Einführung

Kryptologie = Kryptographie (Entwicklung kryptographischer Verfahren) + Kryptoanalyse (Versuch kryptographische Verfahren zu brechen)
Die Kryptographie im engeren und ,,klassischen" Sinn beschäftigt sich mit Verfahren, um in verschiedenen Kommunikationsszenarien eine gegen Angriffe von Gegnern (Mitlesen, Verändern, Unterschieben, Abstreiten) abgesicherte Kommunikation zu ermöglichen.
Die ,,moderne" Kryptologie beschäftigt sich mit kryptographischen Verfahren, mit kryptoanalytischen Verfahren, mit Sicherheitskonzepten und mit mathematischen Methoden zur Untersuchung dieser Dinge, abstrakt und an konkreten Verfahren.
Aufgaben von Kryptosystemen

Konzelation Geheimhaltung/Vertraulichkeit/Zugriffsschutz(kein Unberechtigter kann Nachrichteninhalt mithören oder mitlesen
Integrität/Fälschungsschutz stelle sicher, dass Nachrichten auf dem Übertragungsweg nicht manipuliert worden sind
Authentizität/Signaturen Garantiere Absenderidentität; Bob kann kontrollieren, dass Nachricht vom behaupteten Absender Alice kommt
Nichtabstreitbarkeit Bob kann gegenüber Dritten beweisen,dass die Nachricht in der empfangenen Form vom behaupteten Absender Alice kam

Message Authentication Code (MAC) Funktion, die aus einer Nachricht x einen Code $mac = MAC(x)$ berechnet. Diese Funktion ist ein Geheimnis von legitimen Sendern von Nachrichten an Bob. Insbesondere kann Eva bei gegebener Nachricht x' keinen korrekten MAC für x' berechnen. Bob verfügt über ein Prüfverfahren, das es ihm erlaubt, ein empfangenes Paar (x, mac) darauf zu testen, ob der zweite Teil der zu x gehörende MAC-Wert ist. Wenn Alice die einzige Instanz ist, die die geheime Funktion MAC kennt, dann kann Bob sogar überprüfen, ob Alice tatsächlich die Absenderin ist.

Cäsar-Chiffre Cäsar ließ Texte verschlüsseln, indem man nimmt immer den Buchstaben, der im Alphabet drei Positionen ,,weiter rechts" steht, mit ,,wrap around" am Ende. Nachteil ist offensichtlich: Wer den Trick kannte, konnte jede Nachricht mitlesen. Es gibt einen Schlüssel

Verschiebechiffre Eine einfache ,,Verbesserung" der Cäsar-Chiffre: Verschiebe zyklisch um eine andere Anzahl k von Buchstaben als nur 3. Um hier die Verschlüsselung und die Entschlüsselung durchzuführen, musste man als ,,Schlüssel" nur das Bild von A kennen, alternativ die Verschiebeweite k als Zahl. Es gibt dann 21 Schlüssel.

Substitutionschiffre Sie sagt, dass das Bild eines Buchstabens ein ganz beliebiger anderer Buchstabe sein soll. Dabei müssen natürlich verschiedene Buchstaben auf verschiedene Buchstaben abgebildet werden. Es er gibt sich eine Chiffre, die durch eine Tabelle mit ganz beliebiger Buchstabenanordnung gegeben ist. Wenn man hier ver- und entschlüsseln möchte, muss man die gesamte zweite Tabellenzeile kennen. Diese kann hier also als ,,Schlüssel" dienen. Es gibt $21! \approx 5,11 * 10^{19}$ viele verschiedene Schlüssel.

Vigenère-Verschlüsselung (16.Jhd) Man benutzt ein Schlüsselwort $a_0, ..., a_{s-1}$ größerer Länge $s > 1$ über dem Alphabet $\{A, ..., X\}$. $x = x_0...x_{n-1}$ ist der Klartext, $k = a_0...a_{s-1}$ der Schlüssel. Verschlüssle x mit den durch k gegebenen Verschiebungen wie folgt: x_0 mit $A \to a_0$,x_2 mit $A \to a_2, ..., x_{s-1}$ mit $A \to a_{s-1}$. Wenn der Schlüssel k aufgebraucht ist, benutze ihn wieder von vorne: Verschlüssle $x_i, i \geq s$, mit Verschiebung $A \to a_{i \ mod \ s}$.

kryptographische Verfahren mit Schlüsseln

Symmetrisch (Private-Key) Es gibt einen geheimen Schlüssel k, den beide der kommunizierenden Parteien kennen müssen. Bei Konzelationssystemen bedeutet dies etwa, dass das Verschlüsselungsverfahren und das Entschlüsselungsverfahren beide diesen Schlüssel $k = k'$ benutzen. Bsp. AES (Advanced Encryption Standard).

Asymmetrisch (Public-Key) Nur eine Seite hat einen geheimen Schlüssel k', die andere Seite benutzt einen ,,öffentlichen" Schlüssel $k \neq k'$. Bsp. RSA

Das Kerckhoffs-Prinzip (1883) besagt,dass man davon ausgehen muss, dass Eva die Struktur des Verschlüsselungsverfahrens kennt und die Sicherheit nur von der Geheimhaltung des Schlüssels abhängen darf.
1. Geheimhaltung eines Verfahrens ist schwer sicherzustellen
2. Verfahren sind aufwendig zu entwickeln. Ist das geheime Verfahren einmal bekannt, soist es nutzlos. (Mehrfach passiert: Enigma, GSM-Verfahren ,Stromchiffre RC4)
3. Allgemein bekannte Verfahren können von mehr Experten auf ,,Sicherheit" geprüft werden. Findet niemand einen erfolgreichen Angriff, so kann man eher auf Sicherheit des Verfahrens vertrauen
4. Nur offen gelegte Verfahren können standardisiert werden und weite Verbreitung finden (DES, AES)

Symmetrische Sicherheitsmodelle

Angriffsszenarien/Bedrohungsszenarien

1. ciphertext-only attack (COA): nur mithören
2. known-plaintext attack (KPA): Paare von Klartext und Chiffretext bekannt
3. chosen-plaintext attack (CPA): einige von Eva gewählte Klartexte verschlüsseln
4. chosen-cyphertext attack (CCA): einige von Eva gewählte Chiffretexte entschlüsseln
5. Eva hat hier Möglichkeiten 3. + 4.

Fähigkeiten von Eva (unter anderem)

1. informationstheoretische Sicherheit: unbegrenzte Rechenkapazitäten
2. Konkrete Sicherheit: maximale Anzahl an Rechenoperationen (z.B. 2^{60}) und begrenzter Speicher (z.B. 1000 TB)
3. Im Design des Verschlüsselungsverfahrens gibt es einen Stellhebel, einen ,,Sicherheitsparameter". Je nach Leistungsfähigkeit von Eva kann man durch entsprechende Wahl dieses Parameters die Sicherheit des Systems an eine gegebene (geschätzte) Rechenzeitschranke anpassen.
4. Asymptotische Sicherheit: wenn asymptotisch der Rechenzeitaufwand für Eva zum Brechen des Systems schneller als polynomiell wächst, kann man sagen, dass sie für genügend lange Texte keine Chance mehr hat, das System erfolgreich zu brechen

symmetrische Konzelationssysteme, mit steigender Komplexität. Alice und Bob haben sich auf einen Schlüssel geeinigt.

1. Einmalige Verschlüsselung: Ein einzelner Klartext x vorher bekannter Länge wird übertragen, Eva hört mit (COA)
   - Unvermeidlich: Triviale Information, z.B. dass eine Nachricht übertragen wurde
   - Vermeiden Eva erhält nicht-triviale Information, z.B. Klartext= x oder Klartext weder x_1 noch x_2
   - Gegenstand der Steganographie sind Verfahren, Nachrichten so zu übertragen, dass noch nicht einmal die Existenz der Nachricht entdeckt werden kann.
2. Frische Verschlüsselung: Mehrere Klartexte vorher bekannter Länge werden übertragen, Eva hört mit, kann sich einige Klartexte verschlüsseln lassen (CPA).
   - Triviale Information: z.B. Anzahl der Nachrichten oder Klartext, falls Eva sich zufälligerweise vorher den ,,richtigen" Klartext hat verschlüsseln lassen.
3. Uneingeschränkte symmetrische Verschlüsselung: Mehrere Klartexte verschiedener Länge, Eva hört mit, kann sich einige Klartexte verschlüsseln lassen (CPA).
   - Triviale Information: Analog zur frischen Verschlüsselung

Kryptosysteme und possibilistische Sicherheit

Definition 1.1 Ein Kryptosystem ist ein Tupel $S = (X, K, Y, e, d)$, wobei

- X und K nicht leere endliche Mengen sind [Klartexte bzw. Schlüssel],
- Y eine Menge ist [Chiffretexte], und
- $e : X \times K \to Y$ und $d : Y \times K \to X$ Funktionen sind [Verschlüsselungsfunktion bzw. Entschlüsselungsfunktion],

so dass Folgendes gilt:

1. $\forall x \in X \forall k \in K : d(e(x, k), k) = x$ (Dechiffrierbedingung)
2. $\forall y \in Y \exists x \in X, k \in K : y = e(x, k)$ (Surjektivität)

Bemerkung: Surjektivität kann immer hergestellt werden, indem man Y auf das Bild $Bi(e) = e(X \times K)$ einschränkt. Die Forderung ist für unsere Analysen aber bequem.
Für festes $k \in K$ wird die Funktion $e(., k) : X \to Y, x \to e(x, k)$ als Chiffre bezeichnet.
Beispiel $X = \{a, b\}, K = \{k_0, k_1, k_2\}, Y = \{A, B, C\}$. Die Funktion e ist gegeben durch die erste, die Funktion d durch die zweite der folgenden Tabellen. Dann ist (X, K, Y, e, d) Kryptosystem, denn die Dechiffrierbedingung und die Surjektivität sind erfüllt.

e	a	b
k_0	A	B
k_1	B	A
k_2	A	C

d	A	B	C
k_0	a	b	a
k_1	b	a	a
k_2	a	a	b

Jede Chiffre $e(., k)$ eines Kryptosystems muss injektiv sein. (Die Einträge in jeder Zeile der Tabelle für e müssen verschieden sein.)
Das Vernam-Kryptosystem oder one-time pad der Länge l ist das Kryptosystem $(\{0,1\}^l, \{0,1\}^l, \{0,1\}^l, \oplus_l, \oplus_l)$. Benannt nach Gilbert S. Vernam (1890, 1960), der im Jahr 1918 dieses System für fünf Bits in der Sprache einer Relais-Schaltung beschrieben und zum US-Patent angemeldet hat. In diesem Fall ist es für nicht ganz kleine l offensichtlich unbequem, wenn nicht ganz unmöglich, die Ver-und Entschlüsselungsfunktion durch Tabellen anzugeben. Man benutzt hier und auch üblicherweise mathematische Beschreibungen.
Beispiel: $x = 1011001, k = 1101010$. Dann ist $y = e(x, k) = 1011001 \oplus_7 1101010 = 0110011$. Zur Kontrolle: $d(y, k) = 0110011 \oplus_7 1101010 = 1011001 = x$.
Wir kontrollieren dass das Vernam-System tatsächlich ein Kryptosystem ist.

1. Für $x \in X$ und $k \in K$ gelten $d(e(x, k), k) = (x \oplus_l k) \oplus_l k = x \oplus_l (k \oplus_l k) = x \oplus_l 0^l = x$, d.h. die Dechiffrierbedingung ist erfüllt.
2. Für $y \in Y$ gilt $e(y, 0^l) = y$ und $y \in X, 0^l \in K$. Also gilt Surjektivität.

Definition Ein Kryptosystem $S = (X, K, Y, e, d)$ heißt possibilistisch sicher, wenn $\forall y \in Y \forall x \in X \exists k \in K : e(x, k) = y$.

1. Sei $S = (X, K, Y, e, d)$ Kryptosystem. Dann sind äquivalent:
   - S ist possibilistisch sicher.
   - $\forall x \in X : e(x, K) = \{e(x, k) | k \in K\} = Y$.
2. Das Vernam-Kryptosystem der Länge l ist possibilistisch sicher: Seien $x \in X$ und $y \in Y$. Setze $k = x \oplus_l y$.Dann gilt $e(x, k) = x \oplus_l (x \oplus_l y) = (x \oplus_l x) \oplus_l y = 0^l \oplus_l y = y$.

Definition Für eine endliche, nichtleere Menge X sei $K = P_X$ die Menge der Permutationen (bijektive Funktion $\pi : X \to X$) auf X. Das Substitutionskryptosystem auf X ist das Tupel (X, PX, X, e, d) mit $e(x, \pi) = \pi(x)$ und $d(y, \pi) = \pi^{-1}(y)$. Wenn $\pi : X \to X$ eine Permutation ist, dann ist $\pi^{-1} : X \to X$ die Permutation mit $\pi^{-1}(\pi(x)) = \pi(\pi^{-1}(x)) = x$ für alle $x \in X$.

Bei possibilistischer Sicherheit von Klartexten und Schlüsseln, die Zeichenreihen über einem Alphabet sind, müssen Schlüssel mindestens so lang sein wie der zu übermittelnde Text. unrealistisch → Possibilistisch sichere Systeme kommen daher nur als Bausteine in größeren Systemen vor.

Elementare Wahrscheinlichkeitsrechnung

Definition: Ein (diskreter) Wahrscheinlichkeitsraum ist ein Paar (Ω, Pr), wobei

- Ω eine nichtleere endliche oder abzählbar unendliche Menge und
- $Pr : P(\Omega) \to [0,1]$ eine Abbildung ($P(\Omega) = \{A | A \subseteq \Omega\}$ ist die Potenzmenge)

Definition Sei (Ω, Pr) ein Wahrscheinlichkeitsraum und seien A, B Ereignisse. Dann heißen A und B unabhängig, wenn $Pr(A \cap B) = Pr(A) * Pr(B)$ gilt.
Definition Sei (Ω, Pr) ein Wahrscheinlichkeitsraum und R eine endliche oder abzählbare Menge. Eine Zufallsvariable ist eine Abbildung $X : \Omega \to R$. Zufallsvariablen mit $R \subseteq R$ heißen reelle Zufallsvariable.

Informationstheoretische Sicherheit

Wenn sich durch das Beobachten einer Chiffre die Meinung von Eva über die Wahrscheinlichkeiten der verschiedenen Klartexte von der ursprünglichen Verteilung unterscheidet, hat Eva aus der Beobachtung von y eine gewisse Information erhalten.
In das math. Modell $\Omega = X \times K$ bauen wir die Vorstellung ein, dass $x \in X$ und $k \in K$ nach den Verteilungen Pr_X (Teil der Anwendung) und Pr_K (Teil des Kryptosystems) zufällig und unabhängig gewählt werden. Die Verteilung Pr_X braucht beim Entwurf des Kryptosystems nicht einmal bekannt zu sein.
Definition Ein Kryptosystem mit Schlüsselverteilung (KSV) ist ein 6-Tupel $V = (X, K, Y, e, d, Pr_K)$, wobei

- $S = (X, K, Y, e, d)$ das zugrundeliegende Kryptosystem ist
- $Pr_K : K \to (0, 1]$ die Schlüsselverteilung
- Für $V = (X, K, Y, e, d, Pr_K)$ schreiben wir auch $S[Pr_K]$
- $Pr_K(k) \in (0, 1]$ also $Pr_K(k) > 0$ für alle $k \in K$
- weiter $Pr_X : X \to [0, 1]$ Klartextverteilung
- $Pr : X \times K \to [0, 1]$ durch $Pr((x, k)) := Pr_X(x) * Pr_K(k)$
- Annahme modelliert, dass der Schlüssel k unabhängig vom Klartext durch ein von Pr_K gesteuertes Zufallsexperiment gewählt

Beispiel Sei $X = \{a, b, c\}, K = \{0, 1, 2, 3\}, Y = \{A, B, C\}$ und die Verschlüsselungsfunktion sei durch die folgende Tabelle gegeben:

e	a(0,4)	b(0)	c(0,6)
0 ($\frac{1}{4}$)	A	B	C
1 ($\frac{1}{8}$)	B	C	A
2 ($\frac{1}{2}$)	C	A	B
3 ($\frac{1}{8}$)	C	B	A

Die Wahrscheinlichkeiten $Pr_X(x)$ sind bei den Klartexten, die Wahrscheinlichkeiten $Pr_K(k)$ bei den Schlüsseln in Klammern notiert. Klartexte a und c sind aktiv, Klartext b ist passiv.
Einige einfache Zusammenhänge, für $x_0 \in X, k_0 \in K, y_0 \in Y$

- $X : X \times K \to X, (x, k) \to x$
- $X : X \times K \to K, (x, k) \to k$
- $Pr(x_0) := Pr(\{x_0\} \times K) = Pr_X(x_0) * Pr_K(K) = Pr_X(x_0)$.
- $Pr(k_0) := Pr(X \times \{k_0\}) = Pr_X(X) * Pr_K(k_0) = Pr_K(k_0)$
- Man erhält die ursprünglichen Wahrscheinlichkeiten für Klartexte und Schlüssel zurück
- Im Bsp $Pr(A) = \frac{1}{4} * 0,4 + \frac{1}{8} * 0,6 + \frac{1}{2} * 0 + \frac{1}{8} * 0,6 = 0,25$
- Im Bsp $Pr(c, A) = 0,6 * (\frac{1}{8} + \frac{1}{8}) = 0,15$
- Im Bsp $Pr(c|A) = \frac{P(c,A)}{P(A)} = \frac{0,15}{0,25} = 0,6$

Definition $V = (X, K, Y, e, d, Pr_K)$ ein Kryptosystem mit Schlüsselverteilung

1. Sei Pr_X eine Wahrscheinlichkeitsfunktion auf den Klartexten. Dann heißt V informationstheoretisch sicher bezüglich Pr_X, wenn für alle $x \in X, y \in Y$ mit $Pr(y) > 0$ gilt: $Pr(x) = Pr(x|y)$.
2. Das KSV V heißt informationstheoretisch sicher, wenn es bezüglich jeder beliebigen Klartextverteilung Pr_X informationstheoretisch sicher ist.

Satz: Informationstheoretische Sicherheit des Vernam-Systems
Sei $l > 0$ und $S = (X, K, Y, e, d)$ mit $X = K = Y = \{0, 1\}^l$ und $e = d = \oplus_l$ das Vernam-System der Länge l. Sei weiter $Pr_K : K \to [0, 1]$ die Gleichverteilung. Dann ist $V = S[Pr_K]$ informationstheoretisch sicher.
Es wird sich herausstellen, dass informationstheoretische Sicherheit in bestimmten Fällen Gleichverteilung auf den Schlüsseln erzwingt und dass die informationstheoretische Sicherheit eines KSV nichts mit den konkreten Wahrscheinlichkeiten der Klartextverteilung Pr_X zu tun hat, sondern nur die Menge $\{x \in X | Pr_X(x) > 0\}$ der "aktiven" Klartexte relevant ist.

$Pr(x|y) = \frac{Pr(x,y)}{Pr(y)} = \frac{Pr(y|x)*Pr(x)}{Pr(y)} = \frac{Pr_K(k_x,y)*Pr(x)}{Pr(y)} =^* \frac{\frac{1}{|K|}*Pr(x)}{\frac{1}{|K|}} = Pr(x)$

Satz Sei $V = (X, K, Y, e, d, Pr_K)$ ein KSV mit $|X| = |Y| = |K|$. Dann sind äquivalent:

1. V ist informationstheoretisch sicher.
2. (X, K, Y, e, d) ist possibilistisch sicher und $Pr_K(k) = \frac{1}{|K|}$ für alle $k \in K$.

Der Satz besagt, dass man informationstheoretisch sichere Systeme mit $|X| = |Y| = |K|$ daran erkennt, dass in der Verschlüsselungstabelle in jeder Spalte alle Chiffretexte vorkommen (possibilistische Sicherheit) und dass die Schlüsselverteilung Pr_K uniform ist. Auch in jeder Zeile kommen natürlich alle Chiffretexte vor: das liegt aber einfach an der Dechiffrierbedingung. Die Klartextverteilung ist irrelevant.
Technisch hilfreich sind die folgenden Größen, die nur von der Verschlüsselungsfunktion und der Schlüsselverteilung abhängen

- $P^x(y) := \sum_{k \in K, e(x,k)=y} Pr(k)$ für $x \in X, y \in Y$ (1.1)
- Für alle $x \in X : Pr(x, y) = Pr(x) * P^x(y)$ (1.2)
- wenn $Pr(x) > 0 : Pr(y|x) = \frac{Pr(x,y)}{Pr(x)} = P^x(y)$ (1.3)

Lemma Sei $V = (X, K, Y, e, d, Pr_K)$ KSV und seien Pr_X und Pr'_X Klartextverteilungen mit $Pr'_X(x) > 0 \Rightarrow Pr_X(x) > 0$. Dann gilt: Ist V informationstheoretisch sicher bzgl. Pr_X, so ist V informationstheoretisch sicher bzgl. Pr'_X. Besagt, dass man die Wahrscheinlichkeiten aktiver Klartexte beliebig ändern kann, ohne dass eine bestehende informationstheoretische Sicherheit zerstört wird.
Beweis: Sei V informationstheoretisch sicher bzgl. Pr_X. Wir zeigen nacheinander vier Aussagen.

1. $Pr_X(x) > 0 \Rightarrow Pr^x(y) = Pr(y|x) = Pr(y)$ für alle $y \in Y$
2. $Pr'_X(x) > 0 \Rightarrow Pr'(y|x) = Pr(y)$ für alle $y \in Y$
3. $Pr'(y) = Pr(y)$ für alle $y \in Y$
4. $Pr'(x) = Pr'(x|y)$ für alle $x \in X, y \in Y$ mit $Pr'(y) > 0$

Satz Sei $V = (X, K, Y, e, d, Pr_K)$ KSV und sei Pr_X eine Klartextverteilung. Dann sind äquivalent:

1. V ist informationstheoretisch sicher für Pr_X.
2. Für jedes $x \in X$ und jedes $y \in Y$ gilt: $Pr(x, y) = Pr(x)Pr(y)$ (das Eintreten von x und das Eintreten von y sind unabhängig).
3. Für alle $x \in X$ mit $Pr(x) > 0$ und alle $y \in Y$ gilt $Pr(y) = Pr(y|x)$ (andere Formulierung der Unabhängigkeit).
4. Für alle $x, x' \in X$ mit $Pr(x), Pr(x') > 0$ und alle $y \in Y$ gilt $P^x(y) = P^{x'}(y)$.

Die informationstechnische Sicherheit drückt sich dadurch aus, dass diese Chiffretextwahrscheinlichkeiten auch für jeden Klartext (also jede Spalte) separat auftreten.

Cyphertext-only-Angriffe: Vigenère-Chiffre

Eine Folge $x = x_0...x_{l-1}$ von Buchstaben im Klartextalphabet $\{a, ..., z\}$ der Größe 26. Ein informationstheoretisch sicheres Verfahren ist, für jede Buchstabenposition $0 \geq i < L$ rein zufällig einen Schlüssel $k_i \in \{A, ..., Z\}$ zu wählen und an Position i die Verschiebechiffre mit Schlüssel k_i anzuwenden. Der Schlüssel $k_0, ..., k_{L-1}$ ist dann aber mindestens so lang wie die Klartextfolge. Allerdings ist das nach unseren bisherigen Ergebnissen auch unvermeidlich: Wenn $V = (X, K, Y, e, d, Pr_K)$ informationstheoretisch sicher ist, ist (X, K, Y, e, d) possibilistisch sicher, also $|X| \geq |K|$.

