Vorlesung Datenschutz und Datensicherheit - WS 2001/02
- vorläufig und noch unvollständig -
- Version vom 30.05.2002 -

Prof. Dr. Hanno Lefmann
TU-Chemnitz, Professur für Theoretische Informatik und Informationssicherheit

1 Einführung

Definition 1.1 *Ein kryptographisches System (Kryptosystem) ist ein* 5*-Tupel* (M, K, C, E, D) *bestehend aus:*

- *der Menge* M *der Nachrichten (Klartexte, plaintext),*
  z.B. $M = \{0,1\}^*$*, also die Menge der endlichen 0,1 Folgen*
- *der endlichen Menge* K *der Schlüssel (key),*
  z.B. $K = \{0,1\}^{56}$
- *der Menge* C *der Kryptogramme (verschlüsselte Nachrichten, ciphertext),*
  z.B. $C = \{0,1\}^*$
- *einer Verschlüsselungsfunktion (encryption)* $E\colon M \times K \longrightarrow C$*,*
  Nachricht $m \in M$ *wird mit Schlüssel* $k \in K$ *zu Kryptogramm* $c = E(m,k)$*,*
- *einer Entschlüsselungsfunktion (decryption)* $D\colon C \times K \longrightarrow M$ *mit: für jedes* $k \in K$ *gibt es genau einen Schlüssel* $k' \in K$ *(Gegenschlüssel) mit* $D(E(m,k),k') = m \quad \forall\, m \in M$*. Diese Forderung impliziert, dass die Funktion* $E(\cdot,k)$ *für jedes* $k \in K$ *injektiv ist.*

Wenn Schlüssel k und Gegenschlüssel k' leicht auseinander ableitbar sind, spricht man von einem symmetrischen, ansonsten von einem asymmetrischen Kryptosystem.
Wenn zwei Personen A und B (wie Alice und Bob) miteinander kommunizieren wollen, vereinbaren sie vorab das zugrundeliegende Kryptosystem. Wenn A eine Nachricht an B senden möchte, fixiert

A zunächst einen Schlüssel $k \in K$ und sendet diesen oder den Gegenschlüssel $k' \in K$ auf einem *sicheren* Kanal (Weg) an B.

Wir betrachten zunächst einige einfache kryptographische Systeme, als erstes eine Cäsar-Chiffre.

Bei Cäsar-Chiffren wird das Alphabet $\{A, B, \ldots, Z\}$ zunächst auf die Menge $\mathbf{Z}_{26} = \{0, \ldots, 25\} = M = C = K$ (also den Restklassenring modulo 26) abgebildet, etwa $A \longrightarrow 0, B \longrightarrow 1, \ldots, Z \longrightarrow 25$. Die Verschlüsselungsfunktion E bildet innerhalb des Restklassenringes modulo 26 ab, also $E\colon \mathbf{Z}_{26} \times \mathbf{Z}_{26} \longrightarrow \mathbf{Z}_{26}$, wobei $E(m, k) \equiv m+k \bmod 26$ ist. Mit dem Schlüssel k wird also ein 'shift' ausgeführt. Als Entschlüsselungsfunktion $D\colon \mathbf{Z}_{26} \times \mathbf{Z}_{26} \longrightarrow \mathbf{Z}_{26}$ wird $D(c, k) = c - k \bmod 26$ benutzt. Die Verschlüsselung von Worten, also endlichen Folgen aus $\mathbf{Z}_{26}^*$, erfolgt buchstabenweise. Die Cäsar-Chiffre ist ein symmetrisches Kryptosystem.

Beispiel 1.2 *Das Wort 'OTTO' wird bei Verwendung des Schlüssels* 3 *(wie zu Cäsars Zeiten) chiffriert zu 'RWWR' (oder mit Zahlen* '18,23,23,18'*).*

Wir betrachten nun einige Angriffsarten gegen beliebige Kryptosysteme.

Angriffe gegen ein Kryptosystem:

Wir unterscheiden je nach Wissensstand des Gegners folgende Angriffe gegen beliebige Kryptosysteme:

- ciphertext only attack (nur Kryptogramme)
  Der Gegner kennt nur einige Kryptogramme $c_1, \ldots, c_n$.
- known plaintext attack (bekannte Klartexte)
  Der Gegner kennt einige Klartext-Kryptogramm-Paare $(m_1, c_1), \ldots, (m_n, c_n)$.
- chosen plaintext attack (gewählte Klartexte)
  Der Gegner kennt für von ihm gewählte Klartexte $m_1, \ldots, m_n$ die zugehörigen Kryptogramme $c_1, \ldots, c_n$.
- chosen ciphertext attack (gewählte Kryptogramme)
  Der Gegner kennt für einige von ihm gewählte Kryptogramme $c_1, \ldots, c_n$ die zugehörigen Nachrichten $m_1, \ldots, m_n$.

Ein Kryptosystem heißt sicher bzgl. eines Angriffs, wenn es mit den vorhandenen Ressourcen nicht möglich ist, zu einem gesendeten Kryptogramm $c \in C$ die zugehörige Nachricht $m \in M$ zu ermitteln (bzw. den für die Entschlüsslung erforderlichen Gegenschlüssel).
Allgemein wird von Kerkhoff's Prinzip ausgegangen:
Ein möglicher Angreifer kennt das zugrundeliegende Kryptosystem (aber nicht den individuell benutzten Schlüssel).

Beispiel 1.3 *Die vorgestellte Cäsar-Chiffre (über $\mathbf{Z}_{26}$) ist nicht sicher bezüglich eines known plain text attacks. Kennt man nur ein Nachricht/Kryptogrammpaar $(m, c) \in (\mathbf{Z}_{26})^2$, so braucht man nur die Gleichung $m + k \equiv c \bmod 26$ zu lösen, also $k \equiv c - m \bmod 26$.*

Einige einfache kryptographische Systeme

1. Cäsar Chiffren

   Bei allgemeinen Cäsar-Chiffren wird als Alphabet die Menge $M = \mathbf{Z}_a = \{0, \ldots, a-1\}$ (Nachrichtenmenge) verwendet. Die Schlüsselmenge ist $K = \mathbf{Z}_a$ und die Kryptogrammmenge ist ebenfalls $C = \mathbf{Z}_a$. Die Verschlüsselungsfunktion E bildet innerhalb des Restklassenringes modulo a ab, also $E\colon \mathbf{Z}_a \times \mathbf{Z}_a \longrightarrow \mathbf{Z}_a$, wobei $E(m,k) \equiv m + k \bmod a$ ist. Als Entschlüsselungsfunktion $D\colon \mathbf{Z}_a \times \mathbf{Z}_a \longrightarrow \mathbf{Z}_a$ wird $D(c,k) = c - k \bmod a$ benutzt. Die Verschlüsselung von Worten, also endlichen Folgen aus $\mathbf{Z}_a^*$, erfolgt buchstabenweise.

2. Blockchiffren (Block Ciphers, ECB-Modus 'electronic code book')

   Seien A, A' endliche Alphabete, $M = A^*$, $C = A'^*$ und K eine endliche Menge von Schlüsseln. Eine Chiffrierfunktion $E\colon M \times K \longrightarrow C$ heißt (p,p')-Blockchiffre, wenn es eine Funktion $E'\colon A^p \times K \longrightarrow A'^{p'}$ gibt, wobei $E'(\cdot, k)$ injektiv ist für jeden Schlüssel $k \in K$, so dass gilt: Für jede Nachricht $m \in A^n$ der Länge $n = r \cdot p$, zerlegt in r Blöcke $m_1, \ldots, m_r$ jeweils der Länge p, gilt

   $$E(m,k) = (E'(m_1,k), E'(m_2,k), \ldots, E'(m_r,k)) \,.$$

   Ist n nicht durch p teilbar, also $(r-1) \cdot p < n < r \cdot p$, so fülle die Nachricht m beliebig bis Länge $r \cdot p$ auf. Das Chiffrat $E(m,k)$ hat die Länge $r \cdot p'$.

   Der Nachteil der (p,p')-Blockchiffren ist, dass bei durch p teilbarer Nachrichtenlänge gleiche Klartextblöcke gleiche Kryptogramme erzeugen. Regelmäßig auftretende Textmuster findet

man dann im Kryptogramm wieder ($\longrightarrow$ lange Nachrichten vermeiden!). Ein Vorteil ist hingegen, dass z.B. Bitumkehrfehler bei der Übertragung isoliert bleiben und nur einen Block unbrauchbar machen.

Die Cäsar-Chiffren sind $(1,1)$-Blockchiffren.

3. Flusschiffren (Stream Ciphers)

   Im Gegensatz zu Blockchiffren verwenden Flusschiffren für jeden zu verschüsselnden Buchstaben einen anderen Schlüssel (die Verschlüsselung ist positionsabhängig). Sei etwa $M = C = \{0,1\}^*$ (oder ein anderes zugrunde liegendes Alphabet) und eine endliche Schlüsselmenge, etwa $K = \{0,1\}^d$, gegeben. Ein Schlüssel $k \in K$ wird als Eingabe für einen deterministischen 'Zufallszahlengenerator' verwendet, der eine (im Prinzip beliebig lange) Ausgabefolge $z(k) = (z_1, z_2, \ldots) \in \{0,1\}^*$ produziert.

   Eine Nachricht $m = (m_1, \ldots, m_n) \in M$ wird verschlüsselt mit

$$E(m,k) = (m_1 \oplus z_1, \ldots, m_n \oplus z_n) \; .$$

   Die Entschlüsselung erfolgt dann mit

$$D(c,k) = (c_1 \oplus z_1, \ldots, c_n \oplus z_n) \; .$$

   Hierbei ist $\oplus$ das *Exclusive Or* (XOR) (bitweise Addition ohne Übertrag) mit $0 \oplus 1 = 1 \oplus 0 = 1$ und $x \oplus x = 0$ (in $\mathbf{Z}_2$).

   Zu gegebenem Schlüssel $k = (k_1, \ldots, k_d) \in \{0,1\}^d$ kann man eine Folge $z(k)$ etwa wie folgt finden. Man fixiert für den 'Zufallszahlengenerator' Zahlen $c_1, \ldots, c_d \in \{0,1\}$. Dann setzt man $z_j := k_j$ für $j = 1, \ldots, d$ und für $j > d$:

$$z_j := \sum_{i=1}^{d} c_i \cdot z_{j-i} \bmod 2 \; .$$

   Die Implementierung erfolgt leicht mit linearen Schieberegistern. Natürlich ist die so konstruierte Folge $z(k)$ nicht zufällig.

Beispiel 1.4 *Sei* $d = 3$ *und* $c_1 = c_2 = 1$ *und* $c_3 = 0$. *Dann erhält man die Folge* $1,1,0,1,1,0,1,1,0,\ldots$ *basierend auf der Rekursion* $z_j \equiv z_{j-1} + z_{j-2} \bmod 2$ *für* $j \geq 3$ *und* $z_1 = z_2 = 1$.

Ein Vorteil von Flusschiffren ist, dass Buchstaben/Blöcke positionsabhängig verschlüsselt werden, also gleiche Buchstaben teilweise mit verschiedenen z_i .

4. Chaining (CBC-Modus; cipher block chaining mode)

   Beim Chaining vermeidet man das Auftreten regelmäßiger Muster. Die Verschlüsselung eines Blockes hängt von allen vorherigen Blöcken ab. Eine Nachricht $m \in \{0,1\}^*$ wird zunächst in Blöcke $m_1, \ldots, m_r$ jeweils der Länge p zerlegt. Die Verschüsselung erfolgt auf diesen Blöcken mittels einer Verschlüsselungsfunktion $E'\colon \{0,1\}^p \times K \longrightarrow \{0,1\}^p$.

   Nun wählt man zusätzlich zu dem Schlüssel $k \in K$ eine Initialisierungsfolge $z \in \{0,1\}^p$ und bildet das Chiffrat $E(m,(z,k))$ mit

   $$E(m,(z,k)) = (c_1, \ldots, c_r) = (E'(m_1 \oplus z, k), E'(m_2 \oplus c_1, k), \ldots, E'(m_r \oplus c_{r-1}, k)) \ ,$$

   wobei gilt $c_1 = E'(m_1 \oplus z, k)$ und $c_j = E'(m_j \oplus c_{j-1}, k)$ für $j = 2, \ldots, r$.

   Für die Dechiffrierfunktion $D\colon C \times K \longrightarrow M$ gilt dann

   $$D(c,(z,k)) = (m_1, \ldots, m_r)$$

   mit $m_1 = E'^{-1}(c_1,k) \oplus z$ und $m_j = E'^{-1}(c_j,k) \oplus c_{j-1}$ für $j = 2, \ldots, r$. Hierbei ist E'^{-1} (Dechiffrierfunktion) die Umkehrfunktion zu E'.

   Im Allgemeinen werden gleiche Klartextblöcke verschieden verschlüsselt, da die Verschlüsselung des Kryptogrammblocks c_j von dem Nachrichtenblock m_j und allen vorhergehenden Nachrichtenblöcken $m_1, \ldots, m_{j-1}$ abhängt. Es ist nicht möglich, nur einzelne Kryptogrammblöcke etwa in der Mitte zu entschüsseln. Ein Bitumkehrfehler in Block c_j wirkt sich möglicherweise auf die Entschlüsselung von den Nachrichtenblöcken m_j und m_{j+1} aus, aber nicht auf die folgenden Blöcke $m_{j+2}, \ldots$. Bitumkehrfehler können zum Beispiel bei Störungen auftreten.

5. CFB-Modus (cipher feedback mode)

   Eine Beschleunigung der Entschlüsselung der Chiffrate beim Empfänger gegenüber dem ECB-Modus zeigt das folgend beschriebene Verfahren. Beim ECB-Modus können nämlich Verschlüsselung und Entschlüsselung nur nacheinander ausgeführt werden, was bei zeitaufwendigen Berechnungen ein Problem sein kann.

Man wählt einen Initialisierungsvektor $z \in \{0,1\}^n$ der Länge n. Dann wird eine Zahl $p \leq n$ fixiert. Eine Nachricht $m \in \{0,1\}^*$ wird in Blöcke $m_1, \ldots, m_r$ jeweils der Länge p zerlegt. Gegeben ist eine Chiffrierfunktion E: $\{0,1\}^n \times K \longrightarrow \{0,1\}^*$ mit der endlichen Schlüsselmenge K, die Sender und Empfänger kennen. Anfangs setzt man

$$I_1 := z \, .$$

Für $j = 1, \ldots, r$ werden jeweils folgende Schritte durchgeführt:

1.) $t_j =$ erste p bits von $E(I_j, k)$

2.) $c_j = m_j \oplus t_j$

3.) $I_{j+1} := 2^p \cdot I_j + c_j \bmod 2^n$ (Man entfernt die ersten p bits von I_j, schiebt die verbleibenden bits nach 'links' und an den p freigewordenen Positionen fügt man die Bits von c_j ein.)

Der Empfänger geht analog vor und berechnet:

$$I_1 := z$$

sowie für $j = 1, \ldots, r$:

1.) $t_j =$ erste p bits von $E(I_j, k)$

2.) $m_j = c_j \oplus t_j$

3.) $I_{j+1} := 2^p \cdot I_j + c_j \bmod 2^n$.

Der Vorteil ist, dass Sender und Empfänger gleichzeitig t_{j+1} berechnen können, sobald c_j bekannt ist. Bei Empfang der Chiffrate $c_1, c_2, \ldots$ braucht der Empfänger zur Nachrichtenermittlung nur XOR ausführen.

Ein Bitumkehrfehler in c_j wirkt sich nur auf die nächsten $\lceil n/p \rceil$ Blöcke aus, da pro Schritt 3.) der String c_j jeweils um p Stellen verschoben wird und schließlich nicht mehr im I-String auftritt.

6. Verschiedene oder auch gleiche Kryptosysteme können kombiniert werden, etwa durch Kompositionen oder Produktbildungen.

   So lässt sich aus den beiden Systemen (M, K, C, E, D) und (C, K', C', E', D') durch Kompositionen (Hintereinanderausführung) das System $(M, K \times K', C', E' \circ E, D \circ D')$ bilden mit

Verschlüsselungsfunktion $E' \circ E \colon M \times (K \times K') \longrightarrow C'$, wobei gilt

$$(E' \circ E)(m, (k, k')) := E'(E(m, k), k')$$

und Entschlüsselungsfunktion $D \circ D' \colon C \times (K' \times K) \longrightarrow M$ mit

$$D \circ D'(c, (k', k)) := D(D'(c, k'), k) \; .$$

Als Produkt der beiden Kryptosysteme (M, K, C, E, D) und (C, K', C', E', D') ergibt sich das System $(M \times M', K \times K', C \times C', E \times E', D \times D')$ mit

$$\begin{aligned} (E \times E')((m, m'), (k, k')) &= (E(m, k), E'(m', k')) \\ (D \times D')((c, c'), (k, k')) &= (D(c, k), D'(c', k')) \; . \end{aligned}$$

Beispiel 1.5 *Die Komposition von zwei Cäsar-Chiffren liefert* $E' \circ E(m, (k, k')) = E'(m + k \bmod a, k') \equiv m + k + k' \bmod a$. *Man bekommt also wieder eine Cäsar-Chiffre.*

Das Produkt von zwei Cäsar-Chiffren liefert $E \times E'((m, m'), (k, k')) = (m + k \bmod a, m' + k' \bmod b)$, *wobei* $a = b$ *gelten kann.*

2 Methoden der Kryptoanalyse

Jedes kryptographische System kann, wenn auch eventuell mit großem Aufwand, geknackt werden. Der Beweis, dass ein Kryptosystem nur *mit großem Aufwand geknackt* werden kann, ist nicht immer möglich. Deswegen werden neu entwickelte Systeme mit bekannten Methoden der Kryptoanalyse gemessen und beurteilt. Nehmen wir einmal die Gleichverteilung der Parameter an. Wenn $M = \{0, 1\}^l$ die Menge der Nachrichten ist, wird jede Nachricht $m \in M$ mit einer Wahrscheinlichkeit 2^{-l} gesendet. Ist die Anzahl der Schlüssel $|K| = 2^k$ und wird jedes Kryptogramm $c \in \{0, 1\}^r$ aus gleich vielen Nachricht-Schlüssel-Kombinationen erreicht, so ergeben sich insgesamt 2^{l+k-r} Kombinationen um c zu erhalten. Da die Verschlüsselungsfunktion für jeden Schlüssel $k \in K$ injektiv sein soll, muss $r \geq l$ gelten. Oft wird auch $r = l$ gewählt, damit Nachrichten bei der Verschlüsselung nicht verlängert werden. Im Allgemeinen gilt weiterhin, dass $k > l$ ist. Bilden die Nachrichten sinnvolle Sätze, wird dem Gegner dadurch geholfen, dass manche Nachrichten eine Wahrscheinlichkeit gleich 0 haben.

Grundlegende Methoden der Kryptoanalyse

1. Probable-Word-Methode

   Die Probable-Word-Methode ist eine heuristische Methode, die nach häufig vorkommenden Wörtern sucht (*ciphertext only attack*). Häufig auftretende Muster - wie beispielsweise „Sehr geehrte Damen und Herren“ - werden durch Chaining vermieden.

2. Vollständige Suche

   Bei der vollständigen Suche werden alle Schlüssel auf das Kryptogramm c angewendet, das heißt, es wird $D(c,k)$ für alle $k \in K$ berechnet. Bei einem *known plaintext attack* werden mit dieser Methode alle passenden Schlüssel gefunden, während bei *ciphertext only attack* alle Schlüssel gefunden werden, die zu sinnvollen Texten führen. Um eine vollständige Suche zu verhindern, sollte die Schlüsselmenge sehr groß sein, beispielsweise benutzt DES (Data Encryption Standard) 2^{56} Schlüssel.

3. Strukturelle Zerlegung des Schlüsselraumes

   Falls sich die Schlüssel $k = (k_1, k_2)$ eines Kryptosystems separieren lassen, also getrennt nach k_1 und k_2 gesucht werden kann, wird das System unsicherer. Hängt zum Beispiel die erste Hälfte des Kryptogramms ausschließlich von k_1 und die zweite Hälfte ausschließlich von k_2 ab, so verringert sich die Anzahl der Möglichkeiten sehr stark. Nehmen wir an, dass $|k_1| = |k_2| = 28$ ist, dann verringert sich die Anzahl der durchzuführenden Tests von $2^{28} \cdot 2^{28} = 2^{56}$ auf $2 \cdot 2^{28}$. *Jedes Kryptogrammbit sollte daher möglichst von allen Schlüsselbits abhängen.*

4. Sprachstatistik

   Die zu verschlüsselnden Nachrichten sind im Allgemeinen strukturiert und enthalten Standardredewendungen oder entsprechen einem bestimmten Datenformat. Diese Eigenschaft kann bei der Kryptoanalyse ausgenutzt werden, indem *Häufigkeitstabellen* für Buchstaben, Buchstabenpaare (*Digramme*) oder -tripel (*Trigramme*) erstellt werden. Nach dem Gesetz der großen Zahlen werden diese Häufigkeiten bei langen Texten approximiert. Wir betrachten folgendes Beispiel:

   Beispiel 2.1 *Wir haben eine allgemeine Cäsar-Chiffre mit dem Alphabet* $A = \mathbf{Z}_a = \{0, 1, \ldots, a-1\}$ *vorliegen.*

   1. *Wir nehmen an, dass die Buchstaben des Textes* $m \in Z_a^*$ *unabhängig voneinander gemäß*

der Wahrscheinlichkeitverteilung P auf dem Alphabet A mit

$$P = (p(0), p(1), \ldots, p(a-1))$$

ausgewählt werden, was real annähernd zutreffend ist. Der Wert $p(x)$ gibt also die Wahrscheinlichkeit für das Auftreten des Buchstaben x an. Wenn $k \in \mathbf{Z}_a$ der Schlüssel ist, so hat das Kryptogramm die Wahrscheinlichkeitsverteilung $P' = (p(-k), p(-k+1), \ldots, p(-k+a-1))$.

