The art of computer programming, Vol. 2*, Addison–Wesley, 1969.

Um einordnen zu können auf was für Wege wir uns begeben wollen, zitieren John von Neumann:

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

Unendliche Folgen

Gegeben sei eine Folge von Zahlen $(X_i)_{i \geq 0}$, beispielsweise $X_i \in \mathbb{R}, [0,1]$, $\mathbb{Z}$, $\{0, 1, \ldots, b-1\}$. Letztere werden wir als b-adische Folgen bezeichnen. Wann betrachten wir $(X_i)_{i \geq 0}$ als zufällig?

Definition 4.2. *Sei $S(n)$ eine mögliche Eigenschaft von X_n. Dann definieren wir für $\nu(n) = |\{j \leq n \mid S(j)\}|$, das*

$$Pr(S(n)) = \lambda :\Leftrightarrow \lim_{n \to \infty} \frac{\nu(n)}{n} = \lambda.$$

Definition 4.3. *Eine b-adische Folge heißt* k-verteilt, *falls für alle* $x_1, \ldots, x_k \in \{0, 1, \ldots, b-1\}$ *gilt:*

$$Pr(X_n X_{n+1} \cdots X_{n+k-1} = x_1 \cdots x_k) = \frac{1}{b^k}.$$

Bemerkung 4.1. k-verteilte b-adische Folgen sind auch $(k-1)$-verteilt für $k \geq 2$.

Definition 4.4. *Eine Folge heißt* ∞-verteilt, *falls sie* k*-verteilt ist für alle* $k \geq 1$.

Bemerkung 4.2. Betrachte die durch π definierte 10-adische Folge $3, 1, 4, 1, 5, 9, 2, 6, 5, \ldots$ Ist diese ∞-verteilt? Tatsächlich ist es bis heute sogar unbekannt, ob diese Folge 1-verteilt ist.

Das nächste Theorem zeigt zumindest, dass ∞-verteilte Folgen existieren.

Theorem 4.2 (Franklin 1962). *Für fast alle* $\Theta \in \mathbb{R}$, $\Theta > 1$, *ist* $x_n = \lfloor(\Theta^n \mod 1) \cdot b\rfloor$ ∞*-verteilt.*

Es ist bekannt, dass ein Θ mit dieser Eigenschaft transzendent sein muss. Jedoch ist kein konkreter Wert für Θ bekannt. Es gibt jedoch konkret definierbare Folgen, die ∞-verteilt sind. Hierfür verweisen wir noch einmal explizit auf Knuth.

Endliche Folgen

Definition 4.5. *Sei* $X_1, \ldots, X_N$ *eine endliche b-adische Folge,* $S(n)$ *sei eine Eigenschaft, die* X_n *haben kann,* ν *wie oben definiert. Wir sagen*

$$Pr(S(n)) \approx \lambda :\Leftrightarrow \left|\frac{\nu(N)}{N} - \lambda\right| \leq \frac{1}{\sqrt{N}}.$$

Definition 4.6. *Eine endliche b-adische Folge* $X_1, \ldots, X_N$ *heißt* k-verteilt, *falls für alle* $x_1, \ldots, x_k \in \{0, \ldots, b-1\}$ *gilt:*

$$Pr(X_n X_{n+1} \cdots X_{n+k-1} = x_1 \cdots x_k) \approx \frac{1}{b^k}.$$

Bemerkung 4.3. Beachte, daß in diesem endlichen Fall eine k-verteilte Folge nicht notwendigerweise $(k-1)$-verteilt sein muß.

Wir wollen uns nun an folgende Definition einer zufälligen endlichen Folge halten. Selbstverständlich hat diese Definition ihre Mängel für dessen Diskussion wir auf Knuth verweisen.

Definition 4.7. *Eine b-adische Folge der Länge* N *heißt* zufällig, *falls sie* k*-verteilt ist für alle* $k \leq \log_b N$.

Lineare Schieberegisterfolgen über GF(q)

Nachdem wir eine Definition einer zufälligen Folge zur Verfügung haben stellt sich die Frage nach der Existenz solcher Folgen, und insbesondere wie wir solche (deterministisch) erzeugen können. Wir wollen hier lineare Schieberegisterfolgen betrachten. Als Literatur liegt diesem Abschnitt das folgende Buch zugrunde.

- Heinz Lüneburg, *Galoisfelder, Kreisteilungskörper und Schieberegisterfolgen*, B.I.-Wissenschaftsverlag, 1979.

Bei einem Schieberegister handelt es sich um eine Apparatur mit n nebeneinander liegenden Fächern, in denen sich jeweils eine Zahl aus GF(q) befindet. Die Fächer sind mit einer Startkonfiguration an Zahlen belegt; ferner existiert eine Berechnungsvorschrift, die aus den Zahlen aller Fächer eine neue Zahl generiert. Möchte man eine zufällige Folge mithilfe des Registers erzeugen, so berechnet man zuerst eine Zahl m anhand dieser Berechnungsvorschrift. Siehe auch Abbildung 4.1. Dann entnimmt man die Zahl aus dem letzten Fach (und betrachtet diese als nächstes Glied in der Zufallsfolge) und verschiebt nun die Zahl aus dem vorletzen Fach in das letzte (und nun freie) Fach, die Zahl aus dem vorvorletzten Fach in das vorletzte etc., bis man alle Zahlen ein Fach weiter verschoben hat. Zuletzt fügt man die anfangs berechnete Zahl m in das erste Fach ein, und hat damit das Register wieder für die Entnahme einer weiteren Zufallszahl bereit gemacht. Besteht die Berechnung von m aus einer linearen Funktion in Abhängigkeit der Registereinträge, so spricht man von einem *linearen* Schieberegister.

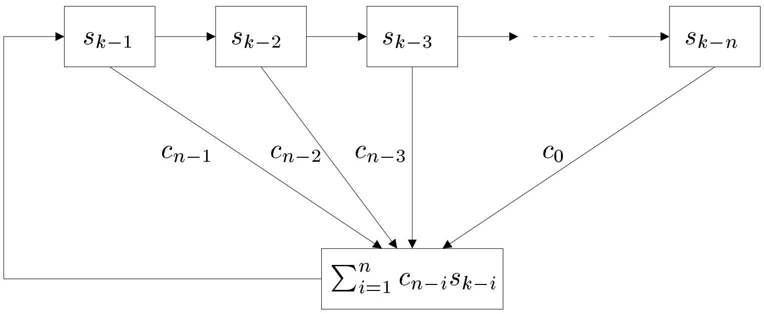


Abb. 4.1. Prinzip eines linearen Schieberegisters

Mathematisch fassen wir ein lineares Schieberegister als eine lineare Rekursion der Länge n auf. Seien der Startvektor $s = (s_{n-1}, s_{n-2}, \ldots, s_0)$ und der Koeffizientenvektor $c = (c_{n-1}, c_{n-2}, \ldots, c_0)$ gegeben. Konstruiere nun eine Folge $(s_i)_{i\geq 0}$ mit $s_k = \sum_{i=1}^{n} c_{n-i}s_{k-i}$ für $k \geq n$. Ohne Einschränkung sei $c_0 \neq 0$. Es sei $S(x) = \sum_{i\geq 0} s_i x^i$ die *erzeugende Funktion* von $(s_i)_{i\geq 0}$. Damit ist

$$\begin{aligned}
S(x) &= \sum_{k=0}^{n-1} s_k x^k + \sum_{k\geq n} s_k x^k \\
&= \sum_{k=0}^{n-1} s_k x^k + \sum_{k\geq n} (\sum_{i=1}^{n} c_{n-i} s_{k-i}) x^k \\
&= \sum_{k=0}^{n-1} s_k x^k \quad + \quad c_{n-1} x \sum_{k\geq n} s_{k-1} x^{k-1} \quad + \quad c_{n-2} x^2 \sum_{k\geq n} s_{k-2} x^{k-2} \\
&\quad + \quad \cdots \quad + \quad c_1 x^{n-1} \sum_{k\geq n} s_{k-n+1} x^{k-n+1} \quad + \quad c_0 x^n \sum_{k\geq n} s_{k-n} x^{k-n} \\
&= \sum_{k=0}^{n-1} s_k x^k \quad + \quad c_{n-1} x \left(S(x) - \sum_{k=0}^{n-2} s_k x^k \right) \quad + \quad c_{n-2}^2 \left(S(x) - \sum_{k=0}^{n-3} s_k x^k \right) \\
&\quad + \quad \ldots \quad + \quad c_1 x^{n-1} \left(S(x) - s_0 x^0 \right) \quad + \quad c_0 x^n \cdot S(x).
\end{aligned}$$

Also $S(x)(1 - c_{n-1}x - c_{n-2}x^2 - \ldots - c_1 x^{n-1} - c_0 x^n) = g_s(x)$, ein Polynom vom Grad $\leq n-1$. Wir setzen $f(x) = 1 - c_{n-1}x - c_{n-2}x^2 - \ldots - c_1 x^{n-1} - c_0 x^n$ als das Polynom, welches die Rekursion definiert. Als Ergebnis erhalten wir wegen $c_0 \neq 0$: $S(x) = \frac{g_s(x)}{f(x)}$ mit $\operatorname{grad} f = n$ und $\operatorname{grad} g_s < n$.

Übung 4.1. Sei $h(x)$ ein Polynom vom Grad $\leq n-1$ über GF(q). Zeige: Es existiert ein Startvektor S mit $g_s(x) = h(x)$.

Periodenlänge einer Schieberegisterfolge

Starten wir mit $s_{n-1} = \ldots = s_0 = 0$, so wird die resultierende Schieberegisterfolge konstant 0. Sei also $s = (s_{n-1}, \ldots, s_0) \neq 0$, so betrachte die *Registerfolgen* $s(k) = (s_{k+n-1}, s_{k+n-2}, \ldots, s_k)$. Da es nur $q^n - 1$ Folgen der Länge n über $GF(q)$ ungleich Null gibt, existiert ein $0 < r < q^n - 1$ mit $s(k+r) = s(k)$ für ein $k \geq 0$. Wie wir sehen werden gilt dann für alle $k \geq 0$, daß $s(k+r) = s(k)$. Wie geht nun $s(k+1)$ aus $s(k)$ hervor? Aus Abbildung 4.1 liest man folgendes ab.

$$s(k+1) = \underbrace{\begin{pmatrix} c_{n-1} & c_{n-2} & \cdots & \cdots & c_0 \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 & 0 \\ 0 & \cdots & 0 & 1 & 0 \end{pmatrix}}_{:=A} s(k)$$

Wegen $\det A = (-1)^{n-1} c_0 \neq 0$ ist die Matrix A also invertierbar. Die Folge der $s(k)$ ist $s(0), A \cdot s(0), A^2 \cdot s(0), \ldots$

Wir hatten bereits festgestellt, daß ein $i < j$ mit $A^i \cdot s(0) = A^j \cdot s(0)$ existiert, sodaß $0 < r := j - i \leq q^n - 1$ gilt. Wegen der Invertierbarkeit von A folgt also $s(0) = A^r \cdot s(0)$, und insbesondere $s_{i+r} = s_i$ für alle $i \geq 0$.

Definition 4.8. *Dieses so definierte r heißt (eine) Periode des Schieberegisters. Die kleinste Periode nennen wir*

$$\mathrm{per}(s) = \min\{r : r \text{ ist eine Periode des Schieberegisters mit Startwert } s\}.$$

Für die erzeugende Funktion $S(x)$ gilt also

$$\begin{aligned} S(x) &= (s_0 + s_1 x + \dots + s_{r-1}x^{r-1})(1 + x^r + x^{2r} + x^{3r} + \dots) \\ &= \frac{s_0 + s_1 x + \dots + s_{r-1}x^{r-1}}{1 - x^r}, \end{aligned}$$

wobei wir in der zweiten Gleichung die Identität $\sum_{i \geq 0}(x^r)^i = \frac{1}{1-x^r}$ für die geometrische Reihe eingesetzt haben. Es gilt also

$$(1 - x^r)S(x) = s_0 + s_1 x + \dots + s_{r-1}x^{r-1} =: s(x)$$

und somit folgt aus $S(x) = \frac{g_s(x)}{f(x)}$ die wichtige Gleichung

$$f(x)s(x) = g_s(x)(1 - x^r).$$

Definition 4.9. $\exp(f) = \min\{t : f \mid 1 - x^t\}$ *heißt der Exponent von f.*

$\exp(f)$ ist wohldefiniert wegen des folgenden Theorems.

Theorem 4.3. *Es existiert immer ein t mit $f \mid 1 - x^t$ und zwar bereits ein $t \leq q^n - 1$.*

Beweis. Sei $S(x) = \frac{1}{f(x)}$. Dies definiert nach Übung 4.1 eine Schieberegisterfolge $(s_i)_{i \geq 0}$, welche eine Periode $r \leq q^n - 1$ besitzt. Das heißt $S(x) = \frac{1}{f(x)} = \frac{s(x)}{1-x^r}$, also $f(x)s(x) = 1 - x^r$. $\square$

Theorem 4.4. *Falls gilt $f \mid 1 - x^t$, so hat jede Schieberegisterfolge, die durch f definiert ist t als Periode.*

Beweis. Da $f \mid 1 - x^t$ gilt, existiert also ein Polynom $h(x)$ mit $f(x)h(x) = 1 - x^t$. Also gilt

$$S(x) = \frac{g_s(x)}{f(x)} = \frac{g_s(x)h(x)}{1 - x^t}.$$

Damit hat die Schieberegisterfolge die Periode t, da

$$\begin{aligned} t &= \mathrm{grad}(fh) = \mathrm{grad}\, f + \mathrm{grad}\, h \\ &> \mathrm{grad}\, g_s + \mathrm{grad}\, h = \mathrm{grad}(g_s h). \end{aligned}$$

$\square$

Korollar 4.2. *Es gilt* $\mathrm{per}(s) \leq \exp(f) \leq q^n - 1$ *für alle* $s \in K^n \setminus \{0\}$.

Korollar 4.3. *Ist* f *irreduzibel, so gilt* $\mathrm{per}(s) = \exp(f)$.

Beweis. Da $f(x)s(x) = g_s(x)(1 - x^{\mathrm{per}(s)})$ und $\mathrm{grad}\, g_s < \mathrm{grad}\, f$ gilt also $f \mid 1 - x^{\mathrm{per}(s)}$, und deshalb $\exp(f) \leq \mathrm{per}(s)$.

Wir fassen unsere Erkenntnisse in folgendem Theorem zusammen.

Theorem 4.5. *Jedes irreduzible Polynom* f *mit* $\exp(f) = q^n - 1$ *und* $f(0) = 1$ *definiert zu jedem Startvektor* $s \neq 0$ *eine Schieberegisterfolge mit Periode* $q^n - 1$. □

Beispiel 4.2. Seien $q = 2$, $n = 3$ und entsprechend $q^n - 1 = 7$. In diesem Fall gibt es zwei Polynome f von der obigen Form:

$$f_1(x) = x^3 + x + 1,$$
$$f_2(x) = x^3 + x^2 + 1.$$

Es gilt $(x^7 + 1) = (x^3 + x + 1)(x^4 + x^2 + x + 1)$. f_1 definiert das lineare Schieberegister mit $c_0 = 1, c_1 = 0$ und $c_2 = 1$.

Es sei $s = (1, 1, 1) \neq 0$ ein Startvektor, so ergibt sich die folgende Schieberegisterfolge (in absteigender Folge der Indizes):

$$\cdots 00101110010111,$$

wobei wir zwei ganze Perioden ausgeschrieben haben. Beginnt man mit einem anderen Startvektor, beispielsweise $s = (0, 1, 0)$, so ergibt sich nur eine Verschiebung:

$$\cdots 11100101110010.$$

Anmerkung 4.1. Zu gegebenen n existieren $\frac{\varphi(q^n-1)}{n}$ viele irreduzible Polynome von Grad n mit $f(0) = 1$ und $\exp(f) = q^n - 1$. Diese f sind die irreduziblen Faktoren des Kreisteilungspolynoms Φ_{q^n-1}. Das m*-te Kreisteilungspolynom* $\Phi_m(x)$ ist gegeben durch $\Phi_m(x) = \prod_\lambda (x - \lambda)$, wobei das Produkt über alle primitive m-te Einheitswurzeln λ gebildet wird.

Daß wir tatsächlich im Sinne unserer Definition von Zufallsfolge erreicht haben, was wir angestrebt haben klärt sich mit der folgenden Übung.

Übung 4.2. Beweise, daß eine Schieberegisterfolge über GF(2) mit Rekursionslänge n und Periodenlänge $2^n - 1$ zufällig im Sinne von Definition 4.7 ist.

5

Der Data-Encryption-Standard

Unser bisheriges Wissen beschert uns mit Kryptoverfahren, die unsicher sind, wie zum Beispiel die Enigmaverschlüsselung, oder mit Verfahren, die sicher aber sehr unpraktisch sind, wie dem One-time-pad.

Deshalb wollen wir uns nun mit einem Verfahren beschäftigen, welches für lange Zeit eingesetzt wurde, in modifizierter From auch noch eingesetzt wird, und einen guten Kompromiß darstellt in Punkto Sicherheit und Praktikabilität: dem Data-Encryption-Standard (DES).

Ausgangspunkt für die Entwicklung des DES war ein Wettbewerb, den das National Bureau of Standards der U.S.A. 1973/1974 ausschrieb. Der Gewinner sollte dann als Standard in den U.S.A. und darüberhinaus eingeführt werden. Eingeführt wurde der Standard DES im Jahre 1977.

Der Vorschlag der sich durchsetzte wurde von IBM entwickelt und von der N.S.A. (National Security Agency, da die USA diesen Geheimdienst lange verleugnet hat, auch als "No Such Agency" benannt) überarbeitet.

Diskussionspunkte

Der Verlauf des Verfahrens hat die Fachwelt zu regen Diskussionen veranlaßt. Bevor wir den DES erklären, wollen wir auf diese Diskussion eingehen.

Das Verfahren

- Entwurfskriterien wurden nie veröffentlicht.
- Anzahl der Schlüssel bei IBM 2^{128} nach der NSA-Überarbeitung nur noch 2^{56}.
- Die NSA hat ihre Überarbeitung damit gerechtfertigt, daß IBM eine Hintertür eingearbeitet haben könnte.
- Es liegt die Vermutung nahe, daß die NSA bereits Mitte der siebziger Jahre in der Lage war den ganzen Schlüsselraum in überschaubarer Zeit zu durchsuchen.

Ist ein Standard sinnvoll?

Pro:	Contra:
• Verbreitung der Nutzung kryptografischer Verfahren • durch hohe Stückzahlen an Spezialhardware preiswert	• ein Standardverfahren ist ein sehr lohnendes Ziel für Angreifer • kann wegen der weit verbreiteten Spezialhardware kostspielig werden, falls das Verfahren gebrochen wird

Feistel-Chiffre

Der DES ist eine Feistel-Chiffre, benannt nach Horst Feistel, damals Mitarbeiter von IBM. Um den DES verstehen zu können, gehen wir zunächst auf die Feistel-Chiffre ein.

Sei $\Sigma = \{0, 1\}$ und seien $f_k : \Sigma^t \longrightarrow \Sigma^t$ Verschlüsselungsfunktionenen zu Schlüsseln k.

Die Feistel-Chiffre ist eine Blockchiffre der Länge $2t$ über Σ. Zu gegebenen Schlüssel $k \in K$, dabei sei K die Schlüsselmenge der Feistelchiffre, werden Rundenschlüssel $k_1, \ldots, k_r$ erzeugt, die zum Schlüsselraum der (f_k) gehören. Nun sei $p = (L_0, R_0) \in \Sigma^t \times \Sigma^t$. Konstruiere eine Folge $((L_i, R_i))_{i=1,\ldots,r}$, definiert durch

$$(L_i, R_i) := (R_{i-1}, L_{i-1} \oplus f_{k_i}(R_{i-1})).$$

Setze für die Verschlüsselungsfunktion der Feistelchiffre

$$E_k = E_k(L_0, R_0) = (L_r, R_r).$$

Um das Dechiffrieren zu untersuchen berechne

$$L_{i-1} = (\underbrace{L_{i-1} \oplus f_{k_i}(R_{i-1})}_{R_i}) \oplus f_{k_i}(\underbrace{R_{i-1}}_{L_i}).$$

Damit gilt $(R_{i-1}, L_{i-1}) = (L_i, R_i \oplus f_{k_i}(L_i))$. Dechiffriert wird also durch Anwendung desselben Verfahrens auf (L_r, R_r) mit umgekehrter Reihenfolge der Rundenschlüssel.

Das DES-Kryptosystem

Grundlegende Definitionen

Wir setzen $P = C = \Sigma^{64}$, $K \subseteq \{0, 1\}^{64}$ mit

$$K := \{(b_1, \ldots, b_{64}) \subseteq \{0,1\}^{64} : \sum_{i=1}^{8} b_{8k+i} \equiv 1 \pmod 2,\ k = 0, \ldots, 7\}.$$

Damit ergibt sich $|k| = 2^{56} \approx 7,2 \cdot 10^{16}$.

Erzeugung der 16 Rundenschlüssel

Sei $k \in K$. Generiere $k_1, \ldots, k_{16} \in \{0,1\}^{48}$. Dazu setze

$$v_i := \begin{cases} 1 & i = 1, 2, 9, 16 \\ 2 & \text{sonst} \end{cases},$$

sowie zwei weitere Abbildungen

$$PC1 : \{0,1\}^{64} \longrightarrow \{0,1\}^{28} \times \{0,1\}^{28}$$
$$PC2 : \{0,1\}^{28} \times \{0,1\}^{28} \longrightarrow \{0,1\}^{48}$$

Die Abbildungen $PC1$ und $PC2$ können wir aus den folgenden beiden Tabellen entnehmen. Dabei sind die Tabellen so zu lesen, daß für $PC1$ sich die Zuordnung $(x_1, \ldots, x_{64})$ gemäß der Tabelle 5.1 auf $(x_{57}, x_{49}, \ldots), (x_{63}, x_{55}, \ldots)$ ergibt und in $PC2$ gemäß der Tabelle 5.2 $(x_1, \ldots x_{28}), (x_{29}, \ldots, x_{56})$ auf $(x_{14}, x_{17}, \ldots, x_{32})$ abgebildet wird.

Die beiden Abbildungen $PC1$ und $PC2$ werden jetzt wie folgt auf k zur Generierung der $k_1, \ldots, k_{16}$ angewandt:

1. Setze $(C_0, D_0) := PC1(k)$.
2. Für jedes $1 \leq i \leq 16$ sei C_i aus C_{i-1} durch zyklische Vertauschung um v_i nach links (rotiert) und D_i durch zyklische Vertauschung um v_i nach rechts aus D_{i-1} hervorgegangen. Dann ist $k_i = PC2(C_i, D_i)$.

$PC1$							$PC2$						
57	49	41	33	25	17	9	63	55	47	39	31	23	15
1	58	50	42	34	26	18	7	62	54	46	38	30	22
10	2	59	51	43	35	27	14	6	61	53	45	37	29
19	11	3	60	52	44	36	21	13	5	28	20	12	4

Tabelle 5.1. Definition der Abbildung $PC1$

Die initiale Permutation

Sei $p \in \{0,1\}^{64} = P$. Wir zerlegen p in zwei Worte der Länge 32: Gemäß der Tabelle 5.3 lesen wir zu jeder zweiten Spalte diese von unten nach

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Tabelle 5.2. Definition der Abbildung $PC2$

oben und erhalten, falls wir mit der zweiten Spalte beginnen die vier Tupel $(58, 50, \ldots, 10, 2)$, $(60, 52, \ldots, 4)$, $(62, \ldots, 6)$ und $(64, \ldots, 8)$. Beginnen wir mit der ersten Spalte erhalten wir die vier Tupel $(57, \ldots, 1)$, $(59, \ldots, 3)$, $(61, \ldots, 5)$ und $(63, \ldots, 7)$. Diese vier Tupel werden jeweils direkt hintereinander geschrieben, sodaß wir zwei 32-Tupel erhalten. Damit wird die initiale Permutation

$$IP(p) := ((p_{58}, \ldots, p_8), (p_{57}, \ldots, p_7)) = (R_0, L_0)$$

definiert.

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	...	...					
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
57	58	59	60	61	62	63	64

Tabelle 5.3. Zur Definition der initialen Permutation IP

Die interne Blockchiffre f_k

Sei $k \in \{0,1\}^{48}$ ein Rundenschlüssel und $R \in \{0,1\}^{32}$ ein rechtes Wort (wie bei den Feistel-Chiffren definiert). Dann wird $f_k(R)$ gemäß der Abbildung 5.1 definiert.

Dabei sind die Abbildungen in der schematischen Darstellung wie folgt definiert:

Die Abbildung $E(R)$

Wir definieren $E : \{0,1\}^{32} \longrightarrow \{0,1\}^{48}$ gemäß der folgenden schematischen Darstellung 5.2.

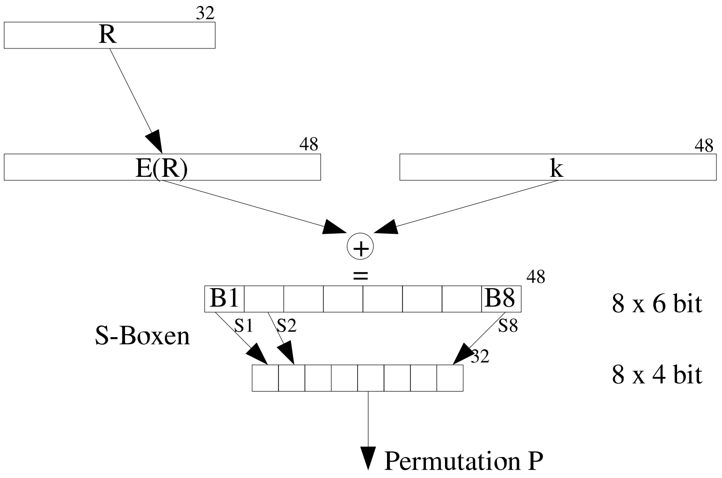


Abb. 5.1. Schematische Darstellung der internen Blockchiffre f_k

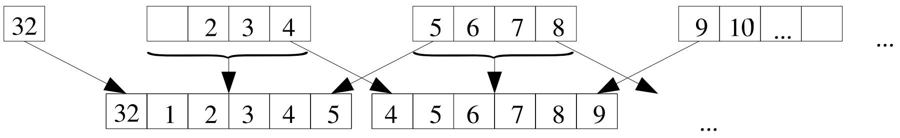


Abb. 5.2. Schematische Darstellung der Abbildung E

Die *S*-Boxen

Schreibe jedes 6-Bit Wort B_i aus der Darstellung 5.1 als

$$b_1 b_2 \dots b_6.$$

Dann lesen wir den Zeilenindex durch $b_1 b_6$ ab und den Spaltenindex durch $b_2 b_3 b_4 b_5$. Der Zeilenindex ist ein 2-Bit Wort, also aus der Menge $\{0, 1, 2, 3\}$, der Spaltenindex ist aus der Menge $\{0, 1, \dots, 15\}$. Mittels der Tabelle 5.4, welche die S-Boxen definiert, können wir mittels Zeilen- und Spaltenindex ein 4-Bit Wort aus der Menge $\{0, 1, \dots, 15\}$ ablesen.

Alle so entstehenden 8 4-Bit Worte binär hintereinander notiert ergeben ein Element der Menge $\{0, 1\}^{32}$.

Die DES-Permutation *P*

Das aus der Anwendung der S-Boxen entstehende 32-Tupel $r = (r_1, \dots, r_{32})$ wird gemäß der Tabelle 5.5 permutiert zu $P(r) = (r_{16}, r_7, \dots, r_{25})$.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S1 : 0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S2 : 0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S3 : 0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S4 : 0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S5 : 0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S6 : 0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S7 : 0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	7	12
S8 : 0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Tabelle 5.4. Die S-Boxen des DES

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Tabelle 5.5. Definition der DES-Permutation P

Die Ausgabe des DES

Ausgegeben werden nicht die sich aus den internen Blockchiffre ergebenen (L_{16}, R_{16}), sondern $C := IP^{-1}((R_{16}, L_{16}))$.