Viginère Angriffe bei bekannter Schlüssellänge

Definition Eine Verschiebechiffre ist ein Kryptosystem $S = (Z_n, Z_n, Z_n, e, d)$ mit $e(x, k) = (x + k) mod\ n$. (Offensichtlich ist dann $d(y, k) = (y - k) mod\ n$.) Unser Beispiel ist der Fall $n = 26$, also $X = Y = K = \{0, 1, 2, ..., 25\}$. Wir identifizieren die Elemente dieser Menge mit den Buchstaben $a, ..., z$ (bei X) bzw. $A, ..., Z$ (bei K und Y). Die einfachste Methode ist folgende Version der Cäsar-Chiffre: Wähle einen Schlüssel k aus $K = \{0, 1, ..., 25\} = \{A, ..., Z\}$ zufällig. Um ,,Texte" (d.h. Wörter über $\mathbb{Z}_n$) zu verschlüsseln, wird S buchstabenweise angewandt: Aus $x_0x_1...x_{l-1}$ wird $e(x_0, k)e(x_1, k)...e(x_{l-1}, k)$.
Diese Methode ist allerdings sehr leicht zu brechen, sogar ,,von Hand". Es gibt mindestens die folgenden naheliegenden Möglichkeiten

1. probiere die 26 möglichen Schlüssel aus, oder
2. zähle, welche Buchstaben am häufigsten im Chiffretext vorkommen und teste die Hypothese, dass einer von diesen für ,,e" steht (Häufigkeitstabellen)

Definition Das Vigenère-Kryptosystem (mit Parametern $(n, S, L) \in \mathbb{N}^3$) ist das Kryptosystem $((\mathbb{Z}_n) \geq L, (\mathbb{Z}_n) \geq S, (\mathbb{Z}_n) \geq L, e, d)$, so dass für alle $s \geq S, l \geq L, x_i, k_j \in \mathbb{Z}_n$ gilt: $e(x_0...x_{l-1}, k_0...k_{s-1}) = y_0...y_{l-1}$ mit $y_i = (x_i + k_{i\ mod\ s}) mod\ n$, für alle $0 \geq i < l$.
Die zentrale Idee ist, dass für die Verschlüsselung des ,,Teiltextes" $y_i = y_iy_{i+s}y_{i+2s}...$, für $0 \geq i < s$, der Buchstabe k_i benutzt wurde, genau wie bei der einfachen Verschiebechiffre. Für $i, 0 \geq i < s$, bestimmt Eva also die in diesem Teiltext $y_i = y_iy_{i+s}y_{i+2s}...$ am häufigsten vorkommenden Buchstaben und testet die Hypothesen, dass diese für ,,e" oder einen anderen häufigen Buchstaben stehen.

- Chiffre: EYRYC FWLJH FHSIU BHMJO UCSEG TNEER...
- $y^0 = EFFBUTFSSMJFPOFT$, häufig: F, Schlüssel → B
- $y^1 = YWHHCNLXSIHXMZCNC$, häufig: Y, Schlüssel → U
- $y^2 = RLSMSEJMTKZMMSRE$, häufig: I,V, Schlüssel → E,R
- $y^3 = YJIJEELVKZVXVIVRYVV$, häufig: V, Schlüssel → R
- $y^4 = CHUOGRVYCSBKWAFHSF$, häufig: S, Schlüssel → O
- Schlüsselwort: BUERO

Der Kasiski-Test

Die Schlüssellänge kann oft durch den Kasiski-Test näherungsweise bestimmt werden. Stimmt der Klartext im Abschnitt $i + s * l$ bis $j + s * (l + h)$ mit dem Klartext im Abschnitt von $i + s * l'$ bis $j + s * (l' + h)$ überein, so gilt dies auch für den Chiffretext $(1 \geq i, j \geq s, l, l', h \in \mathbb{N})$. Kommt ein Teilwort im Klartext an zwei Positionen i und j und ist j-i ein Vielfaches von s, so werden die beiden Vorkommen des Wortes gleich verschlüsselt.
Für möglichst viele ,,lange" (mind. 3) Wörter, die im Chiffretext mehrfach auftreten, notiere die Abstände des Auftretens. Dann suche ein großes s, das viele dieser Abstände teilt (nicht unbedingt alle).

- Der Test funktioniert nur gut, wenn die Schlüssellänge s gering im Verhältnis zur Chiffretextlänge l ist.
- Um ihn anwenden zu können, muss die Klartextsprache bekannt sein.
- Der Test kann auch in der viel allgemeineren Situation benutzt werden, in der Schlüssel nicht s Verschiebungen, sondern s beliebige Substitutionschiffren auf X

Was passiert im Extremfall $s = l$?

- Grundsätzlich informationstheoretisch sicheres one-time pad ...
- ... aber nur dann, wenn die Schlüssel gleichverteilt gewählt werden. Wenn der Schlüssel selbst ein Text ist, so weist der Chiffretext wieder statistische Merkmale auf, die zum Brechen ausgenutzt werden können.

Effektive Verfahren der Schlüsselverlängerung (keine informationstheoretische Sicherheit)

- Autokey-Vigenère: Schlüssel k, Klartext m. Dann wird klassische Vigenère-Chiffre mit Schlüssel km auf m angewendet.
- Pseudozufallszahlen: Geheimer Schlüssel ist seed eines (Pseudo-)Zufallszahlengenerators, mit dem eine lange Schlüsselfolge $k_0, ..., k_{l-1}$ erzeugt wird.

Koinzidenzindex und Friedman-Methode

Die Methode beruht darauf, dass die Buchstabenhäufigkeiten fest stehen und sich bei der Verschlüsselung mit einer einfachen Substitutionschiffre nicht ändert. Ebenso ändert sich nicht die Wahrscheinlichkeit, bei der zufälligen Wahl eines Buchstabenpaars zwei identische Buchstaben zu erhalten.
Sei $x = x_0...x_{l-1}$ ein Klartext, sei $y = y_0...y_{l-1}$ der zugehörige Chiffretext, bei $s = 1$ (an jeder Stelle derselbe Schlüssel). Seien $n_0, ..., n_{25}$ die Anzahlen der Buchstaben $a, ..., z$ in $x, n'_0, ..., n'_25$ die in y. Wir wählen zufällig ein Paar von zwei Positionen in x (ohne ,,Zurücklegen"). Dafür gibt es $\binom{l}{2}$ Möglichkeiten. Genau $\binom{n_i}{2}$ viele davon führen dazu, dass man zweimal den Buchstaben Nummer i zieht, und $\sum_{0\geq i<26}\binom{n_i}{2}$ viele führen dazu, dass man an den beiden Positionen denselben Buchstaben sieht. Wir setzen $IC(x) := \frac{\sum_{0\geq i<26}\binom{n_i}{2}}{\binom{l}{2}} = \frac{\sum_{0\leq i<26} n_i(n_i-1)}{l(l-1)}$. Diese Zahl nennt man den Koinzidenzindex von x. Sie ist die Wahrscheinlichkeit dafür, dass an den beiden zufällig gewählten Positionen der selbe Buchstabe steht. Weil die Verschlüsselung auf den Buchstaben eine Bijektion ist, also sich die vorkommenden Häufigkeiten durch die Verschlüsselung nicht ändern, gilt für $IC(y) := \frac{\sum_{0\geq i<26} n'_i(n'_i-1)}{l(l-1)}$ die Gleichung $IC(x) = IC(y)$. Für lange Texte mit Häufigkeitsverteilung der Buchstaben nähert sich $IC(x)$ einem bestimmten Wert an. Wenn p_i die Häufigkeit von Buchstabe i in der verwendeten Sprache ist, wird für lange Texte x die Näherung $\frac{n_i}{l} \approx \frac{n_i-1}{l-1} \approx p_i$ gelten, also $IC(x) \approx \sum_{0\geq i<26} p_i^2$ sein. Die Summe $\sum_{0\geq i<26} p_i^2$ hat beispielsweise einen Wert von etwa $0,076$ für deutsche und $0,066$ für englische Texte.
Für die Ermittlung eines Schätzwertes für die Schlüssellänge s gehen wir wie folgt vor. Wir nehmen an, die zugrunde liegende Sprache ist Deutsch. Wir berechnen zunächst $IC(y)$ für den Chiffretext y. Die unbekannte Schlüssellänge nennen wir s. Dann berechnen wir eine Näherung für $IC(y)$, auf eine zweite Weise. Dies wird uns eine (Näherungs-)Gleichung für s liefern.
Bilde die Teilwörter $y^0, ..., y^{s-1}$, jedes mit der Länge $\frac{l}{s}$. Innerhalb jedes Teilworts kommen Kollisionen ebenso häufig vor wie in einem gewöhnlichen Text mit nur einem Schlüssel, also erwarten wir zusammen $\binom{l/s}{2}IC(y^0) + ... + \binom{l/s}{2}IC(y^{s-1}) \approx \frac{1}{2}l(l/s-1) * 0,076$ viele ,,Kollisionen" (Paare identischer Chiffretextbuchstaben) aus den einzelnen Teilwörtern. Zwischen zwei Teilwörtern y^u und y^v erwarten wir $(l/s)^2 * 261 \approx 0,0385(l/s)^2$ Kollisionen, aus allen $\binom{s}{2}$ Paaren von Teilwörtern zusammen also $\binom{s}{2}0,0385(l/s)^2 = \frac{s(s-1)}{2} * 0,0385(l/s)^2 = \frac{1}{2} * 0,0385l^2(1-\frac{1}{s})$ viele. Zusammen ist die erwartete Anzahl an Kollisionen in y gleich $\frac{1}{2}l(0,076(l/s-1) + 0,0385l(1-\frac{1}{s}))$. Diese Zahl sollte näherungsweise gleich $\frac{1}{2}l(l-1)IC(y)$ sein. Wir können die resultierende Gleichung $(l-1)IC(y) = 0,076(l/s-1) + 0,0385l(1-\frac{1}{s})$ nach s auflösen und erhalten: $s \approx \frac{(0,076-0,0385)l}{(l-1)IC(y)-0,0385l+0,076}$. Eine tatsächliche Durchführung des Verfahrens mit Chiffretexten erfordert viel Geduld.

Frische Verschlüsselung und Blockchiffren

Szenarium 2 (frische sym. Verschlüsselung): Alice möchte Bob mehrere verschiedene Klartexte vorher bekannter und begrenzter Länge übermitteln. Sie verwendet dafür immer denselben Schlüssel. Eva hört die Chiffretexte mit und kann sich sogar einige Klartexte mit dem verwendeten Schlüssel verschlüsseln lassen (chosen-plaintext attack).
Bemerkung: Das informationstheoretisch sichere Vernam-Kryptosystem ist nutzlos: Aus Kenntnis von $x \in \{0,1\}^l$ und $y = e(x,k)$ für ein einziges Paar $(x,k) \in X \times K$ kann Eva den Schlüssel $k = x \oplus_l y$ berechnen. Gleiches gilt für das Cäsar-System und das Vigenère-System.
Definition Ein Kryptosystem $S = (X, K, Y, e, d)$ ist possibilistisch sicher bzgl. Szenarium 2, wenn für jedes $1 \leq r \leq |X|$, jede Folge von paarweise verschiedenen Klartexten $x_1, x_2, ..., x_r \in X$, jeden Schlüssel $k \in K$ und jedes $y \in Y \backslash \{e(x_i,k) | 1 \leq i < r\}$ ein Schlüssel $k' \in K$ existiert mit $e(x_i,k) = e(x_i,k')$ für alle $1 \leq i < r$ und $e(x_r,k') = y$.
Proposition Für jede nichtleere Menge X ist das Substitutionskryptosystem auf X possibilistisch sicher.
Proposition Sei $S = (X, K, Y, e, d)$ ein Kryptosystem, das possibilistisch sicher ist bzgl. Szenarium 2. Dann ist $\{e(.,k) | k \in K\}$ die Menge aller injektiven Abbildungen von X nach Y.
Ein Kryptosystem, das possibilistisch sicher bzgl. Szenarium 2 ist, hat mindestens $|\{\pi | \pi : X \to Y \text{ ist injektiv}\}| = \frac{|Y|!}{(|Y|-|X|)!} \geq |X|!$ Schlüssel.
Mit $X = \{0,1\}^{128}$ gibt es $\geq 2^{128}!$ viele Schlüssel. Für die Speicherung eines Schlüssels braucht man durchschnittlich (Es gilt $log_2(N!) > N(log_2 N - log_2 e) > N(log_2 N - 1.45)$)
$log(2^{128}!) > 2^{128} * (128 - 1.45) > 2^{134} > (10^3)^{13.4} > 10^{40}$ viele Bits, eine unpraktikable Zahl. Also können wir in realen Situationen im Szenarium 2 keine possibilistische Sicherheit, geschweige denn informationstheoretische Sicherheit erhalten.
Definition Sei $l > 0$. Ein l-Block-Kryptosystem ist ein Kryptosystem $S = (\{0,1\}^l, K, \{0,1\}^l, e, d)$ mit $K \subseteq \{0,1\}^s$ für ein $s > 0$.

Substitutions-Permutations-Kryptosysteme (SPKS)

Grundsätzlich handelt es sich um mn-Block-Kryptosysteme. Dabei werden die Klartexte $x = (x_0, ..., x_{mn-1}) \in \{0,1\}^{mn}$ als m-Tupel $(x^{(0)}, x^{(1)}, ..., x^{(m-1)})$ von Bitvektoren der Länge n betrachtet. Dabei gilt $x(i) = (x_{in}, x_{in+1}, ..., x_{(i+1)n-1})$, für $0 \leq i < m$. Ein Baustein der Verschlüsselungs- und Entschlüsselungsfunktion bei diesen Systemen sind Bitpermutationen auf Bitstrings der Länge l. Sei β eine Permutation von $\{0, ..., l-1\}$ und $x \in \{0,1\}^l$. Dann bezeichnet x^β den Bitvektor $(x_{\beta(0)}, x_{\beta(1)}, ..., x_{\beta(l-1)})$. Kurz: $x^\beta_{(i)} = x_{\beta(i)}$, für $0 \leq i < l$. Sehr oft werden wir annehmen, dass β selbst invers ist, dass also $\beta^{-1} = \beta$ ist für $0 \leq i < l$). Dann erhält man x^β aus x, indem man $x(i)$ an die Stelle $\beta(i)$ des neuen Vektors schreibt.
Definition Ein Substitutions-Permutations-Netzwerk (SPN) ist ein Tupel $N = (m, n, r, s, S, \beta, \kappa)$ wobei

- positive ganzen Zahlen m, n, r und s die Wortanzahl, Wortlängen, Rundenzahl und Schlüssellänge
- $S : \{0,1\}^n \to \{0,1\}^n$ eine bijektive Funktion (S-Box)
- $\beta : \{0, ..., mn-1\} \to \{0, ..., mn-1\}$ selbstinverse Permutation (Bitpermutation)
- $\kappa : \{0,1\}s \times \{0, ..., r\} \to \{0,1\}^{mn}$ Rundenschlüsselfunktion

Das zu N gehörende Substitutions-Permutations-Kryptosystem (SPKS) ist das mn-Block-Kryptosystem
$B(N) = (\{0,1\}^{mn}, \{0,1\}^s, \{0,1\}^{mn}, e, d)$, das durch folgende Operationen beschrieben ist:
Chiffrierung: $e(x,k)$ wird für $x \in \{0,1\}^{mn}$ und $k \in \{0,1\}^s$ wie folgt berechnet:

1. Initialisierung (,,Weißschritt"): Berechne $u = x \oplus_{mn} \kappa(k,0)$.
2. Verschlüsselung in Runden: für $i = 1, ..., r-1$ berechne
   (a) $v(j) = S(u(j))$ für $0 \leq j < m$ (jedes Wort einzeln durch die S-Box)
   (b) $w = v^\beta$ (Bitpermutation auf dem Gesamtwort der Länge mn)
   (c) $u = w \oplus_{mn} \kappa(k,i)$ (XOR mit dem Rundenschlüssel)
   (d) Schlussrunde: $v(j) = S(u(j))$ für $0 \leq j < m$
   (e) Ausgabe: $y = v \oplus \kappa(k,r)$ (ohne Bitpermutation)

Dechiffrierung: $d(y,k)$ wird aus y und k nach demselben Verfahren berechnet, wobei jedoch

1. die S-Box durch ihre Inverse S^{-1} ersetzt wird und
2. die Rundenschlüsselfunktion κ ersetzt wird durch die Funktion $\kappa' : \{0,1\}^s \times \{0, ..., r\} \to \{0,1\}^{mn}$, $(k,i) \to \begin{cases} \kappa(k, r-i) & \text{für } i \in \{0, r\} \\ \kappa(k, r-i)^\beta & \text{für } 0 < i < r \end{cases}$

Vorteil der Struktur: Man kann dieselbe Hardware für Verschlüsselung und Entschlüsselung benutzen. Bei der Entschlüsselung läuft der Vorgang rückwärts ab, wobei die Runden anders gruppiert sind. Anstelle von S verwendet man S^{-1}. Da β selbst invers ist, kann man für die Permutation auch bei der Dekodierung β verwenden. Da Permutation und Schlüsselanwendung in den Runden vertauscht sind, muss man auf die Rundenschlüssel der ,,inneren" Runden bei der Dekodierung auch noch β anwenden.
Beispiel Wortanzahl $m = 3$, Wortlänge $n = 4$, Rundenzahl $r = 3$, Schlüssellänge $s = 24 = 6 * n$ und $\kappa(k,i) = k^{(i)}k^{(i+1)}k^{(i+2)}$. Die Bitpermutation β vertauscht die Bits $(0,4), (1,5), (2,8), (3,9), (6,10)$ und $(7,11)$ und ist damit selbst invers. Die S-Box ist gegeben durch die folgende zweizeilige Tabelle

b	0000	0001	0010	0011	0100	0101	0110	0111
S(b)	0101	0100	1101	0001	0011	1100	1011	1000
b	1000	1001	1010	1011	1100	1101	1110	1111
S(b)	1010	0010	0110	1111	1001	1110	0000	0111

Für $x = 000011110000$ und $k = 000000010010001101000101$ ergeben sich

1. $\kappa(k,0) = 000000010010$ und damit $u = 000011100010$
2. $i = 1$ $v = 010100001101$, $w = 001101010100$, $\kappa(k,1) = 000100100011$ und $u = 001001110111$
3. $i = 2$ $v = 110110001000$, $w = 101011000100$, $\kappa(k,2) = 001000110100$ und damit $u = 100011110000$
4. $v = 101001110101$, $\kappa(k,3) = 001101000101$ und damit $y = 100100110000$

Einschub: Endliche Körper

Die Struktur $\mathbb{Z}_2 = (\{0,1\}, \oplus, \odot, 0, 1)$ ist ein Körper, wobei $\oplus$ für die XOR-Operation und für $\wedge$ (AND) steht. Zu einem endlichen Körper F bildet man den Polynomring $F[X]$. Ein solcher Ausdruck wird mit seiner Koeffizientenfolge $(..., 0, 0, a_n, ..., a_1, a_0)$ identifiziert. Normalerweise lässt man Terme mit Koeffizient 0 weg. Über $\mathbb{Z}_2$ schreibt man wie üblich oft nur den Bitstring $a_n...a_1a_0$ ohne Klammern und Kommas.
Beispiel: In $\mathbb{Z}_2[X]$ liegen die Polynome
$g = 01011 = (..., 0, 1, 0, 1, 1) = X^3 + X + 1$ und
$h = 0110 = (..., 0, 1, 1, 0) = X^2 + X$.
Der Grad eines solchen Polynoms ist n, wenn n maximal mit $a_n \neq 0$ ist. Falls alle Koeffizienten gleich 0 sind ist der Grad $-\infty$. Man addiert und subtrahiert solche Polynome wie üblich und multipliziert sie wie üblich.
Beispiel: Die Summe von g und h ist $g + h = 01101 = (..., 0, 1, 1, 0, 1)$, ihr Produkt ist
$g * h = X^5 + X^4 + X^3 + X = (..., 0, 0, 1, 1, 1, 0, 1, 0) = 111010$.
Mit diesen Operationen erhält $F[X]$ die Struktur eines Rings: Die Addition ist Gruppe mit neutralem Element 0, die Multiplikation ist assoziativ mit neutralem Element $1 = (..., 0, 0, 0, 1)$, die Distributivgesetze gelten. Als Grundmenge des zu konstruierenden Körpers verwenden wir F^k, die Menge aller k-Tupel über F. Dies entspricht der Menge aller Polynome vom Grad bis zu $k-1$. Diese Menge hat genau $|F|^k$ Elemente. Im Fall $F = \mathbb{Z}_2$ erhalten wir $\{0,1\}^k$, die Menge aller Bitstrings der Länge k.
Als Addition $\oplus$ benutzen wir die gewöhnliche Addition von Polynomen, also die komponentenweise Addition der Tupel. Bei $F = \mathbb{Z}_2^k$ ist dies

einfach das bitweise XOR. Wir geben die Additionstafel für $k = 4$ an.
Die Elemente des Körpers $GF(2^4)$ werden dabei wir üblich als Hexadezimalziffern geschrieben: $5 \oplus C = 0101 \oplus 1100 = 1001 = 9$.
Die Multiplikation $\odot$ ist etwas komplizierter. Es wird ein irreduzibles Polynom $f = X^k + a_{k-1} * X^{k-1} + ... + a_1 * X + a_0$ vom Grad k (mit ,,Leitkoeffizient" $a_k = 1$) benötigt, also ein Koeffiziententupel $(1, a_{k-1}, ..., a_1, a_0)$ mit der Eigenschaft, dass man nicht $f = f_1 * f_2$ schreiben kann, für Polynome f_1 und f_2, die Grad ≥ 1 haben. Man kann zeigen, dass es für jedes $k \geq 2$ stets solche Polynome gibt. Sie können mit randomisierten Algorithmen effizient gefunden werden. Hier einige Beispiele für $F = \mathbb{Z}_2$. Wie üblich schreibt man die Koeffizientenfolge als Bitstring. Dieser Bitstring kann dann auch als natürliche Zahl (dezimal geschrieben) interpretiert werden.

k	irreduzibles Polynom vom Grad k über $\mathbb{Z}_2$	Kurzform	Zahl
1	$X + 1$	11	3
2	$X^2 + X + 1$	111	7
3	$X^3 + X + 1$	1011	11
4	$X^4 + X + 1$	10011	19
5	$X^5 + X^2 + 1$	100101	37
6	$X^6 + X + 1$	1000011	67
7	$X^7 + X^3 + 1$	10001001	137
8	$X^8 + X^4 + X^3 + X + 1$	100011011	283

Das Polynom $X^2 + 1 = (1, 0, 1) = 101$ ist nicht irreduzibel, da über $\mathbb{Z}_2$ die Gleichung $(X + 1) * (X + 1) = X^2 + 1$ gilt.
Die Multiplikation funktioniert nun wie folgt: Wenn g und h gegeben sind, berechnet man das Prddukt $g * h$ (Grad maximal $2k - 2$) und bestimmt den Rest r von $g * h$ bei der Division durch f. Dieser Rest (genannt ,,$g * h mod f$") ist das eindeutig bestimmte Polynom r vom Grad höchstens $k - 1$, das $g * h = q * f + r$ erfüllt, für ein ,,Quotientenpolynom" q.
Beispiel: Sei $k = 4$ und $f = (1, 0, 0, 1, 1) = 10011$. Die Faktoren seien $g = (1, 0, 1, 1) = 1011$ und $h = (0, 1, 1, 0) = 110$. Das Produkt ist $g * h = (1, 1, 1, 0, 1, 0) = 111010$. Wenn wir hier von $f * (X + 1) = (1, 1, 0, 1, 0, 1) = 110101$ subtrahieren, erhalten wir das Produkt $(1, 1, 1, 1) = 1111$ im Körper.
Man kann durch geduldiges Multiplizieren und Bilden von Resten (also Dividieren von Polynomen) in dieser Weise eine komplette Multiplikationstabelle aufstellen. Geschickter ist Folgendes: Man berechnet alle Produkte $g * h mod f$, für die in g und in h jeweils die ersten beiden Bits oder die letzten beiden Bits 0 sind. (Wenn man beobachtet, dass $(0, 0, 0, 1) = 0001$ das neutrale Element ist, und die Kommutativität berücksichtigt, muss man nur 15 Produkte berechnen.) Es ergibt sich die folgende Tabelle.