Sei n die Länge des Klartextes $m \in \mathbf{Z}_a^*$ und $Z_n(x)$ eine Zufallsvariable, die angibt, wie oft der Buchstabe x im Klartext m vorkommt. Es ergibt sich für die Wahrscheinlichkeit, dass genau l-mal der Buchstabe x im Klartext m auftritt:

$$Pr\ (Z_n(x) = l) = \binom{n}{l} \cdot p(x)^l \cdot (1 - p(x))^{n-l}\ .$$

denn man hat $\binom{n}{l}$ Möglichkeiten, l der n Positionen im Klartext auszuwählen, und an diesen l Positionen tritt jeweils mit Wahrscheinlichkeit $p(x)$ Buchstabe x auf und jede andere der $(n-l)$ Positionen enthält kein x mit Wahrscheinlichkeit $1-p(x)$. Somit ergibt sich als Erwartungswert für die Zufallsvariable $Z_n(x)$:

$$E(Z_n(x)) = \sum_{l=0}^{n} l \cdot Pr\ (Z_n(x) = l) = \sum_{l=0}^{n} l \cdot \binom{n}{l} \cdot p(x)^l \cdot (1 - p(x))^{n-l} = n \cdot p(x)\ .$$

Als Varianz ergibt sich dann:

$$V(Z_n(x)) = E\left((Z_n(x) - E(Z_n(x)))^2\right) = n \cdot p(x) \cdot (1 - p(x))\ .$$

Wir führen nun eine neue Zufallsvariable $Z_n^*(x)$ ein mit $Z_n^*(x) := \frac{Z_n(x) - E(Z_n(x))}{\sqrt{V(Z_n(x))}}$. Die Zufalsvariable $Z_n^*(x)$ hat eine zentrierte und normierte Binomialverteilung, die sich für große n wie eine Standardnormalverteilung verhält. Inbesondere gilt für große n:

$$Pr\ (a \leq Z_n^*(x) \leq b) \approx \int_a^b \frac{e^{-\frac{x^2}{2}}}{2\pi} dx\ ,$$

und damit

$$\begin{aligned} 0,9973 &\leq Pr\ (-3 \leq Z_n^*(x) \leq 3) \\ &= Pr\left(-3 \leq \frac{Z_n(x) - E(Z_n(x))}{\sqrt{V(Z_n(x))}} \leq 3\right) \end{aligned}$$

$$
\begin{aligned}
&= Pr\left(-3 \le \frac{Z_n(x) - n \cdot p(x)}{\sqrt{n \cdot p(x) \cdot (1-p(x))}} \le 3\right) \\
&= Pr\left(-3 \cdot \sqrt{\frac{p(x)\cdot(1-p(x))}{n}} \le \frac{Z_n(x)}{n} - p(x) \le 3 \cdot \sqrt{\frac{p(x)\cdot(1-p(x))}{n}}\right) \\
&= Pr\left(\left|\frac{Z_n(x)}{n} - p(x)\right| \le 3 \cdot \sqrt{\frac{p(x)\cdot(1-p(x))}{n}}\right) .
\end{aligned}
$$

Mit $p(x) \cdot (1-p(x)) \le {}^1/_4$ *folgt*

$$
\begin{aligned}
Pr\left(\left|\frac{Z_n(x)}{n} - p(x)\right| \le 3 \cdot \sqrt{\frac{p(x)\cdot(1-p(x))}{n}}\right) &\ge 0,9973 \\
Pr\left(\left|\frac{Z_n(x)}{n} - p(x)\right| \le \frac{3}{2\cdot\sqrt{n}}\right) &\ge 0,9973
\end{aligned}
$$

also

$$
Pr\left(\left|\frac{Z_n(x)}{n} - p(x)\right| > \frac{3}{2\cdot\sqrt{n}}\right) \le 0,0027.
$$

Für Wortlänge $n = 10000$ *erhält man etwa:*

$$
Pr\left(\left|\frac{Z_{10000}(x)}{10000} - p(x)\right| > 0,015\right) \le 0,0027 .
$$

Also ist die Wahrscheinlichkeit, dass die relative Häufigkeit $Z_{10000}(x)/10000$ *des Buchstaben* x *um mehr als 0,015 von der Wahrscheinlichkeit* $p(x)$ *des Auftretens von* x *abweicht, kleiner als 0,27%. Betrachtet man das englische Alphabet, so sind die Wahrscheinlichkeiten der beiden häufigsten Buchstaben e und t:* $p(e) = 0,1304$ *und* $p(t) = 0,1045$ *und somit* $p(e) - p(t) = 0,0259 \gg 0,015$. *Es kann also mit großer Wahrscheinlichkeit in einem langen Text zwischen t und e unterschieden werden und Cäsar-Chiffre mit Hilfe einer Häufigkeitsanalyse geknackt werden.*

2. *Eine zweite Möglichkeit, Cäsar-Chiffren über dem Alphabet* $\mathbf{Z}_a$ *durch eine Häufigkeitsanalyse zu brechen, ist die folgende: Sei* Z *eine Zufallsvariable mit der Verteilung* $p = (p(0), \ldots, p(a-1))$, *wobei* $p(x)$ *die Wahrscheinlichkeit von Buchstabe* x *im Klartext*

ist. Seien weiterhin $q = (q(0), \ldots, q(a-1))$ *die (beobachteten)* relativen Häufigkeiten *der Buchstaben im Kryptogramm. Bei Schlüssel* k *ist* $q_k = (q(k), \ldots, q(a-1+k))$ *damit die Buchstabenverteilung im Klartext. Die Idee der Analyse besteht darin, ein* $k \in \{0, 1, \ldots, a-1\}$ *zu bestimmen, bei dem die euklidische Norm*

$$||p - q_k|| = \left(\sum_{x=0}^{a-1} (p(x) - q(x+k))^2 \right)^{1/2}$$

minimal wird, und damit

$$\begin{aligned} \sum_{x=0}^{a-1} (p(x) - q(x+k))^2 &= \sum_{x=0}^{a-1} p(x)^2 + \sum_{x=0}^{a-1} q(x+k)^2 - 2 \cdot \sum_{x=0}^{a-1} p(x) \cdot q(x+k) \\ &= ||p||^2 + ||q||^2 - 2 \cdot \sum_{x=0}^{a-1} p(x) \cdot q(x+k) \end{aligned}$$

minimal ist. Da $||p||^2$ *und* $||q||^2$ *für jedes* k *konstant sind, reicht es, den Ausdruck*

$$\sum_{x=0}^{a-1} p(x) \cdot q(x+k)$$

zu maximieren. Für große n *gilt mit einer Wahrscheinlichkeit von fast* 1 *folgendes:*

$$\begin{aligned} k \text{ korrekt} &\rightarrow ||p - q_k|| \text{ klein} \\ k \text{ nicht korrekt} &\rightarrow ||p - q_k|| \text{ groß} . \end{aligned}$$

Wir haben hier die Korrelation *von zwei Zufallsvariablen berechnet. Dasjenige* k*, für dass der Ausdruck* $||p - q_k||$ *minimal ist, ist also mit Wahrscheinlichkeit fast* 1 *der korrekte Schlüssel, falls* n *groß ist.*

3. *Als dritte Möglichkeit bilden wir jeweils die Mengen* H_M *der* h *häufigsten Buchstaben im Klartext sowie entsprechend die Menge* H_C *der* h *häufigsten Buchstaben im Chiffrat. Ist etwa bei Cäsar-Chiffren der Durchschnitt* $H_M \cap \{x - k \mid x \in H_C\}$ *für ein* $k \in \mathbf{Z}_a$ *groß, so kann dieser Wert* k *als Schlüssel vermutet werden.*

   In einem Beispiel aus dem Englischen bei einer Klartextlänge von 1700 *Buchstaben und* $k = 9$ *liefert ein Durchschnitt der beiden Mengen von* 8 *den richtigen Schlüssel, während bei falschen Schlüsseln der Durchschnitt maximal Kardinalität* 4 *hat.*

3 Klassische Chiffren

Einfache Substitutionchiffren zählen zu den Blockchiffren, wobei $A = A'$ und $p = p' = 1$ gilt. Als Schlüssel werden Permutationen benutzt, wobei gilt $K = \{k\colon A \longrightarrow A \mid k \text{ Permutation}\}$ und $E(m,k) = k(m)$ und $D(c,k) = k^{-1}(c)$. Da offensichtlich $|K| = |A|!$ ist, ergibt sich bei einem Alphabet mit 26 Buchstaben als Anzahl der Schlüssel $|K| = 26! \approx 2^{88}$, also ein sehr großer Schlüsselraum. Solche Substitutionschiffren sind allerdings durch Analysen der Sprachstatistik bei *ciphertext only attack* erfolgreich zu brechen.

Eine Weiterentwicklung sind polygraphische Substitutionschiffren, welche auf Blöcken arbeiten. Dazu gehören zum Beispiel Playfair-Chiffren, die auf Baron Playfair of St. Andrews (1854) zurückgehen. Als Blockgröße wird $p = p' = 2$ gewählt und als Alphabete $A = A' = \{a, b, c, \ldots, z\} \setminus \{j\}$. Als Schlüsselraum wird die Menge der 5×5-Matrizen gewählt, wobei jeder Buchstabe aus A genau einmal in jeder Matrix vorkommt. Im Klartext wird i mit j gleichgesetzt und Doppelbuchstaben wie bb durch Einfügen eines q durch bqb ersetzt. Hierbei wird vorausgesetzt, dass der Doppelbuchstabe qq nicht auftritt. Bei ungerader Klartextlänge wird einfach ein beliebiger Buchstabe an das Ende des Klartextes angefügt. Bei der Verschlüsselung eines Buchstabenpaares (m, m') mit Schlüsselmatrix $k = (k_{i,j})$ werden drei Möglichkeiten unterschieden:

1. *Zeilenregel:* Die Buchstaben m und m' befinden sich in derselben Zeile der Matrix k:
   $(m, m') = (k_{i,s}, k_{i,t}) \qquad s \neq t$
   $E((m, m'), k) = (k_{i,s+1}, k_{i,t+1}) = (c, c')$,

   wobei alle (Index-)Additionen modulo 5 gerechnet werden. Für das Chiffrat (c, c') werden also die in der Matrix „rechts“ von m bzw. m' jeweils daneben liegenden Buchstaben gewählt.

2. *Spaltenregel:* Die Buchstaben m und m' sind in derselben Spalte der Matrix k:
   $(m, m') = (k_{i,s}, k_{j,s}) \qquad i \neq j$
   $E((m, m'), k) = (k_{i+1,s}, k_{j+1,s}) = (c, c')$

   Es werden also für das Chiffrat die jeweils „darunter“ liegenden Buchstaben verwendet.

3. *Rechtecksregel:* Die Buchstaben m und m' sind in verschiedenen Zeilen und Spalten von k:
   $(m, m') = (k_{i,s}, k_{j,t}) \qquad i \neq j, s \neq t$
   $E((m, m'), k)) = (k_{i,t}, k_{j,s}) = (c, c')$.

Die Entschlüsselung eines Buchstabenpaares (c, c') erfolgt in analoger Weise.
Insgesamt gibt es 25! Schlüssel, wobei einige von diesen durch Zeilen- oder Spaltenrotationen gleich arbeiten. Da es jeweils 5 Zeilen- und Spaltenrotationen gibt, ist die Anzahl verschieden arbeitender Schlüssel gleich ${}^{25!}/_{(5\cdot 5)} = 24!$. Bei einem *chosen plaintext attack* kann man einfach alle $25 \cdot 24 = 600$ Buchstabenpaare als Klartext wählen und erhält für jedes Paar das zugehörige Chiffrat. (Kommt man mit weniger als 600 Buchstabenpaaren aus?) Bei einem *ciphertext only attack* hingegen kommen Häufigkeitsanalysen von Digrammen zum Einsatz.

3.1 Hill-Chiffren

Auch die Hill-Chiffren gehören zu den Blockchiffren mit $A = A' = \mathbf{Z}_a$ und $p = p'$. Als Schlüsselmenge K wird die Menge der invertierbaren $p \times p$-Matrizen über $\mathbf{Z}_a$ benutzt. Für eine Nachricht $m \in A^p$ und den Schlüssel $k \in K$ ist $E(m,k) = k \cdot m$ und $D(c,k) = k^{-1} \cdot c$, jeweils als Matrix-Vektor-Produkt in $\mathbf{Z}_a$. Die inverse Matrix k^{-1} erfüllt dabei $k \cdot k^{-1} = I$, wobei I die Einheitsmatrix ist. Die inverse Matrix k^{-1} existiert genau dann, wenn gilt ggT $(\det(k), a) = 1$. Offenbar ist dann $D(E(m,k),k) = k^{-1} \cdot (k \cdot m) = m$ für jeden Klartext $m \in A^p$, also wird die Entschlüsselung korrekt durchgeführt.
Bei einem *chosen plaintext attack* sind Hill-Chiffren leicht zu brechen, wenn man einfach die Einheitsvektoren $e_i \in \mathbf{Z}_a^p$ als Nachrichten m_i wählt, also $e_i = (0, \ldots, 0, 1, 0, \ldots, 0)^t$ für $i = 1, \ldots, p$. Dann erhält man mit $E(e_i, k) = k \cdot e_i = k^i$ die i-te Spalte der Matrix k.
Ein *known plaintext attack* ist schon schwerer durchzuführen. Wenn uns l Klartexte $m_1, \ldots, m_l \in \mathbf{Z}_a^p$ und die zugehörigen Kryptogramme $c_1, \ldots, c_l \in \mathbf{Z}_a^p$ zur Verfügung stehen, können wir hoffen, dass $c_1, \ldots, c_l$ oder $m_1, \ldots, m_l$ ein Erzeugendensystem von $\mathbf{Z}_a^p$ bilden, also sollte zumindest $l \geq p$ gelten. Wir benutzen den Ansatz

a) $e_i = \sum_{j=1}^{l} \alpha_{i,j} \cdot m_j \quad i = 1, \ldots, n$

oder

b) $e_i = \sum_{j=1}^{l} \beta_{i,j} \cdot c_j \quad i = 1, \ldots, n$

und versuchen, eines dieser Gleichungssysteme nach den Unbekannten $\alpha_{i,j}$ bzw. $\beta_{i,j}$ zu lösen (etwa mit dem Eliminationsverfahren von Gauss). Gelingt dieses, so können wir danach mit

a) $k \cdot e_i = \sum_{j=1}^{l} \alpha_{i,j} \cdot k \cdot m_j = \sum_{j=1}^{l} \alpha_{i,j} \cdot c_j$
die i-te Spalte der Matrix k

bzw.

b) $k^{-1} \cdot e_i = \sum_{j=1}^{l} \beta_{i,j} \cdot k^{-1} \cdot c_j = \sum_{j=1}^{l} \beta_{i,j} \cdot m_j$
die i-te Spalte der inversen Matrix k^{-1} berechnen.

Beispiel 3.1 *Im folgenden sei* $a = 26$ *und* $p = 3$.

$$k = \begin{pmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{pmatrix}$$

$$k^{-1} = \begin{pmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{pmatrix}$$

Nehmen wir an, dass wir folgende 12 *plaintext/ciphertext Paare haben:*

DIF	*FER*	*ENT*	*PEO*	*PLE*	*HAV*	*EDI*	*FFE*	*REN*	*TOB*	*JEC*	*TIV*
EAN	*EOD*	*UPF*	*DFS*	*UZY*	*QQX*	*DUK*	*ITS*	*GAD*	*UWH*	*XRM*	*SWL*

Dadurch erhalten wir zwei 3×12 *Matrizen, eine, in der die Nachrichten* m_j *spaltenweise stehen (m-Matrix) und eine mit den Chiffraten* c_j *(c-Matrix) spaltenweise und wir wissen, dass* $k \cdot m_j = c_j$ *ist. Bei der Suche nach der Matrix* k *gehen wir nach dem Ansatz vor, dass wir Zeilen- und Spaltenoperationen beim Umformen einer Matrix auch analog in der anderen Matrix durchführen. Unser Ziel ist dabei,*

a) in der m-Matrix die 3×3 *Einheitsmatrix* I_3 *zu erzeugen, dann enthält die c-Matrix an den entsprechenden Stellen die Matrix* k*, oder*

b) in der c-Matrix die 3×3 *Einheitsmatrix* I_3 *zu erzeugen, dann enthält die m-Matrix an den entsprechenden Stellen die Matrix* k^{-1}.

Wir wollen nun überlegen, wieviele Klartext/Kryptogramm-Paare nötig sind, damit wir mit großer Wahrscheinlichkeit die Matrix k bestimmen können. Dazu nehmen wir an, dass wir l Paare (m_j, c_j), $i = 1, \ldots, l$, zufällig gegeben haben und dass die Nachrichten $m_1, \ldots, m_l \in \mathbf{Z}_a^p$ mit $a \geq 2$ gleichwahrscheinlich und unabhängig voneinander gewählt werden.

Sei U ein linearer Unterraum von $\mathbf{Z}_a^p$ mit $\dim(U) = s$. Wir führen eine Zufallsgröße M aus $\mathbf{Z}_a^p$ ein (zufälliges Ziehen einer Nachricht aus $\mathbf{Z}_a^p$). Wegen der Gleichverteilung der Nachrichten $m_1, \ldots, m_l$ ergibt sich für jede feste Nachricht $m \in \mathbf{Z}_a^p$:

$$\Pr\ (M = m) = a^{-p}$$

und

$$\Pr\ (M \in U) = a^{s-p}\ ,$$

da die Anzahl der Elemente in $\mathbf{Z}_a^p$ gleich a^p ist und jeder s-dimensionale lineare Unterraum von $\mathbf{Z}_a^p$ genau a^s Elemente enthält.

Als nächstes betrachten wir die Zufallsvariable $R_{p,s,a}$, welche die Anzahl der Versuche (zufälliger Nachrichten) angibt, bis zum ersten Mal $M \notin U$ ist. Aufgrund der Gleichverteilung der Nachrichten hängt der Wert von $R_{p,s,a}$ nicht von der speziellen Wahl des Unterraumes U ab:

$$\Pr\ (R_{p,s,a} = t) = (a^{s-p})^{t-1} \cdot (1 - a^{s-p})\ ,$$

da in den ersten $t-1$ Versuchen jeweils $M \in U$ ist und dann $M \notin U$. Als Erwartungswert für die Zufallsvariable $R_{p,s,a}$ erhalten wir

$$\begin{aligned}
E(R_{p,s,a}) &= \sum_{t=1}^{\infty} t \cdot (a^{s-p})^{t-1} \cdot (1 - a^{s-p}) \\
&= (1 - a^{s-p}) \cdot \sum_{t=0}^{\infty} (a^{s-p})^t \cdot (t+1) \\
&= (1 - a^{s-p}) \cdot \sum_{t=0}^{\infty} \left((a^{s-p})^{t+1} \right)' \qquad \text{(Ableiten nach } x := a^{s-p}) \\
&= (1 - a^{s-p}) \cdot \left(1 + \sum_{t=0}^{\infty} (a^{s-p})^{t+1} \right)' \\
&= (1 - a^{s-p}) \cdot \left(\frac{1}{1 - a^{s-p}} \right)' \\
&= (1 - a^{s-p}) \cdot \frac{1}{(1 - a^{s-p})^2} \\
&= \frac{1}{1 - a^{s-p}}\ ,
\end{aligned}$$

also ist $E(R_{p,s,a})$ gleich dem Kehrwert der Wahrscheinlichkeit des Ereignisses, bei einmaligem Ziehen einer Nachricht m nicht in dem Unterraum U zu landen, was ja auch nicht anders zu erwarten war. Für $s = 0$, also $U = \{0\}$, ist die erwartete Anzahl Versuche, bis die erste Nachricht ungleich dem Nullvektor ist, gleich $E(R_{p,0,a}) = 1/(1-a^{-p})$. Als nächstes betrachten wir die zu erwartende Anzahl $E(R_{p,a})$ an Versuchen, um einen p-dimensionalen Raum U (also den ganzen Raum $\mathbf{Z}_a^p$) zu erzeugen. Dazu müssen zuerst $1, 2, \ldots, p-1$-dimensionale Räume erzeugt werden, also

$$\begin{aligned}
E(R_{p,a}) &= \sum_{s=0}^{p-1} E(R_{p,s,a}) \\
&= \sum_{s=0}^{p-1} \frac{1}{1-a^{s-p}} \\
&\leq \sum_{s=0}^{p-1} 1 + 2 \cdot a^{s-p} \quad (\text{da } \frac{1}{1-x} \leq 1 + 2 \cdot x \text{ für } 0 \leq x \leq {}^1/_2) \\
&= p + 2 \cdot a^{-p} \sum_{s=0}^{p-1} a^s \\
&= p + 2 \cdot a^{-p} \frac{a^p - 1}{a-1} \\
&\leq p + \frac{2}{a-1} \, .
\end{aligned}$$

Man benötigt also wenigstens p, aber im Mittel höchstens $p + 2/(a-1)$ Klartext/Kryptogramm-Paare um die Matrix k ermitteln zu können. Speziell in unserem Beispiel ($p = 3, a = 26$) ergibt sich:

$$E(R_{3,26}) = \frac{1}{1-26^{-3}} + \frac{1}{1-26^{-2}} + \frac{1}{1-26^{-1}} \approx 3,042 \, .$$

Es reichen also 4 Tripel, um eine „gute“ Chance zu haben (siehe unten), die Matrix k zu ermitteln. Diese Chance lässt sich mit der Markov-Ungleichung abschätzen:

Theorem 3.2 (Markov Ungleichung) *Für jede nicht-negative Zufallsvariable X gilt für jedes $t > 0$:*

$$Pr(X \geq t) \leq {}^{E(X)}/_t \, .$$

Für $t = 5$ ist in unserem Beispiel die Wahrscheinlichkeit, dass man 5 oder mehr Tripel benötigt, höchstens ${}^{3.042}/_5 \approx 0.61$.

3.2 Mehralphabet-Substitutionschiffren

Mehralphabet-Substitutionschiffren liegen der Idee zu Grunde, unterschiedliche Substitutionen auf einzelne Teile einer Nachricht anzuwenden.

Definition 3.3 *Eine einfache Vigenère-Chiffre der Periode r ist eine Blockchiffre mit $A = A' = Z_a = \{0, 1, \ldots, a-1\}$ und Schlüsselmenge $K = \{0, 1, \ldots, a-1\}^r$. Für einen Schlüssel $k = (k_1, \ldots, k_r) \in K$ und eine Nachricht $(m_1, \ldots, m_r) \in A^r$ ist*

$$E_k(m_1, \ldots, m_r) = ((m_1 + k_1) \bmod a, \ldots, (m_r + k_r) \bmod a) .$$

Also ist eine einfache Vigenère-Chiffre der Periode r ein Produkt von r Cäsar-Chiffren.

Definition 3.4 *Eine verallgemeinerte Vigenère-Chiffre der Periode r ist wie eine einfache Vigenère-Chiffre definiert, nur ist*

$$K = \{(\sigma_1, \ldots, \sigma_r) \mid \sigma_i \colon A \longrightarrow A \text{ Permutation}, i = 1, \ldots, r\} .$$

Für einen Schlüssel $k = (\sigma_1, \ldots, \sigma_r) \in K$ ist

$$E_k(m_1, \ldots, m_r) = (\sigma_1(m_1), \ldots, \sigma_r(m_r)) ,$$

also das Chiffrat ein Produkt von r einfachen Substitutionschiffren.

Ist die Periode r bekannt, so ist die Kryptoanalyse einfach, da $m_i, m_{i+r}, m_{i+2r}, \ldots$ gleich verschlüsselt werden und somit wie bei einfachen Substitutionschiffren vorgegangen werden kann. Ist r hingegen unbekannt, so versucht man, mittels statistischer Methoden (z.B. Koinzidenzmatrix), diesen Wert zu finden.

4 Perfekte Sicherheit

Wir behandeln kurz grundlegende Definitionen der Wahrscheinlichkeitsrechnung.
Gegeben sei eine endliche Menge $S \neq \emptyset$. Die Teilmengen $E \subseteq S$ heißen *Ereignisse* und die einelementigen Teilmengen $\{s\}$, $s \in S$, heißen *Elementarereignisse*. Die Menge $\mathcal{P}(\mathcal{S})$ aller Teilmengen von S heißt *Potenzmenge* von S.
Eine *Wahrscheinlichkeitsverteilung* auf S ist eine Abbildung $\Pr : \mathcal{P}(\mathcal{S}) \longrightarrow [\prime, \infty]$ mit

(i) $\Pr(S) = 1$

(ii) $\Pr(E \cup E') = \Pr(E) + Pr(E')$ für alle $E, E' \subseteq S$ mit $E \cap E' = \emptyset$.