Entschlsseln einer DES-verschlüsselten Nachricht

Tripel-DES

Wir starten mit einer wichtigen Beobachtung: Zu zwei DES-Schlüsseln k_1, k_2 existiert im allgemeinen kein k_3 mit

$$\mathrm{DES}_{k_2} \cdot \mathrm{DES}_{k_1} = \mathrm{DES}_{k_3}.$$

Es konnte gezeigt werden, daß die multiplikative Gruppe erzeugt durch $\{\mathrm{DES}_k : k \in K\}$ mindestens 10^{2499} Elemente hat. Damit kann man das DES-Verfahren mehrfach hintereinander mit verschiedenen Schlüsseln anwenden und erhält tatsächlich neue Verschlüsselungsfunktionen. Beim Tripel-DES wählt man drei Schlüssel $k_1, k_2, k_3 \in K$ und betrachtet als Verschlüsselungsfunktion die Abbildung

$$p \longmapsto E_{k_3}(D_{k_2}(E_{k_1}(p))) = c.$$

Oft wählt man nicht drei verschiedene Schlüssel, sondern setzt $k_1 = k_3$. Damit erzielt man eine Quadrierung des Schlüsselraumes.

Sicherheit des DES

Der Data-Encryption-Standard wird heute noch benutzt, jedoch als Tripel-DES, welcher eine Verbesserung darstellt, da der Schlüsselraum vergrößert wurde. Im Jahre 1993 war es bereits möglich für ca. eine Millionen Dollar einen Spezialcomputer zu bauen, der innerhalb von sieben Stunden den gesamten Schlüsselraum durchsuchen kann. Aus diesem Grund kann man das DES-Verfahren heute nicht mehr als sicher bezeichnen.

Der Advanced Encryption Standard

Der Standard DES wurde im Herbst 2001 vom AES (Advanced Encryption Standard) abgelöst, der wieder in einem Wettbewerb, ausgeschrieben vom National Institute of Standards and Technology (NIST), ausgewählt wurde. In diesem Wettbewerb sind alle Entwurfskriterien dokumentiert und öffentlich einsehbar. Das von der deutschen Telekom ins Rennen geschickte Kryptosystem *Magenta* wurde bereits auf der ersten AES-Konferenz als unsicher erkannt und schied aus dem Wettbewerb aus. In diesem Text wollen wir nicht ausführlich auf AES eingehen. Einige wichtige Bemerkungen zu AES sollen aber hier noch erwähnt werden.

AES ist ein Chiffre mit $\Sigma = \{0,1\}$ und $P = C = \{0,1\}^{128}$. Der Schlüsselraum wird mit $K = \{0,1\}^m$ für $m = 128, 192, 256$ gewählt. Der Schlüsselraum ist also deutlich größer als beim DES. Auch der AES ist ein Rundenverfahren mit 10, 12 oder 14 Runden.

6

Public Key Kryptographie

Die grundlegende Idee

Bislang hatten alle Kryptoverfahren, die wir betrachtet haben, die Eigenschaft, daß die Schlüssel zum Ver- und Entschlüsseln identisch bzw. leicht auseinander zu berechnen waren. Dies hatte zur Folge, daß der Schlüssel geheim von Alice zu Bob gelangen musste. Das Problem des geheimen Übertragens des Schlüssels schien bis in die 70er Jahre hinein ein unlösbares Paradigma zu sein. Doch 1976 hatten indexDiffie, Whitfield Whitfield Diffie und Martin Hellman eine bahnbrechende Idee, um dieses Problem zu überwinden. Sie fanden eine Methode, mit der Nachrichten sicher übermittelt werden konnten, ohne daß zuvor ein Schlüssel ausgetauscht werden musste.

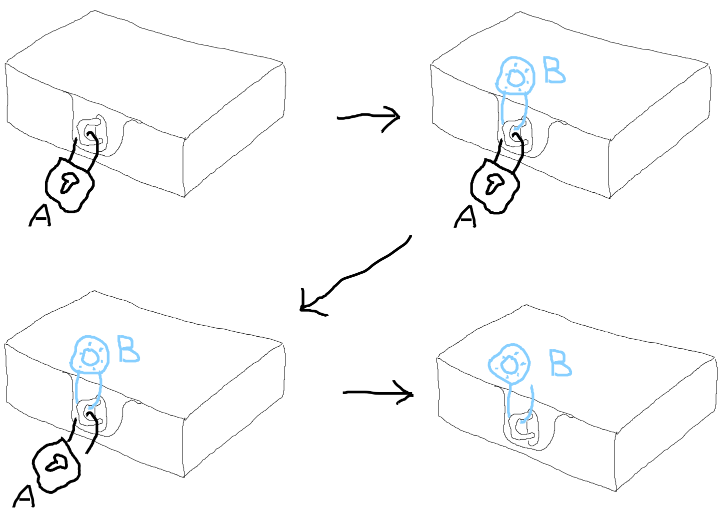


Abb. 6.1. Eine geheime Nachricht von Alice an Bob.

Alice behält also die ganze Zeit ihren Schlüssel, während Bob seinen behält.

Der Diffie-Hellman Schlüsselaustausch

Bob und Alice wollen einen Schlüssel austauschen, um ihn beispielsweise für den DES zu verwenden. Dazu einigen sie sich auf eine zyklische Gruppe (z.B. $G = (\mathbb{Z}_p \setminus \{0\}, \cdot)$) der Ordnung n mit einem Erzeuger g.

Nun wählt Alice zufällig ein $a \in \{1, 2, ..., n-1\}$ und berechnet $A = g^a$. Bob wählt ebenfalls zufällig ein $b \in \{1, 2, ..., n-1\}$ und berechnet $B = g^b$. Dann sendet Alice ihr A an Bob und dieser sein B an Alice. All dies geschieht öffentlich. Daraufhin berechnet Alice $K = B^a = g^{ab}$ und Bob $K = A^b = g^{ba}$. Sie haben also den gemeinsamen Schlüssel K ausgetauscht.

Effizienz des Diffie-Hellman Verfahrens

Sei $g \in G$ und e eine natürliche Zahl und $e = \sum_{i=0}^{k} e_i \cdot 2^i$ die Binärdarstellung von e. Dann ist

$$g^e = g^{\sum_{i=0}^{k} e_i 2^i} = \prod_{i=0}^{k} \left(g^{2^i}\right)^{e_i}$$

Da $e_i \in \{0, 1\}$, ist für $e_i = 1$

$$g^e = \prod_{i=0}^{k} g^{2^i}$$

Zur Berechnung von g^e benutzt man nun folgenden Algorithmus:

1. Berechne alle g^{2^i} mit $i = 0, ..., k$ rekursiv via $g^{2^{i+1}} = g^{2 \cdot 2^i} = \left(g^{2^i}\right)^2$.
2. Bestimme das Produkt aller g^{2^i} mit $e_i = 1$.

Theorem 6.1. *Ist e eine natürliche Zahl und $a \in \{0, ..., m-1\}$, so erfordert die Berechnung von a^e mod m eine Zeit von $\mathcal{O}((size(e))(size(m))^2)$ und Speicherplatz von $\mathcal{O}(size(e) + size(m))$*

Sicherheit des Diffie-Hellman Verfahrens

Betrachte die folgenden Angriffsmöglichkeiten:

1. Man-in-the-middle-attack:
   Ein Angreifer täuscht gegenüber Alice vor Bob zu sein und Bob gegenüber, daß er Alice ist.
2. Der "Lauscher" Eve hört zu und kennt G, g, A und B. Mit Hilfe dieser Informationen will er $K = g^{ab}$ berechnen.
   Dies ist das sogenannte Diffie-Hellman-Problem.

Bemerkung 6.1. Falls ein Angreifer a aus A und b aus B bestimmen kann, so kann er K berechnen. Die Umkehrung dieser Aussage ist bisher nicht bekannt.

Wie aber könnte ein Angreifer a aus A berechnen? Er benötigt dazu

Diskrete Logarithmen

Definition 6.1. *Sei G eine zyklische Gruppe der Ordnung n und $g \in G$ ein Erzeuger. Dann existiert zu jedem $A \in G$ ein Exponent $a \in \{0, ..., n-1\}$ mit $A = g^a$.*
Der Exponent a heißt diskreter Logarithmus von A zur Basis g in G.
geschrieben: $a = log_g A$

Beispiel 6.1. Sei q eine Primzahl und damit $G = (\mathbb{Z}_p \setminus \{0\}, \cdot)$ zyklisch. Betrachte nun $p = 13$ und $G = (\mathbb{Z}_{13} \setminus \{0\}, \cdot)$. Dann ist

A	1	2	3	4	5	6	7	8	9	10	11	12
$\log_2 A$	0	1	4	2	9	5	11	3	8	10	7	6

Es wird hierbei angenommen, daß die Berechnung von $\log_2 A$ in $G = (\mathbb{Z}_p \setminus \{0\}, \cdot)$ schwer ist. Dies muss allerdings nicht unbedingt der Fall sein, denn

Diskrete Logarithmen können leicht sein

Betrachten wir $G = (\mathbb{Z}_n, \odot)$, wobei $g \odot h := g + h$. Dann ist $g^a = g \cdot a$.

Nun sei $[g] = g + n\mathbb{Z}$ ein Erzeuger, d.h., $(g, n) = 1$, und $A \in G$. Gesucht ist $a = \log_{[g]} A$, d.h., gesucht ist ein $a \in \{0, ..., n-1\}$ mit

$$A \equiv g \cdot a \bmod n.$$

Nun gibt uns der erweiterte euklidische Algorithmus $x, y \in \mathbb{Z}$ mit $gx + ny = 1$, also $gx \equiv 1 \bmod n$.

Sei nun $a \in Ax + n\mathbb{Z}$, d.h., $a = Ax + nk$ mit $0 \leq a \leq n-1$. Dann ist

$$g(Ax + nk) = Agx + gnk \equiv A \bmod n.$$

Also ist in diesem Fall der diskrete Logarithmus in G einfach zu berechnen. Mit Algorithmen zur Bestimmung diskreter Logarithmen werden wir uns später noch eingehender beschäftigen.

Das Massey-Omura-Kryptosystem

Wir wollen nun noch ein weiteres kryptographisches Verfahren betrachten, welches auf der Diffie-Hellman-Idee beruht.

Sei K eine zyklische Gruppe der Ordnung $q - 1$, wobei q groß sein sollte. sei also z.B. $K = GF(q)$, q groß.

Alice wählt nun zufällig zwei Elemente $e_A, d_A \in K$ und Bob wählt ebenfalls zufällig $e_B, d_B \in K$ mit $0 < e < q-1$, $(e, q-1) = 1$ und $d = e^{-1} \bmod q - 1$.

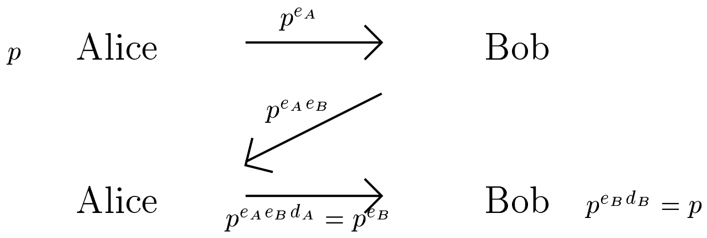
d lässt sich mit Hilfe des euklidischen Algorithmus berechnen:

$$x \cdot e + y(q-1) = 1$$

$$\Longrightarrow x \cdot e \equiv 1 \bmod q-1$$

$$\text{also } d := x$$

Alice codiert nun nach festgelegtem Verfahren ihre Nachricht als $p \in GF(q)$.



Dieses Verfahren entspricht also ganz genau dem Modell der Vorhängeschlösser.

Sicherheit des Massey-Omura-Kryptosystems

- Es besteht wieder die Gefahr einer Man-in-the-middle-attack. Daher sollte Bob seine Sendung an Alice signieren. Zur Signierung bzw. Autentifikation kommen wir später noch.
- Nach einer Weile kennt Bob viele Paare (P, P^{e_A}). Deshalb muss es schwer bleiben, aus den Paaren e_A zu berechnen. Dies basiert wieder auf Schwierigkeit, den diskreten Logarithmus zu bestimmen.

Das ElGamal-Kryptosystem

Wie zuvor sei K eine zyklische Gruppe der Ordnung $q-1$, sei also $K = GF(q)$, wobei q groß ist, $g \in K \setminus \{0\}$ ein Erzeuger. Bob wählt zufällig ein $b \in K$ mit $0 < b < q-1$. b bleibt geheim, während Bob aber $B = g^b$ veröffentlicht.

Alice habe eine Nachricht $p \in K \setminus \{0\}$ und wählt zufällig ein $0 < a < q-1$. Sie berechnet $g^a = A$ und $p \cdot B^a = c$ und sendet dieses Paar (A, c) an Bob. Dieser berechnet $x := q-1-b$, wobei $0 < x < q-1$ ist. Wenn er Alice' Nachricht empfangen hat, decodiert er

$$\begin{aligned} A^x \cdot c &= g^{a(q-1-b)} \cdot \left(g^b\right)^a \cdot p \\ &= \left(g^{q-1}\right)^a \cdot g^{-ab} \cdot g^{ab} \cdot p \\ &= p \end{aligned}$$

Effizienz des ElGamal-Kryptosystems

- In diesem Verfahren müssen zwei Exponentiationen, nämlich g^a und B^a, und eine Multiplikation $(p \cdot B^a)$ berechnet werden. Alice kann jedoch g^a und B^a vorberechnen und auf einer sicheren Umgebung, z.B. einer Chipkarte, speichern. Dann braucht man während des Verfahrens nur noch eine Multiplikation berechnen.
- Der große Nachteil von ElGamal ist die sogenannte Nachrichtenexplosion. Das zu verschickende Paar $(g^a, p \cdot B^a)$ ist doppelt so groß wie die eigentliche Nachricht.

Sicherheit des ElGamal-Kryptosystems

Wieder gilt hier: Um Sicherheit zu gewährleisten, muss das Diskrete-Logarithmus-Problem schwer zu lösen sein. Ferner gilt:

Das ElGamal-System zu brechen $\Longleftrightarrow$ Das Diffie-Hellman-Problem zu lösen.

Beweis. "$\Longleftarrow$": Angenommen, wir können das Diffie-Hellman-Problem lösen, d.h., wir können aus G, g, $A = g^a$ und $B = g^b$ den Schlüssel $K = g^{ab}$ berechnen. Nun aber nach einer ElGamal-Verschlüsselung G, g, $A = g^a$, $B = g^b$ und $c = p \cdot B^a = p \cdot g^{ab}$ bekannt. Berechne also g^{ab}, g^{-ab} und $p = g^{-ab} \cdot c$.

"$\Longrightarrow$": Sei nun ElGamal gebrochen, also G, g, $A = g^a$ und $B = g^b$ bekannt. Bestimme dann den Klartext p zu $c = 1$, also $p \cdot g^{ab} = 1 \Longrightarrow p^{-1} = g^{ab}$.

$\square$

7

Kryptographie in der Praxis

In diesem Kapitel wollen wir darauf eingehen, wie in einer praktischen Implementierung eines Kryptosystems die beteiligten algebraischen Strukturen implementiert und Schlüssel übergeben werden können.

Endliche Körper

Sei $K = GF(q)$, q Primpotenz und $G = (K \setminus \{0\}, \cdot)$ die Einheitengruppe.

Theorem 7.1. *Für jeden Teiler d von $q-1$ gibt es genau $\varphi(d)$ viele Elemente der Ordnung d.*

Beweis. Sei $d|q-1$, und setze $\psi(d) = \sharp$ Elemente in G der Ordnung d. Angenommen $\psi(d) > 0$. Es existiert also ein $a \in G$ der Ordnung d. Also sind $a^0, a^1, \cdots, a^{d-1}$ paarweise verschieden. Diese sind genau die Nullstellen des Polynoms $x^d - 1$. Andererseits ist jedes Element der Ordnung d Nullstelle von $x^d - 1$. Nun hat a^e Ordnung d genau dann, wenn $(e, d) = 1$. Es gibt genau $\psi(d)$ solche e's. Es gilt also folgendes: für jedes $d|q-1$ gilt entweder $\psi(d) = 0$ oder $\psi(d) = \varphi(d)$. Angenommen es existiert ein $d|q-1$ mit $\psi(d) = 0$. Dann ist $q - 1 = \sum_{d|q-1} \psi(d) < \sum_{d|q-1} \varphi(d) = q - 1$. Dies ist ein Widerspruch. □

Korollar 7.1. *$G = (K \setminus \{0\}, \cdot)$ ist eine zyklische Gruppe mit $\varphi(q-1)$ Erzeugern.*

Wichtigster Spezialfall ist $q = p$ Primzahl, $K = \mathbb{Z}_p$. In diesem Fall werden die Erzeuger von $G = (K \setminus \{0\}, \cdot)$ Primitivwurzeln modulo p genannt.

Beispiel 7.1. $K = \mathbb{Z}_{13}, G = (K \setminus \{0\}, \cdot), q - 1 = 12 = 3 \cdot 2^2$

Element	Ordnung	Folge der Potenzen
1	1	1
2	12	1, 2, 4, 8, 3, 6, 12, 11, 9, 5, 10, 7
3	3	1, 3, 9
4	6	1, 4, 3, 12, 9, 10
5	4	1, 5, 12, 8
6	12	1, 6, 10, 8, 9, 2, 12, 7, 3, 5, 4, 11
7	12	1, 7, 10, 5, 9, 11, 12, 6, 3, 8, 4, 2
8	4	1, 8, 12, 5
9	3	1, 9, 3
10	6	1, 10, 9, 12, 3, 4
11	12	1, 11, 4, 5, 3, 7, 12, 2, 9, 8, 10, 6
12	2	1, 12

d	$\varphi(d)$
1	1
2	1
3	2
4	2
6	2
12	4

Übung 7.1. Beweise: Sei G eine endliche Gruppe, und sei $|G| = \prod_{i=1}^{r} p_i^{e_i}$ die Primfaktorzerlegung der Gruppenordnung. Definiere zu $g \in G$ $f_i = \max\{k | g^{\frac{|G|}{p_i^k}=1}\}$. Dann gilt: order $g = \prod_{i=1}^{r} p_i^{e_i - f_i}$.

Folgere daraus: Sei $n \in \mathbb{N}$ mit $g^n = 1$ und $g^{\frac{n}{p_i}} \neq 1$ für alle i, dann ist n die Ordnung von g.

Entwickle daraus einen Algorithmus, der bei bekannter Primfaktorzerlegung von $|G|$ zu gegebenem $g \in G$ dessen Ordnung bestimmt.

Übergabe der Gruppe und Gruppenoperationen

1. Fall: $K = \mathbb{Z}_p$, p prim, $G = (\mathbb{Z}_p \setminus \{0\}, \cdot)$

Übergabe

In diesem Fall übergibt man nur die Primzahl p.

Finden eines Erzeugers

Sei $|G| = n$. Wir wissen, es existieren $\varphi(n)$ Erzeuger.

Falls beispielsweise $p = 2q + 1$ (q prim), so ist $n = 2q$ und $\varphi(n) = \varphi(q) = q - 1$. In diesem Fall ist der Anteil der Erzeuger $\frac{q-1}{2q}$, also fast die Hälfte aller Gruppenelemente. Wähle zufällig ein $g \in G$. Teste dann $g^{\frac{n}{2}} \neq 1$ und $g^{\frac{n}{q}} \neq 1$.

Falls beides gilt, so ist g ein Erzeuger. Allgemein gilt: Das Bestimmen der Ordnung eines Elements ist schnell zu machen, falls die Primfaktorzerlegung von $n = p - 1$ bekannt ist.

Zunächst rufen wir uns noch eine wichtige Erkenntnis aus der Zahlentheorie in Erinnerung:

Theorem 7.2 (Dirichlet 1837). *Sind n, k relativ prim, dann existieren unendlich viele Primzahlen p mit $p \equiv k \mod n$.*

Theorem 7.3. *Es existiert eine Folge von Primzahlen p, so daß die Wahrscheinlichkeit, daß ein zufälliges (d.h. ein gleichverteiltes) $g \in \mathbb{Z}_p \setminus \{0\}$ ein Erzeuger ist, gegen 0 geht.*

Beweis. Setze $n_j = j!$. Wähle nun zu $j \geq 0$ ein p_j (p_j prim) mit

$$p_j \equiv 1 \bmod n_j.$$

Die Existenz des p_j folgt aus dem Satz von Dirichlet. Aus

$$p_j \equiv 1 \bmod n_j \Longleftrightarrow p_j - 1 \equiv 0 \bmod n_j$$

folgt somit für jeden Primteiler p von n_j, daß $p | p_j - 1$. Der Anteil der Elemente der Ordnung $p_j - 1$ bzgl. aller Elemente in $\mathbb{Z}_{p_j} \setminus \{0\}$ ist:

$$\frac{\varphi(p_j - 1)}{p_j - 1} = \frac{(p_j - 1)\prod_{p|p_j-1}\left(1 - \frac{1}{p}\right)}{(p_j - 1)} = \prod_{p|p_j-1}\left(1 - \frac{1}{p}\right) \leq \prod_{\substack{p \leq j \\ p \text{ Prim}}}\left(1 - \frac{1}{p}\right)$$

Für $j \to \infty$ geht $\prod_{p \geq j}(1 - \frac{1}{p})$ gegen 0, denn

$$\begin{aligned}
\prod_{\substack{p \leq j \\ p \text{ Prim}}}\left(1 - \frac{1}{p}\right) &= \left(\frac{1}{\prod_{\substack{p \leq j \\ p \text{ Prim}}} \frac{1}{1 - \frac{1}{p}}}\right)^{-1} \\
&= \left(\prod_{\substack{p \leq j \\ p \text{ Prim}}}\left(\sum_{k \geq 0}\left(\frac{1}{p}\right)^k\right)\right)^{-1} \\
&= \left(\sum_{m \in N_j} \frac{1}{m}\right)^{-1}
\end{aligned}$$

wobei

$$N_j := \{m \in \mathbb{N} : m \text{ hat eine Primfaktorzerlegung mit Primfaktoren } \leq j\}$$

und somit dieser Ausdruck für $j \to \infty$ gegen 0 geht. □

Gruppenoperationen

Die Addition und die Multiplikation in $\mathbb{Z}_p$ sind bekannt. Daher betrachten wir nun die Division - insbesondere die Bestimmung von Inversen. Um also die Gleichung

$$a \cdot x \equiv b \bmod p$$

zu lösen, bestimmt man mit dem euklidischen Algorithmus, der in $\mathcal{O}$(size p) arbeitet, das y mit

$$a \cdot y \equiv 1 \bmod p$$

und setze dann

$$x = by$$

2. Fall: K=GF(q), $q = p^r$ Primzahlpotenz

Übergabe

Um geeignete Übergabeparameter zu finden, betrachten wir K zunächst als Zerfällungskorper. K ist sicher ein Zerfällungskorper von $x^q - x$ über $K_0 = GF(p) = \mathbb{Z}_p$.

Nun ist $G = (K \setminus 0, \cdot)$ zyklisch. Sei $u \in G$ ein Erzeuger. Dann gilt $K = K_0(u)$, d.h., K ist eine Körpererweiterung von $K_0(u)$, also der kleinste Körper, der den Körper K_0 und das Element u enthält.

Sei f das Minimalpolynom von u, d.h., f ist irreduzibel, $f(u) = 0$, f hat den Leitkoeffizienten 1 und für jedes g mit $g(u) = 0$ gilt $f|g$. Dann ist

$$K = K_0(u) \cong K_0[x]/(f).$$

Insbesondere gilt: deg f = $[K : K_0] = r$ und 1, x, x^2,...,x^{r-1} ist eine K_0-VR Basis von $K_0[x]/(f)$ über K_0.

Die Übergabe besteht nun aus

- der Primzahl p (für K_0)
- der Übergabe eines irreduziblen Polynoms mit Leitkoeffizient 1 vom Grad r (am besten ein primitives Polynom, d.h., jede Nullstelle u des Polynoms erzeugt die zyklische Gruppe)

Dann sind die Elemente von $GF(q) = K_0[x]/(f)$ gegeben durch

$$a_0 + a_1 x + ... + a_{r-1} x^{r-1} \text{ mit } a_0 \in K_0 = GF(q).$$

.

Finden eines Erzeugers

Sei $n = |G| = q - 1$. Es existieren $\varphi(n)$-viele Erzeuger. Einen dieser Erzeuger erhalten wir wie folgt.

Betrachte den durch $K_0[x]/(f) \cong K_0(u)$ induzierten Isomorphismus, nämlich

$$K_0[x] \longrightarrow K_0(u)$$
$$g \longmapsto g(u)$$

Also hat $x + (f)$ die Ordnung $q - 1$.

Beispiel 7.2. Sei $K = GF(9)$, $K_0 = GF(3)$, $r = 2$. In diesem Fall existieren zwei primitive Polynome x^2+x-1 und x^2-x-1 (und ein weiteres irreduzibles wäre $x^2 + 1$).

Sei also $K = K_0[x]/(f)$ mit $f = x^2 + x - 1$. Die Potenzen von x sind:

$x^0 = 1$
$x^1 = x$
$x^2 = 1 - x$, denn $x^2 \equiv 1 - x \bmod (f)$
$x^3 = 2x - 1$, denn $x^3 = x - x^2 \equiv 2x - 1 \bmod (f)$
$x^4 = 2$, denn $x^4 = 2x^2 - x \equiv 2 - 3x \equiv 2 \bmod (f)$
$x^5 = 2x$
$x^6 = 2 - 2x$, denn $x^6 = 2x^2 \equiv 2 - 2x \bmod (f)$
$x^7 = x + 1$, denn $x^7 = 2x - 2x^2 \equiv 2x - 2(1 - x) \equiv x + 1 \bmod (f)$
$x^8 = 1$, denn $x^8 = x^2 + x \equiv 1 \bmod (f)$

Gruppenoperationen

Die Addition in K ist bekannt.

Bei der Multiplikation soll folgendes berechnet werden:

$$(a_0 + a_1 x + ... + a_{r-1}x^{r-1})(b_0 + b_1 x + ... + b_{r-1}x^{r-1}) \bmod (f)$$

Zur Berechnung führen wir $\mathcal{O}(r^2)$ Multiplikationen und $\mathcal{O}(r^2)$ Additionen (Cauchy-Produkt) und Polynomdivisionen modulo f, d.h., $\mathcal{O}(r)$ Divisionen und $\mathcal{O}(r^2)$ Multiplikationen in $\mathbb{Z}_p$ durch.

Insgesamt benötigen wir

$$\begin{aligned}\mathcal{O}(r^2(size(p))^2 + r^2(size(p))^2 + r(size(p))^2 + r^2(size(p))^2) = \\ = \mathcal{O}(r^2(size(p))^2) = \mathcal{O}((size(q))^2)\end{aligned}$$

Zur Abschätzung der Division genügt die Abschätzung für die Berechnung von h zu gegebenem g mit $g{\cdot}h+(f) = 1+(f)$. Mit dem erweiterten euklidischen Algorithmus erhalten wir die Schranke $\mathcal{O}((size(q))^2)$.

Beim Potenzieren kostet die Berechnung von $g^k \bmod f$ $\mathcal{O}(size(q) \cdot (size(q))^2)$ Zeit.

8

Berechnung des diskreten Logarithmus

In diesem Kapitel wollen wir einige Algorithmen zur Berechnung des diskreten Logarithmus betrachten. Hierfür möchten wir noch einmal an den chinesischen Restsatz erinnern, dessen Laufzeitverhalten wir noch analysieren wollen. Ferner sind einige der Algorithmen probabilistisch, weshalb wir uns ebenfalls kurz mit dem Geburtstagsphänomen beschäftigen werden.

Der chinesische Restsatz

Der chinesische Restsatz geht zurück auf ein Problem des Chinesen Sun-Tsu, welches er ca. im 4. Jahrhundert n.Chr. in seinem Buch “Master Suns mathematisches Handbuch” veröffentlichte.


孫子算經卷上
度之所起起於忽欲知其忽蠶吐絲爲忽十忽
爲一絲十絲爲一毫十毫爲一氂十氂爲一分
十分爲一寸十寸爲一尺十尺爲一丈十丈爲
一引五十尺爲一端四十尺爲一疋六尺爲一
步二百四十步爲一畝三百步爲一里
稱之所起起於黍十黍爲一絫十絫爲一銖二
十四銖爲一兩十六兩爲一斤三十斤爲一鈞

Abb. 8.1. Beginn des ersten Kapitels des “Sun Tzu Suan Ching”

Seien $m_1, ..., m_n \in \mathbb{N}$ paarweise teilerfremd und $a_1, ..., a_n \in \mathbb{Z}$. Gesucht ist eine Lösung für die simultane Kongruenz:

$$x \equiv a_i \bmod m_i \ , \ i = 1, ..., n$$

Zur Lösung betrachte

$$m := \prod_{i=1}^{n} m_i \text{ und } M_i = \frac{m}{m_i}$$

Offenbar gilt: $(m_i, M_i) = 1$. Nach dem erweiterten euklidischen Algorithmus existieren also y_i mit

$$y_i M_i \equiv 1 \bmod m_i$$

Setze dann

$$x := \sum_{i=1}^{n} a_i y_i M_i \bmod m.$$

Theorem 8.1. *x ist eine Lösung und x ist eindeutig modulo m.*

Beweis. Es gilt $a_i y_i M_i \equiv a_i \bmod m_i$ und ferner $a_j y_j M_j \equiv 0 \bmod m_i$ für $i \neq j$, da $m_i | M_j$. Also folgt $x \equiv a_i \mod m_i$ wie gewünscht. Nun zur Eindeutigkeit: Seien x, x' zwei Lösungen. Dann ist

$$x \equiv x' \bmod m_i \text{ für alle } i \text{, also}$$
$$x \equiv x' \bmod m$$

□

Theorem 8.2. *Das beschriebene Verfahren zur Berechnung von x benötigt die Zeit $\mathcal{O}((\mathrm{size}(m))^2)$ und Speicherplatz von $\mathcal{O}(\mathrm{size}(m))$.*

Beweis. Übung □

Das Geburtstagsphänomen

Man stelle sich eine Gruppe von 23 oder mehr Personen vor. Mit welcher Wahrscheinlichkeit haben wohl zwei Personen der Gruppe am gleichen Tag Geburtstag? Phänomenal, aber wahr: Es haben zwei den gleichen Geburtstag mit einer Wahrscheinlichkeit $\geq \frac{1}{2}$!