$\odot$	0000	0001	0010	0011	0100	1000	1100
0000	0000	0000	0000	0000	0000	0000	0000
0001	0000	0001	0010	0011	0100	1000	1100
0010	0000	0010	0100	0110	1000	0011	1011
0011	0000	0011	0110	0101	1100	1011	0111
0100	0000	0100	1000	1100	0011	0110	0101
1000	0000	1000	0011	1011	0110	1100	1010
1100	0000	1100	1011	0111	0101	1010	1111

Wir notieren weiterhin Bitstrings $(a_3, a_2, a_1, a_0) = a_3a_2a_1a_0$ hexadezimal. Das Polynom $g = (1, 0, 1, 1) = 1011$ ist also $B = 1000 \oplus 0011 = 8 \oplus 3$, das Polynom $h = (0, 1, 1, 0) = 0110$ ist $6 = 0100 \oplus 0010 = 4 \oplus 2$. Die Tabelle sieht dann so aus:

$\odot$	0	1	2	3	4	8	C
0	0	0	0	0	0	0	0
1	0	1	2	3	4	8	C
2	0	2	4	6	8	3	B
3	0	3	6	5	C	B	7
4	0	4	8	C	3	6	5
8	0	8	3	B	6	C	A
C	0	C	B	7	5	A	F

Nun wollen wir beliebige Polynome multiplizieren. Beispiel: Um $g = (1, 0, 1, 1) = 1011 = B$ und $h = (0, 1, 1, 0) = 0110 = 6$ zu multiplizieren, rechnet man unter Benutzung der Tabelle und des Distributivgesetzes wie folgt (Multiplikation modulo f):
$B \odot 6 = (8 \oplus 3)(4 \oplus 2) = (8 \odot 4) \oplus (8 \odot 2) \oplus (3 \odot 4) \oplus (3 \odot 2) = 6 \oplus 3 \oplus C \oplus 6 = 0110 \oplus 0011 \oplus 1100 \oplus 0110 = 1111 = F$.
Mit etwas Geduld und unter Ausnutzung des Distributivgesetzes lässt sich auf diese Weise die folgende Multiplikationstabelle für die Elemente von $\mathbb{Z}_2^4$ finden.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	3	1	7	5	B	9	F	D
3	0	3	6	5	C	F	A	9	B	8	D	E	7	4	1	2
4	0	4	8	C	3	7	B	F	6	2	E	A	5	1	D	9
5	0	5	A	F	7	2	D	8	E	B	4	1	9	C	3	6
6	0	6	C	A	B	D	7	1	5	3	9	F	E	8	2	4
7	0	7	E	9	F	8	1	6	D	A	3	4	2	5	C	B
8	0	8	3	B	6	E	5	D	C	4	F	7	A	2	9	1
9	0	9	1	8	2	B	3	A	4	D	5	C	6	F	7	E
A	0	A	7	D	E	4	9	3	F	5	8	2	1	B	6	C
B	0	B	5	E	A	1	F	4	7	C	2	9	D	6	8	3
C	0	C	B	7	5	9	E	2	A	6	1	D	F	3	4	8
D	0	D	9	4	1	C	8	5	2	F	B	6	3	E	A	7
E	0	E	F	1	D	3	2	C	9	7	6	8	4	A	5	B
F	0	F	D	2	9	6	4	B	1	E	C	3	8	7	B	5

Das neutrale Element der Multiplikation ist das Polynom $1 = (0, ..., 0, 1) = 0...01$. Man beobachtet, dass in jeder Zeile (und ebenso in jeder Spalte) der Tabelle, außer der für das Nullpolynom, alle Elemente genau einmal vorkommen. Dies ist für jeden Körper F und jedes beliebige k so, und es folgt daraus, dass die Elemente der Menge $F^k - \{0\}$ ein Inverses haben.
Fakt: Wenn man die Multiplikation wie beschrieben definiert, mit einem irreduziblen Polynom f mit Leitkoeffizient $a_k = 1$, dann gibt es für jedes Polynom $g \neq (0, ..., 0)$ vom Grad $< k$ (genau) ein Polynom h mit $g * h \ mod \ f = (0, ..., 0, 1) = 1$. Dieses Polynom h ist also g^{-1}, das multiplikative Inverse von g.
(Beweisidee: Man zeigt, dass die Abbildung $h \to g * h \ mod \ f$ in der Menge dieser Polynome injektiv ist. Ist $h_1 * g \ mod \ f = h_2 * g \ mod \ f$, so ist $(h_1 - h_2) * g \ mod \ f = 0$, also ist $(h_1 - h_2) * g$ durch f teilbar. Nach einem Lemma über irreduzible Polynome (analog zur Situation bei Primzahlen) folgt, dass f Teiler von $h_1 - h_2$ oder Teiler von g sein muss. Weil g nicht das Nullpolynom ist und Grad $< k$ hat, kann f kein Teiler von g sein. Also teilt f das Polynom $h_1 - h_2$. Da auch dieses Grad $< k$ hat, muss $h_1 - h_2$ das Nullpolynom sein, also $h_1 = h_2$ gelten. Aus der Injektivität von $h \to g * h \ mod \ f$ folgt die Surjektivität, also gibt es ein h mit $g * h \ mod \ f = 1$.)
Weil die üblichen Rechenregeln in der Struktur $(F^k, \oplus, \odot, 0, 1)$ mit den angegebenen Operationen leicht nachzuweisen sind, bedeutet dies, dass wir einen Körper mit $|F|^k$ Elementen erhalten haben. Wir nennen ihn $F[X]/(f)$, für das gewählte irreduzible Polynom f vom Grad k. Wenn $|F|^k = q$ ist, nennen wir diesen Körper $GF(q)$. Im Fall $F = \mathbb{Z}_2$ erhalten wir so Körper $GF(2^k)$, deren zugrunde liegende Menge einfach $\{0, 1\}^k$ ist, die Menge der Bitstrings der Länge k.
Bemerkung: In der Algebra zeigt man folgende Fakten über endliche Körper:

- Es gibt einen endlichen Körper $GF(q)$ mit q Elementen genau dann wenn $q = p^r$ für eine Primzahl p und einen Exponenten $r \geq 1$.
- Es ist gleichgültig, auf welchem Konstruktionsweg man zu einem Körper mit q Elementen gelangt: alle diese Körper sind isomorph, also strukturell identisch. (Insbesondere ist der Körper $GF(2^k)$ immer ,,der gleiche" , ganz egal welches irreduzible Polynom f man benutzt.) Die Tabellen für die Operationen können eventuell unterschiedlich aussehen, aber nur aufgrund von Umbenennungen von Objekten.

Ab hier schreiben wir + für die Körperaddition und $*$ für die Körpermultiplikation.
Bemerkungen zur Zeit- und Platzeffizienz: Allgemein, und für beliebig große k, kann man für ein festes Polynom f den Rest der Division durch f stets als Produkt des Zeilenvektors, der $g * h$ darstellt, mit einer festen $((2k - 1) \times k)$-Matrix M_f erhalten.
Damit ist die Komplexität der Multiplikation in einem solchen Körper definiert: Es muss das Produkt $g * h$ berechnet werden (,,Konvolution" von zwei Bitvektoren) und dann eine Vektor-Matrix-Multiplikation ausgeführt werden. Über dem Körper $\mathbb{Z}_2$ lässt sich die Vektor-Matrix-Multiplikation mit Hilfe der wortweisen XOR-Operation sehr effizient durchführen. Wenn zusätzlich f nur wenige Terme hat, hat M_f nur wenige 1-Einträge und die Matrix-Vektor-Multiplikation läuft auf einer Addition weniger Shifts eines Vektors hinaus. Für die Konvolution ist effiziente Ausführung etwas schwieriger; es gibt aber moderne Prozessoren, die diese Operation für konstante Wortlängen bereitstellen, was diese Operation auch für beliebige Wortlängen beschleunigt.
Wenn die Bitlänge k kurz und bekannt ist, wird man Tabellen benutzen. (Zum Beispiel benutzt AES einen Körper der Größe 256.) Für die Multiplikation verwendet man auch gerne Potenz- und Logarithmentabellen.
Fakt 2.7: Sei F ein endlicher Körper mit q Elementen. Dann gibt es in F ein ,,primitives Element" g, d.h., g erfüllt $\{g^i | 0 \geq i < q - 1\} = F - \{0\}$. (g ist erzeugendes Element der multiplikativen Gruppe von F.)
Beispiel: Wir hatten oben eine Multiplikationstabelle für $GF(2^4)$ aufgestellt. Mit ihrer Hilfe findet man leicht die ersten 15 Potenzenvon 2: Potenztabelle:

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$exp_2(i) = 2^i$	1	2	4	8	3	6	C	B	5	A	7	E	F	D	9

Diese Potenzen sind alle verschieden, also ist $g = 2$ primitives Element.
Die entsprechende Logarithmentab elle sieht so aus:

x	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$log_2(x)$	0	1	4	2	8	5	10	3	14	9	7	6	13	11	12

Achtung: Der Index der Exponential- und der Logarithmenfunktion ist nicht die natürliche Zahl 2, sondern 2, das ist das Element $X = 0010$ von $GF(2^4)$.
Allgemein: Wenn g primitives Element ist, dann betrachtet man die Exponentialfunktion zur Basis g: $exp_g : \mathbb{Z} \in i \to g^i \in F - \{0\}$.
Diese ist periodisch: Wenn $i - j$ durch $q - 1$ teilbar ist, dann ist $g^i = g^j$. Es genügt also, die Werte g^i für $0 \geq i < q - 1$ zukennen. Man erstellt eine Tabelle mit diesen $q - 1$ Werten. Die Umkehrfunktion ist die Logarithmusfunktion log_g zur Basis g. Sie ordnet jedem $x \in F - \{0\}$ den (eindeutig bestimmten) Wert $i \in \{0, 1, ..., q - 2\}$ mit $x = g^i$ zu, genannt $log_g(x)$. Es gilt $log_g(g^i) = i$ für $i \in \{0, 1, ..., q - 2\}$ und $g^{log_g(x)} = x$ für alle $x \in F - \{0\}$. Man erstellt ebenfalls eine Tabelle mit diesen Werten. Nun kann man leicht multiplizieren.
Aufgabe: Berechne $z = x * y$. Falls $x = 0$ oder $y = 0$ in F, ist $z = 0$. Andernfalls schaue in die log-Tabelle, um $i = log_g(x)$ und $j = log_g(y)$ zu finden. Berechne $k = (i + j) \ mod \ (q - 1)$. Das Ergebnis $z = g^k$ liest man nun in der exp-Tabelle an Stelle k ab.
Beispiel: In $GF(2^4)$ seien $x = 9$ und $y = C$ gegeben. Die log_2-Tabelle liefert $i = log_2(9) = 14$ und $j = log_2(C) = 6$. Wir erhalten $k = (14 + 6) \ mod \ 15 = 5$ und $z = g^5 = 6$ mit der exp_2-Tabelle. Man kontrolliert mit der Multiplikationstabelle für $GF(2^4)$, dass dort tatsächlich $9 * C = 6$ gilt.
Aufwand für eine Multiplikation: Drei Zugriffe auf Tabellen, eine modulare Addition!
Der Rechenaufwand für die Erstellung der Tabellen und auch ihr Platzbedarf ist $O(q)$ und damit akzeptabel, wenn q nicht zu groß ist. Im Fall von $q = 256$ hat man zwei Tabellen mit je 255 Einträgen und kann damit sehr leicht in konstanter Zeit multiplizieren. Auch Tabellen für $q = 2\frac{1}{6} = 65536$ stellen weiter kein Problem dar. Für noch größere q benötigt man weitere Tricks zum Ermitteln diskreter Logarithmen mit Hilfe von Tabellen der Größe etwa $\sqrt{q}$. (Wir werden später darauf zurückkommen.) Sollte q sehr groß werden (Hunderte oder Tausende von Bits), wird man, wie oben skizziert, Konvolution und Matrix-Vektor-Multiplikation benutzen.

AES: Advanced Encryption Standard

Der ,,Advanced Encryption Standard(AES)" ist ein symmetrisches Verschlüsselungsverfahren (d.h., beide Kommunikationspartner benutzen denselben Schlüssel). AES wird in vielen Standardverfahren beider Kommunikation im Internet benutzt, zum Beispiel bei Secure Socket Layer (SSL)/Transport Layer Security (TLS) und bei Secure Shell (SSH). Die AES-Chiffren gehören in die Klasse der Substitutions-Permutations-Kryptosysteme, auch wenn es in Details Abweichungen von der in Abschnitt 2.1 vorgestellten Struktur gibt. Bei der standardisierten Version AES ist die Klartextlänge und Chiffretextlänge stets $l = 128$, die Schlüssellänge 128, 192 oder 256 Bits. Wir konzentrieren uns auf die Variante mit Klartextlänge 128, Schlüssellänge 128. AES umfasst einige Spezialfälle der Rijndael-Familie

von Chiffren, mit der Klartexte der Länge 128, 160,192, 224 oder 256 Bits ver- und entschlüsselt werden können.
In AES wird Arithmetik im Körper $GF(2^8)$ benutzt, dessen Elemente 8-Bit-Vektoren, also Bytes, entsprechen.
Klartext und alle Zwischenergebnisse beider Ver- und Entschlüsselung sind Strings aus 128 Bits, die als 16 Wörter von 8 Bit Länge aufgefasst werden (1 Wort= 1 Byte= 2 Hexziffern). Die Bytes werden als Elemente von $GF(2^8)$ aufgefasst, konstruiert aus $\mathbb{Z}_2$ mit irreduziblem Polynom $f(X) = X^8 + X^4 + X^3 + X + 1 (= (1,0,0,0,1,1,0,1,1))$.
Die 16 Bytes werden als 4×4-Matrix (mit Einträgen aus $GF(2^8) = GF(256)$) aufgefasst. Die Leseanordnung ist dabei spaltenweise von links nach rechts, wobei Spalten von oben nach unten gelesen werden. Ein Bitstring $z \in \{0,1\}^{128}$ wird also in 16 Bytes $z^{(0)}, z^{(1)}, ..., z^{(15)}$ mit $z = z^{(0)}z^{(1)}...z^{(15)}$ aufgeteilt und wie folgt als Matrix $A(z) \in (\{0,1\}^8)^{4\times 4}$ geschrieben:

$$A(z) = \begin{pmatrix} z^{(0)} & z^{(4)} & z^{(8)} & z^{(12)} \\ z^{(1)} & z^{(5)} & z^{(9)} & z^{(13)} \\ z^{(2)} & z^{(6)} & z^{(10)} & z^{(14)} \\ z^{(3)} & z^{(7)} & z^{(11)} & z^{(15)} \end{pmatrix}.$$

Die Einträge einer solchen Matrix sind mit $0 \geq i, j \geq 3$ indiziert, wie folgt: $A = \begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{pmatrix}$.
Aus jeder solchen Matrix ergibt sich durch spaltenweises Auslesen wieder ein Bitstring
$A_{00}A_{10}A_{20}A_{30}A_{01}A_{11}A_{21}A_{31}A_{02}A_{12}A_{22}A_{32}A_{03}A_{13}A_{23}A_{33}$.
Abkürzungen für Zeilen und Spalten einer Matrix A:

- Zeile i : $row_i(A) = (A_{i0}, A_{i1}, A_{i2}, A_{i3})$, für $i = 0, 1, 2, 3$.
- Spalte j : $col_j(A) = \begin{pmatrix} A_{0j} \\ A_{1j} \\ A_{2j} \\ A_{3j} \end{pmatrix}$, für $j = 0, 1, 2, 3$.

Elementare Operationen auf Matrizen:

- Für Matrizen A und B in $GF(2^8)^{4\times 4}$ bedeutet $A \oplus B$ die komponentenweise Anwendung der Addition im Körper $GF(2^8)$. (Diese ist wiederum $\oplus_8$, die bitweise XOR-Operation auf Bytes.)
- Zyklischer Linksshift von Zeile i um $h = 1, 2, 3$ Positionen:
- $rotLeft_1((A_{i0}, A_{i1}, A_{i2}, A_{i3})) = (A_{i1}, A_{i2}, A_{i3}, A_{i0})$ usw.
- Beispiel: $rotLeft_2((A1, 06, 4B, EF)) = (4B, EF, A1, 06)$.
- In AES wird Zeile i um i Positionen nach links rotiert. Die Abbildung ist also folgende:
  $\begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{pmatrix} \rightarrow \begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{11} & A_{12} & A_{13} & A_{10} \\ A_{22} & A_{23} & A_{20} & A_{21} \\ A_{33} & A_{30} & A_{31} & A_{32} \end{pmatrix}$
- Diese Zeilenrotation ist eine Bitpermutation. Ihre Inverse besteht offenbar darin, Zeile i um i Positionen nach rechts zu rotieren. (Sie ist also nicht selbst invers)
- Zyklischer Shift von Spalte j um $h = 1, 2, 3$ Positionen nach oben: $rotUp_1((A_{0j}, A_{1j}, A_{2j}, A_{3j})^T) = (A_{1j}, A_{2j}, A_{3j}, A_{0j})^T$ usw. (Benötigt bei der Rundenschlüsselerzeugung.)
- ,,Lineare Spaltendurchmischung"
- $col_j(A) \rightarrow M * col_j(A)$, für $0 \geq j \geq 3$, für die feste Matrix
  $M = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \in GF(2^8)^{4\times 4}$.

Achtung: Gerechnet wird in $GF(2^8)$!

Beispiel: $M * \begin{pmatrix} 4F \\ B0 \\ 3E \\ A0 \end{pmatrix} = \begin{pmatrix} 02 * 4F \oplus 03 * B0 \oplus 01 * 3E \oplus 01 * A0 \\ ... \end{pmatrix} = \begin{pmatrix} 9E \oplus CB \oplus 3E \oplus A0 \\ ... \end{pmatrix} = \begin{pmatrix} CB \\ ... \end{pmatrix}$. In AES werden einmal alle vier Spalten auf diese Weise transformiert. Dies kann man auch zu einer Matrixmultiplikation zusammenfassen: $A \rightarrow M * A$.

Diese Operation ,,vermischt" die Einträge, ist aber keine Bitpermutation mehr, da Bits nicht nur vertauscht werden, sondern mit Körperoperationen verrechnet. Allerdings ist M invertierbar, sodass man die Operation auch wieder rückgängig machen kann, indem man mit

$M^{-1} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix}$ multipliziert.

- Die S-Box von AES ist eine feste bijektive Abbildung $\{0,1\}^8 \rightarrow 0,1^8$. Der mathematische Hintergrund der S-Box wird weiter unten diskutiert.
- Rundenschlüssel: Aus Schlüssel $k \in \{0,1\}^{128}$ und Rundennummer r wird Rundenschlüssel $\kappa(k,r) \in \{0,1\}^{128} = (\{0,1\}^8)^{4\times 4}$ berechnet. Details hierzu folgen.

AES-Chiffrieralgorithmus

- X:128-Bit-Klartext,
- k:128-Bit-Schlüssel)
- Umarrangiert: $X \in GF(2^8)^{4\times 4}$: Klartextmatrix; $k \in \{0,1\}^{128}$: Schlüssel.

1. Initialschritt (,,Weißschritt"):
   - $U \leftarrow X \oplus \kappa(k,0)$ // addiere Rundenschlüssel für Runde $r = 0$, als 128-Bit-String.
2. Für $r = 1, ..., 9$ tue
   (a) Substitution: Anwendung der S-Box auf jedes Byte in U
   - $V_{ij} \leftarrow S(U_{ij})$, für $0 \geq i, j \geq 3$;

   (b) Zeilenrotation: i-te Zeile um i Positionen nach links
   - $row_i(W) \leftarrow rotLeft_i(row_i(V))$, für $0 \geq i \geq 3$;

   (c) Lineare Spalten durchmischung: M wie oben beschrieben
   - $Z \leftarrow M * W$, für $0 \geq i \geq 3$;

   (d) Schlüsseladdition: Rundenschlüssel für Runde r
   - $U \leftarrow Z \oplus \kappa(k,r)$ // komponentenweise Addition
3. Verkürzte Schlussrunde, $r = 10$, keine Spaltendurchmischung:
   (a) Substitution
   - $V_{ij} \leftarrow S(U_{ij})$, für $0 \geq i, j \geq 3$;

   (b) Zeilenrotation
   - $row_i(Z) \leftarrow rotLeft_i(row_i(V))$, für $0 \geq i \geq 3$;

   (c) Schlüsseladdition
   - $Y \leftarrow Z \oplus \kappa(k,10)$
4. Ausgabe: $Y \in GF(2^8)^{4\times 4}$, geschrieben als 128-Bit-Wort.

Abbildung 3: Die S-Box für AES. $S(z_1z_0)$ für Hexziffern z_1, z_0 steht in Zeile z_1, Spalte z_0. Beispiel: $S(A4) = 49$.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Die S-Box von AES: Die S-Box ist als 16×16-Tabelle gegeben, siehe Abb.3. Zur kompakten Darstellung wird ein Element $(a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)$ von $GF(2^8)$ in zwei Hexziffern zerlegt: (a_7, a_6, a_5, a_4) ist der Zeilenindex, (a_3, a_2, a_1, a_0) der Spaltenindex der Position von $S((a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0))$.
Interessanterweise steckt hinter dieser chaotisch aussehenden Tabelle eine einfache Formel, die eine ,,nichtlineare" Komponente in AES einbringt. Zu $x \in GF(2^8)$ betrachtet man x^{-1}, das multiplikative Inverse. Künstlich definiert man $00^{-1} := 00$ (nur hier und nur für diesen Zweck). Weiter ist h eine invertierbare lineare Funktion auf dem $\mathbb{Z}_2$-Vektorraum $\{0,1\}^8$. Dazu werden die Elemente von $GF(2^8)$ als Spaltenvektoren aufgefasst, die 8 Bits enthalten.

$$h((a_7, ..., a_0))^T = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix}$$

Dann ist die S-Box wie folgt definiert: $S(x) = h(x^{-1}) \oplus_8 63$.
Auch hinter der Matrix M, die im Teil ,,Lineare Spaltendurchmischung" benutzt wird, steckt eine relativ einfache lineare Operation über einer passenden algebraischen Struktur.
Wir nehmen den endlichen Körper $\mathbb{F}_{2^8} = GF(2^8)$ als Ausgangssituation und betrachten Polynome $b_0 + b_1X + b_2X^2 + b_3X^3$ vom Grad maximal 3 über diesem Körper. Diese bilden die Grundmenge des Rings $\mathbb{F}_{2^8}[X]/(X^4 + 1)$: Addition wie gewöhnlich, Multiplikation modulo $X^4 + 1$. Ein solches Polynom wird durch den Spaltenvektor $(b_0, b_1, b_2, b_3)^T \in (F_{2^8})^4$ beschrieben. Wir multiplizieren dieses Polynom über $F_{2^8}[X]/(X^4 + 1)$ mit dem festen Polynom $c(X) = 02 + 01 * X + 01 * X^2 + 03 * X^3$. (Da die Koeffizienten Elemente von F_{2^8} sind, werden sie durch zwei Hexziffern dargestellt.) Der Koeffizientenvektor $(c_0, c_1, c_2, c_3)^T$ des Produktpolynoms bildet die transformierte Spalte. Man kann nachrechnen, dass sie sich als $M * (b_0, b_1, b_2, b_3)^T$ ergibt.
Die Rundenschlüssel werden mit einem iterativen Verfahren berechnet. Dabei wird zunächst Spalte 3 intensiv manipuliert und auf Spalte 0 addiert.