Damit folgt sofort $\Pr(\emptyset) = 0$ und $\Pr(S \backslash E) = 1 - \Pr(E)$. Das Ereignis $\overline{E} = S \backslash E$ heißt auch *Komplementärereignis* zu E. Weiter gilt $\Pr(E) = \sum_{s \in E} \Pr(\{s\})$. Wesentlich ist der Begriff der *bedingten Wahrscheinlichkeit*, definiert als

$$\Pr(A|B) := \frac{\Pr(A \cap B)}{\Pr(B)}$$

für Ereignisse $A, B \subseteq S$ mit $\Pr(B) > 0$. Hierbei ist $\Pr(A|B)$ die Wahrscheinlichkeit für das Ereignis A, gegeben, dass Ereignis B vorliegt.
Zwei Ereignisse $E, E' \subseteq S$ heißen *unabhängig*, falls gilt $\Pr(E \cap E') = \Pr(E) \cdot \Pr(E')$.

Beispiel 4.1 *(a) Angenommen, man weiß für einen idealen 6er-Würfel, dass eine gerade Zahl geworfen wurde, also $B = \{2, 4, 6\}$. Gefragt ist nach der Wahrscheinlichkeit für das Ereignis A mehr als 3 Augen zu haben, also $A = \{4, 5, 6\}$. Dann ist*

$$Pr(A|B) = \frac{Pr(A \cap B)}{Pr(B)} = \frac{{}^2/_6}{{}^1/_2} = {}^2/_3 \, .$$

(b) Die Wahrscheinlichkeit, bei t-maligen Würfeln bei den ersten $(t-1)$ Versuchen nie eine 6 und dann beim t'ten Versuch eine 6 zu würfeln, ist gleich $({}^5/_6)^{t-1} \cdot {}^1/_6$, da die einzelnen Würfe unabhängig voneinander sind.

Erwähnt sei hier das Theorem von Bayes:

Theorem 4.2 *Seien $A, B \subseteq S$ Ereignisse mit $Pr(A), Pr(B) > 0$. Dann gilt*

$$Pr(A) \cdot Pr(B|A) = Pr(B) \cdot Pr(A|B) \, .$$

Beweis: Einsetzen der Definition der bedingten Wahrscheinlichkeit. □

4.1 Perfekte Sicherheit

Shannons Idee bei perfekt sicheren Kryptosystemen war, dass ein Angreifer Oskar aus einem empfangenem Chiffrat c keinerlei Rückschlüsse auf den zugehörigen Klartext ziehen kann.

Wir nehmen an, dass die Klartexte $m \in M$ gemäß einer Wahrscheinlichkeitsverteilung $\Pr_M(m)$ auftreten. Analoges gilt für die verwendeten Schlüssel gemäß einer Wahrscheinlichkeitsverteilung $\Pr_K(k)$. Dann gilt für den Empfang eines bestimmten Chiffrats $c \in C$:

$$\Pr_C(c) = \sum_{(m,k);E(m,k)=c} \Pr_M(m) \cdot \Pr_K(k) \ ,$$

da Klartext und Schlüssel unabhängig voneinander gewählt werden, wobei $E(\cdot,\cdot)$ die Chiffrierfunktion ist.

Definition 4.3 *Das eben vorgestellte Kryptosystem heißt perfekt sicher, wenn für alle Paare* $(m,c) \in M \times C$ *mit* $Pr\,(c) > 0$ *gilt* $Pr\,(m|c) = Pr(m)$.

In perfekt sicheren Systemen ist also bei Vorliegen des Chiffrats $c \in C$ jeder zugehörige Klartext gleichwahrscheinlich, dass heißt, der Gegner erhält aus $c \in C$ keine Information über den zugehörigen Klartext.

Beispiel 4.4 *Sei* $M = \{0,1\}$, $Pr_M(0) = {}^2/_3$ *und* $Pr_M(1) = {}^1/_3$. *Weiter sei* $K = \{k_1, k_2\}$ *mit* $Pr_K(k_1) = {}^1/_4$ *und* $Pr_K(k_2) = {}^3/_4$, *sowie* $C = \{r,s\}$. *Für die Verschlüsselungsfunktion E gelte*

$$E(0,k_1) = r \qquad E(1,k_1) = s$$
$$E(0,k_2) = s \qquad E(1,k_2) = r$$

Dann gilt $Pr_C(r) = {}^2/_3 \cdot {}^1/_4 + {}^1/_3 \cdot {}^3/_4 = {}^5/_{12}$ *sowie* $Pr_C(s) = {}^7/_{12}$. *Weiter ist* $Pr\,(0|r) = \frac{{}^2/_3 \cdot {}^1/_4}{{}^5/_{12}} = {}^2/_5$, $Pr\,(1|r) = \frac{{}^1/_3 \cdot {}^3/_4}{{}^5/_{12}} = {}^3/_5$, *sowie* $Pr\,(0|s) = \frac{{}^2/_3 \cdot {}^3/_4}{{}^7/_{12}} = {}^6/_7$ *und* $Pr\,(1|s) = {}^1/_7$. *Das System ist also nicht perfekt sicher.*

Theorem 4.5 *Ein perfekt sicheres kryptographisches System hat mindestens soviele Schlüssel wie es Klartexte mit positiver Wahrscheinlichkeit hat.*

Beweis: Sei $c \in C$ ein Chiffrat mit $Pr_C(c) > 0$, und $m \in M$ ein Klartext mit $\Pr_M(m) > 0$. Dann gilt wegen der perfekten Sicherheit $\Pr\,(m|c) = \Pr\,(m) > 0$, also existiert ein Schlüssel $k \in K$ mit $E(m,k) = c$. Wegen der Injektivität von $E(\cdot,k)$ sind die gefundenen Schlüssel paarweise verschieden. □

Theorem 4.6 *Sei $|C| = |K|$ und $Pr_{\ M}(m) > 0$ für jeden Klartext $m \in M$. Das Kryptosystem ist perfekt sicher genau dann, wenn die Wahrscheinlichkeitsverteilung $Pr_{\ K}(\cdot)$ auf der Schlüsselmenge K die Gleichverteilung ist und für jedes Paar $(m, c) \in M \times C$ genau ein Schlüssel $k \in K$ mit $E(m, k) = c$ existiert.*

Beweis: Angenommen, unser System hat perfekte Sicherheit. Dann gibt es zu jedem Klartext $m \in M$ und für jedes Chiffrat $c \in C$ einen Schlüssel $k \in K$ mit $E(m, k) = c$, da ansonsten $\Pr(m) \neq 0 = \Pr(m|c)$ gelten würde. Mit $|K| = |C|$ folgt die Eindeutigkeit des Schlüssels k.
Sei nun ein Chiffrat $c \in C$ gegeben. Für einen beliebigen Klartext $m \in M$ sei $k(m)$ der Schlüssel mit $E(m, k(m)) = c$. Dann gilt mit Theorem 4.2:

$$\Pr(m|c) = \frac{\Pr(c|m) \cdot \Pr(m)}{\Pr(c)} = \frac{\Pr(k(m)) \cdot \Pr(m)}{\Pr(c)}$$

Mit $\Pr(m|c) = \Pr(m)$ folgt sofort

$$\Pr(k(m)) = \Pr(c),$$

also ist $\Pr(k(m))$ unabhängig von m. Da für jeden Schlüssel k ein Klartext $m \in M$ existiert mit $k = k(m)$, folgt Gleichverteilung auf K.
Angenommen nun, auf K liege Gleichverteilung vor und für alle Paare $(m, c) \in M \times C$ gibt es genau einen Schlüssel $k = k(m, c) \in K$ mit $E(m, k) = c$. Dann gilt

$$\begin{aligned} \Pr(m|c) = \frac{\Pr(m) \cdot \Pr(c|m)}{\Pr(c)} &= \frac{\Pr(m) \cdot \Pr(k)}{\sum_{m' \in M} \Pr(m') \cdot \Pr(k(m', c))} \\ &= \frac{\Pr(m)}{\sum_{m' \in M} \Pr(m')} \\ &= \Pr(m)\,, \end{aligned}$$

also folgt perfekte Sicherheit. □

5 DES - Data Encryption Standard

DES wurde von IBM im Auftrag der US-Regierung entwickelt. Dieser wurde dann 1977 zum Verschlüsselungsstandard für nicht geheime Nachrichten durch das National Bureau of Standards

(NBS). DES stand in der Kritik, da die Kriterien des Designs geheimgehalten wurden, und da seine Schlüssellänge (56) zu kurz ist. Trotzdem hat sich DES weitgehend durchgesetzt.
Es existieren für diesen Standard schnelle Hardwareimplementierungen. Nach Kerkhoff's Prinzip wurden alle Einzelheiten veröffentlicht. DES ist ein Produkt von Blockchiffren.
Die Klartext-/Kryptogrammmenge ist

$$C = M = \{0,1\}^{64}$$

und die Schlüsselmenge ist

$$K \subseteq \{0,1\}^{64} \qquad \text{mit}$$
$$K = \{(k_1, \ldots, k_{64}) \mid k_1 + \ldots + k_8 \equiv k_9 + \ldots + k_{16} \equiv \ldots \equiv k_{57} + \ldots + k_{64} \equiv 0 \bmod 2\} .$$

Die Anzahl der DES-Schlüssel ist damit $|K| = 2^{56} \approx 7,2 \cdot 10^{16}$, da jedes achte Bit fixiert ist. Dies bedeutet, dass pro Byte ein *Paritätsbit* vorhanden ist, womit eine Fehlerentdeckung möglich wird. Eine Fehlerkorektur ist damit aber nicht realisierbar.

Feistel-Chiffre

Eine Feistel-Chiffre ist eine Blockchiffre mit der Blocklänge $2 \cdot t$. Gegeben ist eine Blockchiffre durch die Verschlüsselungsfunktion $\widehat{E}\colon \{0,1\}^t \times K \longrightarrow \{0,1\}^t$. Nun fixiert man eine Anzahl $r \geq 1$ von Runden und erzeugt zu einem gegebenen *Initialschlüssel* $k \in K$ eine Folge $k_1, k_2, \ldots, k_r$ von *Rundenschlüsseln*.
Für eine Nachricht $m \in \{0,1\}^{2t}$ mit $m = (m_1, \ldots, m_{2t})$ sei $m_L = (m_1, \ldots, m_t)$ bzw. $m_R = (m_{t+1}, \ldots, m_{2t})$ ihre linke bzw. rechte Hälfte. Startend mit $m^{(0)} := m \in \{0,1\}^{2t}$ wird eine Folge $m^{(1)}, \ldots, m^{(r)} \in \{0,1\}^{2t}$ wie folgt konstruiert. Für $i = 1, \ldots, r$ setzt man :

$$m^{(i)} = (m_L^{(i)}, m_R^{(i)}) = (m_R^{(i-1)}, m_L^{(i-1)} \oplus \widehat{E}(m_R^{(i-1)}, k_i)) .$$

Die Chiffrierfunktion $E\colon \{0,1\}^{2t} \times K \longrightarrow \{0,1\}^{2t}$ ist dann definiert durch

$$E(m,k) = E(m^{(0)}, k) = (m_R^{(r)}, m_L^{(r)}) .$$

Die Dechiffrierung erfolgt dann gemäß folgender Rekursion

$$(m_R^{(i-1)}, m_L^{(i-1)}) = (m_L^{(i)}, m_R^{(i)} \oplus \widehat{E}(m_L^{(i)}, k_i))$$

für $i = r, \ldots, 1$.

π								π^{-1}							
58	50	42	34	26	18	10	2	40	8	48	16	56	24	64	32
60	52	44	36	28	20	12	4	39	7	47	15	55	23	63	31
62	54	46	38	30	22	14	6	38	6	46	14	54	22	62	30
64	56	48	40	32	24	16	8	37	5	45	13	53	21	61	29
57	49	41	33	25	17	9	1	36	4	44	12	52	20	60	28
59	51	43	35	27	19	11	3	35	3	43	11	51	19	59	27
61	53	45	37	29	21	13	5	34	2	42	10	50	18	58	26
63	55	47	39	31	23	15	7	33	1	41	9	49	17	57	25

Tabelle 1: Initialpermutation π und ihre inverse π^{-1}.

Bei DES bildet man zum Klartext $m = (m_1, \ldots, m_{64}) \in \{0,1\}^{64}$ nun mittels einer Initialpermutation $\pi \in S_{64}$ (siehe Tabelle 1) die Folge

$$m^{(0)} := (m_{\pi(1)}, \ldots, m_{\pi(64)}) \, .$$

Bei der Initialpermutation π, dargestellt mit einer 8×8-Matrix in Tabelle 1, stehen in Zeile $i \in \{1,2,3,4\}$ alle $x \in \{1, \ldots, 64\}$ mit $x \equiv 2i \bmod 8$. In Zeile $i \in \{5,6,7,8\}$ stehen dann alle $x \in \{1, \ldots, 64\}$ mit $x \equiv 2i - 1 \bmod 8$. An Position $(i,j), i \leq 4$, steht $2i + (8-j) \cdot 8$ und an Position $(i,j), i > 4$, steht $2i - 9 + (8-j) \cdot 8$.

Also gilt $\pi(m_1, \ldots, m_{64}) = (m_{58}, m_{50}, \ldots, m_7)$. Die Initialpermutation π ist kryptographisch nicht relevant.

Man wendet nun eine 16-Runden Feistel-Chiffre an, daraus ergibt sich dann $m^{(16)}$. Das dazugehörige Kryptogramm ist dann $c = \pi^{(-1)}(m_R^{(16)}, m_L^{(16)})$. Diese Feistel-Chiffre hat eine Blocklänge von $2t = 64$. Für die interne Blockchiffre $\widehat{E}$ gilt: $\widehat{E}\colon \{0,1\}^{32} \times K \longrightarrow \{0,1\}^{32}$ mit $K \subseteq \{0,1\}^{48}$.

Wir beschreiben nun die interne Blockchiffre $\widehat{E}\colon \{0,1\}^{32} \times K \longrightarrow \{0,1\}^{32}$. Ausgehend von einer Eingabe $(m_1, \ldots, m_{32}) \in \{0,1\}^{32}$ wird eine Expansion (siehe Tabelle 2) durgeführt. Die Expansionsfunktion $Exp\colon \{0,1\}^{32} \longrightarrow \{0,1\}^{48}$ verlängert die Eingabe der Länge 32 auf Länge 48 mit

$$Exp(m_1, \ldots, m_{32}) = (m_{32}, m_1, m_2, \ldots, m_1) \in \{0,1\}^{48} \, ,$$

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

Tabelle 2: Funktionen Exp und σ

wobei die Bits $4i, 4i+1$ für $i = 1, \ldots, 7$ sowie die Bits 32 und 1 der Ausgangsfolge $(m_1, \ldots, m_{32})$ dupliziert werden, vergleiche Tabelle 2.

Zu dem Rundenschlüssel $k_j = (k_1, \ldots, k_{48}) \in \{0,1\}^{48}$ in Runde j bildet man

$$Exp(m_1, \ldots, m_{32}) \oplus k_j = B_1, B_2, \ldots, B_8 \ ,$$

wobei jeder Block B_i mit $i = 1, \ldots, 8$ eine Länge von 6 hat. Auf diese Blöcke werden die sogenannten S-Boxen angewandt.

S-Boxen: Die S-Boxen werden durch Abbildungen

$$S_i \colon \{0,1\}^6 \to \{0,1\}^4$$

beschrieben, $i = 1, \ldots, 8$. Sie beschreiben hochgradig nichtlineare Abbildungen.

Sei $B_i = (b_1, \ldots, b_6)$ ein Block der Länge 6. Man betrachtet die Bits b_1, b_6 als Binärzahl $x = 2 \cdot b_1 + b_6$ und auch b_2, b_3, b_4, b_5 als Binärzahl $y = 8 \cdot b_2 + 4 \cdot b_3 + 2 \cdot b_4 + b_5$. Betrachtet man nun den Eintrag $S_i(x, y)$ in Binärdarstellung der Länge 4, so ist dies $S_i(B_i)$ (siehe Tabelle 3). Aus der Folge $(m_1, \ldots, m_{32})$ wurde dann die Folge $(S_1(B_1), \ldots, S_8(B_8)) \in \{0,1\}^{32}$. Auf diese Folge wird dann noch die Permutation $\sigma \in S_{32}$ angewandt.

Nun sollen die Rundenschlüssel $k_1, \ldots, k_{16} \in \{0,1\}^{48}$ mit Hilfe des DES-Schlüssels $k \in \{0,1\}^{64}$ berechnet werden. Hierzu verwendet man die Funktion $PC_1 \colon \{0,1\}^{64} \longrightarrow \{0,1\}^{28} \times \{0,1\}^{28}$ mit

$$PC_1(k) = \widehat{k} = (\widehat{k}_L, \widehat{k}_R) = (\widehat{k}_L^{(0)}, \widehat{k}_R^{(0)}) \ .$$

		[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
	[0]	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
S_1	[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
	[0]	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
S_2	[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
	[0]	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
S_3	[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
	[0]	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
S_4	[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
	[0]	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
S_5	[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
	[0]	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
S_6	[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
	[0]	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
S_7	[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	3	12
	[0]	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
S_8	[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 3: S-Boxen $S_1, \ldots, S_8$

PC_1							PC_2					
57	49	41	33	25	17	9	14	17	11	24	1	5
1	58	50	42	34	26	18	3	28	15	6	21	10
10	2	59	51	43	35	27	23	19	12	4	26	8
19	11	3	60	52	44	36	16	7	27	20	13	2
63	55	47	39	31	23	15	41	52	31	37	47	55
7	62	54	46	38	30	22	30	40	51	45	33	48
14	6	61	53	45	37	29	44	49	39	56	34	53
21	13	5	28	20	12	4	46	42	50	36	29	32

Tabelle 4: Die Funktionen PC_1(laut Definition sind die Prüfbits $8, 16, \ldots$ nicht vorhanden) und PC_2.

Für $i = 1, 2, \ldots, 16$ setzt man

$$w_i = \begin{cases} 1 & \text{für } i \in \{1, 2, 9, 16\} \\ 2 & \text{sonst.} \end{cases}$$

(*i*) $\widehat{k}_L^{(i)}$ entsteht aus $\widehat{k}_L^{(i-1)}$ durch *Linksrotation* um w_i Positionen

(*ii*) $\widehat{k}_R^{(i)}$ entsteht aus $\widehat{k}_R^{(i-1)}$ durch Linksrotation um w_i Positionen

(*iii*) $\widehat{k}^{(i)} := PC_2(\widehat{k}_L^{(i)}, \widehat{k}_R^{(i)}) \in \{0,1\}^{48}$, aus den 56 Bits werden also 48 Bits mittels der Funktion PC_2 ausgewählt.

Beispiel 5.1 *Im Hexadezimalsystem sei die folgende Nachricht gegeben:*

$$m = ABCDEF0123456789$$

also gilt etwa $A = 1010$ *und* $F \widehat{=} 1111$.

m							
1	0	1	0	1	0	1	1
1	1	0	0	1	1	0	1
1	1	1	0	1	1	1	1
0	0	0	0	0	0	0	1
0	0	1	0	0	0	1	1
0	1	0	0	0	1	0	1
0	1	1	0	0	1	1	1
1	0	0	0	1	0	0	1

$\pi(m)$							
0	1	1	0		0	1	1	0
0	0	0	0		0	0	0	0
0	1	1	0		0	1	1	0
1	1	1	1		1	1	1	1
1	0	0	0		0	1	1	1
0	1	0	1		0	1	0	1
1	0	0	0		0	1	1	1
0	1	0	1		0	1	0	1

DES-Schlüssel (90D8CFF5ACC0186F)

0	1	0	1	0	0	0	0
1	1	1	0	1	0	0	0
1	1	0	0	1	1	1	1
1	1	1	1	0	1	0	1
1	0	1	0	1	1	0	0
1	1	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	1	1	0	1	1	1	1

$(\widehat{k}_L^{(0)}, \widehat{k}_R^{(0)})$

$PC_1(k)$

0	*0*	*1*	*1*	*1*	*1*	*1*		*0*	*1*	*1*	*1*	*1*	*1*	*0*
0	*1*	*0*	*1*	*0*	*1*	*1*	$\widehat{k}_L^{(1)}$	*1*	*0*	*1*	*0*	*1*	*1*	*1*
1	*1*	*1*	*0*	*0*	*1*	*1*		*1*	*1*	*0*	*0*	*1*	*1*	*0*
0	*1*	*0*	*0*	*1*	*0*	*0*		*1*	*0*	*0*	*1*	*0*	*0*	*0*
1	*0*	*0*	*0*	*0*	*1*	*0*		*0*	*0*	*0*	*0*	*1*	*0*	*0*
0	*1*	*0*	*0*	*1*	*1*	*1*	$\widehat{k}_R^{(1)}$	*1*	*0*	*0*	*1*	*1*	*1*	*0*
0	*0*	*1*	*1*	*0*	*1*	*0*		*0*	*1*	*1*	*0*	*1*	*0*	*1*
1	*1*	*0*	*1*	*0*	*0*	*1*		*1*	*0*	*1*	*0*	*0*	*1*	*1*

PC$_2$ *(... Hier unvollständig!!)*

$PC_2(\widehat{k}_L^{(1)}, \widehat{k}_R^{(1)})$ *liefert dann den Rundenschlüssel* k_1*:*

1	*0*	*0*	*0*	*0*	*1*
1	*0*	*1*	*1*	*0*	*1*
0	*1*	*1*	*1*	*0*	*1*
1	*0*	*0*	*1*	*1*	*1*
1	*1*	*0*	*0*	*1*	*1*
0	*1*	*0*	*1*	*1*	*0*
1	*1*	*1*	*1*	*0*	*0*
0	*0*	*1*	*1*	*0*	*0*

Der Leser möge selbst $Exp(m_1, \ldots, m_{48}) \oplus k_1$ berechnen sowie die S-Boxen anwenden.

Nach 5 Iterationen gilt folgendes:

Jedes Bit der Blöcke (L, R) hängt von jedem Bit der Eingabe und des Schlüssels ab. Hierfür ist besonders die Expansionsfunktion *Exp* verantwortlich.

Es gibt vier schwache Schlüssel (d.h. $k_i = k_j \ \forall i, j$), links unten dargestellt im Hexadezimalsystem.