Allgemeiner:

Seien k Personen gegeben, die an einem von n möglichen Tagen Geburtstag haben (normalerweise $n = 365$). Wie groß ist die Wahrscheinlichkeit, daß zwei am selben Tag Geburtstag haben?

Wir wollen dies modellieren: Wir betrachten den Ereignisraum $\{1, ..., n\}^k$, wobei ein Elementarereignis $g = (g_1, ..., g_n)$ bedeutet, daß Person i am Tag g_i Geburtstag hat. Wir nehmen Gleichverteilung an, also $P(\{g\}) = \frac{1}{n^k}$ für jedes g.

Sei nun $p = P(\{g : g_i = g_j \text{ für ein } i \neq j\})$ die Wahrscheinlichkeit, daß wenigstens zwei Personen am gleichen Tag Geburtstag haben. Ferner sei $q = 1 - p = P(\{g : g_i \neq g_j \text{ für alle } i \neq j\})$. Dann ist

$$q = \frac{1}{n^k} \prod_{i=0}^{k-1} (n - i) = \prod_{i=0}^{k-1} \left(1 - \frac{i}{n}\right) = \prod_{i=1}^{k-1} \left(1 - \frac{i}{n}\right)$$

Mit $1 + x \leq e^x$ folgt:

$$q \leq \prod_{i=1}^{k-1} e^{-\frac{i}{n}} = e^{-\sum_{i=1}^{k-1} \frac{i}{n}} = e^{-\frac{k(k-1)}{2n}}$$

Oftmals ergibt sich daraus auch folgende Frage: Wie groß muss k (Anzahl der Personen) sein, damit für festes n (Tage im Jahr) und festes p_0 gilt, daß $p \geq p_0$ ist? Hierfür werden wir zu gegebenem k den Wert von $q = 1 - p$ berechnen, sodaß $1 - q = p \geq p_0$, also $q \leq 1 - p_0$ gilt.

Gesucht ist also (das kleinste) k mit $e^{-\frac{k(k-1)}{2n}} \leq 1 - p_0 =: q_0$. Es gilt:

$$\begin{aligned} ln(q_0) &\geq \frac{-k(k-1)}{2n} \\ 0 &\geq k^2 - k + 2nln(q_0) \\ k &\geq \frac{1 + \sqrt{1 - 8nln(q_0)}}{2} \\ k &\geq \frac{1 + \sqrt{1 + 8nln(1/q_0)}}{2} \end{aligned}$$

Beispiel 8.1.

$$n = 365 \;,\; p_0 = q_0 = \frac{1}{2} \Longrightarrow k \geq 22,999$$
$$n = 365 \;,\; p_0 = 0,9 \Rightarrow q_0 = 0,1 \Longrightarrow k \geq 41,51$$

Theorem 8.3. *Es sei $0 < p_0 < 1$ und ein Jahr habe n Tage. Dann werden $\mathcal{O}(\sqrt{n})$ viele Personen benötigt, damit wenigstens zwei von ihnen mit Wahrscheinlichkeit $\geq p_0$ am gleichen Tag Geburtstag haben.*

□

Enumeration

Wir erinnern uns noch einmal, daß es bei dem diskreten Logarithmus darum ging, zu einer gegebenen zyklischen Gruppe G der Ordnung n, zu gegebenem Erzeuger $g \in G$ und einem $y \in G$ ein $x \in \{0, ..., n-1\}$ mit $y = g^x$, also $x = \log_g y$, zu finden.

Eine Möglichkeit ist $g, g^2, g^3, ...$ iterativ zu berechnen, bis $g^x = y$ gefunden wird. Dazu werden $x - 1$ Multiplikationen durchgeführt und man benötigt Speicherplatz für g, y, g^x, also für drei Elemente.

Im realistischen Fall, daß $x \geq 2^{160}$ ist, ist dieses Verfahren mit sinnvollem Zeitaufwand nicht durchführbar. Im folgenden werden wir einige weitere Verfahren vorstellen.

Der Baby-Step-Giant-Step-Algorithmus

Der Baby-Step-Giant-Step-Algorithmus basiert auf folgendem Ansatz:

$$\begin{aligned} m &= \lceil \sqrt{n} \rceil, \\ x &= q \cdot m + r \ , \ 0 \leq r < m. \end{aligned}$$

Dann ist

$$y = g^x = g^{qm+r} = g^{qm} g^r$$

also

$$(g^m)^q = y \cdot g^{-r}$$

Es werden die sogenannten Babysteps $B = \{(yg^{-r}, r) : 0 \leq r < m\}$ berechnet:

- Falls $(1, r) \in B$, so ist $x = r$.
- Anderenfalls berechne $z = g^m$ und berechne iterativ $z, z^2, z^3, ...$ und vergleiche in jedem Schritt, ob ein Paar $(z^q, r) \in B$ existiert. Dann gilt $z^q = (g^m)^q = y \cdot g^{-r}$, also $x = qm + r$.

Die $z, z^2, ...$ sind die Giantsteps. Es stellt sich heraus, daß, wenn man das Vergleichen der Babysteps mit den Giantsteps effizient gestaltet, folgendes gilt:

Theorem 8.4. *Der Babystep-Giantstep-Algorithmus benötigt $\mathcal{O}(\sqrt{|G|})$ Multiplikationen und Vergleiche in G. Er muss $\mathcal{O}(\sqrt{|G|})$ viele Elemente aus G speichern.*

Beweis (Skizze für den Fall $G = (\mathbb{Z}_p \setminus 0, \cdot)$.).

Sei $G = (\mathbb{Z}_p \setminus 0, \cdot)$, $n = p-1$, $g \in G$ ein Erzeuger. Gesucht ist zu gegebenem $y \in G$ ein x mit $g^x = y$. Wir machen wieder den Ansatz:

$$\begin{aligned} m &= \lceil \sqrt{n} \rceil, \\ x &= q \cdot m + r \ , \ 0 \leq r < m. \end{aligned}$$

Die Berechnungszeit für die Babysteps $B = \{(y \cdot g^{-1},\ r) :\ 0 \leq r \leq m\}$ beträgt $\mathcal{O}(m(\text{size}(m))(\text{size}(m))^2)$. Betrachte die Hashfunktion

$$\begin{aligned} f : \mathbb{Z}_p &\longrightarrow \mathbb{Z}_{2^k} \\ j + p\mathbb{Z} &\longmapsto j + 2^k\mathbb{Z} \text{ mit } 0 \leq j < p, \end{aligned}$$

wobei 2^k die kleinste Zweierpotenz $\geq m$ ist, d.h., wir betrachten in der Binärdarstellung nur die letzten k Bits. Das Anlegen der Hashtabelle kostet dann $\mathcal{O}(m(\text{size}(m))(\text{size}(n))^2)$, da $m \geq n$. Die Giantsteps $z_j = (g^m)^j$ benötigen $\mathcal{O}((\text{size}(m))(\text{size}(n))^2)$ zur Berechnung von g^m und $\mathcal{O}((\text{size}(n))^2)$ zur Berechnung von $(g^m)^j \to (g^m)^{j+1}$. Unter der Annahme, daß das Berechnen des diskreten Logarithmus schwierig ist, ist das Benutzen einer Hashtabelle sinnvoll. Wir nehmen an, daß die yg^{-1} mod m gleichverteilt in $\mathbb{Z}_m$ sind. Wir berechnen in jedem Schritt $f((g^m)^j)$ in der Zeit $\mathcal{O}((\text{size}(m))^2)$ und prüfen in der Hashtabelle in $\mathcal{O}(1)$, ob ein Paar $(y \cdot g^{-1},\ r)$ mit $y \cdot g^{-1} = (g^m)^j$ existiert. Es werden höchstens $n/m = m$ Iterationen durchgeführt. Somit erhalten wir für die Giantstep-Routine incl. des Prüfens der Hashtabelle die Abschätzung $\mathcal{O}(m(\text{size}(m))^2)$.

Insgesamt benötigt der Babystep-Giantstep-Algorithmus also

- $\mathcal{O}(m(\text{size}(m))(\text{size}(n))^2)$ Zeit und
- $\mathcal{O}(m(\text{size}(n)))$ Platz.

Dies entspricht im Allgemeinen

- $\mathcal{O}(m(\text{size}(m)))$ Gruppenoperationen bei
- Speicherung von $\mathcal{O}(m)$ Gruppenelementen.

$\square$

Der Pollard-ρ-Algorithmus

Sei G eine zyklische Gruppe, $|G| = n$, $g \in G$ Erzeuger und $y \in G$ gegeben. Gesucht ist ein $x \in 0, ..., n-1$ mit $y = g^x$.

Sei $G = G_1 \cup G_2 \cup G_3$ mit $G \neq \emptyset$. Definiere dazu

$$f : G \longrightarrow G$$

$$z = \begin{cases} g \cdot z & : \quad z \in G_1 \\ z^2 & : \quad z \in G_2 \\ y \cdot z & : \quad z \in G_3 \end{cases}$$

Der Algorithmus

Wähle $x_0 \in \{0, ..., n-1\}$ beliebig. Definiere die Folge $(z_i)_{i \geq 0}$ mit $z_0 = y^{x_0}$ und $z_{i+1} := f(z_i)$. Demnach ist $z_i = g^{x_i} y^{y_i}$ mit x_0 wie oben, $y_0 = 0$ und

$$x_{i+1} = \begin{cases} x_i + 1 & : \quad z_i \in G_1 \\ 2x_i & : \quad z_i \in G_2 \\ x_i & : \quad z_i \in G_3 \end{cases}$$

und

$$y_{i+1} = \begin{cases} y_i & : \quad z_i \in G_1 \\ 2y_i & : \quad z_i \in G_2 \\ y_i + 1 & : \quad z_i \in G_3 \end{cases}$$

Da G endlich ist, existiert ein erstes Paar $(s, s+k)$ mit $z_s = z_{s+k}$.

Wieviel Folgenglieder muss ich nun berechnen, damit unter ihnen mit Wahrscheinlichkeit $\geq p$ zu gegebenen $0 < p < 1$ ein Paar (z_s, z_{s+k}) auftaucht?

Da es sich hier um eine Variante des Geburtstagsphänomens handelt ($\mathcal{O}(\sqrt{\text{Gruppenordnung}})$), also erhalten wir eine Abschätzung von $\mathcal{O}(\sqrt{n})$ unter der Annahme, daß die z_i annähernd gleichverteilt sind (das diskrete Logarithmusproblem also schwer zu lösen ist).

Wie verwenden wir nun unser gefundenes Paar $z_s = z_{s+k}$?

Aus

$$\begin{aligned} g^{x_s} y^{y_s} &= g^{x_{s+k}} y^{y_{s+k}} \\ g^{x_s - x_{s+k}} &= y^{y_{s+k} - y_s} \\ &= g^{x(y_{s+k} - y_s)} \end{aligned}$$

folgt, daß letzteres für das gesuchte x gelten muss. Das heißt, es muss gelten

$$x_s - x_{s+k} \equiv x(y_{s+k} - y_s) \bmod n$$

Nun löst man diese Kongruenz. Da es mehrere Lösungen geben kann, testet man die einzelnen Lösungen darauf, ob sie $y = g^x$ wirklich erfüllen. Sind es für einen Test zu viele Lösungen, so starte man den Algorithmus noch einmal mit einem anderen Startwert x_0.

Lineare Kongruenzen

Wie löse ich $ax \equiv b$ mod n?

Sei $g = \mathrm{ggT}(a, n)$, $a = a' \cdot g$, $n = n' \cdot g$. Dann ist

$$\begin{aligned} a'gx &= b + kn'g \\ a'gx - kn'g &= b \end{aligned}$$

1. Falls $g \nmid b$, so existiert keine Lösung.
2. Falls $g|b$, so setze $b' = \frac{b}{g}$ und betrachte

$$a'x \equiv b' \bmod n',$$

wobei jetzt $\mathrm{ggT}(a', n') = 1$. Bestimme nun mit dem euklidischen Algorithmus eine Lösung x_0 dieser Kongruenz.
Die allgemeine Lösung ist dann:

$$x = x_0 + l \cdot n' \text{ mit } 0 \leq l < g$$

Beispiel 8.2. Löse $12x \equiv 8 \bmod 20$. Wir haben $g = \mathrm{ggT}(12, 20) = 4$, $a' = 3$, $b' = 2$, $n' = 5$. Dies führt uns auf $3x \equiv 2 \bmod n$. Das Inverse von 3 mod 5 ist 2, denn $3 \cdot 2 \equiv 1 \bmod 5$. Dann ist

$$\begin{aligned} 2 \cdot 3 \cdot x &\equiv 4 \bmod 5 \\ x &\equiv 4 \bmod 5 \end{aligned}$$

Also ist $x_0 = 4$ eine Lösung. Die Menge der Lösungen ist dann $x = 4, 9, 14, 19$.

Kommen wir zurück zum Pollard-ρ-Algorithmus. Wenden wir uns der Frage nach dem Speicherplatzbedarf zu.
Wir speichern zunächst (z_0, x_0, y_0).

1. Angenommen, (z_i, x_i, y_i) ist gerade im Speicher. Setze dann $j := i + 1$
2. Falls $j < 2i$, berechne (z_j, x_j, y_j) und vergleiche z_i mit z_j
   - Falls $z_i = z_j$ ist, so sind wir fertig.
   - Sonst eröhe j um 1 und gehe zurück zu 2.
3. Falls $j = 2i$, lösche (z_i, x_i, y_i) und berechne und speichere (z_j, x_j, y_j).

Auf diese Weise werden sukzessiv für $j = 2^r$, $r \geq 0$ die Tripel gespeichert und mit den folgenden 2^r verglichen. Spätestens, wenn $j \geq k$ ist, finden wir ein Paar (z_s, z_{s+k}).

Zusammenfassung

Der Pollard-ρ-Algorithmus findet für jedes $0 < p < 1$ in Zeit $\mathcal{O}(\sqrt{n})$ mit einer Wahrscheinlichkeit $\geq p$ ein Paar $z_{s+k} = z_s$ und benötigt nur $\mathcal{O}(1)$ Speicherplatz.

Beispiel 8.3. Sei $G = (\mathbb{Z}_{2017} \setminus \{0\}, \cdot)$, $y = 3$, $g = 5$.
Gesucht ist ein $x \in \{0, ..., 2015\}$ mit $5^x \equiv 3 \bmod 2017$.
Setze

$$\begin{aligned} G_1 &= \{0, ..., 672\} \\ G_2 &= \{673, ..., 1344\} \\ G_3 &= \{1345, ..., 2016\} \end{aligned}$$

Setze $x_0 = 1023$.
Die Folge der Tripel ist:

j	z_j	x_j	y_j
0	986	1023	0
1	2	30	0
2	10	31	0
4	250	30	0
8	1366	136	1
16	1490	277	8
32	613	447	155
64	1476	1766	1000
98	1476	966	1128

Also ist

$$z_{98} = g^{966} \cdot y^{1128} = z_{64} = g^{1766} \cdot 1000$$

$$\begin{aligned} &\Longrightarrow 1766 - 966 \equiv x(1128 - 100) \bmod 2016 \\ &\Longleftrightarrow 128x \equiv 800 \bmod 2016 \end{aligned}$$

Da der $\mathrm{ggT}(128, 2016) = 32$

$$\begin{aligned} &\Longrightarrow 4x \equiv 25 \bmod 63 \\ &\Longrightarrow x' = 22 \end{aligned}$$

Also ist die allgemeine Lösung:

$$x = 22 + l \cdot 63 \text{ , wobei } 0 \leq l < 32$$

.

Für $l = 16$ ergibt sich $x = 1030$ mit $5^{1030} \equiv 3 \bmod 2017$.

Pohlig-Hellman-Algorithmus

Am Beispiel des Pohlig-Hellman-Algorithmus soll untersucht werden, wie der Grundkörper aussehen sollte, damit es möglichst schwierig ist, den diskreten Logarithmus zu bestimmen.
Wie schon zuvor sei G zyklische Gruppe mit einem Erzeugenden $g \in G$, $|G| = n$. Zu einem $\alpha \in G$ wird ein $x \in \{0, ...n-1\}$ gesucht mit $g^x = \alpha$. Das Problem wird in zwei Schritte zerlegt:

1. Reduktion auf Primzahlpotenzordnung.
2. Reduktion auf Primzahlordnung.

1. Sei $p|n$ ein Primteiler von n. Setze

$$n_p := \frac{n}{p^{e_p}} \qquad g_p := g^{n_p} \qquad \alpha_p := \alpha^{n_p}.$$

Wegen $g_p^{p^{e_p}} = 1$ gilt $\operatorname{ord} g_p = p^{e_p}$.
Für das gesuchte x gilt

$$g_p^x = g^{n_p x} = (g^x)^{n_p} = \alpha^{n_p} = \alpha_p.$$

x ist also auch Lösung dieses DL-Problems. Somit haben wir für jeden Primteiler von n ein neues DL-Problem, und unser ursprüngliches Problem hat sich auf ein Problem in einer Gruppe mit Primzahlpotenzordnung reduziert. Dafür wollen wir folgenden Satz anwenden:

Theorem 8.5. *Sei p ein Primteiler von n und x_p eine Lösung von*

$$g_p^{x_p} = \alpha_p$$

Dann ist die Lösung der simultanen Kongruenz

$$x \equiv x_p \mod p^{e_p}$$

eine Lösung des DL-Problems $g^x = \alpha$.

Beweis. Sei x eine Lösung der simultanen Kongruenz. Zeige: $\alpha g^{-x} = 1$. Nach Vorraussetzung gilt zunächst einmal:

$$1 = g_p^{-x_p} \alpha_p = (g^{-x_p} \alpha)^{n_p}.$$

Da dies für alle Primteiler gelten muß, folgt daraus

$$\operatorname{ord}(g^{-x}\alpha) | \operatorname{ggT}\{n_p : p|n, pPrimzahl\}.$$

Wegen

$$\operatorname{ggT}\{n_p : p|n, pPrimzahl\} = 1$$

ist also

$$\operatorname{ord}(g^{-x}\alpha) = 1$$

und somit

$$g^{-x}\alpha = 1.$$

□

2. Sei nun $|G| = n = p^e$, $g \in G$ Erzeuger und $\alpha \in G$.
Gesucht ist ein $x \in \{0, ..., n-1\}$ mit $g^x = \alpha$.
Die p-adische Darstellung von x lautet

$$x = x_0 + x_1 p + x_2 p^2 + ... + x_{e-1} p^{e-1}.$$

Wir wollen die x_i nun einzeln bestimmen, sie werden sich als Lösung eines DL-Problems in einer Gruppe der Ordnung p herausstellen.
Für $p^{e-1}x$ gilt

$$g^{p^{e-1}x} = \alpha^{p^{e-1}}$$

und weiterhin in der p-adischen Darstellung

$$p^{e-1}x = p^{e-1}x_0 + p^e(x_1 + ...).$$

Das bedeutet

$$p^{e-1}x \equiv p^{e-1}x_0 \mod p^e$$

und somit

$$g^{p^{e-1}x} = g^{p^{e-1}x_0} = \alpha^{p^{e-1}}$$

bzw.

$$(g^{p^{e-1}})^{x_0} = \alpha^{p^{e-1}}$$

und

$$\mathrm{ord}(g^{p^{e-1}}) = p.$$

$g^{p^{e-1}}$ erzeugt also eine Gruppe $< g^{p^{e-1}} >$ der Ordnung p, in der x_0 diskreter Logarithmus ist. Seien nun $x_0, ..., x_{i-1}$ bereits bestimmt. Dann ist

$$\begin{aligned} p^{e-i-1}(x - x_0 - x_1p - x_2p^2 - ... - x_{i-1}p^{i-1}) \\ = p^{e-i-1}(x_ip^i + x_{i+1}p^{i+1} + ... + x_{e-1}p^{e-1}) \\ = p^{e-1}x_i + p^e(x_{i+1} + ...). \end{aligned}$$

Damit erhalten wir

$$\begin{aligned} \left(g^{p^{e-1}}\right)^{x_i} &= \left(g^x g^{-(x_0+x_1p+...+x_{i-1}p^{i-1})}\right)^{p^{e-i-1}} \\ &= \left(\alpha * g^{-(x_0+x_1p+...+x_{i-1}p^{i-1})}\right)^{p^{e-i-1}} \\ &\equiv \alpha_i^{e-i-1} \end{aligned}$$

Dabei haben wir im zweiten Schritt $g^x = \alpha$ ausgenutzt. Auch x_i ist also Lösung eines DL-Problems in der selben Gruppe $< g^{p^{e-1}} >$ der Ordunug p. Auf diese Art und Weise können alle x_i bestimmt werden. Indem man den obigen Satz anwendet, findet man die gesuchte Lösung für x.

Beispiel 8.4.

$$5^x \equiv 3 \mod 2017$$

mit der Primfaktorzerlegung

$$2016 = 2^5 * 3^2 * 7$$

Beispiel 8.5.

$$2^x \equiv 28 \mod 37$$

Hier sind $g = 2$, $n = 36 = 2^2 3^2$ und $\alpha = 28$. Das ergibt die folgenden Werte:

$$\begin{aligned} n_2 &= \frac{36}{2^2} = 3^2 = 9 \qquad n_3 = \frac{36}{3^2} = 2^2 = 4 \\ g_2 &= g^{n_2} = 2^9 \equiv 31 \mod 37 \qquad g_3 = 16 \end{aligned}$$

g_2 läßt sich hier schnell mit Hilfe des iterierten Quadrierens berechnen. Für

$$\alpha_2 = 28^9 \equiv -1 \mod 37 \qquad \alpha_3 = 28^4 \equiv 12 \mod 37$$

sind also folgende Gleichungen zu lösen:

$$\begin{aligned} g_2^{x_2} = \alpha_2 : &\qquad 31^{x_2} \equiv -1 \mod 37 \\ g_3^{x_3} = \alpha_3 : &\qquad 16^{x_3} \equiv 12 \mod 37 \end{aligned}$$

Dies sind DL-Probleme in der von g_p erzeugten Gruppe. Diese Gruppen sehen wie folgt aus:

$$\begin{aligned} G_2 := &< g_2 >= \{1, -6, -1, 6\} \\ G_3 := &< g_3 >= \{1, 16, -3, -11, 9, -4, 10, 12, 7\} \end{aligned}$$

Schon an dieser Stelle läßt sich das Ergebnis $x_2 = 2$ und $x_3 = 7$ ablesen. Trotzdem wollen wir einmal den zweiten Schritt durchgehen. Sei also

$$x_2 = y_0 + y_1 2$$

die 2−adische Entwicklung. Damit erhalten wir

$$\left(g_2^{2^{2-1}}\right)^{y_0} = \alpha_2^{2^{2-1}} = (-1)^2 = 1$$

Da y_0 in der 2−adischen Darstellung nur die Werte $0, 1$ annehmen kann, muß $y_0 = 0$ sein. Analog erhält man für y_1:

$$\left(g_2^{2^{2-1}}\right)^{y_1} = \left(\alpha_2 g_2^0\right)^{2^0}$$
$$\Rightarrow (-1)^{y_1} = -1 \qquad \Rightarrow y_1 = 1$$

Auf diese Weise kommen wir also auch zum Ergebnis $x_2 = 2$ und $x_3 = 7$. Jetzt müssen wir nur noch den chinesischen Restsatz anwenden:

$$m_1 = 1 \qquad m_2 = 9 \Rightarrow M_1 = 9 \qquad M_2 = 4$$

Für die zu lösenden Gleichungen

$$y_i M_i \equiv 1 \mod m_i$$

ergeben sich die Lösungen

$$\begin{aligned} y_1 &= 1 \qquad (9 \equiv 1 \mod 4) \\ y_2 &= 7 \qquad (7 \equiv 1 \mod 9) \end{aligned}$$

Aus

$$\Sigma x_i y_i M_i = 2 * 1 * 9 + 7 * 7 * 4 \equiv 34 \mod 36$$

erhalten wir das Endergebnis

$$2^{34} \equiv 28 \mod 37$$

Laufzeit des Pohlig-Hellman-Algorithmus

Wir gehen davon aus, daß die Primfaktorzerlegung von n bekannt ist. Dann muß der Algorithmus folgendes berechnen:

- $g^{n_p} = g_p$
- $\alpha^{n_p} = \alpha_p$
- x_p mit $g_p x^p = \alpha_p$

  – g_p habe Ordnung p^e (mit $e = e_p$)
    Berechnung von $x_0, ..., x_{e-1}$ via DL-Problem in einer Gruppe der Ordnung p

$$\left(g^{p^{e-1}}\right)^{x_i} = (\alpha_i)^{p^{e-i-1}}$$

- Lösen der Kongruenz $x \equiv x_p \mod p^{e_p}$

Die Anzahl der Gruppenoperationen (Addieren, Multiplizieren) ist

$$\mathcal{O}(\sum_{p|n}(size(n) + e_p(size(n) + \sqrt{p}) + 1)),$$

was sich abschätzen läßt zu

$$\mathcal{O}(\sum_{p|n} e_p(size(n) + \sqrt{p}))$$

In diesem Term dominiert $\sqrt{p}$, und das führt zu einem wichtigen Ergebnis: n sollte keine Primfaktorzerlegung haben mit ausschließlich kleinen Primfaktoren, sonst ist der Algorithmus schnell. Das bedeutet, daß für Kryptoverfahren Gruppen G verwendet werden sollten, für die $|G| = n$ eine Primfaktorzerlegung besitzt, in der auch große Faktoren vorkommen.

Beispiel 8.6 (für eine schlechte Gruppe). $p = 2 * 3 * 5^{278} + 1$ ist eine Primzahl, aber die zugehörige Gruppe G hat Ordnung $n = 2 * 3 * 5^{278}$ mit Primfaktoren $2, 3, 5$.
Deutlich besser geeignet sind Körper der Form $K = GF(2^m)$, so daß $2^m - 1$ Mersenne-Primzahl ist.

Übung Berechne den Diskreten Logarithmus von 153 zur Basis 2 in $GF(181)$.

Übung (a) Wie groß ist die Wahrscheinlichkeit, daß ein zufälliges Polynom in $\mathbb{Z}_2[x]$ vom Grad 10 in ein Produkt von Polynomen vom Grad ≤ 2 faktorisiert?
(b) Wie groß ist die Wahrscheinlichkeit, daß ein zufälliges Polynom vom Grad ≤ 10 in ein solches Produkt faktorisiert?

Der Index - Kalkül

Sei $K = GF(q)$, $q = p^n$ mit p prim. $g \in (GF(q) \setminus \{0\})$ sei Erzeuger und $y \in (GF(q) \setminus \{0\})$. Gesucht ist wieder ein $x \in \{0, ..., q-2\}$ mit $g^x = y$. Wir definieren $GF(q) := GF(p)[x]/(f)$, wobei (f) das vom irreduziblen Polynom f erzeugte Ideal ist. Die Elemente von $GF(q)$ haben die Form

$$(a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}) \mod f.$$

Sei $b(x)$ ein Polynom vom Grad $\leq n-1$ mit $g \equiv b(x) \mod f$.
Wähle nun ein Faktorbasis $B \subseteq K \backslash \{0\}$. Dies ist keine Basis im Sinne einer Vektorraumbasis, vielmehr sollen damit Elemente von K als Produkt von Elementen, die diesen Raum erzeugen, geschrieben werden. Normalerweise wählt man hierzu

$$B = \{a(x) | a \text{ irreduzibles Polynom vom Grad } \leq m, \text{ Leitkoeffizient} = 1\}$$

Da wir ein $h \times h-$Gleichungssystem zu lösen haben werden, soll B so gewählt sein, daß $h = |B|$ nicht zu groß ist, gleichzeitig soll aber ein zufällig gewähltes Polynom vom Grad $\leq n$ mit hoher Wahrscheinlichkeit als Produkt darstellbar sein. Zwischen diesen beiden gegenläufigen Forderungen muß also ein Kompromiß gefunden werden. Ein Beispiel ist

$$p = 2 \qquad n = 127 \qquad m = 17 \Rightarrow h = 16510$$
$$2^{127} - 1 \text{ ist Mersenne-Primzahl.}$$

Der Algorithmus funktoniert in zwei Schritten:

1. Vorberechnung für K und g.
2. Bestimmung des Diskreten Logarithmus $\log_g y$.

Die Berechnung aus Schritt 1. kann für alle Paare x, y verwendet werden (wichtig für die Laufzeit).