AES-Rundenschlüsselberechnung

$K_0 = \kappa(k,0) \leftarrow k$ //128-Bit-Schlüssel als 4×4-Matrix
Für $r = 1, ..., 10$ führe Runde r aus.

- Input: Rundenschlüssel K_{r-1} als 4×4-Matrix
- Output: Rundenschlüssel K_r als 4×4-Matrix
- Berechnung von Spalte 0:

1. Rotation (analog zur Zeilenrotation): $U \leftarrow rotUp_1(col_3(K_{r-1}))$ //Spalte 3 zyklisch eine Position nach oben
2. Substitution: $V_i \leftarrow S(U_i)$, für $0 \geq i \geq 3$;
3. Konstantenaddition: $V_0 \leftarrow V_0 + 02^{r-1}$; //Potenz in $GF(2^8)$;
4. Addition auf Spalte 0: $col_0(K_r) \leftarrow col_0(K_{r-1}) + V$; //Vektoraddition;

- Berechnung von Spalten 1 bis 3:
- Iterative Addition für $i = 1, 2, 3$: $col_i(K_r) \leftarrow col_i(K_{r-1}) + col_{i-1}(K_r)$;
- Ergebnis: $K_r = \kappa(k,r)$ für $r = 0, 1, ...10$.

Entschlüsselung: Man geht nach dem schon diskutierten grundsätzlichen Ansatz bei Substitutions-Permutations-Kryptosystemen vor. Erst werden alle Rundenschlüssel berechnet. Diese werden in umgekehrter Reihenfolge benutzt. Für die Substitution wird stets die inverse S-Box S^{-1} benutzt. Rotationen werden in umgekehrter Richtung ausgeführt. Die ,,lineare Spaltendurchmischung" wird mit der zu M inversen Matrix M^{-1} ausgeführt. Ansonsten ist der Ablauf völlig analog der Verschlüsselung.

Aktuelle Lage: Bisher gilt AES als praktisch sicher (insbesondere die 192-Bit- und die 256-Bit-Variante), wenn auch nicht in dem im Folgenden zu besprechenden technischen Sinn. Es wurden einige Angriffe vorgeschlagen, die zu schnellerem ,,Brechen" führen als dem vollständigen Durchsuchen des Schlüsselraums (nur noch 299 Schlüssel müssen getestet werden...), aber dies führt nicht zu praktisch durchführbaren Verfahren.

Bemerkungen zu randomisierten Algorithmen

Wir benötigen ein Algorithmenmodell, um die Angreifer in Eva zu modellieren. Offensichtlich wird sie alle ihr zur Verfügung stehenden Mittel einsetzen, insbesondere auch Zufallsexperimente (wenn es ihr nützt). Also müssen unsere Algorithmen auch Zufallsexperimente ausführen können. Grundsätzlich werden wir klassische Turingmaschinen bzw. üblichen Pseudocode (für gewöhnliche Computer) verwenden und ihre Laufzeit messen, allerdings mit einigen Änderungen/Erweiterungen/Einschränkungen, wie im Weiteren erläutert wird.

Ressourcenverbrauch Da wir nicht erwarten können, dass reale Kryptosysteme in einem idealen Sinn wie der informationstheoretischen Sicherheit sicher sind, wollen wir Sicherheit gegenüber einer Angreiferin Eva studieren, die nur beschränkte Ressourcen hat. Welche Ressourcen kommen in Frage? Sicherlich Zeit verbrauch bzw. Anzahl der Rechenoperationen, aber auch andere.
Vorüberlegung: Ein ,,zeiteffizienter" Angriff (mit Klartextwahl, ,,chosen-plaintext attack", CPA)
Sei $B = (\{0,1\}^l, \{0,1\}^s, \{0,1\}^l, e, d)$ ein Block-Kryptosystem und seien $x_i, y_i \in \{0,1\}^l$ für $1 \geq i \geq t$ paarweise verschiedene Klar- bzw. Chiffretexte. Ein Schlüssel $k \in \{0,1\}^s$ ist konsistent mit den Paaren (x_i, y_i), wenn $e(x_i, k) = y_i$ für alle $1 \geq i \geq t$ gilt. Für realistische (d.h. praktisch relevante) SPKS gibt es im Fall $t \approx 5$ höchstens einen konsistenten Schlüssel. (Man denke an AES. $t = 5$ bedeutet, dass $5 * 128$ Bit Information über k vorliegen, eventuell redundant, aber k ist nur 128 Bit lang.)
Wir können also für Eva in Szenario 2 den folgenden Angriff nach dem CPA-Muster planen:

1. Lasse fünf Klartexte $x_1, ..., x_5$ zu $y_1, ..., y_5$ verschlüsseln.
2. ,,Schaue nach", welcher Schlüssel k mit den Paaren $(x_1, y_1), ..., (x_5, y_5)$ konsistent ist (wenigstens einer ist es und es gibt höchstens einen).
3. Höre Chiffretext y ab und berechne $x = d(y, k)$ aus y.üm in Schritt 2. ,,nachsehen" zu können, benötigt Eva eine Tabelle mit allen möglichen Chiffretextkombinationen für die fünf Klartexte $x_1, ..., x_5$. Dafür gibt es $\geq |K| = 2^s$ Möglichkeiten, nämlich für jeden Schlüssel eine. Die Hashtabelle oder der Suchbaum (oder die entsprechende Struktur in einem Turingmaschinen-Programm) hat Größe 2^s. Wenn der Programmtext oder die Turingmaschine binäre Suche (oder einen binären Suchbaum) benutzt, dann führt dieser ,,Angriff " in Zeit $O(log(|K|)) = O(log(2^s)) = O(s)$ zum Klartext x.

Dieses Vorgehen ist natürlich unrealistisch, denn niemand kann ein so großes Programm schreiben oder speichern, wenn etwa (wie bei AES) die Zahl der Schlüssel $2^{128} > 10^{38}$ beträgt. Um diesen Trick auszuschließen, werden wir den Ressourcenverbrauch eines Algorithmus als Summe von Laufzeit und Länge des Programmcodes (=Größe der Transitionstabelle der Turingmaschine) messen. Die Programmgröße ist eine untere Schranke für die Zeit, die Eva zur Erstellung ihres Programms benötigt. Unser Ressourcenmaß erfasst also die Zeit für die Erstellung und die Zeit für die Ausführung des Algorithmus.ünsere Algorithmen werden oft mit Kryptosystemen einer festen Blocklänge l und einer fixen Anzahl von Klartext-/Chiffretext-Paaren arbeiten. Damit sind nur endlich viele Eingaben möglich, der Ressourcenverbrauch kann also nicht asymptotisch (,,bei großen Eingaben") angegeben werden. Daher betrachten wir zunächst Algorithmen mit konstanter Laufzeit. Das führt dazu, dass alle eventuell vorhandenen Schleifen nur konstant oft durchlaufen werden. Damit können wir aber auf Schleifenanweisungen vollständig verzichten. Die resultierenden Programme heißen ,,Straight-Line-Programme". Man kann sie sich in Pseudocode geschrieben vorstellen (ohne ,,while" und ,,for" und ohne Rückwärtssprünge), oder als Turingmaschinenprogramme mit der Eigenschaft, dass nie eine Programmzeile zweimal ausgeführt wird.

Randomisierung bei Straight-Line-Programmen

In unseren Algorithmen wollen wir zusätzlich die Anweisung ,,$y \leftarrow flip(M)$" erlauben, wobei M eine endliche Menge ist. Die Idee dabei ist, dass der Variable y ein zufälliges Element von M (bzgl. der Gleichverteilung auf M) zugewiesen wird. Damit berechnen unsere Algorithmen keine Funktionen, sondern zufällige Werte, die durch eine Wahrscheinlichkeitsverteilung gesteuert werden.üm den Wahrscheinlichkeitsraum definieren zu können, machen wir die folgenden Annahmen (die keine Einschränkung darstellen):

1. Bei nacheinander ausgeführten Kommandos der Form $y \leftarrow flip(M)$, mit identischem oder unterschiedlichem M, werden immer neue, unabhängige Zufallsexperimente ausgeführt.
2. Wie eben diskutiert, enthalten unsere Algorithmen keine Schleifen. Wir können daher jedes Ergebnis eines Zufallsexperiments in einer eigenen Variable speichern. Zusätzlich können wir die flip-Kommandos aus dem gesamten Programm, auch den bedingten Anweisungen, herausziehen und sie alle (vorab, auf Vorrat) ausführen. Damit können wir annehmen: Wenn die Anweisung $y \leftarrow flip(M)$ im Programmtext vorkommt, dann wird sie in jedem Durchlauf des Algorithmus (unabhängig von der Eingabe und den Ergebnissen der flip-Anweisungen) genau einmal ausgeführt.

Abkürzung: $flip(l)$ für $flip(\{0,1\}^l)$ (l-facher Münzwurf) und $flip()$ für $flip(1)$, also $flip(\{0,1\})$ (einfacher Münzwurf).
Sei nun A ein randomisierter Algorithmus (also ein Straight-Line-Programm), indem die $flip$-Anweisungen $y_i \leftarrow flip(M_i)$ für $1 \geq i \geq r$ vorkommen. Dann verwenden wir als zugrundeliegenden Wahrscheinlichkeitsraum die Menge $M = M_1 \times M_2 \times ... \times M_r$ mit der Gleichverteilung. Dies modelliert die Idee, dass die r Zufallsexperimente jeweils mit der uniformen Verteilung und insgesamt unabhängig voneinander ausgeführt werden.
I sei die Menge der Eingaben, Z sei die Menge der Ausgaben.
Ist $x \in I$ eine Eingabe, so erhalten wir für jedes $m = (m_1, ..., m_r) \in M$ genau eine Ausgabe $A^m(x) \in \mathbb{Z}$, indem wir in A die Anweisung $y_i \leftarrow flip(M_i)$ durch $y_i \leftarrow m_i$ ersetzen. Auf diese Weise erhalten wir

- Für jedes $m \in M$ eine Funktion $A^m : I \to Z, x \to A^m(x)$, und
- für jedes $x \in I$ eine Zufallsgröße $A(x) : M \to Z, m \to A^m(x)$.

Beispiel2.8 Betrachte den folgenden Algorithmus:

- $A(x : \{0,1\}^4) : \{0,1\}$ // nach dem Doppelpunkt: Typ der Eingabe bzw. Ausgabe
- $b_0 \leftarrow flip()$
- $b_1 \leftarrow flip()$
- $b_2 \leftarrow flip()$
- $b_3 \leftarrow flip()$
- $if b_0 = 1$ then return $x(0)$
- $if b_1 = 1$ then return $x(1)$
- $if b_2 = 1$ then return $x(2)$
- $if b_3 = 1$ then return $x(3)$
- return 1

Dann ist $M = \{0,1\}^4$, und es gilt: $A^{0110}(1101) = 1$ und $A^{0010}(1101) = 0$. Kompakt: Wenn $b_0 b_1 b_2 b_3$ mit i Nullen und einer 1 (an Position i) beginnt, dann ist die Ausgabe $x(i)$, für $i = 0, 1, 2, 3$. Ist $b_0 b_1 b_2 b_3 = 0000$, so ist die Ausgabe 1. Also gilt:
$Pr(A(1101) = 0) = Pr(\{w \in \{0,1\}^4 | w_0 = w_1 = 0, w_2 = 1\}) = (\frac{1}{2})^3 = \frac{1}{8}$ und $Pr(b_1 = 1) = Pr(\{w \in \{0,1\}^4 | w(1) = 1\}) = \frac{1}{2}$
Damit erhalten wir auch sinnvolle kombinierte und bedingte Wahrscheinlichkeiten:
$Pr(A(1101) = 0, b_0 = 0) = Pr(\{w \in \{0,1\}^4 | w_0 = w_1 = 0, w_2 = 1\}) = \frac{1}{8}$ und $Pr(A(1101) = 0 | b_0 = 0) = \frac{Pr(A(1101)=0, b_0=0)}{Pr(b_0=0)} = \frac{1}{4}$
Wir können auch den Algorithmus A so ändern, dass eine Anweisung $y_i \leftarrow flip(M_i)$ durch $y_i \leftarrow m_i^{(0)}$ ersetzt wird. (Diesen Algorithmus bezeichnen wir dann mit $A(y_i = m_i^{(0)})$.) Es gilt dann für alle Eingaben x und Ausgaben z: $Pr(A(x) = z | y_i = m_i^{(0)}) = Pr(A(y_i = m_i^{(0)})(x) = z)$.
Die verallgemeinerte Notation $A(y_1 = m_1^{(0)}, ..., y_s = m_s^{(0)})$ erklärt sich von selbst.

Prozeduren als Parameter Wir werden Algorithmen A betrachten, die als Eingabe eine Prozedur B (z.B. die Verschlüsselungsfunktion einer Blockchiffre mit fest eingesetzem Schlüssel) erhalten. Diese Prozedur darf nur aufgerufen werden, sie wird nicht als Text in den Rumpf von A eingefügt. Sie könnte aber wiederum zufallsgesteuert sein. Um den Wahrscheinlichkeitsraum von A mit einem solchen Funktionsparameter zu bestimmen, müssen folgende Informationen gegeben sein:

- B,
- die Anzahl der Aufrufe von B in A,
- welche Aufrufe $y \leftarrow flip(M)$ in B vorkommen.

Wir behandeln dann die Variablen y in verschiedenen Aufrufen von B genau wie die aus einem gewöhnlichen randomisierten Programm (Umbenennen der Variablen, Herausziehen der Zufallsexperimente an den Anfang). In dem resultierenden Wahrscheinlichkeitsraum sind dann die Zufallsexperimente in den verschiedenen Aufrufen von B und die in Teilen von A außerhalb von B unabhängig.

Sicherheit von Block-Kryptosystemen

Wir betrachten hier l-Block-Kryptosysteme $B = (X, K, Y, e, d)$ mit $X = Y = \{0,1\}^l$ und $K \subseteq \{0,1\}^s$ für ein s. e ist die Verschlüsselungsfunktion und d die Entschlüsselungsfunktion. Wir erinnern uns:

1. Im Szenario 2 (chosen-plaintext attack,CPA) kann die Angreiferin Eva sich eine ,,geringe Zahl" von Klartexten verschlüsseln lassen, also hat sie eine Liste von Klartext-Chiffretext-Paaren: $(x_1, y_1), ..., (x_r, y_r)$. Nun kann jedenfalls nur ein ,,neuer" Klartext x, also ein $x \in X\{x_1, ..., x_r\}$, geheim übertragen werden. Die possibilistische Sicherheit verlangt genau, dass dies möglich ist: Wenn $y \in Y\{y_1, ..., y_r\}$ ein neuer gegebener Chiffretext ist, dann gibt es für jeden Klartext $x \in X\{x_1, ..., x_r\}$ einen Schlüssel k, der x_i zu y_i chiffriert, $1 \geq i \geq r$, und x zu y chiffriert.
2. Wenn dabei r beliebig groß sein darf, können nach Prop.2.3 nur Substitutionskryptosysteme in diesem Sinne sicher sein. Da sie $|X|!$ Schlüssel haben müssen und daher immense Schlüssellänge (mindestens $|X|log|X| - O(|X|)$ Bits) erfordern, kommen sie in der Praxis nicht in Frage.

Idee eines neuen Sicherheitsbegriffs (für Block-Kryptosysteme): Gegeben sei eine Angreiferin Eva mit beschränkten Berechnungsressourcen. Wir fragen, wie sehr aus Evas Sicht das l-Block-Kryptosystem $B = (\{0,1\}^l, K, \{0,1\}^l, e, d)$ dem Substitutionskryptosystem $S' = (\{0,1\}^l, P\{0,1\}^l, \{0,1\}^l, e', d')$ (siehe Def.1.9) ähnelt. Ist es ihr mit ,,signifikanter Erfolgswahrscheinlichkeit" möglich, aufgrund der ihr vorliegenden Information und im Rahmen ihrer Ressourcen, zu unterscheiden, welches der beiden Systeme verwendet wird? Wenn dies nicht der Fall ist, kann das Kryptosystem B als sicher gelten, wie die folgende Überlegung zeigt. Wenn Eva aufgrund der ihr vorliegenden Information (2.4) nicht mit nennenswerter Erfolgswahrscheinlichkeit unterscheiden kann, ob sie es mit der Chiffre $e(., k)$ (für ein zufälliges $k \in K$) oder mit $e'(., \pi)$ (für ein zufälliges $\pi \in P_{\{0,1\}^l}$) zu tun hat, dann hat sie aus der ihr vorliegenden Information keine nennenswerte Information über die konkrete Chiffree $(., k)$ gelernt. Insbesondere kann sich ihre Information über die Klartextverteilung nicht (wesentlich) ändern, wenn ihr ein neuer Chiffretext y vorgelegt wird, da bei $S' = (\{0,1\}^l, P\{0,1\}^l, \{0,1\}^l, e', d')$ keine solche Änderung eintritt.

Wir modellieren Verfahren, die eine Chiffre benutzen dürfen und dann ,,raten" sollen, ob es eine Chiffre zu einem BKS B oder eine Zufallspermutation ist, als randomisierte Algorithmen.
Definition 2.9 Ein l-Unterscheider ist ein randomisierter Algorithmus $U(F : \{0,1\}^l \to \{0,1\}^l) : \{0,1\}$, dessen Laufzeit bzw. Ressourcenaufwand durch eine Konstante beschränkt ist.
Das Argument des l-Unterscheiders ist also eine Chiffre F. Diese ist als ,,Orakel" gegeben, das heißt als Prozedur, die nur ausgewertet werden kann, deren Programmtext U aber nicht kennt. Das Programm U kann F endlich oft aufrufen, um sich Paare wie in (2.4) zu besorgen. Danach kann U noch weiter rechnen, um zu einer Entscheidung zu kommen. Das von U gelieferte Ergebnis ist ein Bit.
Am liebsten hätte man folgendes Verhalten, für ein gegebenes Block-Kryptosystem B: Programm U sollte 1 liefern, wenn F eine Chiffre $e(.,k)$ zu B ist, und 0, wenn $F = \pi$ für eine Permutation $\pi \in P\{0,1\}^l$ ist, die keine B-Chiffre ist. Das gewünschte Verhalten wird sich bei U natürlich niemals mit absoluter Sicherheit, sondern nur mit mehr oder weniger großer Wahrscheinlichkeit einstellen, je nach Situation.
Beispiel2.10 Als Beispiel betrachten wir das Vernam-Kryptosystem $B = B_{Vernam}$, siehe Beispiel 1.6. Wir definieren einen l-Unterscheider $U = U_{Vernam}$, der als Parameter eine Funktion $F : \{0,1\}^l \to \{0,1\}^l$ erhält und Folgendes tut:

1. $k \leftarrow F(0^l)$
2. $y \leftarrow F(1^l)$
3. falls $1^l \oplus_l k = y$, dann gib 1 aus, sonst 0.

Dieser Unterscheider benutzt keine Zufallsexperimente, obwohl es ihm erlaubt wäre. Man sieht leicht Folgendes: Wenn $F(.) = e(.,k)$ für die Vernam-Chiffre mit beliebigem Schlüssel k, liefert U_{Vernam} immer das Ergebnis 1. Wenn hingegen F eine zufällige Permutation π von $\{0,1\}^l$ ist, dann ist die Wahrscheinlichkeit, dass $F(1^l) = 1^l \oplus_l F(0^l)$ gilt, genau $\frac{1}{2^l-1}$, also wird die Ausgabe 1 nur mit sehr kleiner Wahrscheinlichkeit ausgegeben (wenn l nicht ganz klein ist).
Wir definieren nun ein Spiel (,,game"), mit dem ein beliebiges Block-Kryptosystem B und ein beliebiger Unterscheider U darauf getestet werden, ob B gegenüber U ,,anfällig" ist oder nicht. (Das Spiel ist ein Gedankenexperiment, es ist nicht algorithmisch.) Die Idee ist folgende: Man entscheidet mit einem Münzwurf (Zufallsbit b), ob U für seine Untersuchungen als $F(.)$ eine zufällige Chiffre $e(.,k)$ von B (,,Realwelt") oder eine zufällige Permutation π von $\{0,1\}^l$ (,,Idealwelt") erhalten soll. Dann rechnet U mit F als Orakel und gibt dann seine Meinung ab, ob er sich in der Realwelt oder in der Idealwelt befindet. U ,,gewinnt", wenn diese Meinung zutrifft.
Definition 2.11 Sei $B = (\{0,1\}^l, K, \{0,1\}^l, e, d)$ ein l-Block-Kryptosystem, und sei U ein Unterscheider. Das zugehörige Experiment (oder Spiel) ist der folgende Algorithmus:

- $GBU() : \{0,1\}$ // kein Argument, Ausgabe ist ein Bit
  1. $b \leftarrow flip(\{0,1\})$
     - if $b = 1$ (,,Realwelt") then $k \leftarrow flip(K); F \leftarrow e(.,k)$ // Zufallschiffre von B
     - if $b = 0$ (,,Idealwelt") then $F \leftarrow flip(P\{0,1\}^l)$ // Zufallspermutation
  2. $b' \leftarrow U(F)$ // Der l-Unterscheider versucht herauszubekommen, ob $b = 0$ oder $b = 1$ gilt.
  3. if $b = b'$ then return 1 else return 0. // 1 heißt, dass U das richtige Ergebnis hat.

Das verkürzte Experiment/Spiel S_U^B (,,short") gibt im 3.Schritt einfach b' aus.
Der Wahrscheinlichkeitsraum für die Analyse des Spiels wird über die Zufallsexperimente zur Wahl von b, von k, der Zufallspermutation π aus $P_{\{0,1\}^l}$ und die Zufallsexperimente in U gebildet. Die Wahrscheinlichkeit dafür, dass der Unterscheider richtig liegt, ist $Pr(G_U^B = 1)$, gemessen in diesem etwas verschachtelten Wahrscheinlichkeitsraum.

Wir erläutern intuitiv, wieso diese Definition eine sehr allgemeine Situation erfasst. Angenommen, Angreiferin Eva kann auf irgendeine algorithmische Weise mit einer gewissen positiven Wahrscheinlichkeit eine ,,nicht triviale Information" über die Klartexte gewinnen, wenn diese mittels einer zufälligen Chiffre des l-Block-Kryptosystems B verschlüsselt worden sind. Damit kann sie einen Unterscheider mit nichttrivialer Erfolgswahrscheinlichkeit bauen! Sie ruft die Verschlüsselungsfunktion F auf, um einige selbstgewählte Klartexte $x_1, ..., x_q$ zu $F(x_1) = y_1, ..., F(x_q) = y_q$ zu verschlüsseln. Dann berechnet sie aus $y_1, ..., y_q$ ihre ,,nichttriviale Information" $I_1, ..., I_q$ zu den entsprechenden Klartexten und prüft, ob I_r auf x_r zutrifft, für $r = 1, ..., q$. Gibt es eine gute Übereinstimmung, so geht sie davon aus, dass F aus der ,,Realwelt" stammt, also eine Chiffre von B ist. Ansonsten vermutet sie, dass F aus der ,,Idealwelt" stammt, also eine zufällige Permutation ist.
Beispiel2.12 Für einen l-Bit-String z sei $p(z) = \oplus_{1 \geq i \geq l} z_i$ seine Parität. Nehmen wir an, das Chiffrierverfahren von N ist unvorsichtig geplant und zwar so, dass $p(e(x,k)) = p(x)$ ist für beliebige $x \in X$ und $k \in K$. Bei der Chiffrierung ändert sich also die Parität nicht. (Die Parität ist sicher ,,nichttriviale Information", auch wenn sie vielleicht nicht unmittelbar nützlich ist.)
$U_{Paritaet}(F)$:

1. Wähle Klartexte $x_1, ..., x_q$ mit $p(x_1) = ... = p(x_q) = 0$.
2. $y_r \leftarrow F(x_r)$, für $r = 1, ..., q$.
3. falls $p(y_1) = ... = p(y_q) = 0$, dann gib 1 aus, sonst 0.