64-Bit Schlüssel mit Paritätsbits	56-Bit Schlüssel
`0101 0101 0101 0101`	`0000000 0000000`
`1F1F 1F1F 0E0E 0E0E`	`0000000 FFFFFFF`
`E0E0 E0E0 F1F1 F1F1`	`FFFFFFF 0000000`
`FEFE FEFE FEFE FEFE`	`FFFFFFF FFFFFFF`

6 Das RSA-System

Bisher wurde immer vorausgesetzt, dass ein sicherer Kanal zum Austausch des geheimen Schlüssels zwischen beiden Parteien, etwa dem Sender A und dem Empfänger B existiert. Ein vermeintlich sicherer Kanal kann natürlich ein großer Unsicherheitsfaktor sein.
Ein weiteres Problem ist, dass bisher der geheime Schlüssel nur den beiden Teilnehmern A und B bekannt war. Ein dritter, etwa C, konnte keine Nachricht an B verschlüsselt übermitteln, die auch von B hätte korrekt entschlüsselt werden können.
Diffie und Hellman schlugen daher 1976 vor, Systeme zu suchen bzw. zu entwickeln, bei denen ein Teil des Schlüssels *öffentlich* ist (C kann an B senden, wenn ein Teil von B's Schlüssel öffentlich ist) und der andere Teil des Schlüssels *privat und geheim* ist. Derartige Kryptosysteme werden *Public-Key-Systeme* (oder auch *asymmetrische Kryptosysteme*) genannt.
Ist dann der öffentliche Schlüssel von B allgemein bekannt, so kann jeder an B Nachrichten (verschlüsselt) senden! Es ist dann kein sicherer Kanal zur Schlüsselübergabe mehr notwendig.
Nach einer Idee von Rivest, Shamir und Adleman entstand das *RSA-System*, benannt nach den Anfangsbuchstaben der Namen der drei Erfinder.
Wir nehmen im folgenden an, dass der Klartext eine Folge von Bits ist. Diese Folge ist in Blöcke gleicher Länge unterteilt und jeder Block, interpretiert als natürliche Zahl, wird einzeln verschlüsselt.
Sei $\mathbf{N} = \{0, 1, 2, \dots\}$ die Menge der natürlichen Zahlen.
Mit ggT (m, n) bezeichnen wir den *größten gemeinsamen Teiler* der natürlichen Zahlen m und n, also die größte natürliche Zahl, die sowohl m als auch n ohne Rest teilt. Es gilt ggT $(0, n) = n$ und ggT $(n, 1) = 1$ sowie ggT $(m, n) =$ ggT $(m, n - l \cdot m)$.
Primzahlen sind alle diejenigen natürlichen Zahlen, die nur durch 1 oder sich selbst (ohne Rest) teilbar sind.
Mit $\mathbf{Z}_n$ sei wie zuvor der *Restklassenring modulo* n bezeichnet. Dann ist

$$\mathbf{Z}_n^* := \{x \in \mathbf{Z}_n \setminus \{0\} \mid \text{ggT } (x, n) = 1\}$$

die Menge der Zahlen aus $\{1, 2, \dots, n-1\}$, die zu n *teilerfremd* bzw. *prim* sind. Hierbei sollte $\mathbf{Z}_n^*$ nicht mit der Menge aller endlichen Folgen aus $\mathbf{Z}_n$ identifiziert werden.
Man kann leicht zeigen, dass die Menge $\mathbf{Z}_n^*$ hinsichtlich der üblichen Multiplikation in $\mathbf{Z}_n$ eine Gruppe bildet.

Die Funktionsweise des RSA-Kryptosystems lässt sich nun wie folgt beschreiben.
Sei $x \in \mathbf{Z}_n^*$ eine Nachricht, die verschlüsselt von A an B gesandt werden soll. Die Nachricht $x \in \mathbf{Z}_n^*$ ist in natürlicher Weise (etwa mit der Binärdarstellung) aus einem $0,1$-Klartextblock der Länge maximal $l = \lfloor \log n \rfloor$ entstanden.

Setup für das RSA-System

(a) B erzeugt zwei große Primzahlen p und q mit $p \neq q$.

(b) B berechnet die Produkte $n := p \cdot q$ sowie $\phi(n) := (p-1) \cdot (q-1)$.

(Der Parameter $\phi(n)$ (*Eulersche ϕ-Funktion*) zählt die Anzahl aller natürlichen Zahlen $x < n$ mit ggT $(x, n) = 1$, also ist $|\mathbf{Z}_n^*| = \phi(n)$.)

(c) B erzeugt eine natürliche Zahl e mit $1 < e < \phi(n)$ und ggT $(e, \phi(n)) = 1$.

(d) B berechnet die (eindeutige) Zahl $d \in \mathbf{Z}_{\phi(n)}$ mit $d \cdot e \equiv 1 \bmod \phi(n)$, also ist $d \equiv e^{-1} \bmod \phi(n)$ das multiplikativ Inverse von e in $\mathbf{Z}_{\phi(n)}^*$.

(e) B veröffentlicht die Zahlen n und e. Dann ist das Paar $k = (n, e)$ der *öffentliche Schlüssel* und das Paar $k' = (\phi(n), d)$ der *private Schlüssel* von B.

Man setzt hier voraus, dass im Setup für das RSA-System alle Schritte (a) bis (e) in Polynomialzeit durchführbar sind (effizient), also bei Eingabe von n in Binärdarstellung (mit $\lceil \log(n+1) \rceil$ Bits) in Zeit $O(\text{poly}(\log n))$. Die effiziente Durchführung der einzelnen Schritte werden wir noch erläutern.

Funktionsweise des RSA-Systems

- A möchte an B eine Nachricht $x \in \mathbf{Z}_n^*$ senden. Dazu nimmt er den öffentlichen Schlüssel $k = (n, e)$ von B, berechnet

$$E(x, k) := x^e \bmod n$$

und sendet das Ergebnis an B.

Diese Verschlüsselung soll auch effizient, das heißt in Polynomialzeit $O(\text{poly}(\log n))$, durchführbar sein.

- Empfängt B die Nachricht $c \in \mathbf{Z}_n^*$ ($\equiv x^e \bmod n$) von A, so nimmt er seinen geheimen privaten Schlüssel $k' = (\phi(n), d)$ und berechnet

$$D(c, k') := c^d \bmod n$$

und erhält die gesendete Nachricht $x \in \mathbf{Z}_n^*$ (Dieses müssen wir natürlich noch verifizieren.).

Wir weisen zunächst die Korrektheit des RSA-Verfahrens nach.

Korrektheit:

Beweis: Wir wollen zeigen, dass für alle $x \in \mathbf{Z}_n^*$ gilt:

$$D(E(x, k), k') \equiv x \bmod n\,. \tag{1}$$

Hierzu benötigen wir den *Satz von Euler Theorem 6.1*:

Theorem 6.1 *Für jedes $a \in \mathbf{Z}_n^*$ gilt:*

$$a^{\phi(n)} \equiv 1 \bmod n\,.$$

Beweis: Mit $\mathbf{Z}_n^* = \{x_1, \ldots, x_{\phi(n)}\}$ gilt auch $\mathbf{Z}_n^* = \{a \cdot x_1, \ldots, a \cdot x_{\phi(n)}\}$ für $a \in \mathbf{Z}_n^*$, da $\mathbf{Z}_n^*$ bezüglich der Multiplikation eine Gruppe ist wegen ggT $(a \cdot x_i, n) = 1$, $i = 1, \ldots, \phi(n)$, also abgeschlossen ist, das heißt $a \cdot x_i \in \mathbf{Z}_n^*$ für $i = 1, \ldots, \phi(n)$. Darüber hinaus folgt aus $a \cdot x_i \equiv a \cdot x_j \bmod n$ sofort $a \cdot (x_i - x_j) \equiv 0 \bmod n$ und mit ggT $(a, n) = 1$ erhält man $x_i \equiv x_j \bmod n$.
Mit $\{x_1, \ldots, x_{\phi(n)}\} = \{a \cdot x_1, \ldots, a \cdot x_{\phi(n)}\}$ erhalten wir dann

$$\prod_{i=1}^{\phi(n)} x_i \equiv \prod_{i=1}^{\phi(n)} (a \cdot x_i) \equiv a^{\phi(n)} \cdot \prod_{i=1}^{\phi(n)} x_i \bmod n\,.$$

Da ggT $(x_i, n) = 1$ für $i = 1, \ldots, \phi(n)$ gilt, ist auch ggT $(\prod_{i=1}^{\phi(n)} x_i, n) = 1$ und es folgt die Behauptung $a^{\phi(n)} \equiv 1 \bmod n$. □
Wir erhalten als Korollar den *Satz von Fermat*.

Korollar 6.2 *Für Primzahlen p und $a \in \mathbf{Z}_p \setminus \{0\}$ gilt*

$$a^{p-1} \equiv 1 \bmod p\,.$$

Beweis: Für Primzahlen p ist $\phi(p) = p - 1$ die Anzahl aller zu p teilerfremden natürlichen Zahlen $x < p$. Die Behauptung folgt dann mit dem Satz von Euler Theorem 6.1. □

Korollar 6.3 *Für $a \in \mathbf{Z}_n^*$, $d \in \mathbf{Z}_{\phi(n)}$ und $i \in \mathbf{Z}$ gilt:*

$$a^{d+i\cdot\phi(n)} \equiv a^d \bmod n \; .$$

Beweis: Wir verwenden die in $\mathbf{Z}_n^*$ geltenden Potenzgesetze:

$$a^{d+i\cdot\phi(n)} \equiv a^d \cdot a^{i\cdot\phi(n)} \equiv a^d \cdot \left(a^{\phi(n)}\right)^i \equiv a^d \cdot 1^i \equiv a^d \bmod n \; .$$

□

Um eine Kongruenz $a^x \equiv a^y \bmod n$ für $a \in \mathbf{Z}_n^*$ zu lösen, kann man daher $x \equiv y \bmod \phi(n)$ betrachten.

Wir sehen nun mit dem Satz von Euler Theorem 6.1 und Korollar 6.3 die Korrektheit (1) des RSA-Verfahrens wie folgt ein:

$$D(E(x,k),k') \equiv D(x^e \bmod n, k') \equiv (x^e)^d \equiv x^{e\cdot d} \equiv x \bmod n \; ,$$

da nach Annahme $e \cdot d \equiv 1 \bmod \phi(n)$ gilt. □

Bevor wir Laufzeitabschätzungen durchführen, betrachten wir das Setup des RSA-Systems genauer. Zunächst einige *kryptanalytische* Überlegungen:

Die Sicherheit des RSA-Systems beruht auf der (höchstwahrscheinlich korrekten) Annahme, dass die Faktorisierung von n nicht effizient durchführbar ist, also nicht in Zeit $O(\text{poly}(\log n))$. Auch ist es schwierig, in $\mathbf{Z}_n^*$ Wurzeln, etwa die e'te, $e \geq 2$, zu ziehen. In $\mathbf{N}$ Wurzeln zu ziehen ist jedoch bei ganzzahligem Ergebnis einfach, da man die binäre Suche anwenden kann.

(a) Angenommen, der Gegner kennt die Primzahlen p und q mit $n = p \cdot q$. Dann kann er auch $\phi(n) = (p-1)\cdot(q-1)$ leicht berechnen. (Die Anzahl der zu $n = p \cdot q$ teilerfremden Zahlen $x < n$ ist gleich $\phi(p \cdot q) = (p-1)\cdot(q-1)$.)

Kennt der Gegner $\phi(n)$, so kann er mit dem sogenannten *Euklidischen Algorithmus* (später) mit e auch den geheimen Schlüssel $d \equiv e^{-1} \bmod \phi(n)$, also $d \cdot e \equiv 1 \bmod \phi(n)$, in Polynomialzeit berechnen und so die Nachricht korrekt entschlüsseln.

(b) Mit $\phi(n) = (p-1) \cdot (q-1) = n - p - q + 1$ folgt

$$(p+q)^2 = (p-q)^2 + 4 \cdot p \cdot q = (p-q)^2 + 4n$$

also

$$(p+q)^2 = (p-q)^2 + 4n \; .$$

Kennt ein möglicher Angreifer nun den Wert $|p-q| = a$, etwa durch Raten oder Durchprobieren bei kleinen Werten von a, so kann er das Gleichungssystem

$$\begin{aligned} (p+q)^2 &= a^2 + 4n \\ |p-q| &= a \end{aligned}$$

nach p und q auflösen, und kennt damit p und q. Denn sei o.B.d.A. $p \geq q$, dann folgt $p = q + a$ und somit $(2q+a)^2 = a^2 + 4n$, also $q = {}^1/_2 \cdot (\sqrt{a^2+4n} - a)$. Die Differenz $|p-q|$ sollte also nicht zu klein gewählt werden, um ein Raten für den Angreifer zu erschweren. Falls gilt $p = q$, so ist die Faktorisierung von $n = p^2$ mit Wurzelziehen einfach durchzuführen.

(c) Wird dieselbe Nachricht x an verschiedene Adressaten $B_1, B_2, \ldots, B_e$ mit $e \geq 3$ gesendet und werden dabei die öffentlichen Schlüssel $(n_1, e), (n_2, e), \ldots, (n_e, e)$ mit ggT $(n_i, n_j) = 1$ für alle $i \neq j$ verwendet - also haben alle öffentlichen Schlüssel die gleiche zweite Komponente und diese ist identisch mit der Anzahl der Empfänger - so kann ein Gegner die verschlüsselte Nachricht leicht entschlüsseln. Dieses erfolgt effizient mit dem *Chinesischen Restsatz*, wie wir gleich sehen werden.

Theorem 6.4 *[**Chinesischer Restsatz**] Sei $n = n_1 \cdot n_2 \cdot \ldots \cdot n_k$, wobei die natürlichen Zahlen $n_1, n_2, \ldots, n_k$ paarweise teilerfremd sind. Dann gibt es zu gegebenen Zahlen $a_i \in \mathbf{Z}_{n_i}$, $i = 1, 2, \ldots, k$, genau ein $a \in \mathbf{Z}_n$ mit*

$$a \equiv a_i \bmod n_i \qquad\qquad \textit{für } i = 1, \ldots, k$$

und dieses Element a kann effizient berechnet werden, also in Zeit $O(poly(k, \log n))$.

Der Chinesische Restsatz garantiert die Lösung eines Systems *simultaner Kongruenzen*. Sind die Zahlen $n_1, \ldots, n_k$ nicht paarweise teilerfremd, so gilt die Aussage des Chinesischen Restsatzes nicht, wie schon das Beispiel $k = 2$ und $n_1 = n_2 = n \geq 3$ sowie $a_1 = 1$ und $a_2 = 2$ zeigt.

Beweis: Da $n_1, n_2, \ldots, n_k$ paarweise teilerfremd sind, gilt für $i = 1, 2, \ldots, k$:

$$\text{ggT} \left(n_i, \frac{n}{n_i} \right) = 1 \; .$$

Mit dem Euklidischen Algorithmus (später) kann man effizient Zahlen $m_i \in \mathbf{Z}_{n_i}$, $i = 1, 2, \ldots, k$, finden mit

$$m_i \cdot \frac{n}{n_i} \equiv 1 \bmod n_i \; ,$$

also existiert das multiplikativ Inverse m_i von ${}^{n}/_{n_i}$ in $\mathbf{Z}_{n_i}^*$.

Es ist offensichtlich ${}^{n}/_{n_i} \equiv 0 \bmod n_j$ für $i \neq j$. Wir setzen

$$a :\equiv \sum_{i=1}^{k} a_i \cdot m_i \cdot \frac{n}{n_i} \bmod n \; .$$

Da jedes n_i ein Teiler von n ist, folgt für $j = 1, 2, \ldots, k$:

$$\begin{aligned} a &\equiv \sum_{i=1}^{k} a_i \cdot m_i \cdot \frac{n}{n_i} \bmod n \\ &\equiv \sum_{i=1}^{k} a_i \cdot m_i \cdot \frac{n}{n_i} \bmod n_j \\ &\equiv a_j \cdot m_j \cdot \frac{n}{n_j} \bmod n_j \\ &\equiv a_j \bmod n_j \end{aligned}$$

und wir haben ein geeignetes Element $a \in \mathbf{Z}_n$ gefunden.

Wir müssen noch die Eindeutigkeit einsehen. Gäbe es ein weiteres Element $b \in \mathbf{Z}_n$ mit obigen Eigenschaften, dann ist $a \equiv b \bmod n_i$ für $i = 1, 2, \ldots, k$ und wegen der paarweisen Teilerfremdheit der n_i auch $a \equiv b \bmod n$, also ist a eindeutig. □

Beispiel 6.5 *Wir wollen das Kongruenzensystem*

$$a \equiv 3 \bmod 7$$

$$a \equiv 4 \bmod 9$$

lösen. Zuerst suchen wir m_1, m_2 mit

$$\begin{aligned} m_1 \cdot 9 &\equiv 1 \bmod 7 \\ m_2 \cdot 7 &\equiv 1 \bmod 9 \; . \end{aligned}$$

Offenbar ist $m_1 \equiv 4 \bmod 7$ *und* $m_2 \equiv 4 \bmod 9$. *Dann setzen wir*

$$\begin{aligned} a &\equiv 3 \cdot m_1 \cdot 9 + 4 \cdot m_2 \cdot 7 \bmod 63 \\ &\equiv 3 \cdot 4 \cdot 9 + 4 \cdot 4 \cdot 7 \bmod 63 \\ &\equiv 45 + 49 \bmod 63 \\ &\equiv 31 \bmod 63 \,. \end{aligned}$$

Also ist $a \equiv 31 \bmod 63$ *die Lösung. Es gilt* $31 \equiv 3 \bmod 7$ *und* $31 \equiv 4 \bmod 9$.

Wir kommen nun auf (c) zurück. Die Nachricht x werde verschlüsselt an e verschiedene Empfänger gesandt und dabei werden die öffentlichen Schlüssel $(n_1, e), (n_2, e), \ldots, (n_e, e)$ verwendet, wobei die Zahlen $n_1, n_2, \ldots, n_e$ paarweise teilerfremd sind. Die Situation, dass die Zahlen $n_1, n_2 \ldots, n_e$ paarweise teilerfremd sind, ist nicht so unwahrscheinlich, da etwa eine Folge von Primzahlen diese Eigenschaft hat! Sind die verschlüsselten Nachrichten $c_1, c_2, \ldots, c_e$ mit

$$c_i \equiv x^e \bmod n_i$$

für $i = 1, 2, \ldots, e$ (und natürlich die Moduli $n_1, n_2, \ldots, n_e$) dem Gegner bekannt, so berechnet dieser in Polynomialzeit mit dem Chinesischen Restsatz Theorem 6.4 das eindeutige $a \in \mathbf{Z}_n$ mit $n := \prod_{i=1}^{e} n_i$ derart, dass

$$a \equiv c_i \bmod n_i$$

für $i = 1, 2, \ldots, e$ gilt. Mit $0 \leq a \leq \prod_{i=1}^{e} n_i$ sowie $0 \leq x < n_i$, $i = 1, 2, \ldots, e$, für die Nachricht x wegen der Injektivität bei der Verschlüsselung, also $x^e < \prod_{i=1}^{e} n_i$, folgt $a = x^e$. Damit kann durch einfaches Ziehen der e'ten Wurzel (in $\mathbf{N}$!) $x = a^{1/e}$ berechnet werden und damit leicht entschlüsselt werden. (Die e'te Wurzel (in $\mathbf{N}$) einer natürlichen Zahl kann leicht (und effizient) mit binärer Suche ermittelt werden.)

Handelt es sich statt einer Nachricht x um verschiedene Nachrichten $x_1, x_2, \ldots, x_e$, so kann diese Technik nicht erfolgreich eingesetzt werden. Warum nicht?

(d) Ist ggT $(x, n) \neq 1$ für eine Nachricht $x \in \mathbf{Z}_n$, so kann zunächst eventuell nicht korrekt entschlüsselt werden, da wir für den Nachweis den Satz von Euler Theorem 6.1 mit $x \in \mathbf{Z}_n^*$

benutzt haben. Aber, da n das Produkt zweier verschiedener Primzahlen ist, wie hier gegeben, gilt

Lemma 6.6 *Für $x \in \mathbf{Z}_n$ und $e \cdot d \equiv 1 \bmod \phi(n)$ ist*

$$(x^e)^d \equiv x \bmod n \ , \tag{2}$$

falls $n = p \cdot q$, wobei p, q verschiedene Primzahlen sind.

Bemerkung: Die Gleichung $x^{\phi(n)} \equiv 1 \bmod n$ gilt nicht für jedes $x \in \mathbf{Z}_n$ wie etwa das Beispiel $n = 6$, also $\phi(n) = 2$, und $x \equiv 2 \bmod 6$ zeigt, denn $x^2 \equiv 2^2 \equiv 4 \not\equiv 1 \bmod 6$. Andererseits ist $x^3 = x^{\phi(2)+1} \equiv 2 \bmod 6$.

Beweis: Ist $x \in \mathbf{Z}_n^*$, so folgt (2) mit dem Satz von Euler Theorem 6.1. Für $x \equiv 0 \bmod n$ ist die Behauptung trivialerweise wahr. Ansonsten ist entweder $x \equiv 0 \bmod p$ oder $x \equiv 0 \bmod q$. O.B.d.A. sei der erste Fall gegeben. Für $x \equiv 0 \bmod p$ ist offenbar $x^e \equiv 0 \bmod p$ für jedes $e \geq 1$, also $x^{e \cdot d} \equiv x \bmod p$. Mit $e \cdot d = 1 + j(p-1)(q-1)$ für ein j ergibt sich

$$(x^e)^d = x \cdot x^{j(p-1)(q-1)} \ ,$$

folglich wegen $x \not\equiv 0 \bmod q$ mit dem Satz von Euler Theorem 6.1

$$\begin{aligned} (x^e)^d = x \cdot (x^{q-1})^{j(p-1)} &\equiv x \cdot 1^{j(p-1)} \bmod q \\ &\equiv x \bmod q \end{aligned}$$

und natürlich

$$x \cdot (x^{p-1})^{j(q-1)} \equiv x \bmod p$$

also $x^{ed} \equiv x \bmod q$ und $x^{ed} \equiv x \bmod p$. Da p und q verschieden sind, also ggT $(p, q) = 1$ ist, folgt $x^{ed} \equiv x \bmod (p \cdot q) \equiv x \bmod n$. □

Beim RSA-System können wir also beliebige Zahlen $x \in \mathbf{Z}_n$ verwenden.

Frage: Gilt die Aussage des Lemmas auch, wenn $n = p_1 \cdot \ldots \cdot p_e$ Produkt von paarweise verschiedenen Primzahlen ist? *Antwort ist 'Ja' $\longrightarrow$ Übung.*

6.1 Der Euklidische Algorithmus

Vielfach trat bisher der Begriff Euklidischer Algorithmus auf. Dieses Verfahren dient zur Bestimmung des größten gemeinsamen Teilers ggT (r_0, r_1) zweier gegebener positiver ganzer Zahlen $r_0 \geq r_1$.

Dieses Verfahren geht iterativ vor. Man bestimmt durch Division mit Rest positive ganze Zahlen $q_1, q_2, \ldots$ (solange wie möglich) sowie nichtnegative ganzzahlige Reste $r_2, r_3, \ldots$ (neben den bekannten Zahlen r_0, r_1) gemäß der Rekursion $r_{i-1} = q_i \cdot r_i + r_{i+1}$ unter den Nebenbedingungen $0 \leq r_{i+1} < r_i$. Man stoppt, wenn zum ersten Mal $r_{m+1} = 0$ für ein m gilt:

$$\begin{aligned}
r_0 &= q_1 \cdot r_1 + r_2 && 0 < r_2 < r_1 \\
r_1 &= q_2 \cdot r_2 + r_3 && 0 < r_3 < r_2 \\
&\vdots \quad \vdots && \vdots \\
r_{m-2} &= q_{m-1} \cdot r_{m-1} + r_m && 0 < r_m < r_{m-1} \\
r_{m-1} &= q_m \cdot r_m && \text{also } r_{m+1} = 0\,.
\end{aligned}$$

Mit der Beziehung ggT $(a, b) =$ ggT $(a, b - n \cdot a)$ für $n \in \mathbf{Z}$ hat man

$$\text{ggT}\ (r_{i-1}, r_i) = \text{ggT}\ (r_i, r_{i-1}) = \text{ggT}\ (r_i, r_{i-1} - q_i \cdot r_i) = \text{ggT}\ (r_i, r_{i+1})\,, \tag{3}$$

und damit folgt mit Induktion

$$\text{ggT}\ (r_0, r_1) = \text{ggT}\ (r_m, r_{m+1} = 0) = r_m\,.$$

Die Korrektheit dieses Verfahrens zur Bestimmung des größten gemeinsamen Teilers zweier Zahlen ist damit bewiesen. Außerdem ist $r_0 \geq r_1 \geq \ldots \geq r_{m+1}$, sogar $r_1 > \ldots > r_{m+1}$, und damit $q_1, \ldots, q_m \geq 1$. Wie sieht es mit der Komplexität (Laufzeit) des Verfahrens aus?