1. Bestimmung von $\log_{b(x)} a(x)$ für ein $a(x) \in B$. Wähle $t \in \{1, ..., q-2\}$ zufällig aus und berechne $c(x) = (b(x))^t$. Jetzt sei c_0 der Leitkoeffizient von $c(x)$. Wir wollen testen, ob sich $c(x)$ durch Elemente von B faktorisieren läßt. Dann hätte $c(x)$ die Form

$$c(x) = c_0 \prod_{a \in B} a(x)^{\alpha_{c,a}}$$

In diesem Fall haben wir

$$\underbrace{\log_{b(x)} c(x)}_{t} - \underbrace{\log_{b(x)} c_0}_{\text{schnell zu berechnen}} = \sum_{a \in B} \underbrace{\alpha_{c,a}}_{\text{bekannt}} \log_{b(x)} a(x)$$

Genaugenommen ist diese Gleichung $\mod (q-1)$ zu rechnen. Durch wiederholte Wahl von t findet man h linear unabhängige Gleichungen dieser Art, die für $\log_{b(x)} a(x)$ in Z_{q-1} gelöst werden müssen.
Es geht tatsächlich schnell, $\log_{b(x)} c_0$ bzw $\log_b c_0$ zu berechnen:

$$b' := b^{\frac{q-1}{p-1}} \qquad \left(b^{q-1} = 1\right) \qquad \frac{q-1}{p-1} \in Z$$

$b^‘$ hat also Ordnung $p-1$ in $GF(q)$ und erzeugt eine Gruppe $GF(p)\backslash\{0\} \subseteq GF(q)\backslash\{0\}$. Diese Gruppe ist relativ klein, so daß sich der Diskrete Logarithmus $y^‘ := \log_{b^‘} c_0$ schnell bestimmen läßt mit einem der vorangegangenen Algorithmen. Aus

$$b^{‘y^‘} = b^{\frac{q-1}{p-1}y^‘} = c_0$$

folgt

$$\log_b c_0 = \frac{q-1}{p-1} y^‘$$

2. Für den zweiten Schritt wählen wir zufällig ein $t \in \{1, ..., q-2\}$ und berechnen $y_1 = yb^t$. Jetzt testen wir, ob sich dieses y_1 über die Faktorbasis darstellen läst:

$$y_1 = y_0 \prod_{a \in B} a(x)^{\alpha_a}$$

Wenn das nicht möglich ist, nehmen wir ein anderes t und prüfen erneut. Mit dieser Darstellung erhalten wir

$$\log_b y_1 = \log_b y + \underbrace{\log_b b^t}_{=t},$$

was wir umformen können zu

$$\begin{aligned} \log_b y &= \log_b y_1 - t \\ \Leftrightarrow \log_b y &= \log_b y_0 + \sum_{a \in B} \alpha_a \underbrace{\log_b a(x)}_{(1)} - t \end{aligned}$$

$\log_b y_0$ ist schnell zu berechnen, da y_0 konstant ist, somit sind wir fertig.

Laufzeit des Index - Kalküls

$$\exp\left[(const + \mathcal{O}(1))\sqrt{q}\sqrt{\log\log q}\right]$$

Das ist durch die Wurzel schon eine deutliche Verbesserung.

9

Turing–Maschinen und Komplexitätstheorie

Unser nächstes Ziel ist die Behandlung von Einwegfunktionen. Funktionen f, für die es „leicht“ ist zu einem gegebenen x den Wert $f(x)$ zu berechnen, aber für die es „schwer“ ist $f^{-1}(y)$ für ein y zu berechnen. Für die Kryptographie brauchen wir genauer Funktionen, für die die Berechnung von $f^{-1}(y)$ *ohne den Schlüssel schwer*, aber mit Schlüssel leicht ist. Bevor wir uns Einwegfunktionen zuwenden, müssen wir klären, was wir mit „leicht“ und „schwer“ meinen.

Der elementare Baustein der Komplexitätstheorie ist der Begriff der Turing–Maschine, den Alan Turing 1936 entwickelte, Jahre bevor er in die Entschlüsselung der Enigma involviert war.

Turing–Maschinen

Eine Turing–Maschine ist eine abstrakte Maschine, die auf einem Band operiert. Dieses Band besteht aus diskreten, linear angeordneten Zellen, die jeweils genau ein Symbol aus einem endlichen Alphabet Σ enthalten. Die Turing–Maschine arbeitet auf diesem Band mittels eines Schreib-/Lesekopfes (Cursors), der über genau einer Zelle des Bandes steht. Der Cursor kann das Symbol in dieser Zelle lesen und überschreiben und hat die Möglichkeit sich eine Zelle nach links oder rechts zu bewegen. Das „Programm“ der Maschine ist nun gegeben durch eine endliche Menge von Zuständen und eine Übergangsfunktion. Die Übergangsfunktion gibt für jeden Zustand und für jedes Symbol, über dem sich der Cursor gerade befindet, an

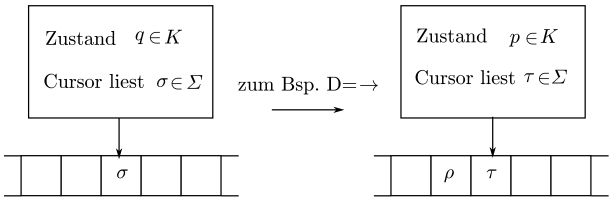
- was im nächsten Schritt der Zustand der Maschine ist
- welches Symbol der Schreib-/Lesekopf in die aktuelle Zelle schreibt und
- ob sich der Kopf eine Zelle nach links oder rechts bewegt.

Genauer definieren wir

Definition 9.1. *Eine* Turing–Maschine *ist ein Quadrupel* $M = (K, \Sigma, \delta, s)$, *für das folgendes gilt:*

- K *ist eine endliche* Zustandsmenge
- $s \in K$ *ist der* Initialzustand
- Σ *ist das (endliche) Alphabet von* M
- $K \cap \Sigma = \emptyset$
- *Es gibt zwei spezielle Symbole, nämlich das Leerzeichen* $\sqcup$ *(blank) und das Startsymbol* $\triangleright \in K$.
- δ *ist die sogenannte Übergangsfunktion mit*
  $\delta : K \times \Sigma \to (K \cup \{h, \text{„}ja\text{“}, \text{„}nein\text{“}\} \times \Sigma \times \{\leftarrow, \rightarrow, -\})$
- h *heißt Haltezustand*
- *„ja“ ist der akzeptierende Zustand*
- *„nein“ ist der verwerfende Zustand*
- *Gilt für* $p, q \in K$ $\quad \delta(q, \triangleright) = (p, \rho, D)$, *so folgt* $\rho = \triangleright, D =\rightarrow$.

Nun sei zum Beispiel $\delta(q, \sigma) = (p, \rho, D)$. Dann löscht die Turing–Maschine σ und schreibt an dieser Stelle ρ auf das Band. Danach geht sie in Zustand p über und bewegt sich einen Schritt in Richtung D, beispielsweise einen Schritt nach rechts für $D =\rightarrow$.

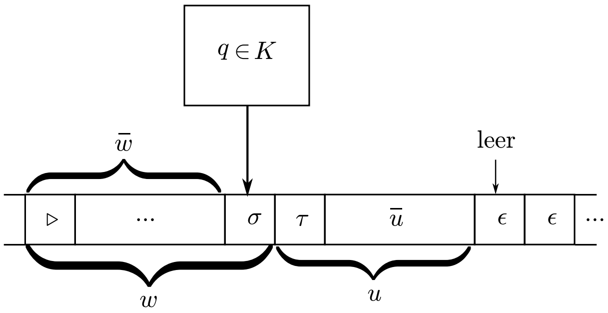


Start und Stop einer Turing–Maschine

- der initiale Zustand von einer Turing–Maschine M ist s
- Sei $x \in (\Sigma \backslash \{\sqcup\})^*$ die Eingabe des Programms, so übergebe $\triangleright x$ an M, d.h. $\triangleright x$ steht auf dem Band und die Cursorposition ist $\triangleright$: $\quad \overset{\downarrow}{\triangleright} x$
- es gibt drei Fälle, in denen die Maschine anhält: h, „ja“ und „$nein$“ In diesem Fall: „ja“ bedeutet, M akzeptiert x und „$nein$“ bedeutet, M verwirft x. Falls h eingenommen wird, so ist $M(x) = y$ die Ausgabe, wobei zum letzten Zeitpunkt $\triangleright y \sqcup \sqcup \cdots \sqcup$ auf dem Band steht. Das letzte Symbol von y ist dabei kein $\sqcup$.
- Die Maschine kannn nie links an $\triangleright$ vorbei laufen, aber unter Umständen unendlich lange nach rechts laufen, in diesem Fall schreiben wir $M(x) =\nearrow$ und sagen M divergiert.

Definition 9.2. *Eine* Konfiguration *einer festen Turing–Maschine* M *ist ein Tripel* (q, w, u), *wobei* $q \in K$ *ein Zustand und* $w, u \in \Sigma^*$ *endliche Zeichenfolgen sind. Eine Konfiguration beschreibt den Zustand von* M *zu einem festen*

Zeitpunkt der Berechnung vollständig: q ist der aktuelle Zustand, w der Inhalt des Bandes links vom Cursor und u der Inhalt des Bandes rechts vom Cursor. Das letzte Zeichen von w befindet sich direkt unter dem Cursor. Allgemeiner



schreiben wir

$$(q, w, u) \xrightarrow{M^k} (q', w', u')$$

falls (q', w', u') durch k Schritte der Form $\xrightarrow{M}$ aus (q, w, u) hervorgeht. Wir schreiben

$$(q, w, u) \xrightarrow{M^*} (q', w', u')$$

falls ein k existiert mit $(q, w, u) \xrightarrow{M^k} (q', w', u')$.

Beispiel

Wir definieren eine Turing–Maschine durch folgende Zustandstabelle

$p \in K$	$\sigma \in \Sigma$	$\delta(\rho, \sigma)$
s	0	$(s, 0, \rightarrow)$
s	1	$(s, 1, \rightarrow)$
s	$\sqcup$	$(q, \sqcup, \leftarrow)$
s	$\triangleright$	$(s, \triangleright, \rightarrow)$
q	0	$(h, 1, -)$
q	1	$(q, 0, \leftarrow)$
q	$\triangleright$	$(h, \triangleright, \rightarrow)$

Geben wir $x = 10011$ ein läuft die Maschine folgendermaßen ab

$$
\begin{array}{rlrl}
 & (s\ , & \triangleright\ , & 10011\ ) \\
\xrightarrow{M} & (s\ , & \triangleright 1\ , & 0011\ ) \\
\xrightarrow{M} & (s\ , & \triangleright 10\ , & 011\ ) \\
\xrightarrow{M} & (s\ , & \triangleright 100\ , & 11\ ) \\
\xrightarrow{M} & (s\ , & \triangleright 1001\ , & 1\ ) \\
\xrightarrow{M} & (s\ , & \triangleright 10011\ , & \epsilon\ ) \\
\xrightarrow{M} & (s\ , & \triangleright 10011\sqcup\ , & \epsilon\ ) \\
\xrightarrow{M} & (q\ , & \triangleright 10011\ , & \sqcup\ ) \\
\xrightarrow{M} & (q\ , & \triangleright 1001\ , & 0\sqcup\ ) \\
\xrightarrow{M} & (q\ , & \triangleright 100\ , & 00\sqcup\ ) \\
\xrightarrow{M} & (h\ , & \triangleright 101\ , & 00\sqcup\ )
\end{array}
$$

Wir erhalten also die Ausgabe $M(x) = 10100$. M berechnet für $x \neq 11...1$ den binären Nachfolger und $M(\underbrace{11...1}_{n}) = \underbrace{00..0}_{n}$.

Übung

Erweitere M, so daß immer der binäre Nachfolger berechnet wird.

Turing–Maschinen, Sprachen und Zeichenkettenfunktionen

Definition 9.3. • *Eine Menge $L \subseteq (\Sigma \setminus \{\sqcup\})^*$ heißt* Sprache.

- *Ist M eine Turing-Maschine mit der Eigenschaft, daß*

$$M(x) = \begin{cases} \text{„ja"} & \text{falls } x \in L \\ \text{„nein"} & \text{falls } x \notin L \end{cases},$$

  so sagen wir M entscheidet *L.*
- *Eine Sprache, für die Turing-Maschine existiert, die sie entscheidet, heißt* rekursiv.
- *Ist M eine Turing-Maschine mit*

$$M(x) = \begin{cases} \text{„ja"} & \text{falls } x \in L \\ \nearrow & \text{falls } x \notin L \end{cases},$$

  so sagen wir M akzeptiert *L.*
- *Wird eine Sprache L von einer Turing–Maschine akzeptiert, so heißt L* rekursiv aufzählbar.

Theorem 9.1. *Ist L rekursiv, so auch rekursiv aufzählbar.*

Beweis. Modifiziere die Turing–Maschine M, die L entscheidet, so daß an der Stelle, an der M den Zustand „nein" annähme, die Maschine in einen neuen Zustand übergeht, der sie ins Unendliche laufen lässt.

Definition 9.4. *Sei $f : (\Sigma \backslash \{\sqcup\})^* \to \Sigma^*$ eine (Zeichenketten-)Funktion. Falls eine Turing-Maschine M existiert mit $M(x) = f(x)$ für alle $x \in (\Sigma \setminus \{\sqcup\})*$, so heißt f* rekursiv *(oder berechenbar) wir sagen M* berechnet *f.*

Definition 9.5. *Ist M eine Turing–Maschine, so definieren wir die* Sprache $E(M)$, die von M aufgezählt wird, *durch*

$$E(M) = \{x : \text{es ex. } q \in k, y \in \Sigma^* \text{mit} (s, \triangleright, \epsilon) \xrightarrow{*} (q, \triangleright y \sqcup x \sqcup, \epsilon)\}$$

Die Sprache $E(M)$ ist also die Menge all jener Wörter, die M im irgendwann einmal von Leerzeichen umschlossen am Ende ihres Bandes stehen hat, wenn M mit einem leeren Band gestartet wird. Man kann Turing–Machinen so konstruieren, daß M dennoch beliebig lange Zwischenrechnungen auf dem Band durchführen kann, die nicht zu $E(M)$ gehören. Daß die $x \in E(M)$ am Ende des Bandes stehen, stellt also keine Einschränkung dar. Man kann sich $E(M)$ als die Menge aller „Werte“ vorstellen, die M während ihrer Laufzeit ausgibt, wobei M zwischen zwei Ausgaben beliebig lange Zwischenrechnungen durchführen kann.

Welche Sprachen L lassen sich auf diese Weise von Turing–Maschinen ausgeben? Das folgende Theorem sagt aus, daß dies genau die rekursiv aufzählbaren Sprachen sind, daher auch der Name „rekursiv aufzählbar“.

Theorem 9.2. *L ist rekursiv aufzählbar genau dann, wenn eine Turing-Maschine M existiert mit $E(M) = L$.*

Beweis (Skizze). „$\Leftarrow$“ Wir konstruieren eine Turing-Maschine A die L akzeptiert, indem wir diejenige Turing-Maschine M mit $E(M) = L$ folgendermaßen modifizieren. Gegeben eine Eingabe x läuft A ab wie M mit folgender Änderung: Immer dann, wenn M ein Ergebnis ans Ende des Bandes schreibt überprüfen wir in A ob das Ergebnis gleich der Eingabe ist. Wenn dem so ist wechseln wir in den „ja“ Zustand und halten an, ansonsten fahren wir mit dem Programm fort. Ist $x \notin L$ so ergibt sich $A(x) = \nearrow$ automatisch falls $M(\epsilon) = \nearrow$. Anonsten schicken wir A künstlich ins unendliche wenn M anhalten würde.

„$\Rightarrow$“ Sei L eine rekursiv aufzählbare Sprache. Dann gibt es eine Turing-Maschine A die L akzeptiert. Wir konstruieren nun eine Turing-Maschine M, die L nach und nach aufs Band schreibt. Wesentlich hierbei ist, daß für endliches Σ die Menge Σ^* abzählbar ist. Sei also $(x_i)_{i \in N}$ eine Aufzählung von Σ^*. Mit A verfügen wir über eine Maschine, die für jedes dieser x_i nach endlicher Zeit „ja“ ausgibt genau dann, wenn $x_i \in L$.

M läuft nun folgendermaßen ab:

1. M berechnet 1 Schritt von $A(x_1)$.
2. M berechnet 2 Schritte von $A(x_1)$ und $A(x_2)$.
   ...

n. M berechnet n Schritte von $A(x_1), ..., A(x_n)$.
   usw.

Für beliebiges k und beliebiges n hat M also nach einer endlichen Zeit n Schritte von $A(x_k)$ berechnet. Für jedes Wort x über Σ, daß von A akzeptiert wird, hat M also nach einer endlichen Zeit ausgerechnet, daß $A(x)$ = „ja" und kann dieses auf sein Band schreiben.

Berechenbarkeit

Turing–Maschinen wurden entwickelt um eine Antwort auf die Frage „Was ist berechenbar?" zu geben. Die intuitive Antwort auf diese Frage ist:

„Jeder Algorithmus, den wir uns ausdenken können, ist berechenbar."

In der ersten Hälfte des 20. Jahrhunderts wurde verschiedene Versuche unternommen, diese Antwort zu präzisieren. In diesem Kontext sind unter anderem Turing–Maschinen, das λ-Kalkül von Alonzo Church und die Zellulären Automaten von John von Neumann entstanden. Es stellte sich heraus, daß all diese Berechenbarkeitsbegriffe äquivalent sind: Was im Sinne des einen Berechenbarkeitsparadigmas berechenbar ist, ist auch in jedem anderen dieser Paradigmen berechenbar. Dies gab Anlass zu der These

Die Church'sche These

Eine Funktion ist genau dann (intuitiv) berechenbar, wenn sie rekursiv ist.
Eine Sprache ist genau dann (intuitiv) entscheidbar, wenn sie rekursiv ist.

Ausblick

Wem bei den in den letzten Beweisen verwendeten Konstruktionen nicht ganz wohl ist, dem sei hier ein kleiner Ausblick auf weitere Konzepte in der Theorie der Turing–Maschinen gegeben, auf die, um nicht zu weit vom Thema abzukommen, hier nicht weiter eingegangen wird.

Mehrere Bänder

Man kann den Begriff von Turing–Maschinen derart erweitern, daß diese über nicht nur ein Band, sondern über n Bänder mit jeweils einem Cursor verfügen. In jedem Schritt würde der Zustandübergang von allen n Cursors abhängen und auf alle n Bänder geschrieben werden. Dies stellt jedoch keine echte Erweiterung des Begriffs der Turing–Maschine dar: gegeben eine Mehr-Band-Maschine M_n lässt sich eine Ein-Band-Maschine M_1 konstruieren, die auf die gleichen Eingaben die gleichen Ausgaben liefert. Das eine Band von M_1 ist dabei gleich denen von M_n unter einer Bijektion zwischen den Zellen des einen Bandes und der Vereinigung der Zellen der n Bänder.

Universelle Turing–Maschinen

Es ist möglich eine sogenannte Universelle Turing–Maschine U zu konstruieren, die andere Turing–Maschinen M simulieren kann. U hat dabei eine feste Zustandsmenge K und eine feste Übergangsfunktion δ. Soll nun eine Maschine M simuliert werden, wird das „Programm" von M (gegeben durch Zustandsmenge und Übergangsfunktion) geeignet als endliche Zeichenkette geschrieben und diese *links* von $\triangleright$ auf dem Band gespeichert.

Berechenbare Zeichenfolgen

Eine interessante Anwendung des Konzeptes der Universellen Turing–Maschine ist folgende. Wir nennen eine unendliche Zeichenfolge „berechenbar", wenn es Turing–Maschine gibt, die diese bei Eingabe von ϵ nach und nach auf eines ihrer Bänder schreibt. Zu jeder Turing–Maschine gibt es ein endliches, diese Maschine vollständig beschreibendes Programm. Da das Alphabet von U endlich ist, gibt es nur abzählbar viele Programme (also abzählbar viele Turing–Maschinen). Aber die Menge von unendlichen Zeichenfolgen über einem mindesten zweielementigen Alphabet ist überabzählbar. Also gibt es unendliche Zeichenfolgen, die nicht berechenbar sind.

Laufzeit und Komplexität

Definition von $\mathcal{P}$ und $\mathcal{NP}$

Wir definieren zunächst $\mathcal{P}$, die Klasse aller in polynomieller Zeit entscheidbaren Sprachen.

Definition 9.6. *Sei M eine Turing-Maschine.*

- *Falls gilt* $(S, \triangleright, x) \xrightarrow{M^t} (H, w, u)$ *wobei* $H \in \{\text{ja}, \text{nein}, \text{h}\}$, *so* benötigt M auf Eingabe x die Zeit t. *Falls $M(x) = \nearrow$, so benötigt sie die Zeit unendlich.*
- *Sei $f : \mathbb{N} \to \mathbb{N}$ eine Funktion. M operiert* innerhalb der Zeitschranke f, *falls sie zur Berechnung auf Eingabe x höchstens $f(|x|)$ Zeit benötigt.*
- *Sei nun $L \subseteq (\Sigma \setminus \{\sqcup\})^*$ eine Sprache.* $L \in \text{Time}(f)$ *genau dann, wenn L von einer Turing-Maschine M entschieden wird, die innerhalb der Zeitschranke f operiert.*
- $\mathcal{P} = \bigcup_{f \text{ ein Polynom}} \text{Time}(f)$

  Nun kommen wir zu $\mathcal{NP}$. $\mathcal{N}$ steht hier für „nicht-deterministisch".

Definition 9.7. *Sei $R \subseteq \Sigma^* \times \Sigma^*$.*

- *R heißt* polynomiell entscheidbar, *falls die Sprache $\{x;y : (x,y) \in R\} \subseteq (\Sigma \cup \{;\})^*$ mit $; \notin \Sigma$ in $\mathcal{P}$ liegt.*
- *R heißt* polynomiell balanciert, *falls ein $k \in N$ existiert, so daß für alle $(x,y) \in R$ gilt, daß $|y| \leq |x|^k$.*

Definition 9.8. *Sei L eine Sprache. $L \in \mathcal{NP}$ genau dann, wenn eine polynomiell entscheidbare, polynomiell balancierte Relation R existiert, so daß $L = \{x : exymit(x, y) \in R\}$*

Was bedeutet das? Eine Sprache L liegt in $\mathcal{NP}$, wenn wir von einem Wort x unter Zuhilfename eines „Beleges“ y in polynomieller Zeit verifizieren können, daß $x \in L$. Wir konstruieren dann die Sprache R so, daß in R nur Paare (x, y) enthalten sind, mit

- $x \in L$ und
- y ist ein Beleg dafür, daß $x \in L$, den wir schnell verifizieren können.

R ist dann polynomiell entscheidbar. Gegeben irgendein $x; y$ müssen wir nur überprüfen, ob y tatsächlich belegt, daß $x \in L$, was in polynomieller Zeit geht. Ist y kein Beleg für x, dann ist $(x, y) \notin R$, selbst wenn $x \in L$.

Warum „nicht-deterministisch“? Der Name „nicht-deterministisch“ gründet sich in der Tatsache, daß man $\mathcal{NP}$ oft äquivalent über nicht-deterministische Turing–Maschinen definiert. Eine nicht-deterministische Turing–Maschine wählt in jedem Schritt zwischen mehreren möglichen nächsten Zuständen zufällig einen aus. Sie akzeptiert ein Wort, wenn es einen Ablauf gibt, der das Wort akzeptiert, und sie lehnt es ab, wenn jeder mögliche Ablauf das Wort ablehnt. Der Zusammenhang zu unserer Definition ist folgender: Für jede „ja“-Instanz $x \in L$ lässt sich ein y raten mit $|y| \leq |x|^k$ und $(x, y) \in R$. Zu richtig geratenem y kann dann die „ja“-Instanz in polynomieller Zeit verifiziert werden.

Beispiele

SAT

Wir betrachten boolesche Ausdrücke wie zum Beispiel $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2) \wedge (x_2 \vee \neg x_3)$. *Frage:* Existiert eine Belegung der x_i mit Wahrheitswerten, so daß der boolesche Ausdruck wahr wird? Die zugehörige Sprache $L = \{x :$ x ist erfüllbarer boolescher Ausdruck$\}$ liegt in $\mathcal{NP}$: Wir wählen

$$R = \{(x, y) : x \in L, y \text{ eine Belegung, die } x \text{ erfüllt }\}$$

R ist polynomiell entscheidbar, da wir, gegeben einen booleschen Ausdruck x und eine Belegung y, mit einer Turing–Maschine in polynomieller Zeit entscheiden können, ob diese Belegung den Ausdruck tatsächlich erfüllt.

Hamiltonwege

Sei $G = (V, E)$ ein Graph. Frage: Existiert ein Weg, der jeden Knoten genau einmal besucht? Wir konstruieren eine polynomiell balancierte Sprache R, indem wir Paare von Graphen x, die einen Hamiltonweg besitzen, und einem solchen Weg y bilden. Wieder können wir in polynomieller Zeit prüfen, ob y tatsächlich ein Hamiltonweg in x ist, also liegt die zur Frage zugehörige Sprache in $\mathcal{NP}$.

$n \in \mathbb{N}$ zusammengesetzt (nicht prim)

Gegeben eine natürliche Zahl n. Frage: existieren $r, s > 1$ mit $n = rs$? Da wir in polynomieller Zeit multiplizieren können liegt die zugehörige Sprache in $\mathcal{NP}$.

Das komplementäre Problem PRIMES, ob n eine Primzahl ist, liegt somit in $co - \mathcal{NP}$, das heißt, daß ein negatives Zertifikat in polynomieller Zeit verifiziert werden kann. 1975 hat V. R. Pratt gezeigt, daß PRIMES auch in $\mathcal{NP}$ liegt, was zunächst recht erstaunlich anmutet. Seit 2002 ist nun allerdings bekannt, daß PRIMES in $\mathcal{P}$ liegt, was die bisherigen Ergebnisse impliziert. Mehr dazu auf Seite 88.

Rucksack-Problem

Gegeben einen Rucksack vom Volumen $V \in \mathbb{N}$ und k Gepäckstücke vom Volumen $v_0, ..., v_{k-1}$. Frage: Existiert ein $I \subseteq \{0, ..., k-1\}$ mit $\sum_{i \in I} v_i = V$? Auch hier liegt die zugehörige Sprache in $\mathcal{NP}$.

$\mathcal{P}$ versus $\mathcal{NP}$

Theorem 9.3. $\mathcal{P} \subseteq \mathcal{NP}$

Beweis. Zu $L \in \mathcal{P}$ definiere $R = \{(x, x) : x \in L\}$.

Es gibt Probleme in $\mathcal{NP}$, für die nicht bekannt ist, ob sie in $\mathcal{P}$ liegen. Von vielen erwartet man es auch nicht, z.B. von SAT. Die große offene Vermutung auf diesem Gebiet ist daher die folgende.

Hypothese 9.1. $\mathcal{P} \neq \mathcal{NP}$

$\mathcal{FP}$ und $\mathcal{FNP}$

Sei L eine Sprache in $\mathcal{NP}$. Das heißt, es existiert eine polynomiell entscheidbare, polynomiell balancierte Relation R_L, so daß $x \in L$ genau dann, wenn ein y existiert mit $(x, y) \in R_L$. Das zugehörige Funktionenproblem ist:

> Gegeben x, finde ein y mit $(x, y) \in R_L$, falls ein solches y existiert, und gebe ansonsten „nein" aus.

Das Problem nennt man FL.

Definition 9.9. *Die Klasse aller FL, die zu $L \in \mathcal{NP}$ korrespondieren, heißt $\mathcal{FNP}$. $\mathcal{FP}$ sei die Unterklasse von $\mathcal{FNP}$ all der Probleme, die in polynomieller Zeit lösbar sind.*

Wichtiges Beispiel: $FSAT \in \mathcal{FNP}$

Theorem 9.4. $\mathcal{P} = \mathcal{NP} \iff \mathcal{FP} = \mathcal{FNP}$

Der Beweis folgt aus

1. $FSAT$ ist $\mathcal{FNP}$-vollständig.
2. $FSAT \in \mathcal{FP} \iff SAT \in \mathcal{P}$.