Wenn wir in der ,,Realwelt" sind ($b = 1$), also F eine Chiffre $e(.,k)$ ist, dann ist die Antwort von U immer ,,1", also korrekt. Wenn wir in der ,,Idealwelt" sind ($b = 0$), also F eine zufällige Permutation ist, dann ist die Antwort nur mit Wahrscheinlichkeit $\frac{2^{l-1}(2^{l-1}-1)(2^{l-1}-2)...(2^{l-1}-q+1)}{2^l(2^l-1)(2^l-2)...(2^l-q+1)} \geq \frac{1}{2^q}$ falsch. (Alle Werte $F(x_1), ..., F(x_q)$ müssen zufällig zu Chiffretexten geführt haben, die Parität 0 haben, von denen es 2^{l-1} viele gibt.) Es ergibt sich $Pr(G_U^B = 1) \geq \frac{1}{2}(1 + 1 - 2^{-q}) = 1 - 2^{-(q+1)}$. Mit der effizienten Berechenbarkeit der ,,nichttrivialen Information" hat man also einen Unterscheider gefunden, der $Pr(G_U^B = 1)$,, groß" macht.
Beispiel: Der folgende triviale l-Unterscheider $U_{trivial}$ erreicht $Pr(G^B_{U_{trivial}} = 1) = \frac{1}{2}$: $b \leftarrow flip(\{0,1\}); return\ b$. Daher interessieren wir uns nicht für die Wahrscheinlichkeit $Pr(G_U^B = 1)$ ansich, sondern für den Abstand von $Pr(G_U^B = 1)$ zu $Pr(G^B_{U_{trivial}} = 1) = \frac{1}{2}$:
Definition 2.13 Sei U ein l-Unterscheider und B ein l-Block-KS. Der Vorteil von U bzgl. B ist $adv(U,B) := 2(Pr(G_U^B = 1) - \frac{1}{2})$
Klar ist:

- Für jeden l-Unterscheider U und jedes l-Block-KS B gilt $-1 \geq adv(U,B) \geq 1$.
- Werte $adv(U,B) < 0$ sind uninteressant. (Wenn man einen Unterscheider U mit $adv(U,B) < 0$ hat, sollte man in U die Ausgaben 0 und 1 vertauschen und erhält einen Unterscheider mit positivem Vorteil.)
- Für den trivialen l-Unterscheider $U_{trivial}$ gilt $adv(U,B) = 0$.üm den Vorteil eines Unterscheiders auszurechnen, sind seine ,,Erfolgswahrscheinlichkeit" und seine ,,Misserfolgswahrscheinlichkeit" hilfreiche Werte: Der Erfolg von U bzgl. B ist (für das verkürzte Spiel $S = S_U^B$) $suc(U,B) := Pr(S\langle b = 1\rangle = 1)$, d.h. die Wahrscheinlichkeit, dass U die Verwendung des l-Block-KS B richtig erkennt. Der Misserfolg ist $fail(U,B) := fail(U) := Pr(S\langle b = 0\rangle = 1)$, d.h. die Wahrscheinlichkeit, dass U die Verwendung des idealen Substitutionskryptosystems nicht erkennt. Man kann in der Notation für ,,fail" das ,,B" auch weglassen, da im Fall $b = 0$ das Kryptosystem B überhaupt keine Rolle spielt.

Lemma 2.14 $adv(U,B) = suc(U,B) - fail(U)$.
Beweis: $Pr(G_U^B = 1) = Pr(S_U^B = b)$

- $= Pr(S_U^B = 1, b = 1) + Pr(S_U^B = 0, b = 0)$
- $= Pr(S_U^B = 1 | b = 1) * Pr(b = 1) + Pr(S_U^B = 0 | b = 0) * Pr(b = 0)$
- $= \frac{1}{2}(Pr(S_U^B\langle b = 1\rangle = 1) + Pr(S_U^B\langle b = 0\rangle = 0))$
- $= \frac{1}{2}(Pr(S_U^B\langle b = 1\rangle = 1) + (1 - Pr(S_U^B\langle b = 0\rangle = 1)))$
- $= \frac{1}{2}((suc(U,B) + (1 - fail(U)))$
- $= \frac{1}{2}(suc(U,B) - fail(U)) + \frac{1}{2}$

Durch Umstellen ergibt sich die Behauptung. Unterscheider mit Werten $adv(U,B)$ substanziell über 0 können als interessant (oder möglicherweise gefährlich) für B gelten. Wir müssen noch die beschränkten Ressourcen des Unterscheiders ins Spiel bringen.
Definition 2.15 Sei $l \in \mathbb{N}$, U ein l-Unterscheider, B ein l-Block-KS. Für $q, t \in \mathbb{N}$ heißt U (q,t)-beschränkt, wenn die Laufzeit des Aufrufs von U innerhalb von G_U^B durch t beschränkt ist und dieser Aufruf die Funktion F mit höchstens q verschiedenen Argumenten ausführt.
Beispiel2.10 (Fortsetzung) Der l-Unterscheider U_{Vernam} wertet die Funktion F an zwei Stellen 0^l und 1^l aus. Die Laufzeit des Aufrufs von U ist beschränkt durch $c * l$ für eine Konstante $c \geq 1$. Also ist U_{Vernam} $(2, c * l)$-beschränkt.
Definition 2.16 Sei $\epsilon > 0$. Dann heißt $B(q,t,\epsilon)$-sicher, wenn für jeden (q,t)-beschränkten l-Unterscheider U gilt $adv(U,B) < \epsilon$.
Man kann diese Bedingung auch umgedreht lesen, nämlich so: ,,Um mit Wahrscheinlichkeit größer als $\frac{1}{2}(1+\epsilon)$ im Experiment G_U^B die richtige Antwort 1 zu erzeugen, braucht ein l-Unterscheider mehr als q Auswertungen oder mehr Laufzeit/Platz als t." Mit beliebig hohen Rechenzeiten lassen sich praktisch alle Block-Kryptosysteme brechen, selbst mit sehr kleinem q.
Beispiel2.10 (Fortsetzung) Sei B das Vernam-System der Länge l. Um den Vorteil von U_{Vernam} bzgl. B zu bestimmen, berechnen wir Erfolg und Misserfolg von U: Es gilt offensichtlich $1 = Pr(S_U^B\langle b = 1\rangle = 1) = suc(U,B)$.
Es gilt $fail(U) = Pr(S_U^B\langle b = 0\rangle = 1) = Pr(S_U^B = 1 | b = 0)$. Das verkürzte Experiment S_U^B gibt 1 aus, wenn der Unterscheider U dies tut. Dieser tut es aber genau dann, wenn $1^l \oplus_l F(0^l) = F(1^l)$ gilt, d.h. $fail(U) = Pr(S_U^B\langle b = 0\rangle = 1) = Pr(1^l \oplus_l F(0^l) = F(1^l)) = \frac{1}{2^l-1}$.
Damit haben wir aber
$adv(U,B) = suc(U,B) - fail(U) = 1 - \frac{1}{2^l-1} = \frac{2^l-2}{2^l-1} \approx 1$.

Das Vernam-System der Länge l ist also nicht $(2, c * l, \frac{2^l-2}{2^l-1})$-sicher.

Dafür realistische Werte von l (z.B. 128) der Wert $\frac{2^l-2}{2^l-1}$ praktisch 1 ist, muss das Vernam-System im Szenario 2 als unsicher angesehen werden. Natürlich sieht man auch mit bloßem Auge, dass die Vernam-Chiffre bei mehrfacher Verwendung nicht ,,sicher" ist, da sie leicht entschlüsselt werden kann. Hier geht es aber um die Formulierung eines Sicherheitskonzepts, das allgemein anwendbar ist. Es ist beruhigend, dass im Fall der Vernam-Chiffre herauskommt, dass sie auch im Sinn dieser Definition unsicher ist.
Am obigen Beispiel eines Block-Kryptosystems, das die Parität von x auf den Chiffretext $e(x,k)$ überträgt, sieht man aber auch, dass das Unterscheiderkonzept sehr empfindlich ist: Obwohl die Information über die Parity klein und harmlos scheint, erklärt es das System für nicht $(q, O(ql), 1 - 2^{-q})$-sicher, weil die Auswertung der Parität nur Zeit $O(l)$ kostet.
Beispiel2.17 Es sei B ein l-Block-Kryptosystem. (Man denke an AES.) Wir nehmen (realistisch) an, dass t (etwa $t = 6$) Klartext-Chiffretext-Paare $(x_1, y_1), ..., (x_t, y_t)$ in B den Schlüssel k der Länge $s = l$ eindeutig bestimmen, für mindestens die Hälfte der Schlüssel (*).
Der l-Unterscheider $U_{brute-force}$ wertet die Funktion F an $t+1$ Stellen $x_1, ..., x_t, x$ aus, mit Ergebnissen $y_1, ..., y_t, y$. Er probiert alle 2^l Schlüssel k, um einen zu finden, der $e(x_1,k) = y_1, ..., e(x_t,k) = y_t$ erfüllt. Wenn kein solches k gefunden wird, gibt U den Wert 0 aus. Sonst wird getestet, ob $e(x,k) = y$ ist. Falls ja, wird 1 ausgegeben, sonst 0.
Das Verhalten lässt sich so beschreiben: In der ,,Realwelt", wenn $F = e(.,k)$ ist für einen zufälligen Schlüssel k, dann findet U dieses k mit

Wahrscheinlichkeit größer als $\frac{1}{2}$ (wegen (*)). Dann ist $e(x,k) = F(x)$ nach Wahl von F und $y = F(x)$, weil y so bestimmt wurde. Also ist $e(x,k) = y$, also gibt U den Wert 1 aus. In der ,,Idealwelt" wird entweder kein passendes k gefunden, oder wenn doch, dann ist die Wahrscheinlichkeit, dass $e(x,k) = y$ ist, wo y einzufälliger Wert in $\{0,1\}^l - \{y_1, ..., y_t\}$ ist, durch $1/(2^l - t)$ beschränkt. Damit ist $adv(U,B) = suc(U,B) - fail(U) \geq \frac{1}{2} - \frac{1}{2^l - t} \approx \frac{1}{2}$. Die Laufzeit des Aufrufs von U ist beschränkt durch $O(|K|) = O(2^l)$.
Daher ist B nicht $(7, O(2^l), \frac{1}{2})$-sicher. Das ist natürlich nicht sehr schlimm, weil die Rechenzeit von U unrealistisch hoch ist.
Schlussbemerkung: Gesucht wäre nun natürlich ein l-Block-Kryptosystem, das (q,t,ϵ)-sicher ist, wobei q und t einigermaßen groß sind und ϵ möglichst klein ist. Es gibt unter bestimmten Annahmen (,,Existenz von Einwegfunktionen") theoretisch konstruierbare Systeme, die für große l einigermaßen effiziente (,,polynomiell in l") Verschlüsselung und Entschlüsselung erlauben und im definierten Sinn für Angreifer mit (im Bezug zu l) nicht zu großen Ressourcen (,,polynomiell in l") ϵ-sicher sind, für recht kleine ϵ (der Wert $1/\epsilon$ darf polynomiell groß sein). Diese Systeme sind aber praktisch nicht verwendbar. Von praktisch relevanten Systemen wie AES weiß man nicht, für welche Werte sie sicher sind.

Uneingeschränkte symmetrische Verschlüsselung

Szenarium 3: Alice möchte Bob mehrere Klartexte beliebiger Länge schicken. Sie verwendet dafür immer denselben Schlüssel. Eva hört die Chiffretexte mit und kann sich einige wenige Klartexte mit dem verwendeten Schlüssel verschlüsseln lassen.
Erweiterungen im Vergleich zu vorher:

1. Beliebig lange Texte sind erlaubt.
2. Mehrfaches Senden desselben Textes ist möglich; Eva sollte dies aber nicht erkennen.

Wir müssen die bisher benutzten Konzepte anpassen, um auch das mehrfache Senden derselben Nachricht zu erfassen. Ein grundlegender Ansatz, um mit identischen Botschaften umzugehen, ist Folgendes: Alices Verschlüsselungsalgorithmus ist randomisiert, liefert also zufallsabhängig unterschiedliche Chiffretexte für ein und denselben Klartext. Allerdings ist normalerweise die Anzahl der Zufallsexperimente fest und nicht von der Klartextlänge abhängig, sodass wir wie vorher davon ausgehen können, dass nur ein Experiment am Anfang ausgeführt wird. Auch der Sicherheitsbegriff und die Rechenzeitbeschränkungen müssen verändert werden.
Klartexte und Chiffretexte sind nun endliche Folgen von Bitvektoren (,,Blöcken") der festen Länge l. Die Menge aller dieser Folgen bezeichnen wir mit $(\{0,1\}^l)^*$ oder kürzer mit $\{0,1\}^{l*}$. Es gibt also unendlich viele Klartexte und Chiffretexte. Die Menge der Schlüssel heißt K. Der Verschlüsselungsalgorithmus E ist randomisiert und transformiert einen Klartext x und einen Schlüssel k in einen Chiffretext. Der Entschlüsselungsalgorithmus D ist deterministisch und transformiert einen Chiffretext y und einen Schlüssel k in einen Klartext. Sicher muss man den Algorithmen eine Rechenzeit zugestehen, die von der Länge der zu verarbeitenden Texte abhängt. Als effizient werden Algorithmen angesehen, die eine polynomielle Rechenzeit haben. Es muss eine verallgemeinerte Dechiffrierbedingung gelten, die die Randomisierung der Verschlüsselung berücksichtigt.
Definition 3.1 Ein symmetrisches l-Kryptoschema ist ein Tupel $S = (K, E, D)$, wobei

- $K \subseteq \{0,1\}^s$ eine endliche Menge ist (für ein $s \in \mathbb{N}$),
- $E(x : \{0,1\}^{l*}, k : K) : \{0,1\}^{l*}$ ein randomisierter Algorithmus und
- $D(y : \{0,1\}^{l*}, k : K) : \{0,1\}^{l*}$ ein deterministischer Algorithmus sind, so dass gilt:
- Die Laufzeiten von E und D sind polynomiell beschränkt in der Länge von x bzw. y.
- Für jedes $x \in \{0,1\}^{l*}, k \in K$ und jedes $m \in M_1 \times ... \times M_r$ (die Ausgänge der flip-Anweisungen in E) gilt: $D(E^m(x,k),k) = x$ (Dechiffrierbedingung).

Die Elemente von

- K heißen ,,Schlüssel"
- $\{0,1\}^{l*}$ heißen ,,Klartexte" bzw. ,,Chiffretexte", je nachdem, welche Rolle sie gerade spielen.

E ist der Chiffrieralgorithmus, D der Dechiffrieralgorithmus.
Bemerkungen:

- Zentral: Die Nachrichtenlänge ist unbestimmt.
- Wir werden immer davon ausgehen, dass der Schlüssel k uniform zufällig aus K gewählt wurde und beiden Parteien bekannt ist. Die Angreiferin Eva kennt k natürlich nicht.
- Etwas allgemeiner ist es, wenn man den Schlüssel nicht uniform zufällig aus einer Menge wählt, sondern von einem randomisierten Schlüsselerzeugungsalgorithmus $G(s : integer) : \{0,1\}^*$ generieren lässt. Konzeptuell besteht zwischen den beiden Situationen aber kein großer Unterschied.
- Jeder Klar-und jeder Chiffretext ist ein Bitvektor der Länge $l * h$ für ein $h \in \mathbb{N}$. (Um auf ein Vielfaches der Blocklänge zu kommen, muss man die Texte notfalls mit Nullen auffüllen.)
- Um einen Klartext x zu verschicken, wird der Algorithmus E mit einem neuen, uniform zufällig gewählten Element m abgearbeitet und es wird $E^m(x,k)$ als Chiffretext verschickt. Insbesondere entsteht bei wiederholter Verschlüsselung eines Textes x (mit sehr großer Wahrscheinlichkeit) jedes mal ein anderer Chiffretext.
- In der Literatur finden sich auch Kryptoschemen, bei denen auch die Dechiffrierung randomisiert ist. Wir betrachten dies hier nicht.

Der Standardansatz zur Konstruktion eines Kryptoschemas besteht darin, von einer Block-chiffre wie in Kapitel 2 auszugehen und sie zu einem Kryptoschema auszubauen. Dies wird im Folgenden beschrieben.

Betriebsarten

Symmetrische Kryptoschemen erhält man z.B. aus Blockchiffren. Durch einfache Regeln wird erklärt, wie ein Klartext, der aus einer Folge von Blöcken besteht, zu chiffrieren ist. Für alle folgenden Konstruktionen sei $B = (\{0,1\}^l, K_B, \{0,1\}^l, e_B, d_B)$ ein l-Block-Kryptosystem für Blöcke einer Länge l. (Man darf sich hier zum Beispiel AES mit $l = 128$ vorstellen, oder eine Variante von DES.)

ECB-Betriebsart (,,Electronic Code Book mode")

Dies ist die nächstliegende Methode. Ein Schlüssel ist ein Schlüssel k von B. Man verschlüsselt einfach die einzelnen Blöcke von x mit B, jedes mal mit demselben Schlüssel k.
Definition: Das zu B gehörende l-ECB-Kryptoschema $S = ECB(B) = (KB, E, D)$ ist gegeben durch die folgenden Algorithmen:

- $E(x : \{0,1\}^{l*}, k : K_B) : \{0,1\}^{l*}$
- zerlege x in Blöcke der Länge $l : x = x_0 x_1 ... x_{m-1}$;
- für $0 \geq i < m$ setze $y_i \leftarrow e_B(x_i, k)$;
- gib $y = y_0 ... y_{m-1}$ zurück.
- $D(y : \{0,1\}^{l*}, k : K_B) : \{0,1\}^{l*}$
- zerlege y in Blöcke der Länge $l : y = y_0 y_1 ... y_{m-1}$;
- für $0 \geq i < m$ setze $x_i \leftarrow d_B(y_i, k)$;
- gib $x = x_0 ... x_{m-1}$ zurück.

Die Verschlüsselung verzichtet auf die Option, Randomisierung zu verwenden. (Sie hat den großen Vorteil, parallel ausführbar zu sein.) Es ist klar, dass die Dechiffrierbedingung erfüllt ist. Jedoch hat dieses Kryptoschema ein ziemlich offensichtliches Problem, nämlich, dass ein Block $x \in \{0,1\}^l$ immer gleich verschlüsselt wird, Eva also ganz leicht nicht-triviale Informationen aus dem Chiffretext erhalten kann. Zum Beispiel kann sie sofort sehen, ob der Klartext die Form $x = x_1 x_1$, mit $x_1 \in \{0,1\}^l$, hat oder nicht.
Das Problem wird augenfällig zum Beispiel bei der Verschlüsselung von Bildern. Ein Bild ist dabei einrechteckiges Schema (eine Matrix) von ,,Pixeln", denen jeweils ein Farbcode (z.B. 1 Byte) zugeordnet ist. Man könnte dabei die Pixel in Gruppen (quadratische Blöcke oder Zeilensegmente) unterteilen. Das Farbmuster jeder solchen Gruppe liefert einen Klartextblock. Wenn viele Bildteile identisch aussehen, etwa gleich konstant gefärbt sind, ergibt sich jeweils derselbe Klartext für den entsprechenden Block, und damit auch derselbe Chiffretext. Wenn man den Chiffretext bildlich darstellt, ergibt sich leicht ein grober Eindruck des Originalbildes. Als Beispiel betrachte diese Bilder aus Wikipedia.
Fazit: Obwohl er so naheliegend ist, sollte der ECB-Modus niemals benutzt werden!

CBC-Betriebsart(,,Cipher Block Chaining mode")

Diese Betriebsart weicht dem zentralen Problem von ECB aus, indem man die Blöcke in Runden $i = 0, 1, ..., m-1$ nacheinander verschlüsselt und das Ergebnis einer Runde zur Modifikation des Klartextblocks der nächsten Runde benutzt. Konkret: Es wird nicht x_i mit B verschlüsselt, sondern $x_i \oplus_l y_{i-1}$ (bitweises XOR). Man benötigt dann einen Anfangsvektor y_{-1} für die erste Runde. Dieser ist Teil des Schlüssels des Kryptoschemas (nicht von B), ein Schlüssel des Schemas ist also ein Paar (k,v) mit $k \in K_B$ und $v \in \{0,1\}^l$.
Definition: Das zu B gehörende l-CBC-Kryptoschema $S = CBC(B) = (KB \times \{0,1\}^l, E, D)$ ist durch die folgenden Algorithmen gegeben:

- $E(x : \{0,1\}^{l*}, (k,v) : KB \times \{0,1\}^l) : \{0,1\}^{l*}$
- zerlege x in Blöcke der Länge $l : x = x_0 x_1 ... x_{m-1}$;
- $y_{-1} \leftarrow v$;
- für $i = 0, ..., m-1$ nacheinander: $y_i \leftarrow e_B(x_i \oplus_l y_{i-1}, k)$;
- gib $y = y_0 ... y_{m-1}$ zurück.
- $E(x : \{0,1\}^{l*}, (k,v) : K_B \times \{0,1\}^l) : \{0,1\}^{l*}$
- zerlege y in Blöcke der Länge $l : y = y_0 y_1 ... y_{m-1}$
- $y_{-1} \leftarrow v$;
- für $i = 0, ..., m-1$ nacheinander: $x_i \leftarrow d_B(y_i, k) \oplus_l y_{i-1}$;
- gib $x = x_0 ... x_{m-1}$ zurück.

Der Vektor v wird Initialisierungsvektor genannt. Man versteht recht gut, was beim Chiffrieren passiert, wenn man sich das Bild auf Seite 104 im Buch von Küsters/Wilke ansieht. Beim Dechiffrieren geht man den umgekehrten Weg: Entschlüssele einen Block y_i mittels B, dann addiere y_{i-1}, um den Klartextblock x_i zu erhalten. Es ist klar, dass die Dechiffrierbedingung erfüllt ist.
Interessant ist die folgende Eigenschaft der Entschlüsselung im CBC-Modus: Wenn bei der Übertragung des Chiffretextes ein einzelner Block y_i durch einen Fehler zu y_i' verfälscht wird, dann ist die Entschlüsselung ab Block y_{i+2} trotzdem wieder korrekt.
In diesem Kryptoschema ist der zentrale Nachteil der ECB-Betriebsart verschwunden: Identische Klartextblöcke führen nun praktisch immer zu verschiedenen Chiffretextblöcken. Die Verschlüsselung von $x = x_1 x_1$ mit $x_1 \in \{0,1\}^l$ liefert i.a. keinen Chiffretext der Form $y = y_1 y_1$. Die oben erwähnte Wikipedia-Seite zeigt auch, dass bei der Verschlüsselung des Bildes mit CBC keine offensichtlichen Probleme mehr auftauchen.
Ein Problem bleibt aber bestehen: Wird zweimal der Klartext x verschlüsselt, so geschieht dies immer durch denselben Chiffretext $y = E(x,(k,v))$. Dies ist eine Folge der Eigenschaft von CBC, deterministisch zu sein. Auch CBC wird man in der Praxis daher nicht benutzen.