(i) Mit der Schulmethode kann man die Addition von zwei Binärzahlen der Länge k (k-Bit Zahlen) in Zeit $O(k)$ (Zeit maximal $c \cdot k$ für eine Konstante $c > 0$) durchführen.

(ii) Das Produkt von zwei k-Bit Zahlen kann nach der Schulmethode mit maximal $(k-1)$ Additionen berechnet werden. Bei jeder Addition werden zwei höchstens $2k$-Bit Zahlen addiert, wobei jede Addition Zeit $O(k)$ kostet. Also ergibt sich als Gesamtzeit $O(k^2)$.

(iii) Analog wie bei der Multiplikation lässt sich einsehen, dass die Division (ohne Rest) von zwei k-Bit Zahlen in Zeit $O(k^2)$ durchgeführt werden kann. Analoges gilt für die Multiplikation modulo einer k-Bit Zahl.

(iv) Das Potenzieren x^y für eine k-Bit Zahl y, gegeben als 0, 1-Folge $(y_1, \ldots, y_k)$, kann mit maximal $2(k-1)$ Multiplikationen durchgeführt werden. Hierbei nutzt man die *modulare Exponentiation* aus, das heißt die Identität $x^{2^{i+1}} = (x^{2^i})^2$. Für die gegebene k-Bit Zahl $y = \sum_{i=0}^{k-1} y_i \cdot 2^i$ berechnet man $x, x^2, x^4, \ldots, x^{2^i}$ durch Quadrieren der jeweils vorhergehenden Zahl $x^{2^{i-1}}$ und multipliziert diejenigen Potenzen x^{2^i}, für die $y_i = 1$ gilt.

(v) Man kann sich überlegen, dass der Euklidische Algorithmus zur Bestimmung von ggT (r_0, r_1) für $r_0 \geq r_1$ maximal $O(\log r_0)$ Iterationen benötigt. Man erhält dann eine Gesamtlaufzeit des Euklidischen Algorithmus von $O((\log r_0)^3)$, da eine Iteration maximal Zeit $O((\log r_0)^2)$ kostet, also ist die Gesamtzeit polynomiell in der Eingabegröße.

Theorem 6.7 *Bei Eingabe natürlicher Zahlen $r_0 \geq r_1$ benötigt der Euklidische Algorithmus zur Bestimmung des größten gemeinsamen Teilers maximal $\log_{1.5}(r_0 + r_1)$ Iterationen.*

Beweis: Ausgehend von r_0, r_1 bestimmt der Euklidische Algorithmus Zahlen $r_{i-1} = q_i \cdot r_i + r_{i+1}$ mit $0 \leq r_{i+1} < r_i$ bis zum ersten Mal gilt $r_{m+1} = 0$. Man zeigt nun

$$r_{i+1} + r_i \leq {}^2/_3 \cdot (r_{i-1} + r_i) \,. \tag{4}$$

Dieses gilt unmittelbar für $r_{i-1} \geq 2r_i$. Andererseits, für $r_{i-1} < 2r_i$, folgt mit $r_{i-1} > r_i$ sofort $r_{i+1} = r_{i-1} - r_i$, also $r_{i+1} + r_i = r_{i-1} < {}^2/_3 \cdot (r_{i-1} + r_i)$. Aus (4) folgt dann per Induktion und der Monotonie $r_1 \geq \ldots \geq r_{m+1}$:

$$1 \leq r_{m+1} + r_m \leq r_{j+1} + r_j \leq ({}^2/_3)^j \cdot (r_0 + r_1) \,.$$

Damit erhalten wir $({}^3/_2)^m \leq r_0 + r_1$, also $m \leq \log_{1.5}(r_0 + r_1)$. □

Es sei hier vermerkt, dass man für ggT $(r_0, r_1) = g$ mit dem Euklidischen Algorithmus auch ganze Zahlen $x, y \in \mathbf{Z}$ erhält mit $x \cdot r_0 + y \cdot r_1 = g$, indem man das Verfahren „rückwärts“ durchläuft.

6.2 Berechnung von multiplikativ Inversen

Mit einer Variante des Euklidischen Algorithmus kann man auch zu gegebenen Elementen $e \in \mathbf{Z}_n^*$ die multiplikativ Inversen $x := e^{-1} \in \mathbf{Z}_n^*$, also $x \cdot e \equiv 1 \bmod n$ berechnen. Hierzu definiert man zusätzlich die Folge $t_0, t_1, \ldots, t_m$ nichtnegativer ganzer Zahlen wie folgt:

$$\begin{aligned} t_0 &:= 0 \\ t_1 &:= 1 \\ t_i &:= t_{i-2} - q_{i-1} \cdot t_{i-1} \bmod r_0 \qquad \text{für } i = 2, \ldots, m \, . \end{aligned}$$

Wir behaupten, dass gilt:

$$r_j \equiv t_j \cdot r_1 \bmod r_0 \qquad \text{für } j = 0, 1, \ldots, m \, . \tag{5}$$

Bevor wir (5) beweisen, folgern wir hieraus für $j = m$:

$$1 \equiv r_m \equiv t_m \cdot r_1 \bmod r_0 \, ,$$

mit ggT $(r_0, r_1) = r_m = 1$ erhalten wir $t_m \equiv r_1^{-1} \bmod r_0$. Wählen wir $r_0 := n$, so ist $t_m \equiv r_1^{-1} \bmod n$ das multiplikativ Inverse von r_1 in $\mathbf{Z}_n^*$.

Beweis: Wir beweisen (5) mit Induktion über j.

$\boldsymbol{i = 0}$: Für $j = 0$ gilt $r_0 \equiv 0 \bmod r_0$ und mit $t_0 = 0$ folgt insbesondere $r_0 \equiv t_0 \cdot r_1 \bmod r_0$.

$\boldsymbol{j-1 \to j}$: Mit Induktionsannahme für $(j-1)$, dem Euklidischen Algorithmus und gemäß Definition der t_j erhalten wir

$$\begin{aligned} r_j &= r_{j-2} - q_{j-1} \cdot r_{j-1} \\ &\equiv r_{j-2} - q_{j-1} \cdot r_{j-1} \bmod r_0 \\ &\equiv t_{j-2} \cdot r_1 - q_{j-1} \cdot t_{j-1} \cdot r_1 \bmod r_0 \\ &\equiv (t_{j-2} - q_{j-1} \cdot t_{j-1}) \cdot r_1 \bmod r_0 \\ &\equiv t_j \cdot r_1 \bmod r_0 \, . \end{aligned}$$

□

Also ist das multiplikativ Inverse von $e \in \mathbf{Z}_n^*$ leicht in Polynomialzeit zu berechnen. Es sei hier vermerkt, dass das multiplikativ Inverse e^{-1} von e in $\mathbf{Z}_n$ genau dann existiert, wenn gilt ggT $(e, n) =$

1. Ist nämlich andernfalls $e \in \mathbf{Z}_n \setminus \mathbf{Z}_n^*$, also ggT $(e, n) = r > 1$, so gilt „r teilt n“, aber aus „r teilt n“ folgt „r teilt nicht $(-1 + l \cdot n)$“ für jedes $l \in \mathbf{Z}$. Wäre also $x \cdot e \equiv 1 \bmod n$ lösbar, so folgte $n|(x \cdot e - 1)$ und damit $r|(x \cdot e - 1)$, Widerspruch.

7 Effizienzüberlegungen beim Setup des RSA-Systems

Wir betrachten im Setup des RSA-Systems Punkt (a) „B erzeugt zwei große Primzahlen p und q“. Wie schnell findet man diese Primzahlen p und q? Hier hilft uns der *Primzahlsatz*:

Theorem 7.1 (Primzahlsatz) *Sei $\Pi(n)$ die Anzahl der Primzahlen p mit $p \leq n$. Dann gilt:*

$$\Pi(n) \sim \frac{n}{\ln n} \quad bzw. \quad \lim_{n\to\infty} \frac{\Pi(n) \cdot \ln n}{n} = 1 \; .$$

Dieses tiefliegende Resultat aus der Zahlentheorie werden wir hier nicht beweisen.
Würfeln wir uns nun zufällig eine k-stellige Binärzahl aus, dann gibt es unter allen k-stelligen Binärzahlen nach dem Primzahlsatz im wesentlichen

$$\frac{2^{k+1}}{\ln 2^{k+1}} - \frac{2^k}{\ln 2^k} = \frac{1}{\ln 2} \cdot 2^k \cdot \left(\frac{2}{k+1} - \frac{1}{k}\right)$$

Primzahlen, also erwischt man eine Primzahl mit Wahrscheinlichkeit $\Theta(1/k)$. Die Wahrscheinlichkeit, dass eine ausgewürfelte Zahl n eine Primzahl ist, beträgt also ungefähr $\Theta(1/\log n)$. Um eine Primzahl der Größe etwa n zu bekommen, muss man daher im Durchschnitt (Erwartungswert) $O(\log n)$ oft würfeln.
Wenn man eine Zahl n ausgewürfelt hat, muss man natürlich testen, ob diese Zahl auch eine Primzahl ist. Dieses sollte effizient geschehen, also in Polynomialzeit, aber leider ist zum Nachweis der Primzahleigenschaft bisher kein Polynomialzeitverfahren bekannt.
Die besten „general purpose Verfahren“ zum Primzahltest haben bei Eingabe von n in Binärdarstellung eine „heuristische Laufzeit“ von

$$e^{O(\sqrt{\log n \cdot \log\log n})}$$

Besser aber auch komplizierter ist hier jedoch das *Number Field Sieve* mit einer „heuristischen Rechenzeit“ von

$$e^{O((\log n)^{1/3} \cdot (\log\log n)^{2/3})} \; .$$

Man bedient sich daher sogenannter Primzahltests. Ein Kriterium heißt *Primzahltest*, wenn es bei Eingabe n die korrekte Antwort „n ist keine Primzahl“ liefert, oder, wenn n diesen Test besteht, die Antwort „n ist vielleicht Primzahl“ liefert. Dieses sollte natürlich quantifiziert werden. Primzahltests sollen effizient durchführbar sein.
Das RSA-Verfahren basiert auf der Annahme, dass es erheblich einfacher ist, zwei große Primzahlen p und q zu finden, als bei gegebener Zahl n das Faktorisierungsproblem zu lösen, also die Primzahlen p, q mit $n = p \cdot q$ zu finden.
Bevor wir einen effizienten Primzahltest, den Solovay-Strassen Test, beschreiben, benötigen wir noch einige Kenntnisse.

Theorem 7.2 (Eulersches Kriterium) *Sei $p \geq 3$ eine Primzahl. Für $a \in \mathbf{Z}_p \setminus \{0\}$ ist die Gleichung $x^2 \equiv a \bmod p$ genau dann lösbar, wenn gilt*

$$a^{\frac{p-1}{2}} \equiv 1 \bmod p\,.$$

Falls die Gleichung $x^2 \equiv a \bmod p$ lösbar ist, so nennt man a einen *quadratischen Rest modulo p* und andernfalls heißt a *quadratischer Nichtrest modulo p.*

Bemerkung: Für jede Primzahl $p \geq 3$ und für jedes $a \in \mathbf{Z}_p \setminus \{0\}$ gilt:

$$a^{\frac{p-1}{2}} \equiv \pm 1 \bmod p\,, \tag{6}$$

da nach dem Satz von Fermat $a^{p-1} \equiv 1 \bmod p$ gilt und die Gleichung $x^2 \equiv 1 \bmod p$ nur die beiden Lösungen $x \equiv \pm 1 \bmod p$ hat.
In einer Gruppe $(G, \cdot)$ heißt $g \in G$ *erzeugendes (primitives) Element*, wenn gilt $G = \{g, g^2, \ldots, g^{|G|}\}$. Man nennt $|G|$ die *Ordnung von G*. Eine Gruppe, die ein erzeugendes Element hat, heißt *zyklisch*. Insbesondere sind für Primzahlen p die Gruppen $\mathbf{Z}_p^* \setminus \{0\}$ zyklisch. Ist g ein erzeugendes Element von G, so gibt es zu jedem Element $h \in G$ eine positive Zahl x mit $h = g^x$. Diese Zahl x heißt *diskreter Logarithmus von h zur Basis g.*
Beweis: $\Longrightarrow$: Ist die Gleichung $x^2 \equiv a \bmod p$ lösbar, und damit $x \not\equiv 0 \bmod p$ wegen $a \in \mathbf{Z}_p \setminus \{0\}$, dann gilt

$$a^{\frac{p-1}{2}} \equiv (x^2)^{\frac{p-1}{2}} \equiv x^{p-1} \equiv 1 \bmod p$$

nach dem Satz von Fermat.

$\Longleftarrow$: Wir nehmen an, dass $a^{(p-1)/2} \equiv 1 \bmod p$ gilt. Die multiplikative Gruppe $\mathbf{Z}_p^*$ hat ein *erzeugendes Element* g, also $\mathbf{Z}_p^* = \{g, g^2, \ldots, g^{p-1}\}$. Da $a \in \mathbf{Z}_p^*$ ist, gibt es eine natürliche Zahl i mit $a \equiv g^i \bmod p$. Damit folgt

$$1 \equiv a^{\frac{p-1}{2}} \equiv g^{\frac{i(p-1)}{2}} \bmod p$$

Aus $g^{i\cdot(p-1)/2} \equiv 1 \bmod p$ folgt nach dem Satz von Fermat und da g erzeugendes Element von $\mathbf{Z}_n^*$ ist, dass $i \cdot (p-1)/2$ ein ganzahliges Vielfaches von $(p-1)$ ist, also i gerade sein muss, etwa $i = 2h$. Dann ist aber $a \equiv g^{2h} \bmod p$ und für $x :\equiv \pm g^h \bmod p$ gilt $x^2 \equiv a \bmod p$. □

Theorem 7.3 *Sei $p \geq 3$ eine Primzahl. In $\mathbf{Z}_p^*$ ist die Anzahl der quadratischen Reste gleich der Anzahl der quadratischen Nichtreste.*

Beweis: Die Behauptung folgt unmittelbar aus folgendem.
Für ein erzeugendes Element $g \in \mathbf{Z}_p^*$ ist g^i genau dann quadratischer Rest, wenn i gerade ist.
$\Longleftarrow$: Ist i gerade, dann ist natürlich $g^{i/2}$ eine Quadratwurzel von g^i.
$\Longrightarrow$: Sei nun i ungerade, etwa $i = 2 \cdot h + 1$. Angenommen, es gibt $x \in \mathbf{Z}_p^*$ mit $x^2 \equiv g^{2h+1} \bmod p$. Da g erzeugendes Element ist, gibt es ein j mit $x \equiv g^j \bmod p$. Also ist $g^{2j} \equiv g^{2h+1} \bmod p$ und nach dem Korollar zum Satz von Euler Theorem 6.1 auch $2j \equiv 2h + 1 \bmod \phi(p)$, also $2(j-h) - 1 \equiv 0 \bmod (p-1)$, aber $(p-1)$ ist gerade und $2(j-h)-1$ ungerade, und wir haben einen Widerspruch. □

Wir behandeln jetzt den *Primzahltest von Solovay und Strassen.*

Primzahltest von Solovay und Strassen:
Eingabe: Natürliche Zahlen n und k, wobei n ungerade ist und in Binärdarstellung gegeben ist. (Die Parameter k ist ein Sicherheitsparameter.)
Der Algorithmus führt die folgenden Schritte durch.

1. Teste, ob n eine nichttriviale Potenz ist, also ob $n = b^m$ für $b, m > 1$ gilt. Falls ja, dann ist die Ausgabe *n ist keine Primzahl.* STOP.

2. Wähle zufällig $a_1, a_2, \ldots, a_k \in \mathbf{Z}_n \setminus \{0\}$, unabhängig voneinander gemäß der Gleichverteilung.

3. Falls ggT $(a_i, n) \neq 1$ für ein $i = 1, 2, \ldots, k$ gilt, dann ist die Ausgabe *n ist keine Primzahl.* STOP.

4. Berechne $z_i :\equiv a_i^{(n-1)/2} \bmod n$ für $i = 1, 2, \ldots, k$.

5. Falls $z_i \not\equiv \pm 1 \bmod n$ für ein i gilt, dann ist die Ausgabe *n ist keine Primzahl.* STOP.

6. Falls $z_i \equiv 1 \bmod n$ für jedes $i = 1, 2, \ldots, k$ gilt, dann ist die Ausgabe *n ist wahrscheinlich keine Primzahl*, STOP,

   sonst ist die Ausgabe *n ist wahrscheinlich eine Primzahl.* STOP.

Theorem 7.4 *Der Primzahltest von Solovay und Strassen ist ein* probabilistischer Polynomialzeit (PP) *Algorithmus (also polynomiell in k und $\log n$) mit einer Irrtumswahrscheinlichkeit von maximal 2^{-k}.*

Die Laufzeit des Algorithmus von Solvay und Strassen ist offenbar polynomiell. Zum Beweis, speziell hinsichtlich der Abschätzung der Irrtumswahrscheinlichkeit, werden wir folgendes Lemma benutzen.

Lemma 7.5 *Sei n eine ungerade positive ganze Zahl, aber keine Potenz einer Primzahl, $n \neq p^l$ für p prim.*
Wenn es mindestens ein Element $a \in \mathbf{Z}_n^*$ mit $a^{(n-1)/2} \equiv -1 \bmod n$ gibt, dann gilt für die Menge $U_n := \{x \in \mathbf{Z}_n^* \mid x^{(n-1)/2} \equiv \pm 1 \bmod n\}$:

$$|U_n| \leq |\mathbf{Z}_n^*|/2 \ .$$

Lemma 7.5 ist zum Beispiel für $n \equiv 3 \bmod 4$ anwendbar, also $(n-1)/2 \equiv 1 \bmod 2$, mit $a \equiv -1 \bmod n$. **Beweis:** Man zeigt, dass U_n eine echte Untergruppe von $\mathbf{Z}_n^*$ bezüglich der Multiplikation ist, also $U_n \neq \mathbf{Z}_n^*$ gilt. Da für jede Untergruppe U einer Gruppe G gilt, dass $|U|$ ein Teiler von $|G|$ ist, folgt die Behauptung.
Offenbar folgt aus $x^{(n-1)/2} \equiv \pm 1 \bmod n$ und $y^{(n-1)/2} \equiv \pm 1 \bmod n$ auch $(x \cdot y)^{(n-1)/2} \equiv \pm 1 \bmod n$, also ist U_n abgeschlossen.
Wir zeigen nun die Existenz eines Elementes $b \in \mathbf{Z}_n^* \setminus U_n$, mit dem $|U_n| \leq |\mathbf{Z}_n^*|/2$ folgt.
Die Primfaktorzerlegung von n sei gegeben durch $n = \prod_{i=1}^{l} p_i^{k_i}$ mit Primzahlen $p_1, p_2 \ldots, p_l$. Man setzt $q := p_1^{k_1}$ sowie $m := n/q$. Nach Voraussetzung ist $l \geq 2$, also $m > 1$, und es gilt ggT $(m, q) = 1$.
Nach dem Chinesischen Restsatz Theorem 6.4 gibt es ein $b \in \mathbf{Z}_n$ mit $b \equiv a \bmod q$ sowie $b \equiv 1 \bmod m$. Dann ist $b \in \mathbf{Z}_q^*$ und $b \in \mathbf{Z}_m^*$, also auch $b \in \mathbf{Z}_n^*$. Damit ist auch $b^{(n-1)/2} \equiv 1 \bmod m$.

Desweiteren ist $b^{(n-1)/2} \equiv a^{(n-1)/2} \equiv -1 \bmod q$, da $b \equiv a \bmod q$ gilt sowie nach Voraussetzung $a^{(n-1)/2} \equiv -1 \bmod n$. Wir unterscheiden zwei Fälle.

Fall (i): Wäre $b^{(n-1)/2} \equiv 1 \bmod n$, dann folgt mit ggT $(q, m) = 1$, dass $b^{(n-1)/2} \equiv 1 \bmod q$ und $b^{(n-1)/2} \equiv 1 \bmod m$ gilt. Widerspruch, da $b^{(n-1)/2} \equiv -1 \bmod q$ ist!

Fall (ii): Wäre $b^{(n-1)/2} \equiv -1 \bmod n$, dann folgt wie in Fall (i): $b^{(n-1)/2} \equiv -1 \bmod q$ und $b^{(n-1)/2} \equiv -1 \bmod m$. Widerspruch, da $b^{(n-1)/2} \equiv 1 \bmod m$ ist.

Also haben wir $b^{(n-1)/2} \not\equiv \pm 1 \bmod n$ und damit ist $b \notin U_n$, also $U_n \neq \mathbf{Z}_n^*$. □

Nun kommen wir zum Beweis von Theorem 7.4.

Beweis: Wir betrachten die einzelnen Schritte im Algorithmus. Die Schritte 2. sowie 4. bedürfen keiner Erklärung.

zu 1.: Ist $n = b^m$ eine nichttriviale Potenz ($m > 1$), so folgt $m \leq \log n$ mit $b \geq 2$. Also kann man einfach für jedes $m = 2, \ldots, \log n$ nach einem $b \leq \sqrt{n}$ suchen mit $b^m = n$. Mit binärer Suche (für jedes m) etwa beträgt die Suchzeit maximal $O(\log n)$, und damit ist Schritt (1.) in Polynomialzeit ausführbar.

zu 3.: Hier wird einfach der Euklidische Algorithmus angewandt.

zu 5.: Die Korrektheit folgt mit dem Satz von Fermat siehe auch 6.

zu 6.: Nur in diesem Schritt können Fehler auftreten. Wir unterscheiden zwei Fälle, abhängig davon, ob n Primzahl ist oder nicht.

n ist Primzahl: Die Ausgabe ist jedoch *n ist keine Primzahl.* Dann ist $z_i \equiv 1 \bmod n$ für $i = 1, 2, \ldots, k$. Es gilt nun nach den vorherigen Ergebnissen (Eulersches Kriterium) für Primzahlen n, dass $a \in \mathbf{Z}_n^*$ quadratischer Rest ist, genau wenn $a^{(n-1)/2} \equiv 1 \bmod n$ ist. Da es genau $(n-1)/2$ quadratische Reste in $\mathbf{Z}_n^*$ gibt, ist die Wahrscheinlichkeit, dass $a_i^{(n-1)/2} \equiv 1 \bmod n$ für ein zufällig gewähltes $a_i \in \mathbf{Z}_n^*$ gilt, gleich $^1/_2$. Bei k „erfolgreichen“ Versuchen ist die Wahrscheinlichkeit für das Ereignis $z_1 \equiv z_2 \equiv \ldots \equiv z_k \equiv 1 \bmod n$ gleich $(^1/_2)^k = 2^{-k}$.

n ist keine Primzahl: Die Ausgabe ist jedoch *n ist Primzahl.* Dann gilt $z_i \equiv \pm 1 \bmod n$ für $i = 1, 2, \ldots, k$, und es gibt ein j mit $z_j \equiv -1 \bmod n$. In diesem Fall können wir Lemma 7.5 anwenden, da n nach Schritt (1) keine Primzahlpotenz ist und ungerade ist. Wegen $a_1, a_2, \ldots, a_k \in U_n$ folgt wegen der zufälligen Wahl der $a_1, a_2, \ldots, a_k$ dann

$$\Pr\,(a_1, a_2, \ldots, a_k \in U_n) \leq (^1/_2)^k = 2^{-k}.$$

□

8 Faktorisierungsverfahren

Als Beispiel eines Verfahrens zur Faktorisierung natürlicher Zahlen wird hier der $(p-1)$-*Algorithmus* von Pollard (1974) vorgestellt.