Public-Key-Kryptographie und die Theorie der Einwegfunktionen

Das allgemeine Prinzip der Public-Key-Kryptographie ist das folgende:
Bob erzeugt ein Paar (e, d) von Schlüsseln und macht e öffentlich. Alice verschlüsselt ihre Nachricht x gemäß des Schlüssels $x \mapsto E_e(x)$, und Bob entschlüsselt sie via $D_d(E_e(x)) = x$. Damit das Kryptoverfahren sicher ist, darf es weder möglich sein, d aus e noch x aus $E_e(x)$ in einem vertretbaren Rechenaufwand zu ermitteln. Wichtiger Unterschied zum one-time-pad ist, daß Eve die Möglichkeit hat, ein x zu raten und nach der Verschlüsselung $x \mapsto E_e(x)$ zu überprüfen, ob dies mit der abgehörten Nachricht übereinstimmt. Deswegen soll E_e eine sogenannte Einweg-Funktion sein:

Definition 9.10. *Sei Σ ein endliches Alphabet und $f : \Sigma^* \to \Sigma^*$ eine Funktion. f heißt* Einweg-Funktion, *falls die folgenden drei Bedingungen gelten.*
(1) *f ist injektiv und $\forall x \in \Sigma^*$ gilt*

$$|x|^{\frac{1}{k}} \leq |f(x)| \leq |x|^k$$

für ein festes $k > 0$.
(2) *f ist in Polynomialzeit berechenbar: f ist in $\mathcal{FP}$.*
(3) *f^{-1} ist nicht in $\mathcal{FP}$, das heißt, es existiert kein polynomieller Algorithmus, der zu gegebenem $y \in \Sigma^*$ entweder ein $x \in \Sigma^*$ mit $f(x) = y$ findet oder ausgibt, daß es kein solches x gibt.*

Die Ungleichung in (1) bedeutet, daß sich die Wortlänge unter f nicht zu sehr verändert.

Bemerkung 9.1.
- zu (3): Grundsätzlich kann immer ein Urbild bestimmt werden: Erzeuge alle x bis zu der Längenschranke, die durch $f(x)$ vorgegebenen ist, und teste, ob dieses x das gesuchte ist. Das geht aber nicht in polynomieller Zeit.
- f^{-1} ist zwar nicht in $\mathcal{FP}$ aber in $\mathcal{FNP}$, denn es ist möglich, ein x zu raten und in polynomieller Zeit zu testen, ob $f(y)$ ein vorher gegebenes $y \in \Sigma^*$ ist.

Mit anderen Worten gilt also folgendes.

Proposition 9.1. *Falls Einwegfunktionen existieren, so gilt $\mathcal{P} \neq \mathcal{NP}$.*

Die Umkehrung dieser Aussage ist nicht bekannt. Der Sachverhalt läßt sich jedoch präzisieren. Da wir aber nicht noch tiefer in die Komplexitätstheorie einsteigen wollen und können, muß das folgende Skizzenhaft bleiben.

Definition 9.11. *Es sei $\mathcal{UP}$ die Klasse der Sprachen, die von einer unzweideutigen, nichtdeterministischen Turingmaschine, die in polynomieller Zeit operiert, akzeptiert werden. Hierbei heißt eine nichtdeterministische Turingmaschine unzweideutig, falls es zu jeder Eingabe höchstens eine akzeptierende Berechnung gibt.*

Es gilt dann offenbar $\mathcal{P} \subseteq \mathcal{UP} \subseteq \mathcal{NP}$. Und nun läßt sich zeigen.

Theorem 9.5. *Es gilt $\mathcal{P} \neq \mathcal{UP}$ genau dann, wenn Einwegfunktionen existieren.*

Man nimmt an, daß sowohl $\mathcal{P} \neq \mathcal{UP}$ als auch $\mathcal{UP} \neq \mathcal{NP}$ gilt.

Kandidaten für Einwegfunktionen

Nun wollen wir die beiden wichtigsten Kandidaten für Einwegfunktionen diskutieren, die für die Public-Key-Kryptographie relevant sind.

Exponenzieren modulo einer Primzahl

Wir betrachten die Funktion

$$f_{\text{exp}}(p,r,x) = \begin{cases} (p, r^x \mod p), & p \text{ prim}, r \text{ Primitivwurzel mod } p,\ 0 \leq x < p, \\ (p,r,x), & \text{sonst.} \end{cases}$$

f_{exp} ist tatsächlich in $\mathcal{FP}$, da inzwischen bekannt ist, daß es einen polynomiellen Algorithmus gibt, der testen kann, ob eine Zahl Primzahl ist. Dieses Primzahltestproblem wird mit $PRIMES$ bezeichnet, und es gilt $PRIMES \in \mathcal{P}$. Es war bereits seit 1975 bekannt, daß $PRIMES \in \mathcal{NP}$ gilt.

Multiplikation zweier Primzahlen

Wir betrachten die Funktion

$$f_{\text{Mult}}(p,q) = \begin{cases} pq, & p,q \text{ prim}, p < q, \\ (p,q), & \text{sonst.} \end{cases}$$

Die Primzahlbedingung ist für die Injektivität notwendig, da Multiplizieren im allgemeinen nicht injektiv ist. Es gilt wieder $f_{\text{Mult}} \in \mathcal{FP}$, da $PRIMES \in \mathcal{P}$.

Die Funktion f_{exp} haben wir bereits im Diffie-Hellman-Verfahren eingesetzt. f_{Mult} werden wir unter anderem im berühmten RSA-Verfahren einsetzen.

Das Rucksack-Kryptosystem

Als nächstes diskutieren wir ein Public-Key-Kryptosystem, welches auf dem Rucksack-Problem (siehe Seite 77) basiert und von Merkle und Hellman 1978 vorgestellt wurde. Das Rucksack-Problem ist $\mathcal{NP}$-vollständig, daß heißt, wenn ein polynomieller Algorithmus für das Rucksack-Problem gefunden wird, so folgt daraus $\mathcal{P} = \mathcal{NP}$. Deshalb war die Hoffnung berechtigt, mit Hilfe des Rucksack-Problems ein sicheres Kryptosystem zu bauen.

Zunächst brauchen wir hierfür noch eine Klasse leichter Instanzen des Rucksack-Problems.

Das superaufsteigende Rucksackproblem

Gegeben seien natürliche Zahlen $V, v_0 < v_1 < \cdots < v_{k-1}$. Falls für alle $j \geq 1$ die Ungleichung $v_j > \sum_{i=0}^{j-1} v_i$ gilt, so nennen wir die Instanz des Problems *superaufsteigend.*

Für diese Klasse von Instanzen existiert ein einfacher polynomieller Algorithmus.

Setze $j := k - 1$ und $W := V$. Falls $v_j \leq W$ gilt, so setze $\varepsilon_j := 1$ und ersetze W durch $W - v_j$, anderenfalls setze $\varepsilon_j := 0$. Ist $j = 0$ und $W > 0$ so gib aus "Keine Lösung!". Ist $j = 0$ und $W = 0$, so gib $I = \{j : \varepsilon_j = 1\}$ aus. Anderenfalls erniedrige j um eins und iteriere.

Im folgenden werden wir das Wort $\varepsilon_{k-1} \cdots \varepsilon_0$ mit der Menge I identifizieren.

Das Kryptosystem

Das Rucksack-Kryptosystem basiert nun aus dem Wechsel allgemeiner Instanzen und superaufsteigender Instanzen des Rucksackproblems. Das System funktioniert folgendermaßen.

Bob wählt zufällig ganze Zahlen $z_0, \ldots, z_k \geq 1$ und setzt

$$\begin{aligned} v_0 &:= z_0 \\ v_j &:= z_j + v_{j-1} + \cdots + v_0 \\ m &:= z_k + \sum_{i=0}^{k-1} v_i. \end{aligned}$$

Dann wählt er ein zufälliges a mit $0 < a < m$ und $(a, m) = 1$ und berechnet b mit $ab \equiv 1 \mod m$ und setzt $w_i := av_i \mod m$. *Bobs geheimer Schlüssel* besteht nun aus $(v_i), m, a, b$. Der *öffentliche Schlüssel* besteht aus (w_i).

Will *Alice eine Nachricht* $\varepsilon_{k-1} \cdots \varepsilon_0$ *versenden*, so berechnet sie die Zahl $c = \sum_{i=0}^{k-1} \varepsilon_i w_i$ mit Hilfe des öffentliche Schlüssels.

Das Entschlüsselungsproblem besteht für Bob nun darin, das Rucksackproblem zur Eingabe $(w_i), C$ zu lösen. *Bob entschlüsselt wie folgt.* Er berechnet $bc = \sum_{i=0}^{k-1} \varepsilon_i b w_i \equiv \sum_{i=0}^{k-1} \varepsilon_i v_i \mod m$ und wählt $V \equiv bc \mod m$ mit $0 \leq V < m$. Das gesuchte Wort $\varepsilon_{k-1} \cdots \varepsilon_0$ ist dann Lösung des superaufsteigenden Rucksackproblems zur Eingabe $(v_i), V$.

Diskussion

Die Transformation $v_i \mapsto w_i$ zerstört die Eigenschaft des superaufsteigens. Es schient also so, als wenn der unberufene Mithörer Eve mit einer allgemeinen Instanz des Rucksackproblems konfrontiert ist.

Wie Shamir 1982 jedoch zeigen sollte ist dem nicht so. Er beschrieb einen Algorithmus, der das Rucksackproblem auf den Instanzen $(w_i), c$ in polynomieller Zeit löst. Nun gibt es zwar Reparaturen dieses Verfahrens, das sogenannte *iterated knapsack cryptosystem,* aber der Glaube an Sicherheit ist dennoch nicht mehr vorhanden. Um Koblitz zu zitieren:

> In any case, most experts, traumatized by Shamir's unexpected breakthrough, do not have much confidence in the security of any public key cryptosystem of this type.

Gibt dies nun Anlaß zu glauben, daß $\mathcal{P} = \mathcal{NP}$ gilt? Nein! Die erzeugte Klasse der Instanzen ist eine echte Unterklasse aller Probleminstanzen. Dies sollte aber Anlaß geben, sich zu fragen, ob Einwegfunktionen, so wie wir sie definiert haben, Sicherheit für Public-Key-Kryptosysteme überhaupt gewährleisten. Hier ist die Antwort ebenfalls nein. Dies liegt an der Tatsache, daß die Komplexitätsklassen über ein"worst case"-Verhalten definiert werden. Für die Kryptographie ist dies aber nicht ausreichend. Man stellt sich nicht damit zufrieden, daß nur einige Nachrichten sicher verschlüsselt werden. Deshalb wird heutzutage der Punkt (3) in der Definition einer Einwegfunktion (auf Seite 78) üblicherweise in folgender Art definiert.

(3') Sei A eine beliebige probabilistsche Turingmaschine mit polynomieller Laufzeitbeschränkung. Dann existiert für jedes Polynom p ein N, sodaß für alle $n \geq N$ und $x \in \Sigma^n$ gilt:

$$\mathrm{Prob}(f(A(f(x))) = f(x)) < \frac{1}{p(n)}.$$

Eine *probabilistsche Turingmaschine* ist eine Turingmaschine, bei der der Wert $\delta(p, \sigma)$ der Übergangsfunktion von einer 0/1-wertigen Zufallsvariable, die die Werte 0 oder 1 jeweils mit Wahrscheinlichkeit $\frac{1}{2}$ annimmt, deterministisch abhängt. Diese können auch durch eine Turingmaschine modelliert werden, die ein zweites Band besitzen, welches zufallig mit einer unendlichen Folge von 0-en und 1-en beschrieben ist. Dieses Band wird von einem Lesekopf in jedem Schritt eingelesen, und um eine Stelle weitergeschoben. δ hängt dann von p, σ und diesem eingelesenen Wert deterministisch ab.

10

Primzahltests

Bei den im letzten Kapitel genannten Einwegfunktionen wurde bereits erwähnt, daß es wichtig ist, schnell entscheiden zu können, ob eine gegebene natürliche Zahl prim ist oder nicht. Erst seit 2002 ist bekannt, daß dieses Problem tatsächlich in polynomieller Zeit lösbar ist. Bevor wir auf dieses Ergebnis eingehen, wollen wir uns mit den klassischen Primzahltests beschäftigen.

Probedivision und Sieb des Erathostenes

Probedivision

Zunächst gilt offenbar folgendes.

Theorem 10.1. *Ist $n \in \mathbb{N}$ zusammengesetzt, so existiert ein Primteiler kleiner gleich $\sqrt{n}$.* □

Zu gegebenem $n \in \mathbb{N}$ reicht es also zu testen, ob es einen Primteiler kleiner gleich $\sqrt{n}$ gibt, um festzustellen, ob n prim ist. Wie erhalten wir nun eine Liste der Primzahlen $p \leq \sqrt{n}$?

Das Sieb des Erathostenes

Um die Primzahlen in einer gegebenen Liste $\{1, \ldots, m\}$ zu erhalten verfahren wir wie folgt.

Streiche die 1 in der Liste. Betrachte die kleinste verbleibende Zahl, die nicht als Primzahl markiert ist, markiere diese als prim, und streiche alle echten Vielfachen dieser Zahl in der Liste. Iteriere.

Aufwand

Für die Verteilung $\pi(x) = \#\{p : p \leq x, p \text{ prim}\}$ der Primzahlen wissen wir nun folgendes.

Theorem 10.2. *Es gilt die Asymptotik* $\pi(x) \sim \frac{x}{\ln x}$. *Genauer gilt für* $x \geq 59$

$$\frac{x}{\ln x}\left(1+\frac{1}{2\ln x}\right) < \pi(x) < \frac{x}{\ln x}\left(1+\frac{3}{2\ln x}\right).$$

□

Größenordnungmäßig gibt es demnach $\frac{\sqrt{n}}{\ln\sqrt{n}}$ viele Primzahlen kleiner gleich $\sqrt{n}$. Da dies exponentiell in $\log_2 n$ ist, ist die Probedivision im allgemeinen für größere n praktisch nicht durchführbar.

Der Fermat-Test und Carmichael-Zahlen

Die Idee des Fermat-Tests beruht auf dem kleinen Satz von Fermat (vergleiche Seite 17). □

Theorem 10.3. *Ist* n *prim und* $(a,n) = 1$, *so gilt* $a^{n-1} \equiv 1 \mod n$.

Findet man also eine Basis a mit $(a,n) = 1$ und $a^{n-1} \not\equiv 1 \mod n$, so weiß man mit Gewissheit, daß n keine Primzahl ist.

Beispiel 10.1. Betrachten wir $n = 341$ und die Basis 2 und 3.

$$2^{340} = (2^{10})^{34} = 1024^{34} \equiv 1^{34} = 1 \mod 341$$
$$3^{340} \equiv 56 \mod 341$$

Die Basis 3 gibt also Aufschluß über die Nichprimalität von $n = 341 = 11 \cdot 31$, im Gegensatz zur Basis 2.

Diesem Sachverhalt wollen wir mit einer Definition begegnen.

Definition 10.1. *Seien* n *und* a *gegeben und es gelte* $(a,n) = 1$. n heißt Pseudoprimzahl zur Basis a, *falls* $a^{n-1} \equiv 1 \mod n$ *gilt.*

Entscheidend für die Konstruktion eines Primzahltests ist die folgende Abschätzung.

Theorem 10.4. *Existiert eine Basis* b *mit* $(b,n) = 1$ *und* $b^{n-1} \not\equiv 1 \mod n$. *Dann gilt dies für mindestens die Hälfte aller Basen* b' *mit* $1 \leq b' \leq n-1$, *für die* $(b',n) = 1$ *gilt.*

Beweis. Sei $A = \{a : 1 \leq a \leq n-1, (a,n) = 1, a^{n-1} \equiv 1 \mod n\}$ die Menge der Basen, bezüglich derer n Pseudoprimzahl ist. Sei nun b gegeben mit $(b,n) = 1$ und $b^{n-1} \not\equiv 1 \mod n$. Für $a \in A$ ist $(ba)^{n-1} = b^{n-1}a^{n-1} \equiv b^{n-1} \not\equiv 1 \mod n$. Also ist n keine Pseudoprimzahl bezüglich ba. Ferner seien $a, a' \in A$ mit $a \neq a'$ gegeben. Angenommen $ba \equiv ba' \mod n$. Dann folgt $n|b(a-a')$ und wegen $(b,n) = 1$ folgt $n|(a-a')$, was aber nicht sein kann, da $1 \leq a, a' \leq n-1$. Demnach hat die Menge

$$\{c : 1 \leq c \leq n-1, \exists a \in A : ba \equiv c \mod n\}$$

genauso viele Elemente wie A und besteht ausschliesslich aus Basen, bezüglich derer n keine Pseudoprimzahl ist. □

Der Fermat-Test

Nun beschreiben wir den Fermat-Test. Wähle $a \in \{1, \ldots, n-1\}$ zufällig gleichverteilt mit $(a, n) = 1$, und teste die Kongruenz $a^{n-1} \equiv 1 \mod n$. Ist die Kongruenz nicht erfüllt so ist n zusammengesetzt. Mit dem letzten Theorem folgt dann also.

Theorem 10.5. *Iteriert man den Fermat-Test k mal, so lässt sich mit Wahrscheinlichkeit grösser gleich $1 - \frac{1}{2^k}$ wahrheitsgemäß sagen, ob n zusammengesetzt ist, oder ob für alle Basen $a \in \{1, \ldots, n-1\}$ mit $(a, n) = 1$ die Kongruenz $a^{n-1} \equiv 1 \mod n$ gilt.*

Carmichael-Zahlen

Das letzte Theorem wirft nun die Frage auf, ob der Fermat-Test nicht mit gleicher Wahrscheinlichkeit zusammengesetzte von Primzahlen unterscheiden kann. Dem ist aber nicht so!

Definition 10.2. *Eine natürliche Zahl n heißt* Carmichael-Zahl*, falls sie zusammengesetzt ist, und für alle Basen $a \in \{1, \ldots, n-1\}$ mit $(a, n) = 1$ die Kongruenz $a^{n-1} \equiv 1 \mod n$ gilt.*

Tatsächlich gibt es solche Zahlen. 1994 wurde von Alford, Granville und Pomerance sogar gezeigt, daß es unendlich viele Carmichael-Zahlen gibt.

Mit diesen Zahlen wollen wir uns ein bißchen beschäftigen. Zunächst bemerken wir, daß gerade zusammengesetzte Zahlen niemals Carmichael-Zahlen sind. Denn es gilt für eine solche

$$(n-1)^{n-1} \equiv (-1)^{n-1} \equiv -1 \not\equiv 1 \mod n.$$

Und wegen $(n-1, n) = 1$ bezeugt $n - 1$, daß n keine Carmichael-Zahl ist.

Theorem 10.6. *Sei n ungerade und zusammengesetzt, dann ist n eine Carmichael-Zahl genau dann, wenn für jeden Primteiler $p|n$ gilt $p^2 \nmid n$ und $p - 1|n - 1$.*

Für den Beweis und auch für den noch folgenden Müller-Rabin-Test brauchen wir folgendes Ergebnis.

Übung 10.1. Sei $g + p\mathbb{Z} \in \mathbb{Z}_p \setminus \{0\}$ ein Erzeuger der multiplikativen Gruppe, das heißt ein Element der Ordnung $p-1$. Zeige, daß $g+p^2\mathbb{Z}$ oder $(p+1)g+p^2\mathbb{Z}$ die prime Restklassengruppe $\mathbb{Z}_{p^2}$ der Ordnung $p(p-1)$ erzeugt.

Der Miller-Rabin-Test

Idee

Angenommen n ist eine Pseudoprimzahl zur Basis a mit $(a, n) = 1$. Also gilt $a^{n-1} \equiv 1 \mod n$. Ziehe sukzessive Quadratwurzeln aus a^{n-1}, d.h. berechne

$a^{\frac{n-1}{2}}, a^{\frac{n-1}{2^2}}, \ldots, a^{\frac{n-1}{2^s}}$, wobei $s = max\{r : 2^r \mid n-1\}$. Die erste Restklasse $\not\equiv 1$ mod n muss $\equiv -1$ mod n sein, falls n prim ist, da die Gleichung $x^2 \equiv 1$ mod n für primes n nur die Lösungen ± 1 mod n besitzt. Setze $d := \frac{n-1}{2^s}$.

Bermerkung: d und s sind schnell zu berechnen.

Der Test

Gegeben $n \geq 3$. Berechne d und s. Wähle $a \in \{2, \ldots, n-1\}$ zufällig gleichverteilt. Berechne $g = (a, n)$. Ist $g > 1$, so ist n zusammengesetzt. Ist $g = 1$, so berechne $a^d, a^{2d}, \ldots, a^{2^s d}$. Prüfe, ob vor dem Auftreten der ersten 1 in dieser Folge -1 steht (mod n). Ist dies nicht der Fall, so gib aus, daß n zusammengesetzt ist.

Theorem 10.7. *Ist n Primzahl, $(a, n) = 1$, so gilt entweder*

(i) $a^d \equiv 1$ mod n oder
(ii) es existiert ein $r \in \{0, \ldots, s-1\}$ mit $a^{2^r d} \equiv -1$ mod n.

Beweis. $\mathbb{Z}_n^*$ hat Ordnung $n - 1 = 2^s d$. Damit hat a^d Ordnung $2^j = k$ für ein $j \in \{0, \ldots, s\}$. Falls $k = 1$, also $j = 0$ gilt *(i)*. Falls $k > 1$ gilt also $1 \leq j \leq s$. Also hat $a^{2^{j-1}d}$ Ordnung 2, d.h. $a^{2^{j-1}d} \equiv 1$ mod n, da n prim. Ferner ist $r = j - 1 \in \{0, \ldots, s-1\}$.

Beispiel 10.2. Es seien $n = 561 = 3 * 11 * 17, a = 2, s = 4, d = 35$. Nun ist $2^{35} \equiv 263, 2^{2*35} \equiv 166, 2^{4*35} \equiv 67, 2^{8*35} \equiv 1$ (mod 561). Also ist n nicht prim.

Theorem 10.8. *Ist $n \geq 3$ ungerade und zusammengesetzt, so gibt es in der Menge $\{1, \ldots, n-1\}$ höchstens $\frac{n-1}{4}$ viele Elemente relativ prim zu n, die keine Zeugen gegen die Primalität von n sind.*

Bemerkung 2. Ist n keine Pseudoprimzahl zur Basis a, $(a, n) = 1$, dann ist a Zeuge gegen die Primalität von n im Sinne des Miller-Rabin-Test.

Lemma 10.1. *Sei $G = \{1, g, \ldots, g^{m-1}\}$ zyklische Gruppe der Ordnung m. Sei $k \geq 1$, dann hat $x^k = 1$ genau $d = (m, k)$ viele Lösungen in G.*

Beweis. Sei $x = g^j$, $j \in \{0, \ldots, m-1\}$ eine Lösung. $x^k = g^{jk} = 1 \iff m \mid jk \iff \frac{m}{d} \mid j\frac{k}{d} \iff \frac{m}{d} \mid j \iff j \in \{0, \frac{m}{d}, \frac{2m}{d}, \ldots, (d-1)\frac{m}{d}\}$.

Lemma 10.2. *Sei p eine ungerade Primzahl, e ungerade. Wie zuvor sei $p - 1 = 2^s d, d$ ungerade. Dann ist die Anzahl der $x \in \mathbb{Z}_p^*$ mit $x^{2^r e} \equiv -1$ mod p gleich $\begin{cases} 0 & r \geq s \\ 2^r(d,e) & r < s \end{cases}$.*

Beweis. Sei $g \in \mathbb{Z}_p^*$ ein multiplikativer Erzeuger und $x = g^j$ für $j \in \{0, \ldots, p-1\}$ (bemerke $g^{\frac{p-1}{2}} \equiv -1$ mod p). Dann ist $x^{2^r e} = g^{j2^r e} \equiv -1$ mod $p \iff j2^r e \equiv \frac{p-1}{2}$ mod $p - 1 \iff j2^r e \equiv 2^{s-1}d$ mod $e^s d$. Gesucht: # der j, die diese Kongruenz erfüllen, $j \in \{0, \ldots, p-1\}$. Falls $r \geq s$: $\quad j2^r e = 2^{s-1}d + k2^s d$ Falls $r < s$: $\quad$ Sei $t = (2^r e, 2^s d) = 2^r(e, d)$ und $2^r(e, d) \mid 2^{s-1}d$. Dann existieren bekannter Maßen genau t viele Lösungen.

Beweis (von Theorem 10.8).

Fall 1:
n teilbar durch ein Primquadrat, etwa $p^\alpha \mid n$, $\alpha \geq 2$, $p^{\alpha+1} \not| n$ Zeigen: Es existieren höchstens $\frac{n-1}{4}$ Basen in $\{1, \ldots, n-1\}$, bezüglich derer n Pseudoprimzahl ist. Angenommen: $b^{n-1} \equiv 1 \bmod n$, also gilt $b^{n-1} \equiv 1 \bmod p^2$. Da $\mathbb{Z}^*_{p^2}$ zyklisch von der Ordnung $p(p-1)$ ist, gilt nach Lemma 1, daß $d = (p(p-1), n-1)$ viele Lösungen für b existieren. Nun gilt $p \mid n$, also $p \not| n-1$, damit gilt $d \leq p-1$. Also gibt es höchstens $\frac{p-1}{n-1} \leq \frac{p-1}{p^2-1} = \frac{1}{p+1} \leq \frac{1}{4}$ (da n ungerade) Basen bezüglich derer n Pseudoprimzahl ist.

Fall 2:
n ist Produkt von zwei Primzahlen $n = pq, p \neq q$. $n-1 = 2^s d, p-1 = 2^{s'} d', q-1 = 2^{s''} d'', d, d', d''$ ungerade und OBdA $1 \leq s' \leq s''$. Für Nichtzeugen b muss gelten entweder (i) $b^\alpha \equiv 1 \bmod p$ und $b^\alpha \equiv 1 \bmod q$ oder (ii) es existiert $r \in \{0, \ldots, s-1\}$ mit $b^{2^r d} \equiv -1 \bmod p$ und $b^{2^r d} \equiv -1 \bmod q$. Für (i) gibt es $(d, p-1)(d, q-1)$ viele Lösungen. $(d, p-1) = (d, d'') =: a'$ $(d, q-1) = (d, d'') =: a''$ Nach Lemma 2 ist für $r < min\{s', s''\} = s'$ die Anzahl der Lösungen $2^r(d, d')2^r(d, d'') = 4^r a' a''$. Die Anzahl der Lösungen ist also höchstens: $a' + a'' + \sum_{r=0}^{s'-1} 4^r a' a''$. Nun ist $(p-1)(q-1) = 2^{s'+s''} d' d'' \leq n-1$, der Anteil der Nichtzeugen ist also höchstens

$$\frac{a'a'' + a'a'' \sum_{r=0}^{s'-1} 4^r}{2^{s'+s''} d'd''} = 2^{-s'-s''} \frac{a'a''(1 + \frac{4^{s'-1}}{4-1})}{d'd''}. \tag{10.1}$$

Falls nun $s' < s''$, dann gilt (10.1) $\leq 2^{-s'-s''} \frac{3+4^{s'}-1}{3} \leq 2^{-2s'-1}(\frac{2}{3} + \frac{4^{s'}}{3}) \leq 2^{-3}\frac{2}{3} + \frac{2^{-2s'-1}2^{2s'}}{3} = 2^{-3}\frac{2}{3} + \frac{1}{6} = \frac{1}{4}$. Falls $s' = s''$, muss eine der Ungleichungen $a' = (d, d') \leq d'$ und $a'' = (d, d'') \leq d''$ strikt sein, denn sonst gälte $d' \mid d$ und $d'' \mid d$. Betrachte $n - 1 = 2^s d = pq - 1 = q - 1 + (p-1)q \equiv q - 1 \bmod p - 1$. Wegen $d' \mid d$ folgt $q - 1 \equiv 0 \bmod d'$ bzw. $d' \mid 2^{s''} d''$ bzw. $d' \mid d''$. Umgekehrt gilt auch $d'' \mid d'$. Damit gälte aber $p = q$ (da auch $s' = s''$). Also muss eine der Ungleichungen strikt sein. Deshalb gilt $a' = a'' \leq 3d'd''$. Somit erhält man für den Quotienten von oben: (1) $\leq 2^{-2s'} \frac{1}{3}(\frac{2+4^{s'}}{3}) \leq \frac{1}{6} \leq \frac{1}{4}$.