R-CBC-Betriebsart(,,Randomized CBC mode")

Um das Problem der identischen Verschlüsselung identischer Klartexte zu beseitigen, muss in die Verschlüsselung eine Zufalls komponente eingebaut werden. Beispielsweise kann man dazu CBC leicht modifizieren. Der Initialisierungsvektor $y_{-1} = v \in \{0,1\}^l$ ist nicht mehr Teil des Schlüssels, sondern wird vom Verschlüsselungsalgorithmus einfach zufällig gewählt, und zwar für jeden Klartext immer aufs Neue. Damit der Empfänger entschlüsseln kann, benötigt er v. Daher wird y_{-1} als Zusatzkomponente dem Chiffretext vorangestellt. Damit ist der Chiffretext um einen Block länger als der Klartext, und Eva kennt auch $v = y_{-1}$.
Definition: Das zu B gehörende l-R-CBC-Kryptoschema $S = R - CBC(B) = (K_B, E, D)$ ist gegeben durch die folgenden Algorithmen:

- $E(x : \{0,1\}^{l*}, k : K_B) : \{0,1\}^{l*}$;
- zerlege x in m Blöcke der Länge $l : x = x_0x_1...x_{m-1}$
- setze $y_{-1} = flip(\{0,1\}^l)$;
- für $i = 0, ..., m-1$ nacheinander: $y_i \leftarrow e_B(x_i \oplus_l y_{i-1}, k)$;
- gib $y = y_{-1}y_0...y_{m-1}$ zurück. //Länge: $m+1$ Blöcke
- $D(y : \{0,1\}^{l*}, k : K_B) : \{0,1\}^{l*}$;
- zerlege y in $m+1$ Blöcke der Länge $l : y = y_{-1}y_0y_1...y_{m-1}$
- für $i = 0, ..., m-1$ nacheinander: $x_i \leftarrow d_B(y_i, k) \oplus_l y_{i-1}$;
- gib $x = x_0...x_{m-1}$ zurück. //Länge: m Blöcke

Es gibt zwei Unterschiede zu CBC:

1. Für jede Verschlüsselung eines Klartextes wird ein neuer zufälliger Initialisierungsvektor verwendet. Dadurch wird ein Klartext x bei mehrfachem Auftreten mit hoher Wahrscheinlichkeit immer verschieden verschlüsselt.
2. Der Initialisierungsvektor ist nicht Teil des geheimen Schlüssels, sondern ist dem Angreifer bekannt, da er Teil des Chiffretextes ist. (Intuitiv würde man vielleicht sagen, dass dies ,,die Sicherheit verringert")

Tatsächlich und etwas unerwartet kann man nach einer sorgfältigen Formulierung eines Sicherheitsbegriffs für Kryptoschemen beweisen, dass die Betriebsart R-CBC zu ,,sicheren" Verfahren führt, wenn die zugrundeliegende Blockchiffre ,,sicher" ist. (Der Beweis ist aufwendig.) Warnung, als Beispiel für harmlos erscheinende Modifikationen, die Kryptoschemen unsicher machen:

1. Man könnte meinen, dass es genügt, bei jeder Verschlüsselung einen neuen Initialisierungsvektor zu benutzen, also zum Beispiel nacheinander $v, v+1, v+2, ...$ Dies führt jedoch zu einem unsicheren Kryptoschema.
2. Um Kommunikation zu sparen, könnte man auf die Idee kommen, dass Alice und Bob sich von einer Kommunikationsrunde zur nächsten den letzten Chiffretextblock merken und ihn bei der nächsten Runde als Initialisierungsvektor benutzen. Dieses Verfahren heißt ,,chained CBC" und wurde in SSL3.0 und TLS1.0 verwendet. Es stellte sich heraus, dass dieses Verfahren mit einem Angriff mit gewähltem Klartext erfolgreich attackiert werden kann!

OFB-Betriebsart(,,Output Feed Back mode")

Wir kommen nun zu zwei Betriebsarten, die einen ganz anderen Ansatz für die Verschlüsselung der einzelnen Blöcke von x verfolgen. Es wird dazu nämlich nicht B mit Schlüssel k benutzt, sondern der Mechanismus des Vernam-Systems (One-Time-Pads, siehe Beispiel 1.6) :$y_i = x_i \oplus_l k_i$, wobei $k_i \in \{0,1\}^l$ ein ,,Rundenschlüssel" für Block x_i ist. Das Kryptosystem B wird nur dafür benutzt, diese Rundenschlüssel herzustellen, die bei Verschlüsselung und bei Entschlüsselung identisch sind. Die Dechiffrierbedingung folgt dann daraus, dass das Vernam-System korrekt dechiffriert. Der Ansatz führt dazu, dass die Entschlüsselungsfunktiond B des Block-Kryptosystems gar nicht benötigt wird.
Zuerst betrachten wir die Betriebsart ,,Output Feedback". Dabei wird ein zufälliger Startvektor v aus $\{0,1\}^l$ gewählt. Man setzt $k_{-1} = v$, und konstruiert die Rundenschlüssel $k_0, ..., k_{m-1}$ dadurch, dass man iteriert den letzten Rundenschlüssel durch die Verschlüsselungsfunktion von B schickt: $k_i = e_B(k_{i-1}, k)$, für $i = 0, 1, ..., m-1$. (Der Name "Output Feedback"rührt daher, dass das Ergebnis einer Verschlüsselung durch e_B wieder als Input des nächsten Aufrufs von e_B benutzt wird.) Der Empfänger benötigt v, um seinerseits die Rundenschlüssel zu berechnen; daher wird v als y_{-1} dem Chiffretext vorangestellt, wie beim R-CBC-Modus.
Definition: Das zu B gehörende l-OFB-Kryptoschema $S = (K_B, E, D) = OFB(B) = (K_B, E, D)$ ist gegeben durch die folgenden Algorithmen:

- $E(x : \{0,1\}^{l*}, k : K_B) : \{0,1\}^{l*}$;
- zerlege x in m Blöcke der Länge $l : x = x_0x_1...x_{m-1}$;
- $k_{-1} \leftarrow y_{-1} \leftarrow flip(\{0,1\}^l)$;
- für $i = 0, ..., m-1$ nacheinander: $k_i \leftarrow e_B(k_{i-1}, k)$ und $y_i \leftarrow x_i \oplus_l k_i$;
- gib $y = y_{-1}y_0...y_{m-1}$ zurück.
- $D(y : \{0,1\}^{l*}, k : K_B) : \{0,1\}^{l*}$;
- zerlege y in $m+1$ Blöcke der Länge $l : y = y_{-1}y_0y_1...y_{m-1}$;
- $k_{-1} \leftarrow y_{-1}$;
- für $i = 0, ..., m-1$ nacheinander: $k_i \leftarrow e_B(k_{i-1}, k)$ und $x_i \leftarrow y_i \oplus k_i$;
- gib $x = x_0...x_{m-1}$ zurück.

Dieses Verfahren hat einen interessanten Vorteil. Oft werden die Blöcke des Chiffretextes beim Empfänger nacheinander eintreffen. Die Hauptarbeit, nämlich die iterierte Verschlüsselung mit e_B zur Ermittlung der Rundenschlüssel k_i, kann unabhängig von der Verfügbarkeit der Klartextblöcke erfolgen, sobald y_{-1} eingetroffen ist. Man kann beweisen, dass die Betriebsarten R-CBC und OFB ,,sicher" sind, wenn die zugrundeliegende Blockchiffre ,,sicher" ist. Dazu weiter unten mehr.

R-CTR-Betriebsart (,,Randomized CounTeR mode")

Dies ist die zweite Betriebsart, bei der das Kryptosystem B nur zur Herstellung von m ,,Rundenschlüsseln" benutzt wird, mit denen dann die Blöcke per $\oplus_l$ verschlüsselt werden. Anstatt iteriert zu verschlüsseln, was bei OFB eine sequentielle Verarbeitung erzwingt, werden hier die mit B zu verschlüsselnden Strings anders bestimmt. Man fasst $\{0,1\}^l$ als äquivalent zur Zahlenmenge $\{0, 1, ..., 2^{l-1}\}$ auf, interpretiert einen l-Bit-String also als Block oder als Zahl, wie es passt. In dieser Menge wählt man eine Zufallszahl r. Man ,,zählt" von r ausgehend nach oben und berechnet die Rundenschlüssel $k_0, ..., k_{m-1}$ durch Verschlüsseln von $r, r+1, ..., r+m-1$ (modulo 2^l gerechnet) mittelse $B(., k)$.
Rundenschlüssel k_i ist also $e_B((r+i) mod\ 2^l, k)$, und Chiffretextblock y_i ist $k_i \oplus_l x_i$. Interessant ist, dass hier die Berechnung der Rundenschlüssel und die Ver- bzw. Entschlüsselung der Blöcke parallel erfolgen kann, also sehr schnell, falls mehrere Prozessoren zur Verfügung stehen.
Definition: Das zu B gehörende l-R-CTR-Kryptoschema $S = R - CTR(B) = (K_B, E, D)$ ist gegeben durch die folgenden Algorithmen:

- $E(x : \{0,1\}^{l*}, k : K_B) : \{0,1\}^{l*}$;
- zerlege x in m Blöcke der Länge $l : x = x_0x_1...x_{m-1}$;
- $r \leftarrow flip(\{0, ..., 2^l - 1\})$;
- für $0 \geq i < m : y_i \leftarrow e_B((r+i) mod\ 2^l, k) \oplus_l x_i$;
- gib $y = ry_0...y_{m-1}$ zurück.
- $D(y : (\{0,1\}^l)+, k : K_B) : \{0,1\}^{l*}$;
- zerlege y in $m+1$ Blöcke der Länge $l : y = y_{-1}y_0y_1...y_{m-1}$;
- $r \leftarrow y_{-1}$;
- für $0 \geq i < m : x_i \leftarrow e_B((r+i) mod\ 2^l, k) \oplus_l y_i$;
- gib $x = x_0...x_{m-1}$ zurück.

Es ist offensichtlich, dass die Dechiffrierbedingung erfüllt ist.
Bemerkungen:

- Wie bei R-CBC und OFB wird hier ein zufälliger Initialisierungswert r verwendet, der als Teil des Chiffretextes dem Angreifer bekannt ist.
- Wie bei OFB wird die Entschlüsselungsfunktion d_B gar nicht verwendet, man kann also anstelle der Verschlüsselungsfunktion e_B eine beliebige Funktion $e_B : \{0,1\}^l \times \{0,1\}^l \rightarrow \{0,1\}^l$ benutzen, bei der die ,,Chiffren" $e_B(., k)$ nicht einmal injektiv sein müssen.
- Man kann dieses Kryptoschema auch wie folgt verstehen: Zu einem gegebenen Klartext $x \in \{0,1\}^{lm}$ wird aus einem zufälligen Initialwert r ein langer Bitstring $k' = E_B(r,k)E_B((r+1) mod\ 2^l, k)...E_B((r+m-1) mod\ 2^l, k)$ berechnet und der Klartext x dann mittels Vernamsystem und diesem Schlüssel verschlüsselt. Der Empfänger erhält r und den Chiffretext, kann also ebenfalls k' b erechnen und damit entschlüsseln. Ist B ein sicheres Block-Kryptosystem, so kann ein Angreifer aus r den Vernam-Schlüssel k' nicht so einfach berechnen, da er k nicht kennt. Die R-CTR-Betriebsart liefert also intuitiv einen hohen Grad an Sicherheit.

Sicherheit von symmetrischen Kryptoschemen

Wir werden hier ein Sicherheitsmodell definieren, das es gestattet, Aussagen wie die folgende zu formulieren (und zu beweisen): Wenn B ein ,,sicheres" l-Block-Kryptosystem ist (bzgl. einer Reihe von Parametern), und das Kryptoschema S wird aus B konstruiert, indem man einen geeigneten Betriebsmodus verwendet, dann ist S ebenfalls ,,sicher" (bzgl. einer variierten Reihe von Parametern). Ziel ist dabei, Betriebsmodi zu identifizieren, die keine unnötigen neuen Unsicherheitskomponenten ins Spiel bringen, die nicht im Block-KS B schon vorhanden waren.
Wir beschränken uns hier auf den Fall, wo Eva begrenzte Ressourcen (Zeit, Orakelaufrufe) hat. Damit müssen wir uns bei Überlegungen zu Kryptoschemen auf das Verhalten auf Klartexten begrenzter Länge und auf feste Rechenzeiten beschränken. Nach dem Kerckhoffs-Prinzip nehmen wir an, dass Eva das Kryptoschema kennt, also zum Beispiel das verwendete Block-Kryptosystem und den Betriebsmodus. Sie kann sich einige Klartexte verschlüsseln lassen (,,known-plaintext attack") und sieht einen Chiffretext y. Ihr Ziel ist, aus y Information über den zugrundeliegenden Klartext x zu gewinnen. Wir können nicht verhindern, dass Eva aus der Länge des Chiffretextes y auf die Länge von x schließt. (Bei allen Betriebsmodi, die wir gesehen haben, ergibt sich die Länge von x direkt aus der Länge von y). Abgesehen hiervon soll sie mit hoher Wahrscheinlichkeit ,,keine signifikante Information" über den Klartext erhalten können.
Wir skizzieren zunächst eine Idee für ein Sicherheitsmodell, das die Fähigkeit, in so einer Situation ,,Information zu ermitteln", formalisiert. Eva behauptet, sie könne ,,aus y Information über x ermitteln, die über die Länge von x hinausgeht". Um das zu überprüfen, stellt ihr ein ,,Herausforderer" Charlie folgende Aufgabe: Eva darf sich zunächst eine Reihe von Klartexten ihrer Wahl verschlüsseln lassen. Dann wählt sie selbst zwei verschiedene, gleichlange Klartexte z_0 und z_1. Charlie verschlüsselt einen von diesen; der Chiffretext ist y. Eva bekommt y. Sie soll herausfinden, ob y von z_0 oder von z_1 kommt. Für diese Entscheidung darf sie sich weitere Klartexte verschlüsseln lassen und weiter rechnen. Wir betrachten ein Kryptoschema als unsicher, wenn Eva eine signifikant von ,,purem Raten" abweichende Erfolgswahrscheinlichkeit hat, sie also mit guten Chancen unterscheiden kann, ob z_0 oder z_1 zu y verschlüsselt wurde.
Wie in Abschnitt 2.4 formulieren wir den Vorgang wieder als Spiel. Gegeben ist also $S = (K, E, D)$. Akteure sind Eva, hier ,,Angreifer" (engl.: adversary) genannt, und Charlie (,,Herausforderer" ,engl.: challenger). Die Parameter, mit denen wir die Erfolgschancen von Eva unter Ressourcenbeschränkungen messen, sind der ,,Vorteil" in Analogie zu Definition 2.13, die Rechenzeit (inklusive Speicherplatz, wie vorher), Anzahl der Orakelaufrufe und Anzahl der bei der Verschlüsselung bearbeiteten Blöcke.

- Charlie wählt zufällig einen Schlüssel k aus K und legt damit die Chiffre $H = E(., k)$ fest, die Klartexte aus $\{0,1\}^{l*}$ in Chiffretexte aus $\{0,1\}^{l*}$ transformiert.
- Eva wählt einige Klartexte und lässt sie sich von Charlie mit H verschlüsseln.
- Eva wählt zwei Klartexte z_0 und z_1 gleicher Länge und gibt sie an Charlie.
- Verdeckt vor Eva: Charlie wirft eine Münze, um zufällig einen der beiden Klartexte zu wählen. Er verschlüsselt ihn mit H, das Ergebnis ist y. Charlie gibt y an Eva.
- Eva kann sich weitere Klartexte verschlüsseln lassen und (mit beschränkten Ressourcen) rechnen und muss schließlich sagen (raten?), ob Charlie z_0 oder z_1 zu y verschlüsselt hat.

Wenn die Wahrscheinlichkeit, dass Eva richtig antwortet, weit von zufälligem Raten abweicht, wollen wir das Kryptoschema als unsicher ansehen.
Man beachte: Unter den vorher oder nachher verschlüsselten Klartexten können auch z_0 und z_1 sein. Ein deterministisches Kryptoschema, also eines, das zu gegebenem Schlüssel k einen Klartext stets gleich verschlüsselt, ist damit sofort disqualifiziert. Wenn aber bei der

Verschlüsselung Randomisierung im Spiel ist, liefern wiederholte Verschlüsselungsanforderungen mit Klartexten z_0 und z_1 (wahrscheinlich) lauter unterschiedliche Antworten, so dass dieser direkte Weg zur Ermittlung des verschlüsselten Klartextes nicht funktioniert.
Wir beschreiben den Part von Eva in diesem Spiel als Algorithmenpaar. Der erste Algorithmus AF (der ,,Finder", ,,find") ist für das Erzeugen von z_0 und z_1 zuständig. Als Argument erhält dieser Teil eine Chiffre H, die er im Sinne eines Orakels benutzen kann. Außerdem werden Aufzeichnungen über die angeforderten Verschlüsselungen und ihre Ergebnisse gemacht. Diese Aufzeichnungen sind als Element v einer endlichen Menge V kodiert. Der zweite Algorithmus AG (der ,,Rater" , ,,guess") ist dafür zuständig, herauszufinden, ob z_0 oder z_1 verschlüsselt wurde. Dieser Algorithmus bekommt H als Orakel, die Aufzeichnungen v von AF und den Chiffretext y als Input.
Definition 3.2 Ein l-Angreifer A ist ein Paar von randomisierten Algorithmen

- $AF(H(\{0,1\}^{l*}) : \{0,1\}^{l*}) : (\{0,1\}^l \times \{0,1\}^l)^+ \times V$
- $AG(v : V, H(\{0,1\}^{l*}) : \{0,1\}^{l*}, y : \{0,1\}^{l*}) : \{0,1\}$

Hierbei ist H ein randomisierter Algorithmus (und nicht als Funktion zu verstehen).
Der ,,Finder" AF bekommt also eine Chiffre $H = E(., k)$ für einen zufälligen Schlüssel k gegeben. (Er darf diese Chiffre nur zum Verschlüsseln benutzen; k kennt er nicht.) Daraus berechnet er zwei verschiedene Klartexte (z_0, z_1) gleicher Länge und ,,Notizen" $v \in V$. Das Ausgabeformat $(\{0,1\}^l \times \{0,1\}^l)^+$ sagt, dass eine Folge $((z_0^{(0)}, z_1^{(0)}), (z_0^{(1)}, z_1^{(1)}), ..., (z_0^{(m-1)}, z_1^{(m-1)}))$ von Blockpaaren ausgegeben werden soll, die wir dann als Paar $(z_0, z_1) = ((z_0^{(0)}, z_0^{(1)}, ..., z_0^{(m-1)}), (z_1^{(0)}, z_1^{(1)}, ..., z_1^{(m-1)}))$ von zwei Folgen gleicher Länge lesen. Danach wird zufällig z_0 oder z_1 zu y verschlüsselt. Im zweiten Schritt verwendet der ,,Rater" AG die Notizen v, die Chiffre $H = E(., k)$ und die ,,Probe" y, um zu bestimmen, ob z_0 oder z_1 verschlüsselt wurde.
Definition 3.3 Sei $S = (K, E, D)$ ein symmetrisches Kryptoschema, und sei $A = (AF, AG)$ ein l-Angreifer. Das zugehörige Experiment (oder Spiel) ist der folgende Algorithmus $G_A^S : \{0,1\}$:

1. $k \leftarrow flip(K)$, $H \leftarrow E(., k)$ (In diesem Schritt wählt Charlie zufällig eine Chiffre des Kryptoschemas S.)
2. $(z_0, z_1, v) \leftarrow AF(H)$ (In dieser Phase berechnet der Finder ein Paar von Klartexten gleicher Länge, von denen er annimmt,ihre Chiffretexte unterscheiden zu können.)
3. $b \leftarrow flip(\{0,1\})$ und $y \leftarrow E(z_b, k)$ (In dieser Phase wählt Charlie zufällig einen der beiden Klartexte und verschlüsselt ihn zu y.)
4. $b' \leftarrow AG(v, H, y)$ (In dieser Phase versucht der Rater herauszubekommen, ob z_0 oder z_1 verschlüsselt wurde.)
5. falls $b = b'$, so gib 1 zurück, sonst 0. (Charlies Auswertung: Hat AG recht oder nicht?)

Das verkürzte Experiment oder Spiel S_A^S gibt im 5.Schritt einfach b' aus. Dann ist $Pr(G_A^S = 1)$ die Wahrscheinlichkeit dafür, dass der Angreifer A sich für den korrekten Klartext entscheidet. Der Wahrscheinlichkeitsraum entsteht durch die expliziten Zufallsexperimente in Schritt 1. und 3. in Kombination mit den Zufallsexperimenten, die bei der Verwendung von H ausgeführt werden.
Man kann jetzt wie in Abschnitt 2.4 (Sicherheit von l-Block-Kryptosystemen) den Vorteil $adv(A, S) = 2(Pr(G_A^S = 1) - \frac{1}{2})$ und die Größen $suc(A, S) = Pr(S_A^S\langle b = 1\rangle = 1)$ (,,Erfolg") und $fail(A, S) = Pr(S_A^S\langle b = 0\rangle = 1)$ (,,Misserfolg") definieren. Allerdings haben die beiden letzten Werte eine etwas andere Semantik. Der Wert $suc(A, S)$ ist die bedingte Wahrscheinlichkeit, dass richtig erkannt wird, dass z_1 verschlüsselt wurde, $fail(A, S)$ ist die bedingte Wahrscheinlichkeit, dass nicht erkannt wird, dass z_0 verschlüsselt wurde. Lemma 2.14 gilt wörtlich. Das heißt:

$$adv(A, S) = suc(A, S) - fail(A, S)$$

Wenn ein Angreifer A mit ,,nicht zu großem Rechenaufwand" einen Vorteil erzielen kann, der deutlich über 0 liegt, wird man das Kryptoschema als unsicher einstufen.
Beispiel 3.4 Ziel ist es, die ECB-Betriebsart anzugreifen, d.h. sei $S = ECB(B)$ für ein l-Block-Kryptosystem B. Wir wollen zeigen, dass es einen l-Angreifer A mit $Pr(G_A^S = 1) = 1$, also $adv(A, S) = 1$, gibt. Dieser ist wie folgt aufgebaut.

- l-Angreifer A mit $V = \{1\}$ ($V = \{1\}$ bedeutet, dass stets $v = 1$ gilt, dass die Aufzeichnung v also keinerlei Information übermittelt.)
- $AF(H)$ arbeitet wie folgt: $z_0 \leftarrow 0^{2l}; z_1 \leftarrow 0^l 1^l$; Ausgabe: $(z_0, z_1, 1)$
- $AG(v, H, y)$ tut Folgendes: falls $y = y_1 y_1$ für ein $y_1 \in \{0,1\}^l$, gib 0 aus, sonst 1.

Im Ablauf des Spiels G_A^S wird der Rater AG mit $y = E(0^{2l}, k) = e_B(0^l, k)e_B(0^l, k)$ oder mit $y = E(0^l 1^l, k) = e_B(0^l, k)e_B(1^l, k)$ gestartet. Im ersten Fall ist $y = y_1 y_1$ für ein $y_1 \in \{0,1\}^l$, im zweiten Fall ist dies nicht so, wegen der Dechiffrierbedingung. Daher gilt $Pr(G_A^S = 1) = 1$, d.h. $adv(A, S) = 1$.
Die Ressourcen, die A benötigt, sind sehr klein: Zwei Aufrufe des Verschlüsselungsverfahrens e_B des Block-Kryptosystems. Wir können schließen, dass es einen effizienten Angreifer A gibt, dem das Sicherheitsmodell Vorteil 1 gibt. Damit gilt das Kryptoschema $ECB(B)$ als komplett unsicher.
Beispiel 3.5 Ziel ist es, die CBC-Betriebsart anzugreifen, d.h. es sei $S = CBC(B)$ für ein Block-Kryptosystem B. Das Problem mit dieser Betriebsart ist, dass ein Klartext bei Wiederholung identisch verschlüsselt wird. Um dies auszunutzen, verwenden wir den folgenden l-Angreifer A mit $V = \{0,1\}^l$, der zwei verschiedene Klartexte benutzt, die nur einen Block enthalten:

- $AF(H)$ arbeitet wie folgt: $z_0 \leftarrow 0^l; v \leftarrow H(z_0); z_1 \leftarrow 1^l$; Ausgabe: (z_0, z_1, v)
- (A merkt sich den Chiffretext zu $x = 0^l$.)
- $AG(v, H, y)$ tut Folgendes: falls $v = y$, so gib 0 aus, sonst 1.