Idee beim Verfahren: Ist p eine Primzahl mit $p|n$ und $q \leq B$ für jede Primzahlpotenz q mit $q|(p-1)$, dann gilt $(p-1)|B!$. Dieses Statement ist folgendermaßen begründet: Für die Primfaktorenzerlegung $p-1 = p_1^{l_1} \cdot \ldots \cdot p_k^{l_k}$ und $p_i^{l_i} \leq B$ für $i = 1, \ldots, k$ gilt $p_i^{l_i} \neq p_j^{l_j}$ für $i \neq j$, und somit folgt aus $p_i^{l_i}|(p-1)$ für $i = 1, 2, \ldots, k$ auch $(p-1)|B!$. Speziell ist dieses Verfahren für Zahlen p geeignet, bei denen $(p-1)$ viele kleine Primfaktoren enthält.

$(p-1)$-Algorithmus:

Eingabe: $n \in \mathbf{N}$ sowie eine Schranke $B \in \mathbf{N}$.

(1.) Berechne $a \equiv 2^{B!} \bmod n$. Dieses kann etwa so erfolgen:

$a := 2$; for $j = 2, 3, \ldots, B$ do $a :\equiv a^j \bmod n$.

(2.) Berechne $d = \text{ggT}\ (a-1, n)$.

(3.) Falls $1 < d < n$, dann ist die Ausgabe „d ist Faktor von n“.

else ist die Ausgabe „kein Faktor von n gefunden“.

Die Korrektheit ist offensichtlich. Falls $(p-1)|B!$ gilt, so ist $a \equiv 1 \bmod p$, also $p|(a-1)$, wobei p ein Primfaktor von n ist.

Laufzeit: In Schritt (1.) zur Berechnung von $a \equiv 2^{B!} \bmod n$ führt man $(B-1)$ modulare Exponentiationen aus, jede kann mit maximal $\lceil 2 \log B \rceil$ modularen Multiplikationen durchgeführt werden. Das Rechnen modulo n kann dann insgesamt in Zeit $O(B \cdot \log B \cdot (\log n)^2)$ durchgeführt werden. In Schritt (2.) kann der ggT in Zeit $O((\log n)^3)$ mit dem Euklidischen Algorithmus berechnet werden. Für die Gesamtzeit T erhalten wir daher die obere Schranke

$$T = O(B \cdot \log B \cdot (\log n)^2 + (\log n)^3)\ .$$

Für $B = O(\text{poly}(\log n))$ ist $T = O(\text{poly}(\log n))$, also Polynomialzeit, aber die Erfolgswahrscheinlichkeit für das Finden eines Teilers ist eventuell klein.

Definition 8.1 *Sei $0 < \varepsilon < 1$ fest. Ein Las Vegas Algorithmus ist ein probabilistisches Verfahren, welches zu jeder (zulässigen) Eingabe mit Wahrscheinlichkeit höchstens ε keine Antwort gibt („Weiss nicht"), aber sonst eine korrekte Antwort liefert.*
Ein Monte Carlo Algorithmus liefert immer eine Antwort, die mit Wahrscheinlichkeit höchstens ε falsch sein kann.

Bemerkung: Wird ein Las Vegas Algorithmus mit Parameter $0 < \varepsilon < 1$ nun l-mal gestartet (*Multi-Start-Ansatz*), so liefert er mit Wahrscheinlichkeit höchstens ε^l keine Antwort. Eine korrekte Antwort erfolgt also mit Wahrscheinlichkeit mindestens $1 - \varepsilon^l$.

Theorem 8.2 *Ein Algorithmus, der zu gegebenem öffentlichen Schlüssel (n, e) im RSA-System den geheimen Schlüssel $(\phi(n), d)$ berechnet, liefert einen Las Vegas Algorithmus mit Parameter $\varepsilon \leq {}^1/_2$ zur Faktorisierung von n.*

Beweis: Zunächst machen wir einige Vorbemerkungen. Sei $n = p \cdot q$ Produkt zweier ungerader Primzahlen p und q. Es gilt dann:

$$\begin{aligned} x^2 \equiv 1 \bmod n \iff & (x^2 \equiv +1 \bmod p \text{ und } x^2 \equiv +1 \bmod q) \\ & \text{oder}(x^2 \equiv -1 \bmod p \text{ und } x^2 \equiv -1 \bmod q). \end{aligned}$$

Dieses bedeutet, dass die Kongruenz $x^2 \equiv 1 \bmod n$ vier Lösungen hat. Also gibt es neben den *trivialen* Lösungen $x \equiv 1 \bmod n$ und $x \equiv -1 \bmod n$ noch zwei weitere *nichttriviale* Lösungen. Diese findet man mit dem Chinesischen Restsatz Theorem 6.4, durch Lösen der simultanen Kongruenzen $x \equiv 1 \bmod p$ und $x \equiv -1 \bmod q$.
Sei $x \not\equiv \pm 1 \bmod n$ nichttriviale Lösung von $x^2 \equiv 1 \bmod n$. Dann gilt

$$n | (x+1) \cdot (x-1) \, ,$$

und folglich ggT $(x+1, n) = p$ oder ggT $(x+1, n) = q$ und analog ist ggT $(x-1, n)$ gleich q oder p. Die größten gemeinsamen Teiler können mit dem Euklidischen Algorithmus berechnet werden. Kennt man nun eine nichtriviale Lösung der Kongruenz $x^2 \equiv 1 \bmod n$, so erhält man mit polynomiellem Zeitaufwand die Faktorisierung von n. Wir haben daher den folgenden Algorithmus:

Algorithmus Faktorisierung bei gegebenem öffentlichen Schlüssel (n, e)

Gegeben: Ein Algorithmus, der zu gegebenen Werten (n, e) ein d berechnet mit $d \cdot e \equiv 1 \bmod \phi(n)$, wobei n Produkt zweier verschiedener Primzahlen ist.

Gesucht: Faktorisierung von n.

(1.) Wähle $w \in \{1, 2, \ldots, n-1\}$ zufällig.

(2.) Bestimme $x := \text{ggT}\ (w, n)$.

(3.) Falls $x > 1$, dann ist die Ausgabe „Faktoren von n sind x und $^n/_x$“, STOP.

(4.) Berechne d mit gegebenem Algorithmus, wobei $d \cdot e \equiv 1 \bmod \phi(n)$ gilt.

(5.) Bestimme $2^t \cdot r := d \cdot e - 1$ mit r ungerade.

(6.) Berechne $v \equiv w^r \bmod n$.

(7.) Falls $v \equiv 1 \bmod n$, dann ist die Ausgabe „keine Antwort“, STOP.

(8.) While $v \not\equiv 1 \bmod n$ do
$v_0 := v; \quad v :\equiv v^2 \bmod n$.

(9.) Falls $v_0 \equiv -1 \bmod n$ ist, dann ist die Ausgabe „keine Antwort“, STOP,
sonst berechne $x := \text{ggT}\ (v_0 + 1, n)$ und die Ausgabe ist „Faktoren sind x und $^n/_x$“.

Bemerkungen:

- Wenn $p|w$ oder $q|w$ gilt, so folgt $x = p$ oder $x = q$.
- Ist ggT $(w, n) = 1$, so wird berechnet $w^r, w^{2r}, w^{4r}, \ldots$ bis gilt $w^{2^s r} \equiv 1 \bmod n$, wobei $s \leq t$ ist wegen $2^t \cdot r \equiv 0 \bmod \phi(n)$.
- Am Ende der While-Schleife in Schritt (8.) hat man $v_0 \not\equiv 1 \bmod n$ und ${v_0}^2 \equiv 1 \bmod n$. Für $v_0 \equiv -1 \bmod n$ erfolgt keine Antwort, sonst ist v_0 nichttriviale Quadratwurzel von 1 mod n, und die Faktorisierung von n ist möglich. Das Verfahren liefert offenbar korrekte Antworten.

Lemma 8.3 *Der Algorithmus liefert keine Antwort mit Wahrscheinlichkeit maximal* $^1/_2$.

Beweis: Der Algorithmus liefert keine Antwort, wenn (a) oder (b) gelten:

(a.) $w^r \equiv 1 \bmod n$

(b.) $w^{2^s r} \equiv -1 \bmod n$ für ein $s \in \{0, 1, \ldots, t-1\}$.

Wir zählen im folgenden die Anzahl der Lösungen der einzelnen Kongruenzen für $n = p \cdot q$.
zu (a.): Gilt $w^r \equiv 1 \bmod n$, so gilt auch $w^r \equiv 1 \bmod p$ und $w^r \equiv 1 \bmod q$. Andererseits können Lösungen von $w^r \equiv 1 \bmod p$ und $w^r \equiv 1 \bmod q$ mit dem Chinesischem Restsatz Theorem 6.4 zu einer Lösung kombiniert werden. Wir betrachten daher die Anzahl der Lösungen der einzelnen Kongruenzen $w^r \equiv 1 \bmod p$ sowie $w^r \equiv 1 \bmod q$.
Zunächst betrachten wir die Kongruenz $w^r \equiv 1 \bmod p$. Sei $g \in \mathbf{Z}_p^*$ erzeugendes Element von $\mathbf{Z}_p^*$, also ist $w \equiv g^u \bmod p$ für ein $0 \leq u \leq p-2$. Dann gilt $g^{ur} \equiv 1 \bmod p$, also nach dem Satz von Fermat

$$(p-1) | u \cdot r \; .$$

Mit

$$\begin{aligned} 2^i \cdot p_1 &:= p-1 \\ 2^j \cdot q_1 &:= q-1 \end{aligned}$$

für ungerade p_1, q_1 gilt

$$\phi(n) | 2^t \cdot r \quad \text{und} \quad e \cdot d - 1 = 2^t \cdot r$$

also

$$2^{i+j} \cdot p_1 \cdot q_1 | 2^t \cdot r \; .$$

Damit gilt $i + j \leq t$ und $(p_1 \cdot q_1) | r$.
Mit $(p-1) | u \cdot r$ folgt $2^i \cdot p_1 | u \cdot r$, also $u = k \cdot 2^i$, da r ungerade ist. Mit $p_1 | r$ und $u \leq p-2$ folgt, dass $k = 0, 1, \ldots, p_1 - 1$ möglich ist. Die Anzahl Lösungen von $w^r \equiv 1 \bmod p$ ist daher p_1. Entsprechend gilt, dass die Anzahl Lösungen von $w^r \equiv 1 \bmod q$ gleich q_1 ist. Die Anzahl Lösungen der Kongruenz $w^r \equiv 1 \bmod n$ ist daher maximal $p_1 \cdot q_1$.
zu (b.): Wir bestimmen für $s = 0, 1, \ldots, t-1$ die Anzahl Lösungen der Kongruenz

$$w^{2^s r} \equiv -1 \bmod n \; .$$

Wie vorher betrachten wir die Kongruenzen $w^{2^s r} \equiv -1 \bmod p$ bzw. $\bmod q$.

Für $w \equiv g^u \bmod p$ und $w^{2^s r} \equiv -1 \bmod p$ folgt $g^{ur2^s} \equiv -1 \bmod p$. Da $g \not\equiv 1 \bmod p$ erzeugendes Element in $\mathbf{Z}_p^*$ ist, gilt $g^{(p-1)/2} \equiv -1 \bmod p$, also nach dem Satz von Fermat

$$\begin{array}{rcll} u \cdot r \cdot 2^s & \equiv & (p-1)/2 \bmod (p-1) & \text{bzw.} \\ 2 \cdot (p-1) & \mid & (u \cdot r \cdot 2^{s+1} - (p-1)) & \text{und mit } p-1 = 2^i \cdot p_1 \\ 2^{i+1} & \mid & \left(\dfrac{u \cdot r \cdot 2^{s+1}}{p_1} - 2^i \right) . & \end{array}$$

Für $s \geq i$ gibt es hier keine Lösungen. Für $s \leq i-1$ sind die Lösungen $u \leq p-2$ genau ungerade Vielfache von 2^{i-s-1}, da r ungerade ist. Die Anzahl dieser Vielfachen ist

$$\frac{p-1}{2^{i-s-1}} \cdot \frac{1}{2} = 2^s \cdot p_1 \; .$$

Analoges folgt für die Kongruenz $w^{2^s r} \equiv -1 \bmod q$. Damit ist die Anzahl Lösungen von $w^{2^s r} \equiv -1 \bmod n$ maximal

$$\begin{array}{rl} 2^{2s} \cdot p_1 \cdot q_1 & \text{für } s \leq \min\{i,j\} - 1 \\ 0 & \text{für } s \geq \min\{i,j\} \; . \end{array}$$

Mit der Annahme $i \leq j$ ist die Anzahl der *schlechten Wahlen* von w dann maximal

$$\begin{array}{rl} & p_1 \cdot q_1 + p_1 \cdot q_1 \cdot (1^2 + 2^2 + \ldots + 2^{2i-2}) \\ = & p_1 \cdot q_1 \cdot \left(\dfrac{2}{3} + \dfrac{2^{2i}}{3} \right) \; . \end{array}$$

Mit $i, j \geq 1$ haben wir

$$p_1 \cdot q_1 = \frac{p-1}{2^i} \cdot \frac{q-1}{2^j} < \frac{p \cdot q}{2^{i+j}} \leq \frac{n}{4} \; .$$

Weiter gilt

$$p_1 \cdot q_1 \cdot \frac{2^{2i}}{3} \leq p_1 \cdot 2^i \cdot q_1 \cdot \frac{2^j}{3} = (p-1) \cdot \frac{(q-1)}{3} < \frac{n}{3}$$

also

$$p_1 \cdot q_1 \cdot \left(\frac{2}{3} + \frac{2^{2i}}{3} \right) \leq \frac{2}{3} \cdot \frac{n}{4} + \frac{n}{3} = \frac{n}{2} \; .$$

Da wir $w \in \{1, 2, \ldots, n-1\}$ gemäß der Gleichverteilung wählen, sind mindestens $(n-1)/2$ *gute Wahlen* für w dabei, also ist die Erfolgswahrscheinlichkeit mindestens $1/2$. □

8.1 Das Merkle-Hellman Knapsack Kryptosystem

Das Merkle-Hellman-Kryptosystem basiert auf dem Subset-Sum-Problem:

Gegeben: Eine Folge $(s_1, s_2, \ldots, s_n)$ natürlicher Zahlen und eine Zahl $T \in \mathbf{N}$.

Gesucht: Ein 0, 1-Vektor $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ mit $\sum_{i=1}^{n} x_i \cdot s_i = T$, falls dieser existiert.

Die Entscheidungsvariante des Subset-Sum-Problems (Gibt es einen solchen Vektor x?) ist als $\mathcal{NP}$-vollständig bekannt. Das Subset-Sum-Problem ist also ein schwieriges Problem, und ist bei $\mathcal{P} \neq \mathcal{NP}$ nicht in Polynomialzeit lösbar.

Definition 8.4 *Eine Folge $(s_1, s_2, \ldots, s_n)$ natürlicher Zahlen heißt superwachsend, wenn gilt:*

$$s_j > \sum_{i=1}^{j-1} s_i \qquad \text{für } j = 2, 3, \ldots, n \; .$$

Theorem 8.5 *Für superwachsende Folgen $(s_1, s_2, \ldots, s_n)$ kann das Subset Sum Problem in Polynomialzeit gelöst werden.*

Beweis: Offenbar ist die Folge $(s_1, \ldots, s_n)$ streng monoton steigend. Für $T \in N$ führen wir folgenden Algorithmus durch:

```
for i = n to 1 do
     if T ≥ s_i then
          T:=T − s_i
          x_i:=1
     else
          x_i:=0
if T = 0 then
     x = (x_1, ..., x_n) ist Lösung.
else
     Es gibt keine Lösung.
```

Korrektheit:

Ist $T \geq s_n$ und $T = \sum_{i=1}^{n} x_i \cdot s_i$, dann gilt:

$$T \geq s_n > s_1 + s_2 + \ldots + s_{n-1} \; ,$$

also $T = \sum_{i=1}^{n-1} x_i \cdot s_i \ + s_n$, falls das Subset Sum Problem für T lösbar ist, da T ohne s_n nicht alleine durch $s_1, \ldots, s_{n-1}$ dargestellt werden kann. Mit $T := T - s_n$ wird dieses Argument iteriert. □

Wie kann man dieses zur Chiffrierung einsetzen?

Eine Nachricht $m = (m_1, m_2, \ldots, m_n) \in \{0,1\}^n$, mit einer (öffentlich bekannten) superwachsenden Folge $s = (s_1, s_2, \ldots, s_n) \in \mathbf{N}^n$ verschlüsselt zu

$$E(m,s) = \sum_{i=1}^{n} m_i \cdot s_i \ ,$$

kann mit obigem Algorithmus leicht entschlüsselt werden, auch durch einen Angreifer. Man verwendet daher eine Transformation $t = (t_1, t_2, \ldots, t_n)$ der superwachsenden Folge $(s_1, s_2, \ldots, s_n)$.

Wir benutzen eine (geheime) superwachsende Folge $s = (s_1, s_2, \ldots, s_n) \in \mathbf{N}^n$ und eine Primzahl p mit $p > \sum_{i=1}^{n} s_i$, und setzen

$$t_i := x \cdot s_i \bmod p$$

für $i = 1, 2, \ldots, n$, wobei $x \in \mathbf{Z}_p$ geheim ist. Die Folge $t = (t_1, t_2, \ldots, t_n) \bmod p$ wird von Bob veröffentlicht.

Dann verschlüsselt Alice eine Nachricht $m = (m_1, m_2, \ldots, m_n) \in \{0,1\}^n$ mit

$$E(m,t) = \sum_{i=1}^{n} m_i \cdot t_i \bmod p$$

und Bob entschlüsselt $y \equiv \sum_{i=1}^{n} m_i \cdot t_i \bmod p$ mit

$$D(y, (p, x, (s_1, s_2, \ldots, s_n))) = (m_1, m_2, \ldots, m_n) \ ,$$

wobei $T \equiv x^{-1} \cdot y \equiv \sum_{i=1}^{n} m_i \cdot s_i \bmod p$ ist. Da $T < p$ ist, ist der Vektor $(m_1, \ldots, m_n)$ als Lösung des Subset Sum Problems für $s_1, \ldots, s_n$ und T leicht aus $x^{-1} \cdot y$ zu ermitteln.

Bemerkung: Shamir zeigte mit Hilfe des sogenannten LLL-Algorithmus (mit polynomieller Laufzeit) allerdings, dass das Merkle-Hellman-System nicht sicher ist. Superwachsende Folgen können durch Multiplikation $\bmod p$ also nicht gut „versteckt“ werden.

9 Digitale Unterschriften

Konventionelle Unterschriften sind physikalischer Teil der Nachricht und können durch Vergleich mit „authentischen Unterschriften“ verifiziert werden.

Digitale Unterschriften sind nicht „physikalisch“ mit der Nachricht verbunden. Also müssen digitale Unterschriften auf irgendeine Weise fest mit der Nachricht verknüpft werden. Die Verifikation geschieht dann über öffentlich bekannte Verifikationsalgorithmen. Der Sender hat ein Paar (k_{pub}, k_{sec}) von Schlüsseln, einen öffentlichen k_{pub}, der zur Verifikation benutzt wird und einen geheimen k_{sec}, welcher zum Unterschreiben genutzt wird.

Definition 9.1 *Ein Signaturschema (P, A, K, S, V) erfüllt folgendes:*

(a) P *= Menge von Nachrichten, z.B.* $M = \{0,1\}^*$

(b) A *= Menge von Signaturen, z.B.* $M = \{0,1\}^*$

(c) K *= endliche Menge von Schlüsseln*

(d) Für jeden Schlüssel $k \in K$ *gibt es einen Signieralgorithmus* $sig_k \in S$ *und einen entsprechenden Verifikationsalgorithmus* $ver_k \in V$*. Für die Funktionen*

$$sig_k \colon P \longrightarrow A$$

und

$$ver_k \colon P \times A \longrightarrow \{0,1\}$$

gilt:

$$ver_k(x,y) = \begin{cases} 1 & falls\ y = sig_k(x) \\ 0 & sonst \end{cases} \qquad \forall x \in D\ \forall y \in A.$$

Die Funktionen sig_k, ver_k sollten in Polynomialzeit berechenbar sein, wobei ver_k öffentlich und sig_k geheim ist. Mit den vorhandenen Ressourcen sollte nur der Signierer eine Nachricht x mit y signieren können, wobei $\mathrm{ver}_k(x,y) = 1$ gilt. Ein Dritter sollte diese Unterschrift nicht fälschen können.

Beispiel 9.2 *RSA-Signatur: Alice signiert eine Nachricht* $x \in \mathbf{Z}_n$ *mit* $sig_k \widehat{=} (\phi(n), d)$*, wobei* k *ihr geheimer Schlüssel ist. Die Verifikation durch Bob geschieht dann mit* $ver_k \widehat{=} (n, e)$*, dem öffentlichen Schlüssel von Alice. Alice nimmt also die Nachricht* $x \in \mathbf{Z}_n$ *und signiert sie durch* $y = sig_k(x) \equiv x^d \bmod n$*, die Verifikation geschieht dann mittels*

$$y^e \equiv \left(x^d\right)^e \equiv x \bmod n\ .$$

Jeder, etwa ein Richter o.ä., kann die Unterschrift von Alice auf Korrektheit prüfen.
Wenn sich nun ein Angreifer Oskar eine beliebige Signatur y wählt und die zugehörige Nachricht x mit dem öffentlichen Schlüssel von Alice zu $x \equiv y^e \bmod n$ berechnet, dann wurde x gültig signiert, da

$$sig_k(x) = x^d \equiv (y^e)^d \equiv y \bmod n$$

gilt. Da aber die Nachricht x im wesentlichen zufällig ist, so ist es für den Angreifer Oskar sehr unwahrscheinlich, eine sinnvolle und für ihn nützliche Nachricht zu signieren, und das Problem ist zunächst nicht von Bedeutung.

Wir wollen nun die Kombination von Verschlüsseln und Signieren betrachten: Alice möchte an Bob eine Nachricht x schicken. Vorausgesetzt ist hier der Einfachheit halber, dass beide den gleichen Modulus n verwenden.

1. *Möglichkeit:* Erst Signieren, dann Verschlüsseln.

   1. Alice berechnet mit ihrem geheimen Schlüssel $(\phi(n), d_{Alice}) = k$ aus dem Klartext $x \in \mathbf{Z}_n$ ihre Signatur $\mathrm{sig}_{Alice/k}(x) = y$.