Fall 3:
$n = p_1 \ldots p_k, k \geq 3, p_i$ paarweise verschiedene Primzahlen Schreibe $p_j - 1 = 2^{s_j} d_j, d_j$ ungerade. Genau wie im Fall 2 folgt mit $(d, d_j) \leq d_j$ für den Quotienten die Abschätzung (OBdA $s_1 \leq s_j f.a.j$) $\leq 2^{-s_1 \cdots - s_k}(1 + \frac{2^{ks_1}-1}{2^{k-1}}) \leq 2^{-ks_1} \frac{(2^{k-1} + s^{ks_1} - 1}{2^{k-1}}) \leq \frac{1}{2^{k-1}} \leq \frac{1}{4}$, da $k \geq 3$.

Laufzeit

Der k-fach iterierte Miller-Rabin-Test läuft in $O(klog^2 n)$.

PRIMES $\in \mathcal{P}$

Schon seit den 70er Jahren ist bekannt, daß PRIMES $\in \mathcal{NP} \cap \text{co} - \mathcal{NP}$ gilt. Mit den im August 2002 veröffentlichen Ergebnissen von Agrawal, Kayal und Saxena ist nun bewiesen, daß PRIMES $\in \mathcal{P}$ ist. Hier wird nicht der gesamte Beweis gebracht, sondern lediglich eine Idee gegeben.

Die grobe Idee

1. Seien $(a, n) = 1$. Dann gilt
$$n \text{ ist prim} \Leftrightarrow (x-a)^n \equiv x^n - a \pmod{n}$$
Die "$\Rightarrow$"-Richtung ist eine direkte Folge des kleinen Fermatschen Satzes.
Die "$\Leftarrow$"-Richtung ist ebenfalls einfach.
Das Problem, dieses Ergebnis anzuwenden, ist, daß die Berechnung von $(x-a)^n$ aufwendig ist.
2. Abschwächung des Satzes:
$$n \text{ ist prim} \Rightarrow (x-a)^n \equiv x^n - a \pmod{x^r - 1, n}.$$
Hierbei gilt die Umkehrung i.A. nicht.
Vorteil: Ist r von der Größenordnung $\log n$, kann man $(x-a)^n \mod x^r - 1$ in polynomialer Zeit berechnen. Frage: Existiert ein solches r und Größenordnung $\log a$ viele a, so daß das Testen dieser a ausreicht, um schließen zu können, daß n prim ist.
3. **Theorem 10.9 (von Agrawal, Kayal, Saxena).** *Sei $n \in \mathbf{N}$ gegeben. Seien q, r prim und $s \leq n$ so gewählt, daß $q \mid r-1$, $n^{\frac{r-1}{q}} \equiv 0, 1 \pmod{r}$ und $\frac{q+s-1}{s} \geq n^{2\lfloor\sqrt{n}\rfloor}$ ist. Falls für alle $1 \leq a \leq s$*
   a) $(a, n) = 1$,
   b) $(n-a)^n \equiv x^r - 1 \pmod{x^r - 1, n}$
   gilt, so ist n eine Primzahlpotenz.
4. Damit lässt sich der folgende Algorithmus erstellen: n ungerade sei gegeben.
   a) Entscheide, ob n eine Potenz einer natürlichen Zahl ist, (dies ist in polynomialer Zeit möglich). Falls ja, stoppt der Algorithmus mit der Ausgabe "zusammengesetzt"
   b) Wähle q, r, s wie im Satz.
   c) Für $a = 2, \ldots, s-1$ teste
      - $a \mid n$. Ist dies wahr, so ist n zusammengesetzt und der Algorithmus stoppt, anderenfalls gilt $(a, n) = 1$.
      - $(x-a)^n \equiv x^n - a \pmod{x^r - 1, n}$. So ist n zusammengesetzt.
   d) n ist prim.

Wesentlich in diesem Algorithmus ist Schritt 4b, denn es war die Existenz eines $\delta \geq \frac{1}{6}$ notwendig, so daß ein c_δ existiert und die folgende Ungleichung gilt:

$$\{r \leq x : q, r \text{ prim}, q \mid r-1, q > x^{\frac{1}{2}+\delta}\} \geq c_\delta \frac{x}{\ln x}.$$

Ein solches δ existiert aber nach einem bekannten Satz der Zahlentheorie. Die Laufzeit des Algorithmus ist $O(\log^{12} n)$.

5. Existieren unendlich viele Primzahlen q, so daß $2q+1$ ebenfalls prim ist (Germain-Primzahlen), mit einer gewissen Dichte, dann lässt sich die Laufzeit auf im wesentlichen $O(\log^6 n)$ verbessern.

Bislang ist dieser Algorithmus allerdings gegenüber schnellen probabilistischen Tests nicht konkurrenzfähig. Der Miller-Rabin-Test hat eine Laufzeit von $O(\log^2 n)$ und entscheidet mit einer sehr hohen Wahrscheinlichkeit. Das Resultat ist aber theoretisch von entscheidener Bedeutung, denn es hat die Existenz eines polynomialen Algorithmus gezeigt. Das Resultat ist also PRIME $\in \mathcal{P}$.

11

Public Key Kryptographie Teil II

Das RSA-Kryptosystem

Dieses Kryptosystem wurde 1977 von Rivest, Shamir und Adleman entwickelt. Man beachte die nicht-alphabetische Reihenfolge, diese ist entstanden, da Adleman überzeugt war, nicht so, wie die beiden anderen, an dem System mitgearbeitet zu haben.

Betrachten wir wieder Alice und Bob.

Bobs Erzeugung der Schlüssel

Bob wählt zufällig zwei Primzahlen $p \neq q \geq 3$ und berechnet $n = pq$. Weiter wählt er $1 < e < \phi(n) = (p-1)(q-1)$ mit $(e, \phi(n)) = 1$ und berechnet mittels Euklidischem Algorithmus d mit $ed \equiv 1 \pmod{\phi(n)}$. Hierbei seien folgende Bezeichnungen eingeführt: n heißt RSA-Modul, e Verschlüsselungsexponent, d heißt Entschlüsselungsexponent sowie das Paar (n,e) der öffentliche und das Paar (n,d) der private RSA-Schlüssel. Bob veröffentlicht das Paar (n, e).

Alices Verschlüsselung

Alice verschlüsselt eine Zahl m mit $0 \leq m < n$ durch die Verschlüsselungsfunktion $m \mapsto m^e \mod n$ mit den von Bob veröffentlichten n und e. Möchte Alice einen Binärtext verschlüsseln, so zerlegt sie diesen in Blöcke der Länge $\lfloor \log_2 n \rfloor$. Jeder Block ist dann eine solche Zahl m. Alice sendet Bob die so verschlüsselte Nachricht.

Bobs Entschlüsselung

Bob entschlüsselt die von Alice erhaltene Nachricht mittels des folgenden Satzes.

Theorem 11.1. *Seien* (n, e) *ein öffentlicher und* (n, d) *ein privater RSA-Schlüssel. Dann gilt* $(m^e)^d \equiv m \pmod{n}$.

Beweis. Es existiert ein $k \in \mathbb{Z}$ *mit* $ed = 1 + k\phi(n)$. *Gilt* $p \mid m$, *so gilt* $(m^e)^d \equiv m \pmod{p}$. *Gilt* $p \nmid m$, *so gilt*

$$m^{ed} = (m^{\phi(n)})^d m \equiv (m^{p-1})^{k(q-1)} m \equiv m \pmod{p}.$$

Für q *analog. Damit gilt* $(m^e)^d \equiv m \pmod{pq}$.

Bemerkungen zur Sicherheit

Ist die Faktorisierung von n bekannt lässt sich $\phi(n)$ berechnen.

Anmerkung 11.1. Weiter ist die Kenntnis von $\phi(n)$ und n äquivalent zur Kenntnis von p, q und n.

Beweis. Die eine Richtung ist trivial. Sei also $\phi(n)$ bekannt. Dann gilt $\phi(n) = (p-1)(q-1) = n - (p+q) + 1$ und mit $r := p + q$ ist dann $p = r - q$. Aus $\phi(n) = pq = (r-q)q$ folgt dann $q^2 - rq - n = 0$.

Ferner gilt, daß wenn n, e, d bekannt sind, so lässt sich n mittels eines schnellen probabilistischen Algorithmus berechnen.

Es ist unbekannt, ob es notwendig ist, die Faktorisierung von n zu kennen, um RSA zu brechen.

Wahl von n bzw. p und q

Momentan empfohlene Bitlänge für p und q in der Größenordnung 1024.

Wahl von e

Es sollte e nicht zu klein gewählt werden, denn sonst kann die "low-exponent-attac" angewandt werden. Diese folgt aus dem folgenden Satz:

Theorem 11.2. *Sei* $e \in \mathbb{N}$, $n_1, \ldots, n_e \in \mathbb{N}$ *paarweise teilerfremd und* $m \in \mathbb{N}$ *mit* $0 \leq m \leq n_i$ *für alle* i. *Sei nun* $c \in \mathbb{N}$ $c \equiv m^e \pmod{n_i}$ *für alle* i *und* $0 \leq c \leq \prod_{i=1}^{e} n_i$. *Dann gilt*

$$c = m^e.$$

Der Beweis wird mittels des chinesischen Restsatzes durchgeführt. Die Attacke ist also die Verschlüsselung der immer gleichen Nachricht m mit den öffentlichen Schlüsseln $(n_1, e), \ldots, (n_e, e)$.

Vertiefende Sicherheitsdiskussion

Kommen wir jetzt zum Beweis, daß, wenn neben n und e auch d bekannt ist, n einfach faktorisiert werden kann. Seien hierzu s, k (k ungerade), so daß $ed - 1 = 2^s k$ gilt.

Lemma 11.1. *Gilt* $(a, n) = 1$, *so hat* $a^k \in \mathbb{Z}_n$ *Ordnung* 2^i *für* $0 \leq i \leq s$.

Beweis. Es gilt $(a^k)^{2^s} = a^{k2^s} \equiv 1 \pmod{n}$. Also $\operatorname{ord}(a^k \mod n) \mid 2^s$.

Anmerkung 11.2. Analoges gilt für $\mod p$ und $\mod q$.

Theorem 11.3. *Ist* $(a, n) = 1$. *Falls die Ordnung von* $a^k \mod p$ *und* $\mod q$ *verschieden ist, so gilt*

$$1 < (a^{2^t k} - 1, n) < n$$

für ein $t \in \{0, 1, \ldots, s-1\}$.

Beweis. O.B.d.A. sei $\operatorname{ord}(a^k \mod p) > \operatorname{ord}(a^k \mod q) = 2^t$. Dann gilt $a^{2^t k} \equiv 1 \pmod{q}$ und $a^{2^t k} \not\equiv 1 \pmod{q}$. Damit gilt $(a^{2^t k} - 1, n) = q$.

Theorem 11.4. *Die Anzahl der* $a \in \{1, \ldots, n-1\}$ *mit* $(a, n) = 1$ *für die* ($a^k \mod p$) *und* ($a^k \mod q$) *in* $\mathbb{Z}_n$ *verschiedene Ordnung haben, ist mindestens* $\frac{\phi(n)}{2} = \frac{(p-1)(q-1)}{2}$.

Beweis. Sei g Primitivwurzel $\mod p$ und $\mod q$. Für diesen Beweis sei $o_r(h) = \operatorname{ord}(h \mod r)$ eine Abkürzung.

1. Angenommen es gilt $o_p(g^k) > o_q(g^k)$. Sei nun $x \in \{1, \ldots, p-1\}$ ungerade und $y \in \{1, \ldots, p-2\}$ beliebig. Betrachte die Lösungen a der Kongruenzen
$$\begin{aligned} a &\equiv g^x \pmod{p} \\ a &\equiv g^y \pmod{q}. \end{aligned} \tag{11.1}$$
Dann gilt $o_p(a^k) = o_p(g^{xk}) = o_p(g^k)$, da $o_p(g^k)$ 2er-Potenz ist und x ungerade gewählt wurde. Ferner gilt $o_q(a^k) = o_q(g^{yk}) \leq o_q(g^k) < o_p(g^k) = o_p(a^k)$. Also gilt $o_q(a^k) \neq o_p(a^k)$. Da g Primitivwurzel $\mod p$ und $\mod q$ ist, sind alle Lösungen von a verschieden. Dies sind aber genau $\frac{p-1}{2}(q-1)$ viel.
2. Betrachte den Fall $o_p(g^k) = o_q(g^k) \geq 2$. $o_p(g^k) \geq 2$ gilt, da $p-1$ und $q-1$ gerade ist. Sei a wieder Lösung von 11.1, wobei jetzt entweder x gerade und y ungerade oder umgekehrt x ungerade und y gerade gilt.
Betrachte Fall x gerade und y ungerade. Es gilt $o_q(a^k) = o_q(g^k)$ und $o_p(a^k) = o_p(g^{xk}) < o_p(g^k)$, da x gerade und ≥ 2 ist. Also gilt $o_q(a^k) > o_p(a^k)$.
Der umgekehrte Fall geht analog.
Zählen wir jetzt diese Fälle, so erhält man $\frac{p-1}{2}\frac{q-1}{2} + \frac{p-1}{2}\frac{q-1}{2} = \frac{(p-1)(q-1)}{2}$.

Damit existiert der folgende

Faktorisierungsalgorithmus

Wähle $a \in \{1, \ldots, n-1\}$ zufällig gleichverteilt. Gilt $(a, n) > 1$, so ist man fertig. Anderenfalls berechne $g_z = (a^{2^t k} - 1, n)$ für $t = s-1, s-2, \ldots, 0$. Findet man $g_z > 1$, so ist man fertig. Iteriere dies r mal.

Theorem 11.5. *Die Wahrscheinlichkeit, daß das Verfahren* n *faktorisiert ist* $\geq 1 - (\frac{1}{2})^r$.

Beweis.

$$\frac{\frac{(p-1)(q-1)}{2}}{\phi(n)} = \frac{1}{2}.$$

Das Rabin-Verfahren

Dies Verfahren ist ähnlich zu RSA, besitzt jedoch den großen Vorteil, daß es äquivalent zum Faktorisieren des Moduls ist.

Das Verfahren

Berechne $n = pq$, wobei p, q prim mit $p \equiv q \equiv 3 \pmod 4$ ist. Öffentlicher Schlüssel ist n, der geheime Schlüssel ist das Paar (p, q). Die Verschlüsselung geschieht, indem man den Klartext $m \in \{0, \ldots, n-1\}$ mittels $m \mapsto c \equiv m^2 \pmod n$ abbildet.

Die Entschlüsselung geschieht nach folgendem Verfahren. Berechne

$$\begin{aligned} m_p &\equiv c^{\frac{p+1}{4}} \pmod p \\ m_q &\equiv c^{\frac{q+1}{4}} \pmod q. \end{aligned}$$

Dann gilt: $(\pm m_p)^2 \equiv c^{\frac{p+1}{2}} \equiv m^{p+1} \equiv m^2 \pmod p$, analog für m_q. Nach dem euklidischen Algorithmus existieren $x, y \in \mathbb{Z}$ mit $px + qy = 1$ und setze

$$\begin{aligned} r &\equiv xpm_q + yqm_p \pmod n, \\ s &\equiv xpm_q - yqm_p \pmod n. \end{aligned}$$

Dann sind $(\pm r)$ und $(\pm s)$ die Quadratwurzeln von n, da

$$\begin{aligned} (\pm r)^2 &= x^2p^2m_q^2 + 2xpm_qyqm_p + y^2q^2m_p^2 \\ &\equiv y^2q^2m_p^2 \pmod p \\ &\equiv m_p^2 \pmod p \\ &= c \end{aligned}$$

gilt, analog berechnet man $\bmod q$. Eine der vier Quadratwurzeln ist m. Zur Auswahl der richtigen Quadratwurzel versehe m mit zusätzlicher Struktur.

Zur Erzeugung der Schlüssel

Theorem 11.6 (von Dirichlet). *$(n, k) = 1$, so existieren unendlich viele Primzahlen p mit $p \equiv k \pmod{n}$. Die asymptotische Dichte ist ebenfalls bekannt.*

Für das hier vorgestellte Rabin-Verfahren ist nur der folgende einfach zu beweisende Spezialfall notwendig:

Theorem 11.7. *Es existieren unendlich viele Primzahlen p mit $p \equiv 3 \pmod{4}$.*

Beweis. Angenommen es existieren nur endlich viele solcher Primzahlen. Sei p die größte davon. Betrachte

$$N := \underbrace{3 \cdot \ldots \cdot p}_{\text{ungerade Primzahlen } \leq p} \cdot 4 - 1$$

Dann gilt:

- $N \equiv 3 \pmod{4}$
- N ist nicht prim, da $N > p$.
- keine Primzahl $\leq p$ teilt N.

Sind q, r prim mit $q \equiv r \equiv 1 \pmod{4}$, so folgt $qr \equiv 1 \pmod{4}$. Sei $q = 1+4k$, $r = 1 + 4l$. Dann gilt

$$qr = 1 + 4(k + l) + 16kl \equiv 1 \pmod{4}.$$

Also muss N eine Primfaktorzerlegung mit $q > p$ und $q \equiv 3 \pmod{4}$ besitzen. Dies ergibt einen Widerspruch.

Zur Sicherheit des Rabin-Verfahren

Wird das Rabin-Verfahren gebrochen, lässt sich mit hoher Wahrscheinlichkeit n faktorisieren. Genauer definieren wir, wann das Rabin-Verfahren gebrochen ist, wie folgt: Es existiert ein polynomialer Algorithmus, der bei gegebenen Eingabewert $c \in \{1, \ldots, n-1\}$ eine ganze Zahl $R(c) =: m \in \{1, \ldots, (n-1)\}$ berechnet, so daß $m^2 \equiv c \pmod{n}$ gilt.

Betrachte ein zufällig gewähltes $x \in \{1, \ldots, (n-1)\}$ mit $(x, n) = 1$. Definiere $c \equiv x^2$ und setze $m := R(c)$. Es können dann die folgenden vier Fälle eintreten:

1. $m \equiv x \pmod{p}$ und $m \equiv x \pmod{q}$,
2. $m \equiv -x \pmod{p}$ und $m \equiv -x \pmod{q}$,
3. $m \equiv x \pmod{p}$ und $m \equiv -x \pmod{q}$ und
4. $m \equiv -x \pmod{p}$ und $m \equiv x \pmod{q}$.

Für die beiden ersten Fälle gilt: $m \equiv x \pmod{n}$ bzw. $m \equiv -x \pmod{n}$.

Im dritten Fall gilt: $m - x \equiv 0 \pmod{p}$ und $m + x \equiv 0 \pmod{q}$. Klar ist, daß $p \mid (m - x)$ gilt, wir behaupten zusätzlich, daß $q \nmid (m - x)$ gilt. Angenommen es gilt $q \mid (m - x)$, so folgt $m - x \equiv 0 \pmod{q}$. Zusammen mit $m + x \equiv 0 \pmod{q}$ folgt also $2x \equiv 0 \pmod{q}$. Damit folgt aber $q \mid x$ und somit $(x, n) = q$. Dies ist aber ein Widerspruch zur Voraussetzung $(x, n) = 1$. Der vierte Fall folgt analog.

Festzustellen ist, daß mit Wahrscheinlichkeit $\frac{1}{2}$ ein Teiler von n gefunden wird. Damit ergibt sich folgender

Algorithmus

1. Wähle $x \in \{1, \ldots, (n-1)\}$ und berechne $q = (x, n)$. Ist > 1, so ist g ein Faktor von n.
2. Anderenfalls bestimme $c \equiv x^2 \pmod{n}$ und $m = R(c)$. Berechne weiter $(m - x, n)$. Ist $(m - x, n) \neq 1$ so ist ein Faktor von n gefunden.

Iteriere den Algorithmus r mal.

Zusammenfassend gilt: Ist das Rabin-Verfahren gebrochen, so lässt sich mit Wahrscheinlichkeit $\geq 1 - (\frac{1}{2})^r$ ein Teiler von n finden.

12

Faktorisierungsalgorithmen

Wie wir in den letzten Abschnitten gesehen haben, steht und fällt die Sicherheit vieler Kryptosysteme mit der Annahme, daß es schwierig ist, große Zahlen zu faktorisieren. In der Tat ist bis heute nur sehr wenig über die Komplexität des Faktorisierens bekannt. Natürlich ist klar, daß das Problem in $\mathcal{FNP}$ liegt, da man die Faktoren einer zusammengesetzten Zahl einfach raten kann und schnell überprüfen kann, ob man richtig geraten hat, indem man die geratenen Zahlen miteinander multipliziert. Außerdem ist seit 2002 bekannt, daß das *Entscheidungsproblem*, ob eine Zahl zusammengesetzt ist, in Polynomialzeit lösbar ist, da sich effizient testen läßt, ob eine Zahl Primzahl ist. Das bedeutet allerdings noch lange nicht, daß es einfach ist, die Primfaktoren einer Zahl zu bestimmen.

Bemerkung 12.1. Es ist interessant zu bemerken, daß das Problem, natürliche Zahlen zu Faktorisieren, in der Komplexitätsklasse $\mathcal{BQP}$ (=Bounded Error Quantum Probabilistic Polynomial Time) liegt, da Peter Shor 1994 einen Polynomialzeitalgorithmus für Quantencomputer gefunden hat, der mit hoher Wahrscheinlichkeit die Faktoren einer natürlichen Zahl n findet. Sollte es also eines Tages möglich sein, Quantencomputer zu konstruieren, wäre dies das Ende vieler heutzutage gängigen Kryptosysteme.

In diesem Abschnitt wollen wir ein paar Faktorisierungsalgorithmen vorstellen und ihre Eigenschaften untersuchen.

Fermat-Faktorisierung

Sei n eine natürliche Zahl. Die einfachste Möglichkeit, die Zahl n zu faktorisieren, ist natürlich die sogenannte *Probedivision*, bei der man einfach alle Zahlen von 2 bis $\lfloor\sqrt{n}\rfloor$ ausprobiert, bis eine von ihnen n teilt. Eine ähnliche Methode ist die Fermat-Faktorisierung, die auf der folgenden einfachen Beobachtung beruht:

Theorem 12.1. *Sei $n \in \mathbb{N}$ eine ungerade natürliche Zahl. Dann existiert eine Bijektion zwischen allen Faktorisierungen $n = a \cdot b$ mit natürlichen Zahlen $a \geq b$ und allen Darstellungen von n der Form $n = t^2 - s^2$, wobei $s, t \geq 0$ ganze Zahlen sind.*

Beweis. Ist $n = t^2 - s^2 = (t + s)(t - s)$, so erhält man die Faktorisierung durch $a = t + s$ und $b = t - s$.

Löst man die beiden Gleichungen nach t und s auf, so ergibt sich $t = (a + b)/2$ und $s = (a - b)/2$. □

Wenn mir nun annehmen, daß die beiden Faktoren a und b von n nahe beieinander liegen, so können wir n mit Theorem 12.1 leicht faktorisieren, indem wir für $t = \lfloor\sqrt{n}\rfloor + 1, \lfloor\sqrt{n}\rfloor + 2, \ldots$ die Differenz $t^2 - n$ berechnen und testen, ob sie ein Quadrat s^2 ist. Ist dies der Fall, so folgt, daß n von der Form $n = t^2 - s^2$ ist und sich somit als $n = (t + s)(t - s)$ faktorisieren läßt. Das ist schon das ganze Geheimnis hinter der Fermat-Faktorisierung.

Faktorbasen

Faktorbasen stellen eine Verallgemeinerung und Weiterführung der Grundidee aus der Fermat-Faktorisierung dar. Nehmen wir nämlich an, wir kennten natürliche Zahlen s und t, so daß

$$\begin{aligned} s^2 &\equiv t^2 \pmod{n}, \\ \text{aber } s &\not\equiv \pm t \pmod{n} \end{aligned} \tag{12.1}$$

ist. Dann wäre doch n ein Teiler von $t^2 - s^2 = (t + s)(t - s)$, aber n teilte weder $t + s$ noch $t - s$. Also wäre dann $a := (t + s, n)$ ein echter Teiler von n, und wir könnten n faktorisieren als $n = a \cdot \frac{n}{a}$.

Beispiel 12.1. Sei $n = 4633$, und $s = 118, t = 5$. Dann gilt $118^2 \equiv 5^2 \pmod{n}$, aber $118 \not\equiv \pm 5 \pmod{n}$, also ist $(118 + 5, n) = 41$ ein echter Teiler von n, und wir erhalten $n = 41 \cdot 113$. □

Wie können wir uns nun zu gegebenem n zwei Zahlen s und t mit der Eigenschaft (12.1) beschaffen? Die Idee besteht darin, eine Menge von b_i's zu finden, so daß wir die gesuchte Zahlen aus den Repräsentanten von $(b_1 \cdot \cdots \cdot b_k)^2 \pmod{n}$ konstruieren können. Dies führt uns zum Begriff der Faktorbasis:

Definition 12.1. *Sei $n \in \mathbb{N}$ ungerade.*

- *Der* kleinste absolute Rest *einer Zahl a modulo n sei diejenige ganze Zahl $a' \in [-n/2, n/2] \cap \mathbb{Z}$, für die $a \equiv a' \pmod{n}$ ist. Wir schreiben $a' = a \pmod{n}$.*
- *Eine* Faktorbasis *ist eine Menge $B = \{p_1, \ldots, p_h\}$ mit $p_i \in \mathbb{P} \cup \{-1\}$, wobei $\mathbb{P}$ die Menge aller Primzahlen bezeichne.*

- *Ist B eine Faktorbasis, so nennen wir eine ganze Zahl b B-glatt, falls sich der kleinste absolute Rest $b^2 \pmod n$ mit Elementen aus B faktorisieren läßt.*

Beispiel 12.2. Sei $n = 4633$, und wähle $B = \{-1, 2, 3\}$ als Faktorbasis. Dann sind die Zahlen 67, 68 und 69 B-glatt, denn es gilt

$$\begin{aligned} 67^2 &= -144 = -1 \cdot 2^4 \cdot 3^2 \pmod n \\ 68^2 &= -9 = -1 \cdot 3^2 \pmod n \\ 69^2 &= 128 = 2^7 \pmod n \end{aligned}$$

□

Unser Ziel besteht darin, aus verschiedenen B-glatten Zahlen b_i zwei Zahlen s und t zu konstruieren, die Eigenschaft (12.1) erfüllen. Dabei wird s das Produkt der b_i sein, und t werden wir mit Hilfe der absoluten Reste der b_i^2 modulo n konstruieren. Dazu brauchen wir noch folgende Schreibweise:

Definition 12.2. *Sei $B = \{p_1, \ldots, p_h\}$ eine Faktorbasis und b ein B-glattes Element mit*

$$b^2 \pmod n = \prod_{j=1}^{h} p_j^{\alpha_j}.$$

Dann definieren wir den Vektor $\varepsilon_b \in \mathrm{GF}(2)^h$ durch

$$\varepsilon_b(j) = \begin{cases} 1, \text{ falls } \alpha_j \text{ ungerade,} \\ 0, \text{ falls } \alpha_j \text{ gerade.} \end{cases}$$

Offenbar ist für eine B-glatte Zahl $b^2 \pmod n$ genau dann ein Quadrat, wenn ε_b der Nullvektor ist.

Beispiel 12.3. Sei wieder $n = 4633$, und wähle $B = \{-1, 2, 3\}$ als Faktorbasis. Wie wir in Beispiel 12.2 gesehen haben, sind die Zahlen 67, 68 und 69 B-glatt. Es gilt

$$\begin{aligned} \varepsilon_{67} &= (1, 0, 0)\,, \\ \varepsilon_{68} &= (1, 0, 0) \text{ und} \\ \varepsilon_{69} &= (0, 1, 0)\,. \end{aligned}$$

□

Sind $b_1, \ldots, b_k$ B-glatte Zahlen mit

$$b_i^2 \pmod n = \prod_{j=1}^{h} p_j^{\alpha_{ij}},$$

und gilt für die zugehörigen Vektoren ε_{b_i}, daß

$$\sum_{i=0}^{k} \varepsilon_{b_i} = 0,$$

so ist

$$(b_1 \cdot \dots \cdot b_k)^2 \pmod n \equiv \prod_{j=1}^{h} p_j^{\sum_{i=1}^{k} \alpha_{ij}} \pmod n,$$

und alle Exponenten $\sum_{i=1}^{k} \alpha_{ij}$ sind gerade. Nun können wir uns also wie angekündigt unser s und t definieren durch

$$s := b_1 \cdot \dots \cdot b_k \text{ und}$$
$$t := \prod_{j=1}^{h} p_j^{\frac{1}{2}\sum_{i=1}^{k} \alpha_{ij}}$$

Nach Konstruktion ist dann $s^2 \equiv t^2 \pmod n$. Um n faktorisieren zu können, brauchen wir noch, daß $s \not\equiv \pm t \pmod n$ ist. Dabei hilft die folgende Übung:

Übung 12.1. (a) Ist $m = p^\alpha$ oder $m = 2p^\alpha$, wobei p eine ungerade Primzahl ist, so hat $x^2 \equiv 1 \pmod m$ die einzigen Lösungen $x \equiv \pm 1 \pmod 1$.
(b) Hat m nicht diese Form, so gibt es mehr Lösungen.
(c) Ist m eine ungerade Zahl mit r Primfaktoren, so hat $x^2 \equiv 1 \pmod m$ genau 2^r viele Lösungen.