Im Ablauf des Spiels G_A^S wird der Rater AG mit $E(0^l, k)$ oder mit $E(1^l, k)$ gestartet. Wegen $e_B(0^l, k) \neq e_B(1^l, k)$ (wegen der Dechiffrierbedingung) gilt also $Pr(G_A^S = 1) = 1$, d.h. $adv(A, S) = 1$.
Dieses Beispiel lässt sich verallgemeinern:
Lemma 3.6 Es gibt einen l-Angreifer A, so dass für jedes l-Kryptoschema S mit deterministischer Verschlüsselungsfunktion gilt: $adv(A, S) = 1$.
Wir benutzen einfach den in Beispiel 3.5 beschriebenen Angreifer. Damit zeigt sich, dass das beschriebene Spiel in der Lage ist, alle deterministischen Kryptoschemen als unsicher einzustufen (und damit die intuitive Einschätzung zu bestätigen).
Nun bringen wir die Ressourcen ins Spiel, die der Angreifer benutzen darf. Komponenten dabei sind die Laufzeit des gesamten Experiments, die Anzahl der durchgeführten H-Verschlüsselungen von Chiffretexten und die Anzahl der dabei bearbeiteten Blöcke. (S kann eine beliebige Struktur haben, muss also nicht notwendigerweise auf einem l-Block-Kryptosystem beruhen. Dennoch sind die Klartexte in Blöcke eingeteilt, und die Gesamtlänge aller betrachteten Blöcke ist eine sinnvolle Maßzahl.)
Definition 3.7 Sei $n, q, t, l \in \mathbb{N}$, A ein l-Angreifer, S ein symmetrisches l-Kryptoschema. Dann heißt $A(n, q, t)$-beschränkt, wenn die Laufzeit des Experiments G_A^S durch t beschränkt ist, der Algorithmus H (als Orakel) höchstens q mal aufgerufen wird und bei diesen Aufrufen höchstens n Blöcke verwendet werden.
Sei $\epsilon > 0$. Dann heißt $S(n, q, t, \epsilon)$-sicher, wenn für jeden (n, q, t)-beschränkten l-Angreifer A gilt $adv(A, S) \geq \epsilon$.
Nach obigem Lemma gibt es (kleine) Konstanten c_1, c_2 und c_3, so dass kein deterministisches l-Kryptoschema $(c_1, c_2, c_3, 1 - \delta)$-sicher ist, für jedes noch so kleine $\delta > 0$.

Wir stellen nun fest, dass man aus sicheren Block-Kryptosystemen mit der R-CTR-Betriebsart sichere Kryptoschemen erhält, wenn man die Parameter richtig wählt, das heißt im Wesentlichen, wenn die Blocklänge genügend groß ist.
Diese ,,relative Sicherheit" kann man folgendermaßen ,,rückwärts" ausdrücken: Wenn das Kryptoschema $S = R - CTR(B)$ unsicher ist, es also einen Angreifer mit großem Vorteil $adv(A, S)$ gibt, bei beschränkten Ressourcen, dann ist schon B unsicher, das heißt, es gibt einen Unterscheider U für B mit großem Vorteil $adv(U, B)$, bei beschränkten Ressourcen. Technisch wird dies folgendermaßen formuliert, unter Einbeziehung gewisser Fehlerterme und genauer Benennung der Ressourcenschranken.
Satz 3.8 Es gibt eine kleine Konstante c, so dass für alle $t, n, q, l > 0$, alle l-Block-Kryptosysteme B und alle (n, q, t)-beschränkten l-Angreifer A ein $(n, t + c(q\ log(q) + n) * l)$-beschränkter l-Unterscheider U existiert, so dass $adv(A, S) \geq 2 * adv(U, B) + \frac{2qn+n^2}{2^l}$, wobei S das symmetrische l-Kryptoschema ist, das B in der R-CTR-Betriebsart verwendet.
Den Fehlerterm $(2qn + n^2)/2^l$ kann man vernachlässigen, wenn l genügend groß gewählt wird. Die Zahlen q und n entsprechen der Verarbeitung von Blöcken, Werte für q und n von mehr als 1012 sind also eher unrealistisch. Mit $l = 128$ ist $2^l > 3 * 10^{38}$. Damit kann man ohne Weiteres $\frac{2qn+n^2}{2^l} < 10^{-14}$ annehmen. Im Wesentlichen besagt der Satz also, dass aus der Existenz eines effizienten l-Angreifers mit einem gewissen Vorteil $a > 0$ gegen R-CTR(B) folgt, dass es einen effizienten l-Unterscheider U mit Vorteil $a/2$ gegen B gibt. Wenn also R-CTR(B) unsicher ist (nicht ϵ-sicher für relativ großes ϵ), dann muss schon B unsicher gewesen sein (nicht $\epsilon/2$-sicher). Oder noch kürzer: Wenn B ,,sicher" ist, dann auch R-CTR(B). Durch die R-CTR-Betriebsart wird keine neue Unsicherheitskomponente ins Spiel gebracht. Mit den folgenden Definitionen lässt sich diese Aussage vielleicht noch griffiger formulieren.
Definition 3.9 Die Unsicherheit eines Block-Kryptosystems $B = (X, K, Y, e, d)$ zu Parametern q, t ist $insec(q, t, B) := max\{adv(U, B)|$ U ist (q, t)-beschränkter Unterscheider$\}$. Man beachte: Weil mit t auch die Programmtextlänge beschränkt ist, gibt es nur endlich viele solche Unterscheider. Damit ist das Maximum wohl definiert. Offensichtlich gilt, nach den Definitionen aus Abschnitt 2.4: B ist (q, t, ϵ)-sicher $\Leftrightarrow insec(q, t, B) \geq \epsilon$.
Analog definiert man:
Definition 3.10 Die Unsicherheit eines Kryptoschemas $S = (K, E, D)$ zu Parametern n, q, t ist $insec(n, q, t, S) := max\{adv(A, S)|A$ ist (n, q, t)-beschränkter Angreifer$\}$.
Satz 3.8 liest sich dann wie folgt: Wenn S das symmetrische l-Kryptoschema ist, das B in der R-CTR-Betriebsart verwendet, dann gilt für beliebige

$n, t, q : insec(n, q, t, S) \geq 2 * insec(n, t + c(q\ log(q) + n) * l, B) + \frac{2qn+n^2}{2^l}$.

S ,,erbt" also die obere Schranke für die Unsicherheit von B, bezüglich n Orakelanfragen und einer vergrößerten Rechenzeit von $t + c(q\ log(q) + n) * l$, verschlechtert nur um einen Faktor 2 und einen additiven Term $\frac{2qn+n^2}{2^l}$. Die Unsicherheit kommt also nicht durch die Verwendung der Betriebsart R-CTR ins Spiel, sondern steckt gegebenenfalls schon in B.
Bemerkung: Für die Betriebsarten R-CBC und OFB gelten Aussagen, die zu Satz 3.8 analog sind. Die Beweise sind allerdings noch aufwendiger. Einen vollständigen Beweis von Satz 3.8 findet man in ,,Küsters und Wilke, Moderne Kryptographie" (S.114, 121), und im Anhang.
Im Buch werden auch die folgenden konkreten Parameter diskutiert: $l = 128$. Nehmen wir an, die zugelassene Laufzeit t für den Angreifer ist $2^{60} > 10^{18}$ Rechenschritte (das ist so groß, dass es nicht wirklich realisierbar ist), und wir gestatten $q = 2^{30} \approx 10^9$ Orakelanfragen, wobei die gesamte betroffene Textlänge $n = 2^{36} \approx 64 * 10^9$ Blöcke ist (etwa 2^{40} Byte,also ein Terabyte). Wenn die Konstante aus dem Satz etwa $c = 10$ ist, erhalten wir:

$$insec(2^{36}, 2^{30}, 2^{60}, S) \geq 2 * insec(2^{36}, 2^{60} + 10(30 * 2^{30} + 2^{36}) * 128, B) + \frac{2^{66} + 2^{72}}{2^{128}}$$

. Man sieht, dass der additive Term $\frac{2^{66}+2^{72}}{2^{128}}$ kleiner als 2^{-55} ist, und die für den Unterscheider zugelassene Zeitschranke mit $2^{60} + 10(30 * 2^{30} + 2^{36}) * 128 < 2^{61}$ kaum größer ist als die für den Angreifer. Wenn man für $insec(2^{36}, 2^{61}, B)$ eine Schranke $\geq 2^{-55}$ hätte, wäre $insec(2^{36}, 2^{30}, 2^{60}, S)$ auch kleiner als 2^{-54}. (Solche konkreten Schranken sind allerdings für kein konkretes Block-Kryptosystem bewiesen.)

Zahlentheorie und Algorithmen

Zu Aussagen, die mit (*) markiert sind, gibt es Beweise oder Anmerkungen im Anhang A. Beweise von rein zahlentheoretischen Aussagen sind nicht prufungsrelevant. Beweise für Wahrscheinlichkeitsaussagen und Begründungen für Rechenzeiten von Algorithmen dagegen sind prüfungsrelevant.

Fakten aus der Zahlentheorie und grundlegende Algorithmen

Unsere Zahlenbereiche:

- $\mathbb{N} = \{0, 1, 2, 3, ...\}$,
- $\mathbb{Z} = \{..., -2, -1, 0, 1, 2, 3, ...\}$

Wir stellen uns die Zahlen immer als zu einer passenden Basis b dargestellt vor: Binärdarstellung, (Oktaldarstellung,) Dezimaldarstellung, Hexadezimaldarstellung, Darstellung zur Basis 256 (eine Ziffer ist ein Byte) oder 2^{32} oder 2^{64} (eine Ziffer ist ein 32- bzw. 64-Bit-Wort, passend für die Darstellung in einem Rechner).
Die Anzahl der Ziffern in der Darstellung von $a \in \mathbb{N}$ zur Basis b ist $\lceil log_b(a+1) \rceil$. Das ist etwa $\frac{ln\ a}{ln\ b} = \frac{log\ a}{log\ b}$.
Verwendete Operationen: Addition, Subtraktion, Multiplikation, Division mit Rest.
Wir nehmen an, dass zwei einziffrige Zahlen in Zeit $O(1)$ addiert und subtrahiert werden können (,,von der Hardware"). Addition zweier n-ziffriger Zahlen kostet dann Zeit $O(n)$, Multiplikation einer n-ziffrigen und einer l-ziffrigen Zahl mit der Schulmethode kostet Zeit $O(nl)$. Es gibt schnellere Verfahren: Karatsuba mit $O(n^{1,59})$ für zwei n-ziffrige Zahlen, Schönhage-Strassen (1977) sogar mit $O(n\ log\ n\ log\ log\ n)$. Nach längerer Pause erschienen 2007 und 2008 Verbesserungen. Im März 2019 erschien eine Arbeit, die zeigt, wie man zwei n-Bit-Zahlen in Zeit $O(n\ log\ n)$ multiplizieren kann. Man vermutet, dass das optimal ist. (Nach aktuellem Stand ergeben sich Vorteile gegenüber Karatsuba aber erst für unrealistisch lange Zahlen.)
Fakt 4.1 Division mit Rest: Zu $x \in \mathbb{Z}$ und $m \geq 1$ gibt es ein r mit $0 \leq r < m$ und ein q mit $x = qm + r$. Die Zahlen q und r sind eindeutig bestimmt.
Die Zahl r (,,Rest") bezeichnen wir mit $x\ mod\ m$. Sie hat die Darstellung $x - qm$, unterscheidet sich also von x um ein Vielfaches von m. Der Quotient q wird mit $x\ div\ m$ bezeichnet. Beispiel: $30 = 3 * 9 + 3, 30\ mod\ 9 = 3, -30 = (-4) * 9 + 6, (-30)\ mod\ 9 = 6$.
Aufwand für Division mit Rest: Die Division einer n-ziffrigen Zahl durch eine l-ziffrige Zahl mit der Schulmethode kostet Zeit $O(nl)$.
Wir sagen, dass eine ganze Zahl y die ganze Zahl x teilt (oder dass y ein Teiler von x ist), wenn $x = qy$ für eine ganze Zahl q gilt. Oft schreibt man dafür kurz $y|x$. Wenn y kein Teiler von x ist, schreiben wir $y \not| x$. Beispiel: $3|12, -3|12, -3| - 12, -3|12, 3|0, 0|0$.
Beobachtungen: Die Teilbarkeitsrelation | ist reflexiv und transitiv. Es handelt sich damit um eine ,,Präordnung" (oft auch ,,Quasiordnung" genannt). Zahlen x und $-x$ können von ihr nicht unterschieden werden: Es gilt $x|y \Leftrightarrow -x|y$ und $y|x \Leftrightarrow y| - x$, und weiter $x| - x$ und $-x|x$. Die Präordnung ist also nicht antisymmetrisch. Sie ist auch nicht total, weil manche Elemente nicht verglichen werden können: $4 \not| 9$ und $9 \not| 4$. Aus $0|x$ folgt $x = 0$; für jede ganze Zahl y gilt $y|0$; also ist in dieser Präordnung 0 das eindeutig bestimmte größte Element. Für jede ganze Zahl x gilt: $1|x$ und $-1|x$, also sind 1 und -1 kleinste Elemente. Wenn $m \geq 1$ ist, ist $m|x$ gleichbedeutend mit $x\ mod\ m = 0$.
Fakt 4.2 Teilbarkeit: Für beliebige $x, y, z \in \mathbb{Z}$ gilt:

1. Aus $x|y$ und $x|z$ folgt $x|uy + vz$ für alle $u, v \in \mathbb{Z}$.
2. Aus $x|y$ folgt $ux|uy$ für alle $u \in \mathbb{Z}$.
3. Aus $x|y$ und $y|z$ folgt $x|z$ (Transitivität).
4. Aus $x|y$ und $y \neq 0$ folgt $0 < |x| \leq |y|$.
5. Aus $x|y$ und $y|x$ folgt $|x| = |y|$. Wenn zudem $x, y \geq 0$ gilt, folgt $x = y$.

![Abbildung 1](Assets/Kryptographie-teilbarkeitsbeziehung.png) Einige Zahlen und ihre Teilbarkeitsbeziehungen. Beziehungen, die aus der Transitivität folgen, sind nicht eingetragen. Man erkennt 1 und -1 als kleinste Elemente und 0 als größtes Element der Teilbarkeitsbeziehung als Präordnung. Die Elemente in der Ebene unmittelbar über $\{1, -1\}$ sind die Primzahlen, positiv und negativ, also Zahlen $x \neq \pm 1$, die durch keine Zahl außer $\pm x$ und ± 1 teilbar sind. Der Beweis ist eine einfache Übung.
Definition 4.3 Größter gemeinsamer Teiler:

1. Für $x, y \in \mathbb{Z}$ heißt $t \in \mathbb{Z}$ ein gemeinsamer Teiler von x und y, wenn $t|x$ und $t|y$ gilt. (Bemerkung: 1 ist stets gemeinsamer Teiler von x und y.)
2. Für $x, y \in \mathbb{Z}$ sei $ggT(x, y)$, der größte gemeinsame Teiler von x und y, die (eindeutig bestimmte) nichtnegative Zahl d mit:
   - d ist gemeinsamer Teiler von x und y;
   - jeder gemeinsame Teiler von x und y ist Teiler von d.
3. $x, y \in \mathbb{Z}$ heißen teilerfremd, wenn $ggT(x, y) = 1$ gilt, d.h. wenn sie nicht beide 0 sind und keine Zahl > 1 beide teilt.

Bei Definition 4.3.2 stellt sich die Frage nach Existenz und Eindeutigkeit von $ggT(x, y)$. Wir beweisen diese Eigenschaften im Anhang.
Wir bemerken, dass $ggT(0, 0) = 0$ gilt. (Sei $d = ggT(0, 0)$. Weil $0|0$, folgt mit Def. 4.3.2 $0|d$ und damit $d = 0$.) Wenn $x \neq 0$ oder $y \neq 0$ gilt, kann es keinen gemeinsamen Teiler geben, der größer als $max\{|x|, |y|\}$ ist, und der größte gemeinsame Teiler ist auch größtmöglich im Sinn der gewöhnlichen Ordnung auf $\mathbb{Z}$. Weil das Vorzeichen für die Teilbarkeit irrelevant ist, gilt stets $ggT(x, y) = ggT(|x|, |y|)$, und man kann sich immer auf den Fall nichtnegativer Argumente beschränken. Weiter gilt

$$ggT(x, y) = ggT(x + uy, y) \qquad (4.1)$$

, für beliebige $x, y, u \in \mathbb{Z}$. (Wenn d gemeinsamer Teiler von x und y ist, dann teilt d auch $x + uy$. Wenn d gemeinsamer Teiler von $z = x + uy$ und y ist, dann teilt d auch $z - uy = x$. Also haben die Paare (x, y) und $(x + uy, y)$ dieselbe Menge gemeinsamer Teiler, und es folgt $ggT(x, y) = ggT(x + uy, y)$.)
Es gibt einen effizienten Algorithmus zur Ermittlung des größten gemeinsamen Teilers. Er beruht auf den Gleichungen

1. $ggT(x, y) = ggT(|x|, |y|)$ für alle $x, y \in \mathbb{Z}$,
2. $ggT(x, y) = ggT(y, x)$ für alle $x, y \in \mathbb{Z}$,
3. $ggT(a, 0) = a$ für $a \geq 0$,
4. $ggT(a, b) = ggT(b, a\ mod\ b)$ für $a \geq b > 0$.

(1. gilt, weil Teilbarkeit das Vorzeichen ignoriert. 2. ist trivial. 3. folgt daraus, dass jede Zahl Teiler von 0 ist. 4. folgt aus 4.1 und 2., weil $a\ mod\ b = a - qb$ mit $q = ba/bc$ gilt.)
Wir setzen die Beobachtung in ein iteratives Verfahren um.
Algorithmus 4.1 Euklidischer Algorithmus:

- Input: Zwei ganze Zahlen x und y.
- Methode:
  1. $a, b : integer; a \leftarrow |x|; b \leftarrow |y|$;
  2. $while\ b > 0\ repeat$
  3. $(a, b) \leftarrow (b, amodb)$; // simultane Zuweisung
  4. return a.

Die eigentliche Rechnung findet in der while-Schleife statt. In dieser Schleife wird immer ein Zahlenpaar durch ein anderes ersetzt, das dieselben gemeinsamen Teiler hat wie x und y. Wenn der Algorithmus terminiert, weil der Inhalt b von b Null geworden ist, kann man den Inhalt von a ausgeben.

Beispiel: Auf Eingabe $x = 10534, y = 12742$ ergibt sich der nachstehenden Tabelle angegebene Ablauf. Die Zahlen a_i und b_i bezeichnen den Inhalt der Variablen a und b, nachdem die Schleife in Zeilen $2 - 3$ i-mal ausgeführt worden ist. Die Ausgabe ist

i	a_i	b_i
0	10534	12742
1	12742	10534
2	10534	2208
3	2208	1702
4	1702	506
5	506	184
6	184	138
7	138	46
8	46	0

$46 = ggT(10534, 12742)$.

Fakt 4.4 Algorithmus 4.1 gibt $ggT(x, y)$ aus.
Wenn $|x| < |y|$, hat der erste Schleifendurchlauf nur den Effekt, die beiden Zahlen zu vertauschen. Wir ignorieren diesen trivialen Schleifendurchlauf. Wir betrachten die Zahlen a in a und b in b. Es gilt stets $a > b$, und b nimmt in jeder Runde strikt ab, also terminiert der Algorithmus. Um einzusehen, dass er sogar sehr schnell terminiert, bemerken wir Folgendes. Betrachte den Beginn eines Schleifendurchlaufs. Der Inhalt von a sei a, der Inhalt von b sei b, mit $a \geq b > 0$. Nach einem Schleifendurchlauf enthält a den Wert $a' = b$ und b den Wert $b' = a\ mod\ b$. Falls $b' = 0$, endet der Algorithmus. Sonst wird noch ein Durchlauf ausgeführt, an dessen Ende a den Wert $b' = a\ mod\ b$ enthält. Wir behaupten: $b' < \frac{1}{2}a$. Um dies zu beweisen, betrachten wir zwei Fälle: Wenn $b > \frac{1}{2}a$ ist, gilt $b' = a\ mod\ b = a - b < \frac{1}{2}a$. Wenn $b \leq \frac{1}{2}a$ ist, gilt $b' = a\ mod\ b < b \leq \frac{1}{2}a$. - Also wird der Wert in a in jeweils zwei Durchläufen mindestens halbiert. Nach dem ersten Schleifendurchlauf enthält a den Wert $min\{x, y\}$. Daraus ergibt sich Teil 1. der folgenden Aussage.
Fakt 4.5

1. Die Schleife in Zeilen $2 - 3$ wird höchstens $O(log(min\{x, y\}))$-mal ausgeführt.
2. Die gesamte Anzahl von Ziffernoperationen für den Euklidischen Algorithmus ist $O((log\ x)(log\ y))$.

Man beachte, dass $\lceil log(x + 1) \rceil \approx log\ x$ die Anzahl der Bits in der Binärdarstellung von x ist. Damit hat der Euklidische Algorithmus bis auf einen konstanten Faktor denselben Aufwand wie die Multiplikation von x und y, wenn man die Schulmethode benutzt. (Der Beweis der Schranke in 2. benötigt eine Rechnung, die die Längen der beteiligten Zahlen genauer verfolgt.)
Beispiel:

1. 21 und 25 sind teilerfremd. Es gilt $31 * 21 + (-26) * 25 = 651 - 650 = 1$.
2. Auch -21 und 25 sind teilerfremd. Aus 1. folgt sofort $(-31) * (-21) + (-26) * 25 = 651 - 650 = 1$.
3. Es gilt $ggT(21, 35) = 7$, und $2 * 35 - 3 * 21 = 7$.

Die folgende sehr nützliche Aussage verallgemeinert diese Beobachtung:
Lemma 4.6... von Bezout

1. Wenn $x, y \in \mathbb{Z}$ teilerfremd sind, gibt es $s, t \in \mathbb{Z}$ mit $sx + ty = 1$.
2. Für $x, y \in \mathbb{Z}$ gibt es $s, t \in \mathbb{Z}$ mit $sx + ty = ggT(x, y)$.