   2. Anschließend berechnet sie für das Paar (x, y) (geeignet codiert) mit dem öffentlichen Schlüssel (n, e_{Bob}) von Bob das Chiffrat

      $$E_{Bob}((x, y), e_{bob}) = z$$

      und schickt z an Bob.

   3. Bob erhält z und berechnet mit seinem geheimen Schlüssel $(\phi(n), d_{Bob})$

      $$D_{Bob}(z, d_{Bob}) = (x, y)\ .$$

   4. Danach überprüft er mit der öffentlichen Funktion $\mathrm{ver}_{Alice/k}$, ob $\mathrm{ver}_{Alice/k}(x, y) = 1$ ist. Er berechnet also $y^{e_{Alice}} \bmod n$ und vergleicht das Ergebnis mit der Nachricht x.

2. *Möglichkeit:* Erst Verschlüsseln, dann Signieren.

   1. Alice berechnet mit dem öffentlichen Schlüssel (n, e_{Bob}) von Bob das Chiffrat

      $$E_{Bob}(x, e_{Bob}) = z.$$

2. Alice berechnet mit ihrem geheimen Schlüssel $k = (\phi(n), d_{Alice})$ die Signatur $\text{sig}_{Alice/k}(z) = y$ des Chiffrats und schickt (z, y) an Bob.

3. Bob erhält (z, y) und berechnet mit seinem geheimen Schlüssel $(\phi(n), d_{Bob})$

$$D_{Bob}(z) = x \ .$$

4. Danach überprüft er mit der öffentlichen Funktion $\text{ver}_{Alice/k}$, ob $\text{ver}_{Alice/k}(z, y) = 1$ ist.

Das Problem bei der zweiten Variante ist, dass ein Angreifer Oskar das Paar (z, y) abfangen und y durch y' ersetzen kann, wobei $y' = \text{sig}_{Oskar/k'}(E_{Bob}(x, e_{Bob}))$. Danach sendet er (z, y') an Bob. Bob überprüft nun die Signatur mit $\text{ver}_{Oskar/k'}$ und schließt, dass die Nachricht von Oskar kommt.
Man sollte also immer erst signieren und dann verschlüsseln!

9.1 Angriffe gegen digitale Signaturen

1. key only attack: Der Angreifer kennt nur den öffentlichen Schlüssel des Unterschreibers.

2. known signature attack: Der Angreifer kennt den öffentlichen Schlüssel sowie einige Nachricht/Unterschrift Paare, also $(m_i, \text{sig}_k(m_i))$ für $i = 1, \ldots, l$.

3. chosen message attack: Der Angreifer erhält zu von ihm gewählten Nachrichten $m_1, \ldots, m_l$ die entsprechenden Unterschriften $\text{sig}_k(m_1), \ldots, \text{sig}_k(m_l)$.

Ein Signaturangriff ist erfolgreich, wenn der Angreifer eine gültige fremde Unterschrift unter eine beliebige Nachricht setzen kann.
Das RSA-System ist nicht sicher gegen einen *known signature attack*. Der Angreifer kenne einige (mindestens zwei) Nachricht/Signatur Paare, zum Beispiel (m_1, y_1) und (m_2, y_2) mit $\text{sig}_k(m_i) = y_i$ also $y_i \equiv m_i^d \bmod n, i = 1, 2$. Dann ist

$$\begin{aligned} \text{sig}_k(m_1 \cdot m_2, d) &\equiv (m_1 \cdot m_2)^d \bmod n \\ &\equiv m_1^d \cdot m_2^d \bmod n \\ &\equiv y_1 \cdot y_2 \bmod n \end{aligned}$$

eine gültige Signatur unter der Nachricht $m_1 \cdot m_2$. Analog sieht man, dass y^{-1} eine gültige Unterschrift unter die zu m inverse Nachricht m^{-1} ist, so diese jeweils in $\mathbf{Z}_n$ existieren. Vermeiden kann

man diese Unsicherheiten, indem man statt der Nachricht m die Folge mm (jeweils als $0,1$-Folge) signiert, denn das Produkt zweier Zahlen der Form mm oder das Inverse hat im Allgemeinen nicht wieder diese Form.

9.2 Das El Gamal Kryptosystem

Das El Gamal Kryptosystem basiert auf dem Diskreten Logarithmus Problem in $\mathbf{Z}_p^*$.

Gegeben: Ein Tripel (p, a, b), wobei p Primzahl und a primitiv ist, also Erzeuger von $\mathbf{Z}_p^* \setminus \{0\}$, d.h. $\mathbf{Z}_p^* = \{a, a^2, \ldots, a^{p-1}\}$, und $b \in \mathbf{Z}_p^*$.

Gesucht: Ein x mit $0 \leq x \leq p-2$ mit $a^x \equiv b \bmod p$.

Zur Lösung dieses Problems ist kein Polynomialzeitverfahren (d.h. in Zeit $O(\text{poly}(\log p))$) bekannt. Für kryptographische Anwendungen sollte die Primzahl p mindestens etwa 800 Bits haben.
Alice möchte an Bob eine Nachricht senden. Beim El Gamal Kryptosystem ist das Tripel (p, a, b) öffentlich, während x mit $a^x \equiv b \bmod p$ geheim ist. Zum Verschlüsseln einer Nachricht $m \in Z_p^*$ wählt Alice eine zufällige Zahl $k \in \mathbf{Z}_{p-1} \setminus \{0\}$ und berechnet

$$E(m,k) = (a^k \bmod p, m \cdot b^k \bmod p) =: (y_1, y_2) \; .$$

Zum Dechiffrieren wird von Bob

$$D(y_1, y_2) = y_2 \cdot ({y_1}^x)^{-1} \bmod p$$

berechnet.

Korrektheit:

$$\begin{aligned} D(E(m,k),k) &\equiv D((a^k \bmod p, m \cdot b^k \bmod p), k) \\ &\equiv m \cdot b^k \cdot (a^{k \cdot x})^{-1} \bmod p \\ &\equiv m \cdot b^k \cdot \left((a^x)^k\right)^{-1} \bmod p \\ &\equiv m \cdot b^k \cdot (b^k)^{-1} \bmod p \\ &\equiv m \bmod p \; . \end{aligned}$$

Der Nachteil des EL Gamal Kryptosystems ist, dass das Chiffrat doppelt so lang wie der Klartext ist.

Wird zweimal der gleiche Exponent k verwendet, so kann ein Angreifer, wenn er zwei Chiffrate (y_1, y_2) und (z_1, z_2) hat und die zu (y_1, y_2) zugehörige Nachricht m_1, so kann er auch die zu (z_1, z_2) gehörige Nachricht m_2 mit $m_2 \equiv z_2 \cdot y_2^{-1} \cdot m_1 \bmod p$ berechnen.

9.3 Der Pohlig-Hellman Algorithmus

Wir wollen versuchen das folgende Problem zu lösen.

Diskreter Logarithmus Problem (DLP)

Gegeben: Ein Tripel (p, a, b), wobei p Primzahl ist, $a, b \in \mathbf{Z}_p^*$ und a primitiv in $\mathbf{Z}_p^*$ ist.

Gesucht: Das $x \in \{0, 1, \ldots, p-2\}$ mit $a^x \equiv b \bmod p$.

Trivialer Algorithmus

Für $y = 0, 1, \ldots, p-2$ berechne a^y bis gilt $a^y \equiv b \bmod p$, welches dann die Lösung $x := y$ liefert. Die Anzahl der Multiplikationen bei diesem trivialen Algorithmus ist im worst case $\Theta(p)$, also nicht polynomiell in der Eingabelänge $\Theta(\log p)$.

Der Pohlig-Hellman Algorithmus berechnet zu gegebenem Tripel (p, a, b) (p prim, $a, b \in \mathbf{Z}_p^*$ und a primitiv in $\mathbf{Z}_p^*$) ein $y \in \{0, 1, \ldots, p-2\}$ mit $y \equiv x \bmod q^c$, wobei gilt $a^x \equiv b \bmod p$ und wobei q eine Primzahl ist mit $q^l | (p-1)$, $l \geq 1$, aber $q^{l+1} \not| (p-1)$.
Kennt man alle paarweise verschiedenen Primteiler $q_1, \ldots, q_m$ von $(p-1)$, also $p - 1 = \prod_{i=1}^{m} q_i^{c_i}$, so liefert das Pohlig-Hellman Verfahren Werte $z_j \equiv x \bmod q_j^{c_j}$, $j = 1, \ldots, m$. Damit kann der Wert $x \in \mathbf{Z}_{p-1}$ mit $x \equiv z_j \bmod (p-1)$, $j = 1, \ldots, m$, mit dem Chinesischen Restsatz Theorem 6.4 leicht berechnet werden, sodass gilt $a^x \equiv b \bmod p$.
Bevor wir das Verfahren anwenden, machen wir zunächst einige Vorbemerkungen. Sei $q^l | (p-1)$ und $q^{l+1} \not| (p-1)$ für eine Primzahl q. Der gesuchte Wert y mit $y \equiv x \bmod q^l$ hat eine Darstellung der Form

$$y := \sum_{i=0}^{l-1} y_i \cdot q^i$$

mit $y_i \in \{0, 1, \ldots, p-1\}$ für $i = 0, 1, \ldots, l-1$. Man berechnet nun sukzessive die Werte $y_0, y_1, \ldots, y_{l-1}$.

Lemma 9.3

$$b^{\frac{(p-1)}{q}} \equiv a^{\frac{(p-1)y_0}{q}} \bmod p\ .$$

Beweis: Aus $a^x \equiv b \bmod p$ und $y \equiv x \bmod q^l$ folgt die Existenz von $s \in \mathbf{N} \cup \{0\}$ mit $x = y + s \cdot q^l$. Mit $b^{\frac{p-1}{q}} \equiv a^{\frac{(p-1)x}{q}} \bmod p$ wegen $a^x \equiv b \bmod p$ reicht es daher zu zeigen, dass gilt

$$a^{\frac{(p-1)(y+s\cdot q^l)}{q}} \equiv a^{\frac{(p-1)y_0}{q}} \bmod p\ .$$

Dieses ist nach dem Satz von Fermat äquivalent zu

$$\begin{aligned}
& \frac{p-1}{q} \cdot (y + s \cdot q^l) \equiv \frac{p-1}{q} \cdot y_0 \bmod (p-1) \\
\iff\ & \frac{p-1}{q} \cdot (y + s \cdot q^l - y_0) \equiv 0 \bmod (p-1) \\
\iff\ & \frac{p-1}{q} \cdot (\sum_{i=0}^{l-1} y_i \cdot q^i + s \cdot q^l - y_0) \equiv 0 \bmod (p-1) \\
\iff\ & (p-1) \cdot (\sum_{i=1}^{l-1} y_i \cdot q^{i-1} + s \cdot q^{l-1}) \equiv 0 \bmod (p-1)\ ,
\end{aligned}$$

was offenbar erfüllt ist. □

Beim Pohlig-Hellman Verfahren wird $a^{\frac{(p-1)i}{q}} \bmod p$ für $i = 0, 1, \ldots, q-1$ berechnet bis gilt $a^{\frac{(p-1)i}{q}} \equiv b^{\frac{(p-1)}{q}} \bmod p$, und dann setzt man $y_0 := i$.

Für $l = 1$ ist man fertig, sonst wird y_1 wie folgt berechnet:

Sei $b_0 := b$ und setze $b_1 := b_0 \cdot a^{-y_0}$ und $y^{(1)} := \log_a b_1 \bmod q^l$. Dann gilt $y^{(1)} = \sum_{i=1}^{l-1} y_i \cdot q^i$ und somit folgt wie in Lemma 9.3

$$b_1^{\frac{p-1}{q^2}} \equiv a^{\frac{(p-1)y_1}{q}} \bmod p.$$

Also wird $a^{\frac{(p-1)j}{q}} \bmod p$ berechnet, $j = 0, 1, \ldots, q-1$, bis gilt $a^{\frac{(p-1)j}{q}} \equiv b_1^{\frac{(p-1)}{p^2}} \bmod p$ und dann setzt man $y_1 := j$.

Das Verfahren wird fortgesetzt. Allgemein wird wie folgt vorgegangen:

Pohlig-Hellman Verfahren:

Gegeben: Ein Tripel (p, a, b) mit p prim, $a, b \in \mathbf{Z}_p^*$, a primitiv in $\mathbf{Z}_p^*$ sowie eine Primzahl q mit $q^l | (p-1)$ aber $q^{l+1} \not| (p-1)$.

Gesucht: Ein y mit $y \equiv x \bmod q^l$ und $y = \sum_{i=0}^{l-1} y_i \cdot q^i$, wobei gilt $a^x \equiv b \bmod p$.

1. Berechne $c_i = a^{\frac{(p-1)i}{q}} \bmod p$ für $i = 0, \ldots, q-1$.

2. $j := 0$ `und` $b_j := 0$

   `Solange` $j \leq l-1$ `do`

   $d := b_j^{(p-1)/q^{j+1}} \bmod p$

   `Bestimme` i `mit` $c_i \equiv d \bmod p$

   $y_j := i$

   $b_{j+1} := b_j \cdot a^{-y_i \cdot q^i} \bmod p$

   $j := j + 1$

   `Lösung ist` $(y_0, \ldots, y_{l-1})$.

Man muss maximal $q \cdot l$ Schritte durchführen, von denen jeder einzelne in Zeit $O(\text{poly}(\log p))$ durchführbar ist. Mit $q \geq 2$ und $q^l | (p-1)$ folgt $l = O(\log p)$. Gilt für jeden einzelnen Primfaktor q mit $q|(p-1)$ nun $q = O(\text{poly}(\log p))$ so löst das Pohlig-Hellman Verfahren das Diskrete Logarithmus Problem in Polynomialzeit, wenn die Primfaktorzerlegung von $(p-1)$ gegeben ist. Letzteres ist natürlich nicht unproblematisch.
Als Konsequenz haben wir:
„Ist die Primzahlzerlegung von $p-1$ „leicht“ zu finden, so ist das Diskrete Logarithmus Problem mit dem Pohlig-Hellman Verfahren „effizienz“ zu lösen.

9.4 Das El Gamal Signaturschema

Alice wählt eine Primzahl p, so dass das Diskrete Logarithmus Problem „schwierig“ zu lösen ist, weiterhin wählt sie ein primitives Element $a \in \mathbf{Z}_p^*$, sowie ein geheimes $x \in \mathbf{Z}_{p-1}^*$ und berechnet b mit $a^x \equiv b \bmod p$. Als Menge der Nachrichten wählt sie $P = \mathbf{Z}_p^*$ und als Menge der Signaturen $A = \mathbf{Z}_p^* \times \mathbf{Z}_{p-1}$. Alice veröffentlicht das Tripel (p, a, b) und hält x geheim. Zum Signieren der Nachricht $m \in \mathbf{Z}_p^*$ wählt Alice noch zufällig ein $k \in \mathbf{Z}_{p-1} \setminus \{0\}$ und berechnet

$$\text{sig}_k(m) \quad = \quad (\gamma, \delta) \; ,$$

wobei

$$\gamma \quad = \quad a^k \bmod p \quad \text{und}$$

$$\delta \quad = \quad (m - x \cdot \gamma) \cdot k^{-1} \bmod (p-1) \, .$$

Ist $k \equiv 1 \bmod (p-1)$ eine gute Wahl?

Insbesondere gilt dann

$$k \cdot \delta + x \cdot \gamma \equiv m \bmod (p-1) \, .$$

Zur Verifikation der Signatur (γ, δ) setzen wir

$$\mathrm{ver}(m, \gamma, \delta) = 1 \quad \Longleftrightarrow \quad b^{\gamma} \cdot \gamma^{\delta} \equiv a^{m} \bmod p \, .$$

Korrektheit:

$$\begin{aligned} b^{\gamma} \cdot \gamma^{\delta} &\equiv (a^{x})^{\gamma} \cdot \gamma^{\delta} \bmod p \\ &\equiv a^{x\gamma} \cdot \gamma^{\delta} \bmod p \\ &\equiv a^{m-k\delta} \cdot (a^{k})^{\delta} \bmod p \\ &\equiv a^{m-k\delta} \cdot a^{k\delta} \bmod p \\ &\equiv a^{m} \bmod p \, . \end{aligned}$$

Sicherheitsaspekte:

(a) Oskar wählt ein $\gamma \in \mathbf{Z}_p^*$ und möchte ein entsprechendes $\delta \in \mathbf{Z}_{p-1}$ finden, um die Unterschrift einer Nachricht $m \in \mathbf{Z}_p^*$ zu fälschen. Er versucht die Gleichung

$$\delta \equiv \log_{\gamma}(a^{m} \cdot b^{-\gamma}) \bmod p$$

zu lösen, was aber das Berechnen des diskreten Logarithmus ist.

(b) Oskar wählt $\delta \in \mathbf{Z}_{p-1}$ und möchte ein $\gamma \in \mathbf{Z}_p^*$ finden, um die Unterschrift zur Nachricht m zu fälschen. Dazu muss er

$$b^{\gamma} \cdot \gamma^{\delta} \equiv a^{m} \bmod p$$

lösen, wofür kein effizientes Verfahren bekannt ist.

(c) Oskar wählt $\delta \in \mathbf{Z}_{p-1}$ und $\gamma \in \mathbf{Z}_p$ und möchte eine passende Nachricht m finden:

$$m \equiv \log_{a}(b^{\gamma} \cdot \gamma^{\delta}) \bmod p \, ,$$

was das Berechnen des diskreten Logarithmus erfordert.

(d) Wird zufällig zweimal das gleiche $k \in \mathbf{Z}_{p-1}$ zum Signieren der Nachrichten $m_1 \in \mathbf{Z}_p^*$ und $m_2 \in \mathbf{Z}_p^*$ verwendet, so ergibt sich (γ, δ_1) als Signatur unter m_1 und (γ, δ_2) als Signatur unter m_2. Weiterhin gilt

$$\begin{aligned} b^\gamma \cdot \gamma^{\delta_1} &\equiv a^{m_1} \bmod p \\ b^\gamma \cdot \gamma^{\delta_2} &\equiv a^{m_2} \bmod p \end{aligned}$$

und mit dem Satz von Euler Theorem 6.1, da $a \in \mathbf{Z}_p^*$ ist, erhalten wir

$$\begin{aligned} \gamma^{\delta_2 - \delta_1} &\equiv a^{m_2 - m_1} \bmod p \\ a^{k \cdot (\delta_2 - \delta_1)} &\equiv a^{m_2 - m_1} \bmod p \\ k \cdot (\delta_2 - \delta_1) &\equiv m_2 - m_1 \bmod (p-1) \; . \end{aligned}$$

Nun kann $g = \text{ggT}\ (\delta_2 - \delta_1, p-1)$ mit dem Euklidischen Algorithmus berechnet werden. Dann gilt:

$$k \cdot \frac{\delta_2 - \delta_1}{g} \equiv \frac{m_2 - m_1}{g} \bmod \frac{p-1}{g}$$

und man kann eine Zahl z berechnen, so dass $z \equiv k \bmod {}^{(p-1)}/_g$ ist. Nun gilt

$$k \equiv z + l \cdot \frac{p-1}{g} \bmod (p-1) \qquad l = 0, 1, \ldots, g-1 \; .$$

Also kann k durch Probieren für $l = 0, 1, \ldots, g-1$ gefunden werden und dieses mittels des Tests, ob $\gamma \equiv a^k \bmod p$ gilt, überprüft werden.

Was kann ein Angreifer machen, wenn dreimal das gleiche $k \in \mathbf{Z}_{p-1}$ verwendet wird?

Im El Gamal Kryptosystem kann man anstelle von Restklassenringen $\mathbf{Z}_p$ beliebige endliche Gruppen $(G, \circ)$ benutzen. Man verwendet dann ein Element $a \in G$, so dass das Diskrete Logarithmus Problem in der Untergruppe $U = \{\ a^0, a^1, a^2, \ldots\}$ vermutlich schwierig zu lösen ist. Sodann wählt man ein geheimes $x \in \mathbf{N}$ mit $1 < x < |U|$ und fixiert $b \in U$ mit $a^x = b$. Für eine Nachricht $m \in G$ und eine zufällig gewählte Zahl $k \in \mathbf{Z}_{|U|}$ wird wie folgt verschlüsselt:

$$E(m, k) = (y_1, y_2)$$

mit

$$y_1 = a^k \quad \text{und} \quad y_2 = m \circ b^k \; .$$

Die Entschlüsslung erfolgt dann durch

$$D((y_1, y_2), x) = y_2 \circ (y_1^x)^{-1} .$$

Die algorithmische Schwierigkeit des Diskreten Logarithmus Problems in einer Gruppe $(G, \circ)$ hängt von der Darstellung der Gruppe $(G, \circ)$ ab. So ist das Diskrete Logarithmus Problem in $(\mathbf{Z}_n, +)$ einfach lösbar, man muss nur die Kongruenz $x \cdot a \equiv b \bmod n$ lösen, also schnell durchführbar.
Dagegen scheint das Diskrete Logarithmus Problem in $(\mathbf{Z}_n^*, \cdot)$ schwierig zu sein, obwohl es einen kanonischen Homomorphismus gibt, d.h. $a^{x+y} = a^x \cdot a^y$.
Algorithmisch schwierig ist das Diskrete Logarithmus Problem

- für endliche Körper $GF(q)$ (Galois Körper), multiplikativ
- sowie die Grupppe einer elliptischen Kurve über einem endlichen Körper.

9.5 Endliche Körper

Man weiß aus der Algebra, dass es zu jeder Primzahlpotenz $q = p^l$ einen (und bis auf Isomorphie genau einen) endlichen Körper mit q Elementen gibt. Weitere endliche Körper existieren nachweislich nicht.
Diese eindeutig bestimmten Galois Körper $GF(q)$ mit q Elementen, $q = p^l$ mit p Primzahl, konstruiert man wie folgt. Man identifiziert die Elemente mit Polynomen aus $\mathbf{Z}_p[x]$, also alle Koeffizienten sind aus $\mathbf{Z}_p$, wobei man Vielfache eines festen Polynoms $q(x) \in \mathbf{Z}_p[x]$ weglässt. Man betrachtet also den Restklassenring $\mathbf{Z}_p[x]_{q(x)}$ in Analogie zu $\mathbf{Z}_n$, geschrieben jedoch als $\mathbf{Z}_p[x]/q(x)$.
Das Polynom $q(x)$ hat hierbei für $q = p^l$ den Grad l und ist *irreduzibel* über $\mathbf{Z}_p[x]$, d.h. nicht in nichttriviale Faktoren zerlegbar.