Betrachten wir nun die Kongruenz $s^2 \equiv x^2 \pmod n$, so sind zwei dieser Lösungen $\pm s \pmod n$, und nach der vorhergehenden Übung gibt es noch $2^r - 2$ andere Lösungen. Gehen wir nun davon aus, daß die Faktorbasis so gewählt ist, daß die Werte von t zufällig verteilt sind, so haben wir also mit Wahrscheinlichkeit $1 - 2/2^r$ ein Paar (s, t) gefunden, das die Bedingung (12.1) erfüllt, und können n faktorisieren.

Wie findet man nun eine günstige Faktorbasis? Typischerweise gibt es zwei Möglichkeiten:

1. Wähle ein x in Abhängigkeit von n und setze $B := \{-1\} \cup \{p \leq x \,|\, p \text{ prim}\}$. Praktisch kann man B zum Beispiel mit dem Sieb der Eratosthenes berechnen.
2. Wähle im Prozeß die b_i's mit $b_i^2 \pmod n$ "klein", und füge die Primzahlen aus der Primfaktorzerlegung von $b_i^2 \pmod n$ zu B hinzu, wobei B nicht zu groß werden sollte.

Beispiel 12.4. Sei wieder $n = 4633$ und $B = \{-1, 2, 3\}$. Setzte $b_1 = 67$ und $b_2 = 68$. Wie wir schon gesehen haben, ist $\varepsilon_{67} = (1, 0, 0)$ und $\varepsilon_{68} = (1, 0, 0)$, also ist $\varepsilon_{67} + \varepsilon_{68} = 0$.

Also setzen wir $s := 67 \cdot 68 \equiv -77 \pmod n$, und da $67^2 \equiv -2^4 \cdot 3^2 \pmod n$ und $68^2 \equiv -3^2 \pmod n$, setzen wir $t := 2^2 \cdot 3^2 \equiv 36 \pmod n$. Offenbar ist $s \not\equiv \pm t \pmod n$, also ist $(-77 + 36, 4633) = 41$ ein echter Teiler von n. □

Fassen wir zum Abschluß diese Abschnittes noch einmal die Schritte des Faktorbasisalgorithmus zusammen: Gegeben sei eine ungerade natürliche Zahl n. Im ersten Schritt wählen wir ein x mit der Eigenschaft, daß die Anzahl der Dezimalstellen von x ein fest gegebener Bruchteil der Anzahl der Dezimalstellen von n ist. Nun setzen wir $B := \{-1\} \cup \{p \leq x | p \text{ Primzahl}\}$, und definieren $h := |B|$. Wir wählen dann zufällig $k \geq h+1$ viele paarweise verschiedene b_i's, so daß alle b_i's B-glatt sind, und finden ein $I \subseteq \{1, \ldots, k\}$, so daß die Summe der durch I indizierten ε-Vektoren verschwindet. Solch ein I existiert, da die ε-Vektoren in einem Vektorraum der Dimension h leben, und läßt sich zum Beispiel mit Hilfe des Gauß-Algorithmus effizient berechnen. Als nächstes berechnen wir s und t wie oben, und testen, ob $s \not\equiv \pm t \pmod{n}$ ist. Falls nicht, so suchen wir nach anderen linearen Abhängigkeiten. Führt dies immer noch nicht zum Erfolg, so generieren wir weitere b_i's.

Durch eine mehr ins Detail gehende Analyse kann man sehen, daß die Laufzeit dieses Verfahrens im Erwartungswert die Laufzeit

$$O\left(\exp\left(c\sqrt{\log n \log\log n}\right)\right)$$

besitzt, wobei c eine kleine Konstante darstellt. Der beste derzeit bekannte Faktorisierungsalgorithmus ist das Zahlenkörpersieb von John Pollard aus dem Jahr 1988, das eine erwartete Laufzeit von

$$O\left(\exp\left(c\,(\log n)^{\frac{1}{3}}\,(\log\log n)^{\frac{2}{2}}\right)\right)$$

erreicht. Darüber hinaus gibt es noch weitere interessante Verfahren, wie zum Beispiel die elliptische Kurvenmethode, die Hendrik Lenstra im Jahre 1987 vorgestellt hat und welche in Zeit

$$O\left(\exp\left(c\sqrt{\log p \log\log p}\right)\right)$$

läuft, wobei p die zweitgrößte Primzahl in der Zerlegung von n bezeichne. Der erste dieser fortgeschrittenen Faktorisierungsalgorithmen ist das quadratische Sieb, das Carl Pomerance 1981 veröffentlicht hat. Für Zahlen mit weniger als 112 Stellen läuft es gewöhnlich besser als das Zahlenkörpersieb.

Die Kettenbruchmethode

Der Faktorbasisalgorithmus aus dem letzten Abschnitt läßt noch einiges zu wünschen übrig. Besonders unbefriedigend ist es, daß die b_i's alle zufällig gewählt werden. Im folgenden wollen wir eine Methode kennenlernen, um *deterministisch* b_i's mit der Eigenschaft zu bestimmen, daß die kleinsten absoluten Reste der b_i^2 modulo n kleiner als $2\sqrt{n}$ sind. Dazu benötigen wir die Kettenbruchdarstellung für reelle Zahlen, die wir uns als nächstes genauer ansehen wollen.

Kettenbrüche

Sei $x > 0 \in \mathbb{R}$ eine positive reelle Zahl. Bekanntlich läßt sich x durch eine Folge von rationalen Zahlen beliebig gut approximieren. Wir möchten nun eine solche approximierende Folge von rationalen Zahlen finden, die besonders gute Eigenschaften hat. Dies können wir mit Hilfe der von Legendre erfundenen *Kettenbrüche* erreichen. Dabei wird x dargestellt durch

$$x = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \cdots}}},$$

mit gewissen $a_i \in \mathbb{Z}$. Zur besseren Lesbarkeit schreiben wir auch

$$x = a_0 + \frac{1}{a_1+} \cdot \frac{1}{a_2+} \cdot \frac{1}{a_3+} \cdot \ldots \cdot \frac{1}{a_i+} \cdot \ldots .$$

Nun stellt sich natürlich sofort die Frage, ob zu jedem x eine Darstellung als Kettenbruch existiert. Sei dazu $x \in \mathbb{R}$ gegeben. Die a_i lassen sich leicht induktiv bestimmen. Es ist klar, daß

$$a_0 = \lfloor x \rfloor$$

sein muß. Zusätzlich definieren wir noch den Restterm

$$x_0 = a_0 - x_0.$$

Haben wir nun für ein $i > 0$ schon die ersten $i - 1$ a_j und x_j bestimmt, so daß

$$x = a_0 + \frac{1}{a_1+} \cdot \frac{1}{a_2+} \cdot \frac{1}{a_3+} \cdot \ldots \cdot \frac{1}{a_{i-1} + x_{i-1}}$$

ist, so ist es leicht, im nächsten Schritt a_i und x_i zu berechnen. Wir wollen nämlich x_{i-1} in der Form $1/\left(a_i + x_i\right)$ schreiben, also muß $1/x_{i-1} = a_i + x_i$ sein, das heißt,

$$a_i = \left\lfloor \frac{1}{x_{i-1}} \right\rfloor \text{ und } x_i = \frac{1}{x_{i-1}} - a_i.$$

Offenbar terminiert dieser Prozeß, sobald ein $x_i = 0$ wird, und das ist nur dann der Fall, wenn x rational ist. Die Umkehrung ist auch richtig: Sei dazu x rational. Dann sind auch alle x_i rational, da in diesem Fall alle Rechenoperationen innerhalb des Körpers $\mathbb{Q}$ stattfinden. Wir schreiben x_i als gekürzten Bruch r_i/s_i. Da die $x_i \in [0, 1)$ sind, gilt $r_i < s_i$. Nach Konstruktion gilt für x_{i+i}:

$$x_{i+1} = \frac{s_i}{r_i} - a_i = \frac{s_i - r_i a_i}{r_i} = \frac{r_{i+1}}{s_{i+1}},$$

also ist $1 \leq s_{i+1} \leq r_i < s_i$. Die s_i bilden also eine streng monoton fallende Folge von natürlichen Zahlen, und daher muß ein j existieren, so daß $s_j = 1$ ist. Dann ist $x_j = 0$, denn es ist $x_j = r_j/s_j = r_j$ eine ganze Zahl in $[0, 1)$.

Nun interessieren wir uns für den Fall, daß x irrational und positiv ist. Unser Ziel ist es, zu zeigen, daß in diesem Fall die Kettenbruchentwicklung von x (sogar ziemlich geschwind) gegen x konvergiert. Dazu betrachten wir zunächst die *Konvergenten* der Kettenbruchentwicklung von x.

Definition 12.3. *Sei $x \in \mathbb{R}^+$ irrational mit Kettenbruchentwicklung*

$$x = a_0 + \frac{1}{a_1+} \cdot \frac{1}{a_2+} \cdot \frac{1}{a_3+} \cdot \ldots \cdot \frac{1}{a_i+} \cdot \ldots .$$

Dann heißt der Bruch

$$\frac{b_i}{c_i} = a_0 + \frac{1}{a_1+} \cdot \frac{1}{a_2+} \cdot \frac{1}{a_3+} \cdot \ldots \cdot \frac{1}{a_i}$$

die i-te Konvergente *von x, wobei b_i und c_i teilerfremde ganze Zahlen sind.*

Einige einfache Eigenschaften der Konvergenten sind in dem folgenden Lemma zusammengestellt:

Lemma 12.1. *Sei $x \in \mathbb{R}^+$ irrational und (a_i) die Folge der Koeffizienten der Kettenbruchentwicklung von x. Wir definieren rekursiv zwei Folgen (b_i) und c_i durch $b_0 := a_0$, $b_1 := a_0a_1 + 1$ und $b_i := a_ib_{i-1} + b_{i-2}$ sowie $c_0 := 1$, $c_1 := a_1$ und $c_i := a_ic_{i-1} + c_{i-2}$.*

Dann gelten:

(a) Für $i \geq 1$ ist

$$b_ic_{i-1} - b_{i-1}c_i = (-1)^{i-1}. \tag{12.2}$$

(b) Die Zahlen b_i und c_i sind teilerfremd.

(c) Der Bruch b_i/c_i ist die i-te Konvergente von x.

Beweis. (a) Wir führen wieder eine Induktion nach i. Für $i = 1$ ist

$$b_1c_0 - b_0c_1 = (a_0a_1 + 1) \cdot 1 - a_0a_1 = 1 = (-1)^{1-1}.$$

Auch der Induktionsschritt von $i-1$ nach i ist eine einfache Rechnung:

$$\begin{aligned} b_ic_{i-1} - b_{i-1}c_i &= (a_ib_{i-1} + b_{i-2}) \cdot c_{i-1} - b_{i-1} \cdot (a_ic_{i-1} + c_{i-2}) \\ &= -\left(b_{i-1}c_{i-2} + b_{i-2}c_{i-1}\right) \\ &= -(-1)^{i-2} = (-1)^{i-1}. \end{aligned}$$

(b) Dies ist eine unmittelbare Folgerung aus der soeben bewiesenen Aussage, denn ist $g = (b_i, c_i)$, so gilt offenbar $g_i | b_ic_{i-1} - b_{i-1}c_i = \pm 1$, also ist $g = 1$, und b_i und c_i sind teilerfremd.

(c) Wir führen den Beweis durch Induktion nach i. Die Aussage läßt sich leicht für $i = 0, 1$ verifizieren. Für den Induktionsschritt beobachten wir, daß die $i+1$-te Konvergente konstruktionsgemäß aus der i-ten Konvergente hervorgeht, indem a_i durch $a_i + 1/a_{i+1}$ ersetzt wird. Nach Induktionsannahme ist die i-te Konvergente von der Form $(a_i b_{i-1} + b_{i-2})/(a_i c_{i-1} + c_{i-2})$, wobei $b_{i-1}, b_{i-2}, c_{i-1}, c_{i-2}$ nicht von i abhängen. Das heißt,

$$
\begin{aligned}
(i+1)\text{-te Konvergente} &= \frac{\left(a_i + \frac{1}{a_{i+1}}\right) \cdot b_{i-1} + b_{i-2}}{\left(a_i + \frac{1}{a_{i+1}}\right) \cdot c_{i-1} + c_{i-2}} \\
&= \frac{a_{i+1}\left(a_i b_{i-1} + b_{i-2}\right) + b_{i-1}}{a_{i+1}\left(a_i c_{i-1} + c_{i-2}\right) + c_{i-1}} \\
&= \frac{a_{i+1} b_i + b_{i-1}}{a_{i+1} c_i + c_{i-1}} \\
&= \frac{b_{i+1}}{c_{i+1}},
\end{aligned}
$$

und da wir bereits gesehen haben, daß b_{i+1} und c_{i+1} teilerfremd sind, folgt die Behauptung.

□

Nun verfügen wir über die nötigen Werkzeuge, um die angekündigte Konvergenzaussage zu beweisen.

Lemma 12.2. *Sei $x \in \mathbb{R}^+$ irrational und b_i/c_i die i-te Konvergente von x. Dann ist*

$$\lim_{i \to \infty} \frac{b_i}{c_i} = x.$$

Beweis. Der Beweis zerfällt in zwei Teile.

b_i/c_i konvergiert: Dividiert man die Beziehung (12.2) aus Lemma 12.1 durch $c_i c_{i+1}$, so erhält man

$$\frac{b_i}{c_i} - \frac{b_{i-1}}{c_{i-1}} = \frac{(-1)^{i-1}}{c_i c_{i-1}}. \tag{12.3}$$

Da für $i \geq 1$ der Wert von $a_i \geq 1$ und $c_i = a_i c_{i-1} + c_{i-2}$ ist, folgt, daß die Folge der c_i streng monoton wächst. Somit schließen wir aus dem Leibniz-Kriterium, daß b_i/c_i konvergiert.

Der Grenzwert ist x: Zunächst beobachten wir, daß wir x aus der $(i+1)$-ten Konvergenten

$$\frac{b_{i+1}}{c_{i+1}} = a_0 + \frac{1}{a_1+} \cdot \frac{1}{a_2+} \cdot \frac{1}{a_3+} \cdot \ldots \cdot \frac{1}{a_i} \cdot \frac{1}{a_{i+1}}$$

erhalten, indem wir a_{i+1} durch $1/x_i$ ersetzen, denn nach Konstruktion ist

$$x = a_0 + \frac{1}{a_1+} \cdot \frac{1}{a_2+} \cdot \frac{1}{a_3+} \cdot \ldots \cdot \frac{1}{a_i + x_i}.$$

Folglich ist

$$x = \frac{\frac{1}{x_i} b_i + b_{i-1}}{\frac{1}{x_i} c_i + c_{i-1}} = \frac{b_i + x_i b_{i-1}}{c_i + x_i c_{i-1}}. \tag{12.4}$$

Nun ist $x_i \in (0,1)$, und daher folgt aus (12.4), daß x zwischen b_i/c_i und b_{i+1}/c_{i+1} liegt, wie man leicht nachrechnet. Wer nicht rechnen möchte, kann sich von der Richtigkeit der Aussage mit einem Bild überzeugen. Betrachtet man nämlich die Vektoren $u = (c_i, b_i)$ und $v = (c_{i-1}, d_{i-1})$, so liegt der Vektor $u + x_i \cdot v$ im Inneren des von u und v aufgespannten Parallelogrammes, d.h., die Steigung von $u + x_i \cdot v$ liegt zwischen den Steigungen von u und v, was genau die Aussage ist, die wir zeigen wollten (siehe Abbildung 12.1).

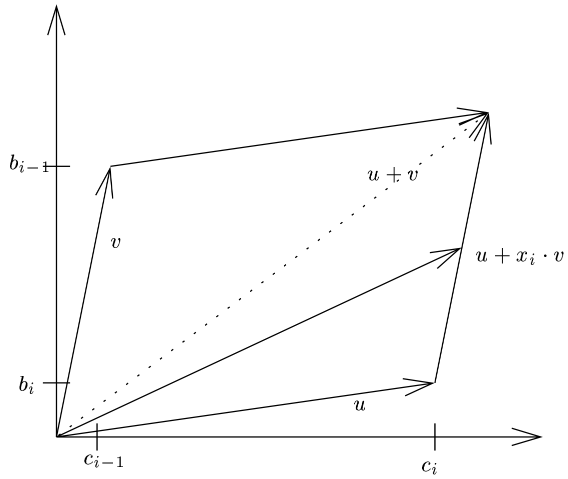


Abb. 12.1. Die Steigung von $u + x_i \cdot v$ liegt zwischen den Steigungen von u und v.

Es folgt daher, daß die Glieder der Folge (b_i/c_i) um x herum oszillieren, also konvergiert die Folge gegen x.

□

Für die oben angekündigte Anwendung im Faktorisierungsalgorithmus ist das folgende Ergebnis wichtig, das uns später helfen wird, b_i's mit kleinen absoluten Resten zu bestimmen.

Theorem 12.2. *Sei $x > 1$ eine irrationale reelle Zahl mit Konvergenten b_i/c_i. Dann gilt für alle i:*

$$\left| b_i^2 - x^2 c_i^2 \right| < 2x.$$

Beweis. Es gilt:

$$\begin{aligned}\left|b_i^2 - x^2c_i^2\right| &= c_i^2 \cdot \left|\frac{b_i^2}{c_i^2} - x^2\right| \\ &= c_i^2 \cdot \left|x - \frac{b_i}{c_i}\right| \cdot \left|x + \frac{b_i}{c_i}\right|\end{aligned}$$

Wegen Gleichung (12.2) ist nun $|x - b_i/c_i| \leq |b_{i+1}/c_{i+1} - b_i/c_i|$ und somit auch $b_i/c_i \leq x + |b_{i+1}/c_{i+1} - b_i/c_i|$. Daher können wir folgendermaßen abschätzen:

$$\begin{aligned}\left|b_i^2 - x^2c_i^2\right| &\leq c_i^2 \cdot \left|\frac{b_{i+1}}{c_{i+1}} - \frac{b_i}{c_i}\right| \cdot \left(2x + \left|\frac{b_{i+1}}{c_{i+1}} - \frac{b_i}{c_i}\right|\right) \\ &\leq c_i^2 \cdot \frac{1}{c_{i+1}c_i}\left(2x + \frac{1}{c_{i+1}c_i}\right) = \frac{c_i}{c_{i+1}} \cdot 2x + \frac{1}{c_{i+1}^2},\end{aligned}$$

wobei die letzte Ungleichung aus (12.3) folgt. Es folgt also:

$$\begin{aligned}\left|b_i^2 - x^2c_i^2\right| - 2x &< -2x + \frac{c_i}{c_{i+1}} \cdot 2x + \frac{1}{c_{i+1}^2} \\ &= 2x \cdot \left(-1 + \frac{c_i}{c_{i+1}} + \frac{1}{2xc_{i+1}^2}\right).\end{aligned}$$

Weil nun $x > 1$ und $c_{i+1} \geq 1$ ist, folgt weiter:

$$\begin{aligned}\left|b_i^2 - x^2c_i^2\right| - 2x &< 2x \cdot \left(-1 + \frac{c_i}{c_{i+1}} + \frac{1}{c_{i+1}}\right) \\ &= 2x \cdot \left(-1 + \frac{c_i + 1}{c_{i+1}}\right) \\ &\leq 2x \cdot \left(-1 + \frac{c_{i+1}}{c_{i+1}}\right) = 0,\end{aligned}$$

wobei sich die letzte Abschätzung daraus ergibt, daß die c_i streng monoton wachsen. Damit ist die Ungleichung bewiesen. □

Faktorisierung mit Hilfe der Kettenbruchentwicklung

Nun wollen wir die Ergebnisse aus dem letzten Abschnitt verwenden, um den Faktorbasisalgorithmus aus dem letzten Abschnitt zu verbessern. Sei dazu eine ungerade natürliche Zahl $n \geq 16$ gegeben. Wir betrachten die Konvergenten b_i/c_i von $x = \sqrt{n}$, und wählen die b_i als die Zahlen, die für den Faktorbasisalgorithmus verwendet werden. Das ist eine kluge Wahl, denn es gilt $b_i^2 \equiv b_i^2 - nc_i^2 \pmod{n}$, und nach Theorem 12.2 ist der kleinste absolute Rest von b_i^2 gegeben durch $|b_i^2 - nc_i^2| > 2\sqrt{n} \leq n/2$. Die Faktorbasis wird gebildet, indem die b_i faktorisiert werden, was hoffentlich einigermaßen schnell geht, da die b_i^2 relativ klein sind (aber nicht klein genug, um eine polynomielle Laufzeit zu erreichen).

Abschließend betrachten wir ein kleines Beispiel, um die Effektivität der Kettenbruchmethode zu verdeutlichen.

Beispiel 12.5. Es sei $n = 9073$. Dann ist $x := \sqrt{n} \approx 95{,}252$ und $2x \approx 190$. Nach der Kettenbruchmethode errechnen sich nun die b_i folgendermaßen: Wir setzen $b_0 := a_0 := \lfloor\sqrt{n}\rfloor$ und $x_0 := \sqrt{n} - a_0$ sowie $b_1 = a_0 a_1 + 1$. Dann definieren wir rekursiv $a_i := \lfloor 1/x_i \rfloor$ und $x_i := 1/x_{i-1} - a_i$ sowie $b_i := a_i b_{i-1} + b_{i-2} \pmod{n}$. Es ergeben sich die folgenden Werte:

i	a_i	b_i	$b_i^2 \pmod{n}$
0	95	95	$-48 = -3 \cdot 2^4$
1	3	286	139
2	1	381	$-7 = -7$
3	26	1119	87
4	2	2619	$-27 = -3^3$

Wir betrachten nur diejenigen b_i, deren kleinste absolute Reste b_i^2 sehr klein sind (also b_0, b_2 und b_4). somit erhalten wir die Faktorbasis $B = \{-1, 2, 3, 7\}$ und die ε-Vektoren $\varepsilon_0 = (1,0,1,0)$, $\varepsilon_2 = (1,0,0,1)$ und $\varepsilon_4 = (1,0,1,0)$. Offenbar ist $\varepsilon_0 + \varepsilon_4 = 0$, also setzen wir $s := 95 \cdot 2619 \equiv 3834 \pmod{n}$. Und $t := 3^2 \cdot 2^2 = 36$. Dann ist $s^2 \equiv t^2 \pmod{n}$ und $s \not\equiv \pm t \pmod{n}$, also ist $(3834 + 36, 9073) = 43$ ein echter Teiler von $n = 43 \cdot 211$. □

13

Kryptographische Hashfunktionen

Sei Σ ein Alphabet.

Definition 13.1. • *Eine* Hashfunktion *ist eine Funktion $h : \Sigma^* \to \Sigma^n$ für ein festes n.*
- *Eine* Kompressionsfunktion *ist eine Funktion $h : \Sigma^m \to \Sigma^n$ für feste m, n mit $m < n$.*

Ein einfaches Beispiel ist die Paritätsfunktion: $\Sigma = \{0, 1\}$, $n = 1$ mit $\sigma_1 \dots \sigma_m \mapsto \sigma_1 \oplus \cdots \oplus \sigma_m$.

Bemerkung 13.1. Hash- und Kompressionsfunktionen sind *nie* injektiv.

Definition 13.2. *Eine Hash-/Kompressionsfunktion heißt* kryptographisch*, falls gilt*

- *zu gegebenem x ist $h(x)$ effizient berechenbar, und*
- *für jede probabilistische polynomielle Turing-Maschine ist die Wahrscheinlichkeit ein Urbild zu $s \in \Sigma^n$ zu finden verschwindend gering.*

Spric: h^{-1} ist für jede Instanz schwer zu berechnen. Siehe Teil 3' der Definition der Einwegfunktion.

Definition 13.3. *h heißt* schwach kollisionsresistent*, falls zu vorgegebenem $x \in D$ (wobei $D = \Sigma^*$ oder $D = \Sigma^m$) jede probabilistische ploynomielle Turing-Maschine nur mit verschwindend geringer Wahrscheinlichkeit eine Kollision $x' \neq x$ mit $h(x') = h(x)$ findet.*

h heißt stark kollisionsresistent*, falls jede probabilistische ploynomielle Turing-Maschine nur mit verschwindend geringer Wahrscheinlichkeit eine Kollision $x' \neq x$ mit $h(x') = h(x)$ findet.*

Bemerkung 13.2. Kollisionsresistente, effizient berechenbare Hashfunktionen sind kryptographisch.

Beweis (Skizze). Angenommen nicht. Wähle $x \in D$ zufällig. Berechne $y = h(x)$. Berechne nach Annahme ein Urbild x'. Dann ist mit nicht verschwindender Wahrscheinlichkeit $x \neq x'$. □

Geburtstagsattacke auf Hashfunktionen

Sei $|\Sigma| = \alpha$ und $h : \Sigma^* \to \Sigma^n$ eine Hashfunktion. Nehmen wir an, daß wir Elemente aus Σ^* zufällig wählen können und daß die Hashwerte dabei gleichverteilt sind. Sei $q = \frac{1}{2}$, so benötigen wir

$$k \geq \frac{1 + \sqrt{1 + 8\alpha^n \cdot ln\frac{1}{q}}}{2} \tag{13.1}$$

viele Elemente $x \in \Sigma^*$, so daß mit Wahrscheinlichkeit $\geq q = \frac{1}{2}$ eine Kollision auftritt. Die Größenordnung ist also $\alpha^{\frac{n}{2}}$.

Im Fall $\Sigma = 0, 1$ haben wir $2^{\frac{n}{2}}$. Für den digitalen Signaturstandard fordert man deshalb $n \geq 160$. Zum Vergleich: Der DES-Schlüsselraum hat eine Größe von $2^5 6$.

Kompressionsfunktionen aus Verschlüsselungsfunktionen

Seien $e_k : \{0,1\} \to \{0,1\}^n$ Verschlüsselungsfunktionen mit Schlüsseln $k \in \{0,1\}^n$. Konstruiere beispielsweise $h : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ (also $m = 2n$) via $h(k,x) = e_k(x)$ (oder $h(k,x) = e_k(x) \oplus x$ oder $h(k,x) = e_k(x) \oplus k$ oder $h(k,x) = e_k(x \oplus k)$ oder ...).

Hashfunktionen aus Kompressionsfunktionen

Sei $g : \Sigma^m \to \Sigma^n$ eine Kompressionsfunktion mit $r = m - n \geq 1$ und $\Sigma = \{0,1\}$. Wir wollen eine Funktion $h : \Sigma^* \to \Sigma^n$ konstruieren. Dafür sei $x \in \Sigma^*$ beliebig und y die binäre Darstellung der Länge von x. Definiere $\bar{y}$ als das $0/1$-Wort welches aus y hervorgeht, indem von hinten an jeder r-ten Stelle eine 1 eingefügt wird und vorne das Wort $10\ldots0$, so daß $\bar{y}$ als Länge Vielfaches von r hat. Dann sei

$$\bar{x} = 0\ldots0x\underbrace{0\ldots0}_{\text{r viele}}\bar{y},$$

wobei am Anfang so viele Nullen angefügt werden, daß $\bar{x}$ eine Länge hat, die ein Vielfaches von r ist. Ziel der Konstruktion ist es, daß kein Wort ein Endstück eines anderen ist.

Beispiel 13.1. $r = 3$, $x = 1010011010$, Länge 10, $y = 1010$, $\bar{y} = 110110$ (hat bereits Länge $k \cdot r$) Dann ist

$$\bar{x} = 00\underbrace{1010011010}_{x}\underbrace{000}_{r=3}\underbrace{110110}_{\bar{y}}. \tag{13.2}$$

Zerlege nun $\bar{x} = x_1 \dots x_t$ in t Blöcke x_i der Länge r. Setze nun $H_0 = \underbrace{0 \dots 0}_{n}$, das 0-Wort der Länge n. Bestimme rekursiv

$$H_i = g(\underbrace{H_{i-1} \circ x_i}_{\text{Block der Laenge m}})$$

für $1 \leq i \leq t$. Setze schließlich $h(x) = H_t$.

Theorem 13.1. *Ist g kollisionsresistent, so auch h.*

Beweis. Sei also (x, x') eine Kollision von h. Das heißt $h(x) = h(x')$ und $x \neq x'$. Seien $\bar{x} = x_1 \dots x_s$ und $\bar{x'} = x'_1 \dots x'_s$ und $H_0, \dots, H_s$ bzw. $H'_0, \dots, H'_t$ die zugehörigen Folgen der H_i wobei oBdA $s \leq t$. Es gilt $H_s = H'_t$.

Fall 1: Es existiert $0 \leq i < s-1$. $H_{s-i} = H'_{t-i}$ und $H_{s-i-1} \not H'_{t-i-1}$. Dann ist $H_{s-i-1} \circ x_{s-i} \neq H'_{t-i-1} \circ x_{t-i}$ und $g(H_{s-i-1} \circ x_{s-i}) = H_{s-i} = H_{t-i} = g(H'_{t-i-1} \circ x_{t-i})$ eine Kollision von g.

Fall 2: $H_{s-i} = H'_{t-i}$ für alle $0 \leq i \leq s$. Dann existiert ein i mit $0 \leq i \leq s-1$ mit $x_{s-i} \neq x'_{t-i}$ (Beweis siehe unten). Dann ist also $H_{s-i-1} \circ H_{s-i} \neq H'_{t-i-1} \circ x'_{t-i}$. Wie oben ist dies eine Kollision.

Seien $\bar{x} = 0 \dots 0x0 \dots 0\bar{y}$ und $\bar{x'} = 0 \dots 0x'0 \dots 0\bar{y'}$

Fall a: Die Längen von $\bar{y}$ und $\bar{y'}$ sind verschieden. Dann steht ein Block mit Anfangs-1 über/unter dem 0-Block der Länge r.

Fall b: $\bar{y}$ und $\bar{y'}$ sind gleich lang, aber x und x' haben verschiedene Länge. Dann existiert ein paar von r-Blöcken in $\bar{y}$ und $\bar{y'}$, die übereinanderstehen und verschieden sind.

Fall c: $\bar{y}$ und $\bar{y'}$ sind gleich lang, und x und x' haben gleiche Länge. Dann klappt alles da $x \neq x'$. □

Beispiel einer arithmetischen Kompressionsfunkion

Sei $p = 2q + 1$, p, q Primzahlen.

Sei a Primitivwurzel $\mod p$, b zufällig aus $\{1, \dots, p-1\}$. Betrachte $h : \{0, \dots, q-1\}^2 \to \{1, \dots, p-1\}$ mit $(x_1, x_2) \mapsto a^{x_1} b^{x_2} \mod p$.

Sei (x, x') eine Kollision von h mit $x = (x_1, x_2)$ und $x' = (x_3, x_4)$. Es gilt $a^{x_1} b^{x_2} \equiv a^{x_3} b^{x_4} \mod p$, also $a^{x_1 - x_3} \equiv b^{x_4 - x_2} \mod p$. Sei $y = log_a b$, also $a^y = b$ dann gilt

$$a^{x_1 - x_3} \equiv a^{y(x_4 - x_2)} \mod p.$$

Es ist $x_1 - x_3 \equiv y(x_4 - x_2) \mod p-1$ $(p-1 = 2q)$. Sei $d = (\underbrace{x_4 - x_2}_{\text{Betrag} \leq q-1}, 2q)$. Es ist $d = 1$ oder $d = 2$.

Ist $d = 1$ so existiert eine eindeutige Lösung für y.

Ist $d = 2$ so existieren genau zwei Lösungen, da der diskrete Logarithmus $log_a b$ mit Sicherheit eine Lösung ist. Test welche der beiden Lösungen tatsächlich eine der diskrete Lögarithmus ist.

Wir haben gezeigt: Eine Kollision (x, x') von h führt zur Lösung des diskreten logarithmus Problems $log_a b$. Also: Wenn das DL-Problem schwer ist, so ist h kollisionsresistent mit hoher Wahrscheinlichkeit.

Authentifizierung

Grundsätzliche Anforderung

- Fester Teilnehmerkreis.
- Jeder Teilnehmer kann effizient Authentifizierungen zu Dokumenten erstellen.
- Jeder Teilnehmer soll effizient die Gültigkeit einer Authentifizierung prüfen können.
- Für nicht Teilnehmende ist es nicht mit vertretbarem Aufwand mäglich eine gültige Authentifizierung zu einem Dokument zu erstellen.

Hierfür betrachtet man sogennante message authentication codes, kurz MAC genannt.

Definition 13.4. *Eine parametrisierte Hashfunktion ist eine Familie* $\{h_k : k \in K\}$ *von Hashfunktionen,* K *eine Schlüsselmenge.*

Beispiel 13.2. Ist $g : \{0,1\}^* \to \{0,1\}^n$ eine Hashfunktion, so ist die Menge der $h : \{0,1\}^* \times \{0,1\}^n \to \{0,1\}^n$ definiert durch $h_k(x) = h(x,k) = g(x) \oplus k$ eine parametrisierte Hashfunktion.

Bemerkung 13.3. Konstruktion von MACs meist analog zu symmetrischen Verschlüsselungsverfahren.

Anwendung

Alice und Bob tauschen geheimen Schlüssel k aus. Alice schickt Nachricht x zusammen mit $y = h_k(x)$ an Bob. Bob prüft, ob $h_k(x) = y$ gilt.

Problem

Bob kann nicht nachweisen, daß die Nachricht von Alice kam (er selbst kann y ja berechnen und hätte x somit Alice unterschieben können). Eignet sich nicht zum Abschließen von Verträgen.

Signaturen

Grundsätzliche Anforderungen

- Jeder Teilnehmer kann die eigene Signatur zu einem Dokument effizient erstellen.
- Jeder Teilnehmer kann effizient verifizieren, ob ein gegebenes Wort Signatur eines anderen Teilnehmers zu einem gegeben Dokument ist.
- Es ist nicht mit vertretbarem Aufwand möglich die Signatur eines anderen Teilnehmers zu einem Dokument zu fälschen.

Grundsätzliches Public-Key-Setup

Sei $h : \{0,1\}^* \to \{0,\dots,n-1\}$ die kollisions resistente Hashfunktion, öffentlich bekannt. Seien E und D Public-Key Verschlüsselungs- bzw. Entschlüsselungsfunktionen mit öffentlichem Schlüssel (n,e) und geheimem Schlüssel d. Klartextraum und Schlüsseltextraum: $\{0,\dots,n-1\}$. Dann gilt $D(E(m,e),d) = m$. Es gelte $E(D(m,d),e) = m$.

Signaturschema

Alice will eine Nachricht $x \in \{0,1\}^*$ signieren.

$$x \mapsto h(x) = m \mapsto D(m,d) = s$$

Alice schickt (x,s) an Bob. Bob berechne $E(s,e)$ und vergleicht mit $h(x)$.

Problem

Man-in-the-middle-Attack: Bob muss sich auf die Authentizität des öffentlichen Schlüssels (n,e) verlassen können.

RSA-Signatur

RSA $n = q \cdot p$ $1 < e < \varphi(n) = (p-1)(q-1)$ d mit $d \cdot e \equiv 1 \mod \varphi(n)$. Öffentlicher Schlüssel $(n,.e)$. Geheimer Schlüssel d. Verschlüsseln $m \in \{0,\dots,n-1\} \mapsto m^e \mod n$ Entschlüsseln $c \mapsto c^d \mod n$. Es gilt $(m^d)^e = (m^e)^d \equiv m \mod n$.

Das *Rabin-Verfahren* erfüllt ebenfalls die Bedingung $E(D(m,d),e) = m$ und ist somit ebenfalls geeignet.

ElGamal-Signatur

Zunächst erinnern wir an die

ElGamal-Verschlüsselung

Öffentlicher Schlüssel: (p, g, A), p prim, g Primitivwurzel $\mod p$. $A = g^a$ für ein von Alice zufällig gewähltes $a \in \{1, \ldots, p-1\}$. Geheimer Schlüssel: a.

Bob will Nachricht an Alice schicken, wählt b und berechnet $B = g^b$ und $c = A^b \cdot m$. Er schickt (B, c) an Alice. Alice setzt $x = p - 1 - a$ und berechnet $m = B^x \cdot c$.

Signaturverfahren

Sei $h : \{0,1\}^* \to \{1, \ldots, p-2\}$ eine Hashfunktion, öffentlich bekannt, p prim. Alice öffentlicher Schlüssel (p, g, A) wie oben. Geheimer Schlüssel a.

Sei $x \in \{0,1\}^*$ eine Nachricht. Man berechnet $m = h(x)$. Wählt zufällig ein $k \in \{1, \ldots, p-2\}$ teilerfremd zu $p-1$. Die Signatur ist dann $r = g^k \mod p$, $s = k^{-1}(m - a \cdot r) \mod p - 1$. Alice verschickt $(x, \underbrace{(r, s)}_{\text{Signatur}})$.

Prüfen der Signatur

$A^r \cdot r^s \equiv g^{h(x)} \mod p$, denn

$$\begin{aligned} A^r \cdot r^s &\equiv g^{ar} \cdot g^{ks} \\ &\equiv g^{ar} \cdot g^{kk^{-1}(m-ar)} \\ &\equiv g^m \mod p \end{aligned}$$

Angenommen $k = log_g r$ lässt sich berechnen. $k \cdot s = m - ar$ und somit $a = r^{-1}(m - ks)$. Lösen des DL-Problems $log_g A$ oder $log_g r$ führt zur Bestimmung des geheimen Schlüssels a.

Wahl von p

Wichtig, daß das DL-Problem $\mod p$ schwer ist.

Wahl von k

Angenommen Texte x_1 und x_2 werden mit gleichem $r = g^k$ und s_1 bzw. s_2 signiert.

$$s_1 - s_2 = k^{-1}(h(x_1) - h(x_2)) \mod p - 1,$$

also

$$k = (h(x_1) - h(x_2))(s_1 - s_2)^{-1} \mod p - 1,$$

falls $(s_1 - s_2)$ invertierbar $\mod p - 1$ ist. Je nachdem, wie groß $\varphi(p-1)$ ist, kann dies sehr wahrscheinlich sein. Kenntnis von k führt offenbar zur Kenntnis von a.

Elektronischer Umschlag

Der sogenannte elektronische Umschlag ist das folgende Verfahrensprotokoll. Zunächst folgt Alice den Schritten:

1. Hashwert zum Dokument berechnen,
2. Hashwert mit Alices privatem Schlüssel verschlüsseln,
3. verschlüsselten Hashwert an Nachricht anfügen,
4. gesamte Nachricht mit Bobs öffentlichem Schlüssel verschlüsseln.

Dieses Dokument schickt sie an Bob und er folgt den Schritten:

1. Gesamte Nachricht mit Bobs privatem Schlüssel entschlüsseln,
2. den verschlüsselten Hashwert abtrennen,
3. den Hashwert mit Alices öffentlichem Schlüssel entschlüsseln,
4. den Hashwert der Nachricht berechnen,
5. die Hashwerte auf ihre Gleichheit überprüfen.

Digitaler Signatur Algorithmus

Der digitale Signatur Algorithmus ist als Standard 1991 eingeführt worden.

Schlüsselerzeugung

Alice wählt Primzahl q mit $2^{159} < q < 2^{160}$, d.h. $sizeq = 160$. Wählt Primzahl p mit $2^{511+64t} < q < 2^{512+64t}$ für ein $t \in \{0,..,8\}$, so daß $q|p-1$. Damit ist $sizep = 512 + 64t$. Alice wählt Primitivwurzel $x \mod p$. Setzt $g = x^{\frac{p-1}{q}}$. g erzeugt Untergruppe der Ordnung q in $\mathbb{Z}_p^x$. Alice wählt $a \in \{1, \ldots, q-1\}$ zufällig. Bestimmt $A = g^a$.

Öffentlicher Schlüssel

(p, q, g, A).

Geheimer Schlüssel

a.

Bemerkung

Das DL-Problem $log_g A$ findet in der von g erzeugten Untergruppe der Ordnung $\approx 2^{160}$ statt.

Signatur

$h : \{0,1\}^* \to \{1, \ldots, q-1\}$ sei öffentlich bekannte Hashfunktion. Alice wählt zufällig $k \in \{1, \ldots, q-1\}$,$(k, q) = 1$. Sei $x \in \{0,1\}^*$.

$$r = (g^k \mod p) \mod q$$
$$s = k^{-1}(h(x) + ar) \mod q$$

Die Signatur von x ist dann (r, s).

Verifikation

Bob testet die Identität

$$r \equiv ((g^{s^{-1}h(x)} A^{rs^{-1}}) \mod p) \mod q.$$

Denn

$$\begin{aligned} g^{s^{-1}h(x)} A^{rs^{-1}} &\equiv g^{s^{-1}h(x)} g^{ars^{-1}} \mod p \\ &\equiv g^{s^{-1}(h(x)+ar)} \mod p \\ &\equiv g^k \mod p. \end{aligned}$$

Der einzige bekannte Angriff ist, das DL-Problem in der von g erzeugten Untergruppe zu lösen.

14

Zero knowledge

Historisches Beispiel

Der venezianische Rechenmeister Niccolò Tartaglia (1499–1557) entdeckte 1535 eine Methode zum Bestimmen der Wurzeln einer kubischen Gleichung. Er verkündet die Entdeckung öffentlich. Ohne jedoch seine Methode zu verraten gelingt es ihm, die ganze Welt davon zu überzeugen, dass er tatsächlich eine Methode gefunden hat.

Und zwar auf sehr einfache Weise. Man gab ihm eine Gleichung 3. Grades der Form $p(x) = 0$. Mit Hilfe seiner Methode hat er die Nullstellen a, b, c bestimmt, und es ließ sich einfach $p(x) = (x - a)(x - b)(x - c)$ nachprüfen.

Geronimo Cardano (1501–1576) gelang es jedoch Tartaglia dazu zu überreden, das Geheimnis preiszugeben, unter der Bedingung, dass Cardano es geheim halten würde. Es kam wie es kommen musste: Cardano veröffentlichte die Methode 1545 in seiner "Ars Magna". Obgleich Cardano den Ursprung der Formel nicht verheimlicht ist sie heute unter *Cardanosche Formel* bekannt.

Das Zero knowledge–Problem

Alice kennt ein Geheimnis, zum Beispiel den Beweis eines Satzes. Sie will Bob von ihrer Kenntnis überzeugen, ohne jedoch das Geheimnis preiszugeben.

Typischerweise sind Protokolle zur Realisierung dieses Problems zufallsbasiert. Die grundsätzliche Idee wird durch das Schema in Abbildung 14.1 veranschaulicht. Alice betritt allein die Höhle und entscheidet sich zufällig für eine der beiden Türen und tritt durch sie hindurch. Anschliessend betritt Bob die Hühle, wählt zufällig eine der beiden Türen und bittet Alice aus dieser herauszutreten. Dies kann sie immer tun, wenn sie mittels ihres Geheimnisses die magische Tür öffnen kann, denn dies ist die einzige Möglichkeit die magische Tür zu öffnen. Kennt sie das Geheimnis nicht kann sie nur aus der richtigen Tür heraustreten, falls Bob sich zufällig für die Tür entschieden hat, hinter der sie steht. Dieses Experiment wiederholen Alice und Bob k mal. Tritt Alice jedes mal aus der richtigen Tür heraus, so ist die Wahrscheinlichkeit dafür, dass sie das Geheimnis nicht kennt nur $\frac{1}{2^k}$.

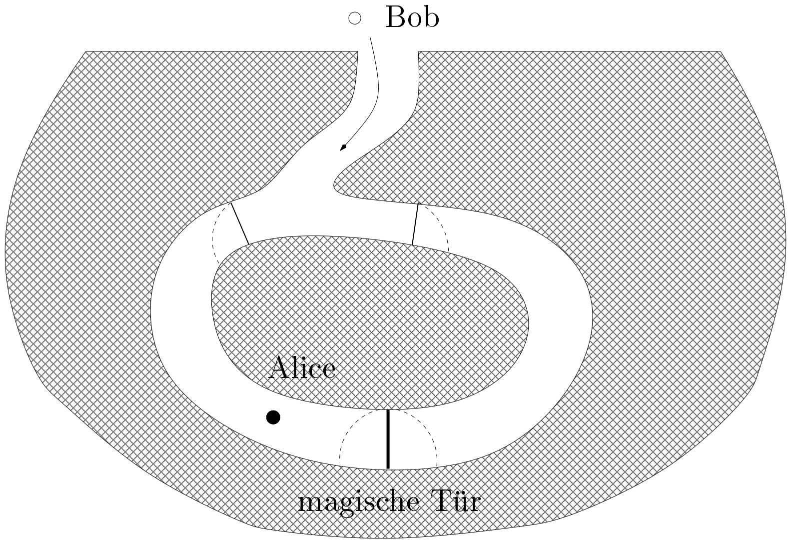


Abb. 14.1. Die Zero knowledge–Höhle

Zero–knowledge–Protokoll

Zunächst begnügen wir uns mit einer anschaulichen Definition. Wir kürzen Alice mit P für *prover* und Bob mit V für *verifier* ab.

Definition 14.1. *Ein Verfahrensprotokoll zwischen P und V heißt* Zero–knowledge–Protokoll, *falls folgende Bedingungen gelten.*

(A) Vollständigkeit: Falls P das Geheimnis besitzt, so akzeptiert V mit Wahrscheinlichkeit 1.
(B) Korrektheit: Falls P das Geheimnis nicht besitzt, so akzeptiert V nur mit verschwindend geringer Wahrscheinlichkeit.
(C) Zero–knowledge: V erfährt durch das Protokoll nichts von dem Geheimnis, was er nicht auch ohne Protokoll erfahren könnte.

Das Fiat–Shamir–Protokoll

Wir wollen nun ein erstes Beispiel eines Zero–knowledge–Protokolls angeben. Es seien $n = pq$ Produkt zweier Primzahlen, $s \in \{1, \dots, n-1\}$ mit $(s, n) = 1$

und $v = s^2 \mod n$. Alice kennt das Paar (v, s) und will Bob, der nur v kennt, davon überzeugen, daß sie eine Quadratwurzel von v kennt.

Das Protokoll

1. P wählt zufällig $r \in \{1, \ldots, n-1\}$ mit $(r, n) = 1$ und schickt $x = r^2 \mod n$ an V. (Alice betritt einen der Gänge.)
2. V wählt zufällig ein Bit $e \in \{0, 1\}$ und sendet dieses an P. (Bob ruft, durch welche Tür Alice treten soll.)
3. P schickt $y = rs^e \mod n$ an V zurück. (Alice öffnet für $e = 1$ die magische Tür.)
4. V prüft die Identität $y^2 = xv^e \mod n$. (Bob schaut, ob Alice aus der richtigen Tür tritt.)

Dieses Protokoll wird k mal durchgeführt. Gilt jedesmal $y^2 = xv^e \mod n$, so akzeptiert V, sonst verwirft er.

Analyse

(A) Die *Vollständigkeit* ist klar. (B) Für die *Korrektheit* stellen wir fest, dass P, wenn er keine Quadratwurzel s von v kennt, höchstens eine der beiden Fragen richtig beantworten kann. Denn angenommen P kennt ein r und ein y mit $r^2 \equiv x \mod n$ und $y^2 \equiv xv \mod n$. Dann ist $s = yr^{-1}$ eine Quadratwurzel von v. Das heißt also, dass V in diesem Fall nur mit Wahrscheinlichkeit $\frac{1}{2^k}$ akzeptieren würde. (C) Die *Zero–knowledge–Eigenschaft* zeigen wir, indem wir zeigen, dass die gesamte Information die V während des Protokolls erhält auch in einer Simulation ohne Kenntnis von s erzeugt werden kann. Die Information, die V erhält sind in jeder Runde ein x und ein y, sodass $y^2 \equiv xv^e \mod n$ gilt. Für die Simulation wähle man zufällig $y \in \{1, \ldots, n-1\}$ mit $(y, n) = 1$, und $e \in \{0, 1\}$. Mit $x = y^2 v^{-e} \mod n$ gilt dann $y^2 \equiv xv^e \mod n$ und x ist ein gleichverteiltes zufälliges Quadrat mit $(x, n) = 1$. Die Folge der x und y die bei k-facher Wiederholung erzeugt werden unterscheidet sich in Verteilung nicht von der Folge, die im Verlaufe des Protokolls erhalten wurde.

Bei genauerer Betrachtung des Protokolls mag sich die Frage stellen, warum V nicht immer nur $e = 1$ an P schickt. Dies macht er nicht, weil er nicht davon ausgehen kann, dass P sich an die Details des Protokolls hält. Tatsächlich kann P sich bereits im ersten Schritt überlegen, in welchen der beiden Fälle, $e = 0$ oder $e = 1$ sie ein richtiges y schicken können will. Wie das? Ähnlich wie in der Simulation! Will sie für $e = 0$ die richtige Antwort schicken können, so verfährt sie wie im Protokoll. Will sie hingegen für $e = 1$ die richtige Antwort schicken können, so wählt sie zufällig $y \in \{1, \ldots, n-1\}$ mit $(y, n) = 1$ und sendet im ersten Schritt $x = y^2 v^{-1} \mod n$ an V. Der oben geführte Korrektheitsbeweis erweist sich aber zum Glück auch dann als gültig, wenn P auf irgendeine Weise vom Protokoll abweicht.

Formale Definition

Die Formalisierung der diskutierten Zero–knowledge–Protokolle geht auf eine Arbeit von Goldwasser, Micali und Rackoff von 1989 zurück.

Es sei L eine Sprache, die Bob (V) und Alice (P) bekannt ist. Alices Geheimnis sei von der Form $x \in L$.

Definition 14.2. *Ein Paar* (P, V) *von* kommunizierenden Turing–Maschinen mit Eingabe x *besteht aus*

- *probabilistischen Turing–Maschinen* P *und* V,
- *wobei* V *polynomiell beschränkt ist.*
- P *und* V *können mittels eine Kommunikationsbandes Nachrichten miteinander austauschen. Nachdem eine der Maschinen eine Nachricht auf das Kommunikationsband geschrieben hat, geht sie in den Wartezustand über, bis sie eine Antwort erhält.*
- P *und* V *haben ein gemeinsames Eingabeband, welches mit* x *beschrieben wird.*
- *Falls* V *anhält, so schreibt* V *eine "1" für "akzeptiert", oder eine "0" für "verwirft" auf das Band, und die Maschine hält. Dieser Ausgabewert sei durch* $[P(x), V(x)]$ *bezeichnet.*

Damit Bob nicht schon selbst in polynomieller Zeit testen kann, ob $x \in L$ gilt, ist dies also nur interessant für Sprachen, für die nicht bekannt ist, ob sie in $\mathcal{P}$ liegen.

Definition 14.3. *Ein* interaktiver Beweis für eine Sprache L *ist ein Paar* (P, V) *von kommunizierenden Turing-Maschinen, sodass*

(A) Vollständigkeit: für alle $x \in L$ *gilt* $Prob([P(x), V(x)] = 1) = 1$.
(B) Korrektheit: für jede Turing-Maschine P^*, *jedes Polynom* p *und jedes* $x \notin L$ *von genügend großer Länge* $|x|$ *gilt*

$$Prob([P^*(x), V(x)] = 1) < \frac{1}{p(|x|)}.$$

Die Bedeutung von (A) ist offensichtlich: Falls Alice (P) $x \in L$ beweisen kann, so akzeptiert Bob (V) dies mit Wahrscheinlichkeit 1. Gilt jedoch $x \notin L$, so sagt (B), dass selbst wenn Alice vom Protokoll in Form von P^* abweichen sollte, Bob nur mit verschwindend geringer Wahrscheinlichkeit akzeptiert.

Definition 14.4. *Ein interaktiver Beweis ist* zero–knowledge, *falls zusätzlich gilt*

(C) für jede probabilistische polynomielle Turing-Maschine V^* *existiert eine probabilistische Turing-Maschine* M_{V^*}, *die in Erwartung polynomiell in* $|x|$ *läuft, sodass für alle* $x \in L$ *gilt, dass* $M_{V^*}(x)$ *und* $[P(x), V^*(x)]$ *identisch verteilt sind.*

M_{V^*} heißt die V^* simulierende Maschine.

Das heisst, selbst wenn Bob vom Protokoll durch V^* abweicht, so ist er doch nur in der Lage Informationen zu erhalten, die er in (erwarteter) Polynomzeit auch selbst berechnen könnte.

Das Graphenisomorphie–Protokoll

Sei L die Sprache der Paare von Isomorphen Graphen, genauer sei

$$L = \{(n, G_0, G_1) : G_0, G_1 \text{ Graphen mit } V(G_0) = V(G_1) = [n], G_0 \cong G_1\}.$$

Es ist klar, dass $L \in \mathcal{NP}$ liegt, $L \in \mathcal{P}$ hingegen ist nicht bekannt. Ebenso ist unbekannt, ob L $\mathcal{NP}$-vollständig ist.

Das Protokoll

Sei $(n, G_0, G_1) \in L$ und $\varphi : G_0 \to G_1$ ein Isomorphismus.

1. P wählt eine zufällige Permutation $\pi : [n] \to [n]$ und schickt $H = \pi G_1$ an V.
2. V wählt zufällig $e \in \{0, 1\}$ und schickt dies an P.
3. P sendet $\pi \circ \varphi^e$ an V.
4. V testet, ob $\pi \circ \varphi^e : G_{1-e} \to H$ ein Isomorphismus ist.

Dies wird k-fach iteriert. Ist in jeder Runde $\pi \circ \varphi^e$ ein Isomorphismus, so akzeptiert V, ansonsten verwirft er.

Coin Flipping

Alice und Bob benötigen das zufällige Ergebnis eines Münzwurfes. Sie mißtrauen sich und können sich auch nicht treffen, um einen Münzwurf durchzuführen. Wie können sie ihr Dilemma lösen?

Zunächst ist eine Variante denkbar, in der ein *vertrauenswürdiger Agent* eingeschaltet wird. An einen solchen könnten sowohl Alice als auch Bob ein Bit schicken. Das Ergebnis des Münzwurfes, die Summe der Bits modulo zwei, würde der Agent dann an beide zurückschicken.

Eine *technische Variante* ist ebenfalls denkbar. Bob wählt ein Bit b, schreibt es auf einen Zettel und verschliesst diesen in einer Kiste. Diese Kiste schickt er an Alice. Alice wählt ihr Bit b, schreibt es auf einen Zettel und schickt diesen an Bob. Sobald Bob den Zettel erhält, schickt er den Schlüssel für die Kiste an Alice. Das Ergebnis des Münzwurfes ist für beide wieder die Summe $a \oplus b$.

Ein kryptographisches Protokoll

Sei $q|p-1$ und g ein Element der Ordnung q in $\mathbb{Z}_p^\times$.

1. Alice wählt $x \in \{0, \dots, q-1\}$ zufällig und schickt $y = g^x \mod p$ an Bob.
2. Bob wählt ein zufälliges Bit b, ein zufälliges $k \in \{0, \dots, q-1\}$ und schickt $r = y^b g^k \mod p$ an Alice.
3. Alice wählt ein zufälliges Bit a und schickt es an Bob.
4. Bob schickt b und k an Alice.
5. Alice prüft $r = y^b g^k \mod p$. Ist dies der Fall, so ist das Ergebnis des Münzwurfes $a \oplus b$.

Es ist leicht zu sehen, dass das Coin Flipping–Protokoll sicher ist, falls das diskrete Logarithmus Problem schwer ist. Die Umkehrung ist ebenfalls klar, da in Schritt 4. Bob ein b seiner Wahl schicken kann, und das k mit Hilfe des diskreten Logarithmus entsprechend bestimmen kann.

Ein weiteres kryptographisches Protokoll

Hier nehmen wir an, dass Alice und Bob eine Ausgangspräferenz haben: Kopf oder Zahl.

1. Alice wählt p, q und berechnet $n = pq$.
2. Alice sendet n an Bob. Alice gewinnt, falls Bob am Ende des Spieles n nicht faktorisieren kann.
3. Bob wählt $r \in \mathbb{Z}_n$ zufällig und schickt $k = r^2 \mod n$ an Alice.
4. Alice berechnet die vier Quadratwurzeln $\pm r, \pm s$, wählt eine davon aus, nennen wir sie y, und schickt y an Bob.
5. Ist $y = \pm r$, so hat Bob keine neue Information. Ist jedoch $y = \pm s$, so kann er n faktorisieren.

Bob kann also mit mindestens 50%-iger Wahrscheinlichkeit n faktorisieren. Ist die Wahl von p und q ungünstig, so steigt die Chance für Bob. Diese Wahl liegt jedoch bei Alice, dass heißt sie würde sich selbst schaden.

Sachverzeichnis