Beispiel 9.4 *Wir wollen $GF(8)$ konstruieren. Wegen $8 = 2^3$ suchen wir zuerst ein irreduzibles Polynom $q(x) \in \mathbf{Z}_2[x]$ über $\mathbf{Z}_2$. Kandidaten hierfür sind Polynome vom Grad 3 mit konstantem Term 1, ansonsten hat man sofort x als Faktor (wie z.B. in $x^3 + x^2 + x = x \cdot (x^2 + x + 1)$), also sind mögliche Kandidaten:*

$$\begin{aligned} q_1(x) &= x^3 + 1 \\ q_2(x) &= x^3 + x + 1 \end{aligned}$$

$$\begin{aligned} q_3(x) &= x^3 + x^2 + 1 \\ q_4(x) &= x^3 + x^2 + x + 1 \,. \end{aligned}$$

Wegen $x^3 + 1 = (x+1) \cdot (x^2 + x + 1)$ *und* $x^3 + x^2 + x + 1 = (x+1) \cdot (x^2 + 1)$ [*man arbeitet über* $\mathbf{Z}_2$*!*], *verbleiben die irreduziblen Polynome* $q_2(x)$ *und* $q_3(x)$.
Wir nehmen etwa das Polynom $q(x) := q_3(x) = x^3 + x^2 + 1$. *Die Elemente von* $GF(8)$ *sind dann* $\mathbf{Z}_2[x]/q(x)$ *bzw.:*

$$0, 1, x, x+1, x^2+1, x^2+x+1, x^3+1, x^3+x+1$$

Warum findet sich nicht das Element x^2 *in der Aufzählung? Nun, es gilt*

$$x^2 + q_3(x) = x^3 + x^2 + x^2 + 1 = x^3 + 1 \,,$$

also ist es schon mit $x^3 + 1$ *vertreten.*
Addition zweier Elemente in $GF(8)$ *geschieht komponentenweise. Multiplikation erfolgt wie in* $\mathbf{Z}_n$, *wobei wir statt n das Polynom* $q_3(x)$ *herausrechnen, also z.B:*

$$\begin{aligned} (x^2+x+1)(x^3+1) &= (x^2+x+1) \cdot x^2 \\ &= (x^3 + x^2 + x) \cdot x \\ &= (x^3 + x^2 + 1 + (x+1)) \cdot x \\ &= x^2 + x \\ &= x^3 + x + 1 \,. \end{aligned}$$

9.6 Elliptische Kurven

Für Primzahlen $p > 3$ ist die Elliptische Kurve $y^2 = x^3 + ax + b$ über $\mathbf{Z}_p$ (mit $a, b \in \mathbf{Z}_p$ konstant) die Menge aller Lösungen $(x, y) \in (\mathbf{Z}_p)^2$ mit

$$y^2 = x^3 + ax + b \,,$$

wobei $4a^3 + 27b^2 \not\equiv 0 \bmod p$ gilt.
Wir nehmen zu allen Lösungen $(x, y) \in (\mathbf{Z}_p)^2$ einen speziellen Punkt O (Punkt „im Unendlichen“) hinzu und erhalten mit nachfolgenden Operationen eine Gruppe mit Operation „+ “. Der Punkt O hat die Funktion eines *neutralen Elements* mit $P + O = O + P$. Seien $P = (x_1, y_1) \in (\mathbf{Z}_p)^2$ und $Q = (x_2, y_2) \in (\mathbf{Z}_p)^2$ Punkte der elliptischen Kurve. Dann gilt:

- Für $x_1 = x_2$ und $y_1 = -y_2$ ist $P + Q =: O$.
- Ansonsten für $x_1 \neq x_2$ oder $y_1 \neq -y_2$ ist $P + Q := (x_3, y_3)$ mit

$$\begin{aligned} x_3 &:= \lambda^2 - x_1 - x_2 \\ y_3 &:= \lambda \cdot (x_1 - x_3) - y_1 \end{aligned}$$

  wobei gilt

$$\lambda = \begin{cases} \frac{y_1 - y_2}{x_2 - x_1} & \text{für } P \neq Q \\ \frac{3x_1^2 + a}{2y_1} & \text{für } P = Q \, . \end{cases}$$

  Offenbar ist zum Punkt $P = (x, y)$ der inverse Punkt $P^{-1} = -(x, y) = (x, -y)$.

Bei praktischen Anwendungen werden heutzutage zunehmend elliptische Kurven benutzt.
Frage: Was ist das Diskrete Logarithmus Problem für eine elliptische Kurve?

9.7 Hashing

Bei den bisherigen Signaturverfahren wurden Dokumente $x \in \mathbf{Z}_n$ (geeignet kodiert) mit einer Signatur $\text{sig}(x) \in \mathbf{Z}_n$ versehen, Signatur und Dokument haben also die gleiche Länge, was bei langen Dokumenten natürlich unvorteilhaft ist.
Eine Aufteilung des Dokuments in Blöcke, wobei jeder Block signiert wird, ist kein guter Ausweg, da ein Angreifer Blöcke entfernen oder umstellen kann. Als Ausweg verwendet man Hash-Funktionen, z.B.

$$h\colon \{0,1\}^N \longrightarrow \{0,1\}^n \, ,$$

wobei N sehr groß im Vergleich zu n ist, $N \gg n$.

Attacke I:
Angenommen, ein Angreifer Oskar habe eine gültige Unterschrift (x, y) (hier ist x das Dokument und y die Signatur von dem Hashwert $h(x)$). Oskar versucht nun in einem Angriff ein anderes Dokument x' zu finden mit $h(x') = h(x)$. Das Nachrichtenpaar (x, x') ergibt dann eine Kollision. Hat er Erfolg, so ist (x', y) eine gültige Unterschrift. Wie kann man versuchen, diesen Angriff zu verhindern?

Definition 9.5 *Sei x eine Nachricht. Eine Hash-Funktion h ist schwach kollisions-frei für x, falls es mit den vorhandenen Ressourcen nicht möglich ist, eine andere Nachricht $x' \neq x$ zu finden mit $h(x') = h(x)$.*

Attacke II:

Ein Angreifer Oskar erhält zwei Dokumente x und x' und $x \neq x'$ mit $h(x) = h(x')$. Oskar gibt Dokument x an Bob und überredet ihn, das Dokument x zu signieren, etwa den Hashwert von x mit y. Dann ist die Unterschrift (x', y) eine Fälschung.

Definition 9.6 *Eine Hash-Funktion h heißt stark kollisons-frei, wenn es mit den vorhandenen Ressourcen nicht möglich ist, verschiedene Nachrichten x und x' zu finden mit $h(x) = h(x')$.*

Beispiel 9.7 *Die Funktion $h\colon \mathbf{Z}_p^n \longrightarrow \mathbf{Z}_p$ mit $h(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i \bmod p$ ist nicht stark kollisions-frei, denn $x = (x_1, \ldots, x_n)$ sowie $x' = (x_1 - 1, x_2 + 1, x_3, \ldots, x_n)$ liefern $h(x) = h(x')$.*

Wir betrachten nun eine Hashfunktion $h\colon \{0,1\}^N \longrightarrow \{0,1\}^n$ mit $N > n$. Die Anzahl K an Kollisionen lässt sich dann wie folgt mit der Cauchy-Schwartz Ungleichung (für $x_1 + \ldots + x_l = m$, $x_1, \ldots, x_l \geq 0$, ist $\sum_{i=1}^{l} x_i^2 \geq m^2/l$) nach unten abschätzen

$$\begin{aligned} K &= \sum_{y \in \{0,1\}^n} \binom{|h^{-1}(y)|}{2} \\ &= \sum_{y \in \{0,1\}^n} \frac{|h^{-1}(y)|^2}{2} - \frac{1}{2} \cdot \sum_{y \in \{0,1\}^n} |h^{-1}(y)| \\ &\geq \frac{1}{2} \cdot \frac{2^{2N}}{2^n} - \frac{1}{2} \cdot 2^N \\ &= 2^{2N-n-1} - 2^{N-1}\,, \end{aligned}$$

also gibt es für $N \gg n$ sehr viele Kollisionen.

Wie findet man diese? Eine Möglichkeit hierzu ist ein Birthday Attack:

Wir nehmen an, dass für die Hashfunktion $h\colon \{0,1\}^N \longrightarrow \{0,1\}^n$ gilt $(h^{-1}(y)) = (h^{-1}(y'))$ $\forall y, y' \in \{0,1\}^n$

Wähle zufällig Nachrichten $x_1, \ldots, x_l \in \{0,1\}^N$ und berechne die Hashwerte $h(x_1), \ldots, h(x_l)$. Dann gilt

$$\begin{aligned}
\text{Prob}(\exists \text{ keine Kollision}) &= \prod_{i=0}^{l-1} \frac{2^n - i}{2^n} \\
&= \prod_{i=0}^{l-1} \left(1 - \frac{i}{2^n}\right) \\
&\leq \prod_{i=0}^{l-1} e^{-\frac{i}{2^n}} \\
&= e^{-\prod_{i=1}^{l-1} \frac{i}{2^n}} \\
&= e^{-\frac{l(l-1)}{2^n}} ,
\end{aligned}$$

also $\text{Prob}(\exists \text{ Kollision}) \geq 1 - e^{-\frac{l(l-1)}{2^n}}$.
Für $l \approx c \cdot 2^{n/2}$ für eine Konstante $c > 0$ folgt

$$\text{Prob}(\exists \text{ Kollision}) \geq 1 - e^{-c}$$

und diese Wahrscheinlichkeit ist für großes c nahe bei 1, man hat also mit $c \cdot 2^{n/2}$ zufällig gewälten Werten eine gute Chance, eine Kollision zu finden.

Chaum-van Heigst-Pfitzmann Hashfunktion

Diese Hashfunktion beruht auf dem Diskreten Logarithmus Problem.

- Man wählt eine große Primzahl p, so dass $q = (p-1)/2$ ebenfalls eine ungerade Primzahl ist. (Dieses ist zunächst nicht trivial!) Weiter werden zwei primitive Elemente $\alpha, \beta \in \mathbf{Z}_p^*$ gewählt, mit $\alpha \neq \beta$. Der Wert x mit $\alpha^x \equiv b \bmod p$ bleibt geheim.

- Die Hashfunktion $h\colon \{0, \ldots, q-1\}^2 \longrightarrow \mathbf{Z}_p^*$ ist definiert als

$$h(x_1, x_2) = \alpha^{x_1} \cdot \beta^{x_2} \bmod p \, .$$

Für diese Hashfunktion ist es vermutlich schwer, Kollisionen zu finden wie der folgende Satz zeigt.

Theorem 9.8 *Wenn eine Kollision für die Chaum-van Heigst-Pfitzmann Hashfunktion h gegeben ist, dann ist $x = \log_\alpha \beta \bmod p$ effizient berechenbar.*

Beweis: Gegeben ist eine Kollision, also zwei Paare $(x_1, x_2) \neq (x_3, x_4)$ mit

$$h(x_1, x_2) = h(x_3, x_4) \; .$$

Dann gilt

$$\begin{aligned} & \alpha^{x_1} \cdot \beta^{x_2} \equiv \alpha^{x_3} \cdot \beta^{x_4} \bmod p \\ \iff \quad & \alpha^{x_1 - x_3} \equiv \beta^{x_4 - x_2} \bmod p \; . \end{aligned}$$

Wir setzen $d := \text{ggT}\,(x_4 - x_2, p-1)$. Aus $p - 1 = 2 \cdot q$ und q prim folgt $d \in \{1, 2, q, p-1\}$. Entsprechend der möglichen Werte von d unterscheiden wir vier Fälle.

Fall 1: Sei $d = 1$. Dann existiert das Inverse von $(x_4 - x_2) \bmod (p-1)$ und mit $y := (x_4 - x_2)^{-1} \bmod (p-1)$ folgt

$$\begin{aligned} \beta &\equiv \beta^{(x_4 - x_2) \cdot y} \bmod p \\ &\equiv \alpha^{(x_1 - x_3) \cdot y} \bmod p \; , \end{aligned}$$

also

$$\log_\alpha \beta \equiv (x_1 - x_3) \cdot y \bmod p \; .$$

Man kann das Problem in diesem Fall also mit dem Euklidischen Algorithmus lösen (Bestimmung von y).

Fall 2: Sei $d = 2$. Mit $p - 1 = 2q$ und q ungerade ergibt sich $\text{ggT}\,(x_4 - x_2, q) = 1$. Mit $y := (x_4 - x_2)^{-1} \bmod q$, gilt

$$(x_4 - x_2) \cdot y = l \cdot q + 1 \qquad \text{für ein } l \in \mathbf{Z},$$

und es folgt mit $\beta^q \equiv -1 \bmod d$:

$$\begin{aligned} \alpha^{(x_1 - x_3) \cdot y} &\equiv \beta^{(x_4 - x_2) \cdot y} \bmod p \\ &\equiv \beta^{l \cdot q} \cdot \beta \bmod p \\ &\equiv (\beta^q)^l \cdot \beta \bmod p \\ &\equiv (-1)^l \cdot \beta \bmod p \end{aligned}$$

also

$$\begin{aligned} \log_\alpha \beta &\equiv (x_1 - x_3) \cdot y \bmod (p-1) \qquad \text{oder} \\ \log_\alpha \beta &\equiv (x_1 - x_3) \cdot y + q \bmod (p-1)\,. \end{aligned}$$

Durch Einsetzen von x in $\alpha^x \equiv \beta \bmod p$ kann dann leicht die richtige Lösung ermittelt werden.

Fall 3: Angenommen, es ist $d = q = (p-1)/2$. Mit $0 \le x_2, x_4 \le q-1$ folgt $-(q-1) \le x_4 - x_2 \le q-1$, also ist $d = q$ nicht möglich.

Fall 4: Für $d = p-1$ folgt $x_4 = x_2$. Mit $\alpha^{x_1} \cdot \beta^{x_2} \equiv \alpha^{x_3} \cdot \beta^{x_4} \bmod p$ ergibt sich $\alpha^{x_1} \equiv \alpha^{x_3} \bmod p$, also $x_1 = x_3$ da α primitiv ist. Nach Annahme war jedoch $(x_1, x_2) \neq (x_3, x_4)$. □

9.7.1 Extension von Hashfunktionen

Bisher haben wir Hashfunktionen der Art $h\colon \{0,1\}^N \longrightarrow \{0,1\}^n$ mit $N > n$ und festen Werten N und n betrachtet. Wir wollen nun die Funktion h ausdehnen, indem wir die Hashfunktionen für beliebige endliche 0, 1-Folgen definieren, also Hashfunktionen der Art $h^*\colon \{0,1\}^* \longrightarrow \{0,1\}^n$ betrachten.
Hierzu zerlegen wir einen String $x \in \{0,1\}^*$ mit Länge $|x| = L > N \ge n+2$ in k Teilstrings $x_1, \ldots, x_k$, $k = \lceil L/(N-n-1) \rceil$, wie folgt:

$$\begin{aligned} |x_i| &= N - n - 1 \qquad \text{für } i = 1, \ldots, k-1 \\ |x_n| &= N - n - 1 - d\,. \end{aligned}$$

Dann bildet man die Strings $y_1, \ldots, y_k$ mit $y_i := x_i$ für $i = 1, \ldots, k-1$ und $y_k := x_k 0^d$ jeweils der Länge $N - n - 1$. Weiter sei y_{k+1} die Binärdarstellung der Zahl d. Mit $g_1 := h(0^{n+1}\, y_1)$ und $g_{i+1} := h(g_1\, 1\, y_{i+1})$, $i = 1, \ldots, k$, ist der Hashwert dann $h^*(x) := g_{k+1}$.

9.8 Schlüsselverteilung

Bei Public-Key Kryptosystemen ist der Vorteil, dass keine Schlüssel ausgetauscht bzw. vorab verteilt werden müssen. Ihr Nachteil ist jedoch, dass sie im allgemeinen im Vergleich zu symmetrischen

Kryptosystemen wie DES langsam sind. Die Verwendung symmetrischer Verfahren erscheint sinnvoll, wenn Schlüsselverteilung bzw. -austausch sicher durchgeführt werden kann.
Eine Möglichkeit der Schlüsselverteilung ist, dass ein übergeordneter Teilnehmer (Trust Center) die geheimen Schlüssel wählt und diese an die n Teilnehmer (User) verteilt. Hierbei wählt das Trust Center für jedes Paar $\{A, B\}$ von Usern einen geheimen Schlüssel $k_{A,B} = k_{B,A}$ und verteilt diesen an A und B auf einem sicheren Kanal. Problematisch ist hier zum einen der geforderte sichere Kanal sowie, dass jeder User $(n-1)$ Schlüssel speichern muss. Eine andere Möglichkeit liefert

Blom's Schema der Schlüsselverteilung:

- Bei n Usern sind die Schlüssel Elemente von $\mathbf{Z}_p$, wobei p eine Primzahl ist mit $p \geq n$, die öffentlich ist.
- Es gibt einen *Sicherheitsparameter* k, $1 \leq k \leq n-2$, der die größte Anzahl Teilnehmer einer Koalition (Austausch von Information unter diesen ist möglich) angibt, so dass das Schema noch sicher ist.
- Für jeden User A gibt es eine Zahl $r_A \in \mathbf{Z}_p$, die öffentlich ist. Diese Zahlen sind paarweise verschieden.
- Das Trust Center wählt zufällig Zahlen $a_{i,j} \in \mathbf{Z}_p$, $i = 0, \ldots, k$ und $j = 0, \ldots, k$, mit $a_{i,j} = a_{j,i}$ für $0 \leq i, j \leq k$, die das folgende Polynom in den Variablen x und y bestimmen

$$f(x, y) = \sum_{i=0}^{k} \sum_{j=0}^{k} a_{i,j} \cdot x^i \cdot y^j \ .$$

- Für jeden User A berechnet das Trust Center in $\mathbf{Z}_p$ den Wert

$$\begin{aligned} f(x, r_A) &= \sum_{i=0}^{k} \sum_{j=0}^{k} a_{i,j} \cdot x^i \cdot r_A^j \\ &= \sum_{i=0}^{k} \left(\sum_{j=0}^{k} a_{i,j} \cdot {r_A}^j \right) \cdot x^i \\ &=: \sum_{i=0}^{k} r(A, i) \cdot x^i \\ &=: g_A(x) \end{aligned}$$

und sendet dem User A das Polynom $g_A(x)$ über einen sicheren Kanal.

- Falls die User A und B kommunizieren wollen, berechnen sie jeweils $g_A(r_B)$ und $g_B(r_A)$ und erhalten den Kommunikationsschlüssel

$$k_{A,B} = k_{B,A} = f(r_A, r_B) = f(r_B, r_A) \; .$$

Der Vorteil bei Blom's Schema ist, dass jeder User je nach Sicherheitsparameter nur $(k+1)$ Elemente aus $\mathbf{Z}_p$ speichern muss.

Lemma 9.9 *Blom's Schema für $k = 1$ ist sicher bzgl. eines Angriffs jedes anderen einzelnen Users.*

Beweis: Das Trust Center habe die Funktion

$$f(x, y) = a_{0,0} + a_{1,0} \cdot (x + y) + a_{1,1} \cdot x \cdot y$$

gewählt.
Ein User $C \neq A, B$ möchte den Kommunikationsschlüssel $k_{A,B}$ mit

$$k_{A,B} = a_{0,0} + a_{1,0} \cdot (r_A + r_B) + a_{1,1} \cdot r_A \cdot r_B$$

berechnen. Hierbei sind r_A und r_B öffentlich bekannt, aber $a_{0,0}$, $a_{1,0}$ sowie $a_{1,1}$ sind unbekannt.
Der Angreifer C kennt jedoch sein Polynom

$$\begin{aligned} g_C(x) &= a(C,0) + x \cdot a(C,1) \\ &= (a_{0,0} + a_{1,0} \cdot r_C) + x \cdot (a_{1,0} + a_{1,1} \cdot r_C) \; . \end{aligned}$$

Der Angreifer nimmt nun an, dass $k(= k_{A,B})$ der Kommunikationsschlüssel von A und B ist und versucht mit folgendem Gleichungssystem $a_{0,0}$, $a_{1,0}$ und $a_{1,1}$ in $\mathbf{Z}_p$ zu bestimmen:

$$\begin{pmatrix} 1 & r_A + r_B & r_A \cdot r_B \\ 1 & r_C & 0 \\ 0 & 1 & r_C \end{pmatrix} \begin{pmatrix} a_{0,0} \\ a_{1,0} \\ a_{1,1} \end{pmatrix} = \begin{pmatrix} k \\ a(C,0) \\ a(C,1) \end{pmatrix}$$

Wir betrachten die Determinante der Matrix:

$$\begin{aligned}
& \det \begin{pmatrix} 1 & r_A + r_B & r_A \cdot r_B \\ 1 & r_C & 0 \\ 0 & 1 & r_C \end{pmatrix} \\
= \; & \det \begin{pmatrix} r_C & 0 \\ 1 & r_C \end{pmatrix} - \det \begin{pmatrix} r_A + r_B & r_A \cdot r_B \\ 1 & r_C \end{pmatrix} \\
= \; & {r_C}^2 - r_C \cdot (r_A + r_B) - r_A \cdot r_B \\
= \; & (r_C - r_A) \cdot (r_C - r_B) \\
\not\equiv \; & 0 \bmod p \,,
\end{aligned}$$

da $r_C \not\equiv r_A \bmod p$ sowie $r_C \not\equiv r_B \bmod p$. Da die Determinante ungleich Null ist, ist damit für jeden angenommenen Schlüssel k das Gleichungssystem eindeutig nach $a_{0,0}$, $a_{1,0}$ und $a_{1,1}$ auflösbar. Der Schlüssel $k_{A,B}$ kann also nicht von C identifiziert werden.

□

Zwei User C und D in einer Koalition können jedoch den Schlüssel $k_{A,B}$ berechnen für $\{C, D\} \cap \{A, B\} = \emptyset$. Sie kennen nämlich

$$a(C,0) \equiv a_{0,0} + a_{1,0} \cdot r_C \bmod p \tag{7}$$

$$a(C,1) \equiv a_{1,0} + a_{1,1} \cdot r_C \bmod p \tag{8}$$

$$a(D,0) \equiv a_{0,0} + a_{1,0} \cdot r_D \bmod p \tag{9}$$

$$a(D,1) \equiv a_{1,0} + a_{1,1} \cdot r_D \bmod p. \tag{10}$$

Aus (7) und (9) folgt

$$a(C,0) - a(C,1) \equiv a_{1,0}(r_C - r_D) \bmod p,$$

also

$$a_{1,0} \equiv (a(C,0) - a(C,1)) \cdot (r_C - r_D)^{-1} \bmod p \,.$$

Damit lässt sich $a_{0,0}$ mit (7) oder (9) leicht berechnen. Analog erhält man aus (8) und (10) den Wert von $a_{1,1}$, und damit kennen C und D das geheime Polynom $f(x, y)$ und sie können alle Schlüssel berechnen. Analog kann man zeigen:

Satz: Blom's Schema für beliebiges k ist sicher bzgl. jeder Koalition von k anderen Usern, aber unsicher bzgl. einer Koalition von $(k+1)$ Usern.

Für den Austausch eines Schlüssels zwischen *zwei* Kommunikationspartnern gibt es das

Verfahren von Diffie und Hellman:

- Operiert wird in $\mathbf{Z}_p^*$, wobei p eine Primzahl ist, oder in einer beliebigen Gruppe G, in der das Diskrete Logarithmus Problem wohl schwer zu lösen ist.
- Die beiden Partner A und B einigen sich auf ein primitives Element $x \in \mathbf{Z}_p^*$, was öffentlich bekannt sein kann.
- Jeder User A und B wählt nun einen geheimen Exponenten d_A und d_B, $1 \leq d_A, d_B \leq p-2$.
- User A berechnet $x^{d_A} \bmod p$ und schickt das Ergebnis $y \equiv x^{d_A} \bmod p$ an B. User B erhält y, berechnet $y^{d_B} \bmod p$ und schickt das Ergebnis $z \equiv y^{d_B} \equiv (x^{d_A})^{d_B} \bmod p$ an A. Der Wert $z \in \mathbf{Z}_p^*$ ist der vereinbarte Kommunikationsschlüssel, den beide kennen, ohne ihre privaten Schlüssel d_A bzw. d_B dem anderen preisgegeben zu haben.