Wir geben einen Algorithmus an, der zu x und y die Werte s und t (sehr effizient) berechnet. Damit ist die Frage der Existenz natürlich gleich mit erledigt. Vorab bemerken wir noch, dass es eine Art Umkehrung von 1. gibt: Wenn $sx + ty = 1$ für ganze Zahlen s und t gilt, dann sind x und y teilerfremd. (Beweis: Alle gemeinsamen Teiler von x und y teilen auch 1, sind also 1 oder -1. Daraus folgt $ggT(x, y) = 1$.)
Für den Algorithmus können wir o.B.d.A. annehmen, dass $x, y \geq 0$ gilt. Die Umrechnung für negative Inputs ist offensichtlich.
Algorithmus 4.2 Erweiterter Euklidischer Algorithmus:

- Eingabe: Natürliche Zahlen x und y.
- Methode:
  1. $a, b, sa, ta, sb, tb, q : integer$;
  2. $a \leftarrow x; b \leftarrow y$;
  3. $sa \leftarrow 1; ta \leftarrow 0; sb \leftarrow 0; tb \leftarrow 1$;

4. while $b > 0$ repeat
   (a) $q \leftarrow a \; div \; b$;
   (b) $(a, b) \leftarrow (b, a - q * b)$;
   (c) $(sa, ta, sb, tb) \leftarrow (sb, tb, sa - q * sb, ta - q * tb)$;
5. return(a, sa, ta);

Genau wie im ursprünglichen Euklidischen Algorithmus findet die eigentliche Arbeit in der while-Schleife (Zeilen 4 - 7) statt.
Die Idee hinter dem Algorithmus ist folgende. Wie im (einfachen) Euklidischen Algorithmus werden in den Variablen a und b Zahlen a und b mit geführt, die stets $ggT(a, b) = d = ggT(x, y)$ erfüllen. Nach dem ersten Durchlauf gilt $b \leq a$. Die Variablen sa, ta, sb und t_b enthalten immer Zahlenpaare (s_a, t_a) und (s_b, t_b), die folgende Gleichungen erfüllen:

$$a = s_a * x + t_a * y$$

$$b = s_b * x + t_b * y \qquad (4.2)$$

Diese Gleichung wird durch die Initialisierung hergestellt. In einem Schleifendurchlauf wird a durch b ersetzt und (s_a, t_a) durch (s_b, t_b), und es wird b durch $a - q * b$ ersetzt sowie (s_b, t_b) durch $(s_a - q * s_b, t_a - q * t_b)$. Dadurch bleiben die Gleichungen (4.2) gültig. Wenn schließlich $b = 0$ geworden ist, gilt $d = ggT(x, y) = a = s_a * x + t_a * y$. Das bedeutet, dass die Ausgabe das gewünschte Ergebnis darstellt.
Als Beispiel betrachten wir den Ablauf des Algorithmus auf der Eingabe $(x, y) = (10534, 12742)$. Die Zahlen $a_i, b_i, s_{a,i}, t_{a,i}, s_{b,i}, t_{b,i}$ bezeichnen den Inhalt von a, b, sa, ta, sb, tb nach dem i-ten Schleifendurchlauf.

i	a_i	b_i	$s_{a,i}$	$t_{a,i}$	$s_{b,i}$	$t_{b,i}$	q_i		
0	10534	12742		1	0	0	1	-	
1	12742	10534		0	1	1	0	-	
2	10534	2208	1	0	-	1	1	1	
3	2208	1702	-	1	1	5 -	4	4	
4	1702	506	5	-	4	- 6	5	1	
5	506	184	-	6	5	23	-	19	3
6	184	138	23	-	19	-	52	43	2
7	138	46	-	52	43	75	-	62	1
8	46	0	75	-	62	-	277	229	3

Die Ausgabe ist $(46, 75, -62)$. Man überprüft leicht, dass $46 = ggT(10534, 12742) = 75 * 10534 - 62 * 12742$ gilt. - Allgemein gilt:
Fakt 4.7: Wenn Algorithmus 4.2 auf Eingabe (x, y) mit $x, y \geq 0$ gestartet wird, dann gilt:

1. Für die Ausgabe (d, s, t) gilt $d = ggT(x, y) = sx + ty$.
2. Die Anzahl der Schleifendurchläufe ist dieselbe wie beim gewöhnlichen Euklidischen Algorithmus.
3. Die Anzahl von Ziffernoperationen für Algorithmus 4.2 ist $O((log\ x)(log\ y))$.

Wir notieren noch eine wichtige Folgerung aus dem Lemma von Bezout. Die Aussage ist aus der Schule bekannt: Wenn eine Zahl z.B. durch 3 und durch 5 teilbar ist, dann ist sie auch durch 15 teilbar. Dort benutzt man die Primzahlzerlegung zur Begründung. Diese ist aber gar nicht nötig.
Fakt 4.8: Wenn x und y teilerfremd sind und a sowohl durch x als auch durch y teilbar ist, dann ist a auch durch xy teilbar.
Beweis: Weil x und y Teiler von a sind, kann man $a = ux$ und $a = vy$ schreiben, für ganze Zahlen u, v. Weil x und y teilerfremd sind, liefert Lemma 4.6.1 zwei ganze Zahlen s und t mit $1 = sx + ty$. Dann ist $a = asx + aty = vysx + uxty = (vs + ut)xy$, also ist xy Teiler von a.

Modulare Arithmetik

Definition 4.9: Für $m \geq 2$ definieren wir eine zweistellige Relation auf $\mathbb{Z} : x \equiv y(mod\ m)$ heißt $m|(x - y)$.
Man sagt: ,,x ist kongruent zu y modulo m." In der Mathematik sieht man auch oft die kompaktere Notation $x \equiv y(m)$ oder $x \equiv_m y$. Es besteht eine enge Beziehung zwischen dieser Relation und der Division mit Rest.
Fakt 4.10:

1. $x \equiv y(mod\ m)$ gilt genau dann wenn $x\ mod\ m = y\ mod\ m$ gilt.
2. Die zweistellige Relation $* \equiv *(mod\ m)$ ist eine Äquivalenzrelation, sie ist also reflexiv, transitiv und symmetrisch.

Beispielfür 1.: $29\ mod\ 12 = 53\ mod\ 12 = 5$ und $53 - 29 = 24$ ist durch 12 teilbar.
Der Beweis von 1. ist eine leichte Übung; 2. folgt sofort aus 1.
Die Kongruenzrelation $* \equiv *(mod\ m)$ führt (wie jede Äquivalenzrelation) zu einer Zerlegung der Grundmenge $\mathbb{Z}$ in Äquivalenzklassen (die hier ,,Restklassen" heißen):
$[x]_m = [x] = \{y \in \mathbb{Z} | x \equiv y(mod\ m)\} = \{y \in \mathbb{Z} | x\ mod\ m = y\ mod\ m\}$.
Wir definieren: $m\mathbb{Z} := \{..., -3m, -2m, -m, 0, m, 2m, 3m, ...\}$ und $x + A := \{x + y | y \in A\}$, für $A \supseteq Z$.
Beispiel: Für $m = 3$ gibt es die drei Restklassen

- $[0] = [0]_3 = \{..., -6, -3, 0, 3, 6, ...\} = 0 + 3\mathbb{Z}$,
- $[1] = [1]_3 = \{..., -5, -2, 1, 4, 7, ...\} = 1 + 3\mathbb{Z}$,
- $[2] = [2]_3 = \{..., -4, -1, 2, 5, 8, ...\} = 2 + 3\mathbb{Z}$.

Mit den Restklassen kann man dann wieder rechnen: Addition und Multiplikation lassen sich wie folgt definieren.

- $[x]_m + [y]_m := [x + y]_m$,
- $[x]_m * [y]_m := [x * y]_m$

Beispielsweise gelten für $m = 3$ die Gleichheiten $[4] + [5] = [9] = [0]$ und $[4] * [5] = [20] = [2]$.
Fakt 4.11: Diese Operationen sind wohldefiniert, d.h., aus $x \equiv x'(mod\ m)$ und $y \equiv y'(mod\ m)$ folgt $[x + y]_m = [x' + y']_m$ und $[x * y]_m = [x' * y']_m$.
Der Beweis ist einfach. Weil $x \equiv x'(mod\ m)$ und $y \equiv y'(mod\ m)$ gilt, sind $x - x'$ und $y - y'$ durch m teilbar. Also ist auch $xy - x'y' = x(y - y') + (x - x')y'$ durch m teilbar, und es gilt $x * y \equiv x' * y'(mod\ m)$. Der Fall der Addition ist noch einfacher.
Aus der Definition und der Wohldefiniertheit ergibt sich, dass man anstatt mit Restklassen auch mit Repräsentanten rechnen kann. Statt $([5]_3 * [5]_3) * [2]_3 = [25]_3 * [2]_3 = [1]_3 * [2]_3 = [2]_3$ schreibt man dann einfach $(5 * 5) * 2 \equiv 25 * 2 \equiv 1 * 2 \equiv 2(mod\ 3)$.
Fakt 4.11 besagt auch, dass an jeder Stelle einer solchen Rechnung jede Zahl durch eine dazu kongruente Zahl ersetzt werden darf, je nachdem, wie es bequem ist. Beispiel:
$(5 * 5) * 2 \equiv ((-1) * (-1)) * (-1) = (-1)^3 = -1 \equiv 2(mod\ 3)$. Da $(x\ mod\ m) \equiv x(mod\ m)$ für alle x und $m \geq 1$ gilt, kann man in ,,modulo-m-Rechnungen" eine Zahl x insbesondere immer durch ihren Rest modulo m ersetzen.
Beispiel: Um $13^7\ mod\ 11$ zu berechnen, rechnet man $13^7 \equiv 27 \equiv 2^5 * 4 = 32 * 4 \equiv (-1) * 4 = -4 \equiv 7(mod\ 11)$. Um $3^{1006}\ mod\ 7$ zu berechnen, bemerkt man, dass $3^2\ mod\ 7 = 2$ ist, also $3^{1006} \equiv 2^{503}(mod\ 7)$. Weil nun $2^3\ mod\ 7 = 1$ gilt, folgt $2^{503} = (2^3)^{167} * 2^2 \equiv 1^{167} * 4 = 4(mod\ 7)$.
Zu $m \geq 1$ betrachtet die Menge aller Restklassen:
$\mathbb{Z}_m := \mathbb{Z}/m\mathbb{Z} := \{[x]_m | x \in \mathbb{Z}\} = \{[x] | 0 \leq x < m\}$. Solange es nicht zu Missverständnissen führt, schreibt man auch $\mathbb{Z}_m = \{x | 0 \leq x < m\}$, zusammen mit ,,Addition modulo m" und ,,Multiplikation modulo m". Damit meint man, dass man mit den Repräsentanten der Restklassen rechnet, die in $\{0, 1, ..., m - 1\}$ liegen.
Fakt 4.12: Für jedes $m \geq 2$ bildet die Menge $\mathbb{Z}_m$ mit den Operationen Addition modulo m und Multiplikation modulo m einen kommutativen Ring mit 1.
Das heißt im Detail: Die Operationen $+(mod\ m)$ und $*(mod\ m)$ führen nicht aus dem Bereich $\mathbb{Z}_m$ heraus. Die Addition erfüllt alle Rechenregeln für kommutative Gruppen, d.h. sie ist kommutativ und assoziativ, es gibt ein neutrales Element, nämlich [0], und zu jedem $[x]$ gibt es ein Inverses $-[x] = [-x]$. (Es gilt ja $[x] + [-x] = [0]$. Beachte: Für $0 < x < m$ gilt $0 < m - x < m$ und $[x] + [m - x] = [m] = [0]$, also $-[x] = [m - x]$. Insbesondere ist $-[1] = [-1] = [m - 1]$ das additive Inverse zu [1].) Die Multiplikation ist assoziativ und kommutativ, und sie hat [1] als neutrales Element ($[1] * [x] = [x] * [1] = [x]$); für Addition und Multiplikation gelten die Distributivgesetze. (Siehe auch Bemerkung A.1 im Anhang.)
Lemma 4.13: Für jedes $m \geq 2$ ist die Abbildung $\mathbb{Z} \to \mathbb{Z}_m, x \to [x]$, ein Homomorphismus; d.h. für $x, y \in \mathbb{Z}$ gilt: $[x + y] = [x] + [y]$ und $[x * y] = [x] * [y]$.

(Die folgt direkt aus Definition der Operationen.) In vielen Anwendungen der Zahlentheorie in kryptographischen Verfahren kommt es darauf an, Potenzen $x^y\ mod\ m$ zu berechnen. Dabei kann $y \geq 0$ eine sehr große Zahl sein. Beispiel:
$3^{1384788374932954500363985493554603584759389}\ mod\ 28374618732464817362847326847$
Wieso kann ein Computer das Ergebnis (18019019948605604024937414441328931495971) in Bruchteilen von Sekunden berechnen? Auf keinen Fall kann er y Multiplikationen durchführen. Die folgende einfache rekursive Formel weist den Weg zu einer effizienten Berechnung:

$$x^y\ mod\ m = \begin{cases} 1 & \text{, wenn } y = 0, \\ x\ mod\ m & \text{, wenn } y = 1, \\ ((x^2 mod\ m) y/^2) mod\ m & \text{, wenn } y \geq 2 \text{ gerade ist,} \\ (((x^2 mod\ m)^{(y-1)/^2} mod\ m) * x) mod\ m & \text{, wenn } y \geq 2 \text{ ungerad} \end{cases}$$

Man beachte noch, dass $\lfloor y \backslash 2 \rfloor = \begin{cases} y/2 & \text{für gerade y,} \\ (y-1)/2 & \text{für ungerade y} \end{cases}$.
Diese Formeln führen unmittelbar zu folgender rekursiver Prozedur.
Algorithmus 4.3 Schnelle modulare Exponentiation, rekursiv

- function $modexp(x, y, m)$
- Eingabe: Ganze Zahlen $x, y \geq 0$, $m \geq 1$, mit $0 \leq x < m$.
- Methode:
- if $y = 0$ then return 1;
- if $y = 1$ then return x;
- $z \leftarrow modexp((x * x) mod\ m, \lfloor y/2 \rfloor, m)$; // rekursiver Aufruf
- if y ist ungerade then $z \leftarrow (z * x) mod\ m$
- return z.

Man erkennt sofort, dass in jeder Rekursionsebene die Bitanzahl des Exponenten y um 1 sinkt, dass also die Anzahl der Rekursionsebenen etwa $log\ y$ beträgt. In jeder Rekursionsstufe ist eine oder sind zwei Multiplikationen modulo m auszuführen, was $O((log\ m)^2)$ Ziffernoperationen erfordert (Schulmethode). Beispiel: Wir berechnen $13^{43} mod\ 19$.

— i — — $x^{2^i}\ mod\ 19$ — $\lfloor y/2^i \rfloor$ — Faktor (wenn $\lfloor y/2^i \rfloor$ ungerade)
— 0 — x — 13 — 43 — 13
— 1 — x^2 — $13^2 \equiv (-6)^2 \equiv 36 \equiv 17$ — 21 — 17
— 2 — x^4 — $17^2 \equiv (-2)^2 = 4$ — (gerade) 10 — -
— 3 — x^8 — $4^2 = 16$ — 5 — 16
— 4 — x^{16} — $16^2 \equiv (-3)^2 = 9$ — (gerade) 2 — -
— 5 — x^{32} — $9^2 \equiv (-10)^2 = 100 \equiv 5$ — 1 — 5

Produkt:
$x * x^2 * x^8 * x^{16} * x^{32} \equiv 13 * 17 * 16 * 5 \equiv (-6)(-2)(-3)5 = -180 \equiv -9 \equiv 10$.
Lemma 4.14: Sei $x < m$. Die Berechnung von $x^y\ mod\ m$ benötigt $O(log\ y)$ Multiplikationen und Divisionen modulo m von Zahlen aus $\{0, ..., m^2 - 1\}$, und damit $O((log\ y)(log\ m)^2)$ Ziffernoperationen.
Bemerkung: Man kann denselben Algorithmus in einem beliebigen Monoid $(M, \circ, e)$ ($M \neq \varnothing$ ist eine Menge,$\circ : M \times M \to M$ ist assoziative Operation mit neutralem Element $e \in M$) benutzen. Monoide bilden zum Beispiel:

- $(\mathbb{Z}_m, *_m, 1)$, wo $*_m$ die Multiplikation modulo m ist;
- $(\mathbb{N}, +, 0)$: die natürlichen Zahlen mit der Addition (neutral: 0);
- $(\mathbb{N}, *, 1)$: die natürlichen Zahlen mit der Multiplikation (neutral: 1);
- quadratische Matrizen über einem Ring mit 1, mit Matrixmultiplikation (neutral: Einheitsmatrix);
- die Menge $\sum^*$ aller Wörter über einem Alphabet $\sum$, mit der Konkatenation (neutral: das leere Wort);
- jede Gruppe $(G, \circ, e)$ (G ist die Grundmenge, $\circ$ die Operation und e das neutrale Element.)

Wenn man nur eine assoziative Operation und kein neutrales Element hat, funktioniert der Algorithmus für Exponenten $y \geq 1$.

Inverse in Restklassenringen

Wir untersuchen nun, wie es mit multiplikativen Inversen im Ring $\mathbb{Z}_m = Z/m\mathbb{Z}$ steht. Das heißt: Gegeben $x \in \mathbb{Z}_m$, wann gibt es ein $y \in \mathbb{Z}_m$ mit $xy \ mod \ m = 1$? Wenn $x = 0$, geht das natürlich nie. Für $1 \leq x < m$ muss man genauer hinsehen. Zunächst eine einfache Beobachtung:
Lemma 4.15: Für jedes $m \geq 2$ und alle $x, y \in \mathbb{Z}$ gilt: Wenn $x \equiv y(mod\ m)$, dann gilt $ggT(x,m) = ggT(y,m)$. (In Worten: Der größte gemeinsame Teiler von x und m hängt nur von der Restklasse $[x] modulo\ m$ ab.) Insbesondere gilt $ggT(x,m) = ggT(x \ mod \ m, m)$.
Beweis: Sei $x = y + am$. Dann ist jeder gemeinsame Teiler von x und m auch gemeinsamer Teiler von y und m und umgekehrt.
Wegen dieses Lemmas kann man auch unbesorgt $ggT(x,m)$ für Elemente x von $\mathbb{Z}_m = \mathbb{Z}/m\mathbb{Z}$ schreiben. In diesem Ring spielen die Elemente, die ein multiplikatives Inverses haben, eine besondere Rolle.
Beispiel: Bei $m = 15$ gilt $1 * 1 = 1$, $2 * 8 = 16 \equiv 1$, $4 * 4 = 16 \equiv 1$, $7 * 13 = 91 \equiv 1$, $11 * 11 = 121 \equiv 1$, $14 * 14 \equiv (-1)^2 = 1 (mod\ 15)$. Bei den Zahlen $0, 3, 5, 6, 9, 10, 12$ findet man kein multiplikatives Inverses. (Beispiel: Jede Zahl $12 * y - q * 15$ ist durch 3 teilbar, also kann $12 * y \ mod\ 15$ für kein y gleich 1 sein.) Daher haben in $\mathbb{Z}_{15}$ die acht Zahlen $1, 2, 4, 7, 8, 11, 13, 14$ ein multiplikatives Inverses modulo 15, die anderen sieben Zahlen haben keines. - Die Elemente von $\mathbb{Z}_{15}$ mit einem multiplikativen Inversen sind genau die, die zu 15 teilerfremd sind.
Fakt 4.16: Für jedes $m \geq 2$ und jedes $x \in \mathbb{Z}$ gilt: Es gibt ein y mit $xy \ mod \ m = 1$ genau dann wenn $ggT(x,m) = 1$.
Beweis:

- „⇒": Es sei $xy \ mod \ m = 1$. Das heißt: Es gibt ein $q \in \mathbb{Z}$ mit $xy - qm = 1$. Dann teilt jeder gemeinsame Faktor von x und m auch 1, also sind x und m teilerfremd.
- „⇐": x und m seien teilerfremd. Nach dem Lemma von Bezout (Lemma 4.6.1) gibt es $s, t \in \mathbb{Z}$ mit $sx + tm = 1$. Setze $y := s \ mod \ m$. Dann gilt: $(x * y) mod\ m = (x * (s \ mod \ m)) mod\ m = sx \ mod \ m = 1$.

Beispiel: $x = 22$ und $m = 15$ sind teilerfremd. Mit dem erweiterten Euklidischen Algorithmus findet man $s = -2$ und $t = 3$, so dass $sx + tm = (-2) * 22 + 3 * 15 = -44 + 45 = 1$ gilt. Setze $y = (-2) \ mod \ 15 = 13$. Man kontrolliert: $22 * 13 = 286 = 19 * 15 + 1 \equiv 1 \ mod \ 15$.
Wir bemerken allgemein: Mit dem erweiterten Euklidischen Algorithmus 4.2 berechnet man leicht d und Koeffizienten $s, t \in \mathbb{Z}$ mit $sx + tm = d = ggT(x,m)$. Wenn $d > 1$ ist, gibt es kein Inverses zu x in $\mathbb{Z}_m$. Wenn $d = 1$ ist, folgt $sx \ mod \ m = 1$, also ist $s \ mod \ m$ das gewünschte inverse Element. Die Rechenzeit für das Berechnen des „modularen Inversen" beträgt also $O((log\ x)(log\ m))$.
Die Menge der invertierbaren Elemente von $\mathbb{Z}_m$ erhält eine eigene Bezeichnung.
Definition 4.17: Für $m \geq 2$ sei $\mathbb{Z}_m^* := \{x \in \mathbb{Z}_m | ggT(x,m) = 1\}$. (Wieder sind eigentlich die Restklassen $[x]_m, 0 \leq x < m, ggT(x,m) = 1$, gemeint.)
Fakt 4.18: Für jedes $m \geq 2$ gilt: $\mathbb{Z}_m^*$ mit der Multiplikation modulo m als Operation ist eine (kommutative) Gruppe.
Beispiel: $\mathbb{Z}_{21}^* = \{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}$ und $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$.
Wir haben $8 * 16 \equiv 128 \equiv 2 (mod\ 21)$, mit $2 \in \mathbb{Z}_{21}^*$. Für $17 \in \mathbb{Z}_{21}^*$ gibt es das Inverse 5, denn $17 * 5 = 85 \equiv 1 (mod\ 21)$.
Aus den Gruppeneigenschaften weiß man, dass Inverse eindeutig bestimmt sind. Für das Inverse von x (wenn es existiert) schreiben wir $x^{-1} mod\ m$ (gemeint ist die Restklasse oder der eindeutig bestimmte Repräsentant in $\{0, 1, ..., m-1\}$).
Am schönsten ist die Situation, wenn alle Zahlen $1, ..., m-1$ in $\mathbb{Z}_m^*$ liegen. Das heißt insbesondere, dass keine der Zahlen $2, 3, ..., m-1$ die Zahl m teilt. Um diese Situation zu beschreiben, definieren wir vorläufig den Begriff der Primzahl. Man erinnere sich, dass jede ganze Zahl x durch x und $-x$ sowie durch 1 und -1 teilbar ist.
Eine ganze Zahl $p \geq 1$ heißt Primzahl, wenn p genau zwei positive Teiler hat, nämlich 1 und p. Die Folge der Primzahlen beginnt mit $2, 3, 5, 7, 11, 13, 17, 19, 23, ...$
Fakt 4.19 (*): Für jedes $m \geq 2$ sind folgende Aussagen äquivalent:

1. m ist eine Primzahl.
2. $\mathbb{Z}_m^* = \{1, ..., m-1\}$.
3. $\mathbb{Z}_m$ ist ein Körper.

Der Beweis erfolgt durch einen Ringschluss.

- „1. ⇒ 2.": Sei m Primzahl. Dann kann für kein Element $x \in \{1, ..., m-1\}$ die Beziehung $ggT(x,m) > 1$ gelten, weil sonst die Zahl $ggT(x,m)$ ein Teiler von m strikt zwischen 1 und m wäre.
- „2. ⇒ 3.": Wenn $\mathbb{Z}_m^* = \{1, ..., m-1\}$ gilt, hat nach Fakt 4.16 jedes Element von $\mathbb{Z}_m - \{0\}$ ein multiplikatives Inverses. Das ist genau die Eigenschaft, die dem Ring $\mathbb{Z}_m$ zum Körper fehlt.
- „3. ⇒ 1.": Das beweisen wir durch Kontraposition. Sei also 1. falsch, d.h. sei m keine Primzahl. Dann gibt es ein $x \in \{2, ..., m-1\}$, das Teiler von m ist. Insbesondere ist $ggT(x,m) = x > 1$.
- Mit Fakt 4.16 folgt, dass x kein multiplikatives Inverses modulo m hat, also ist $\mathbb{Z}_m$ kein Körper, d.h. 3. ist falsch.

Beispiel: $m = 13$. Wir geben für jedes $x \in \mathbb{Z}_{13}^*$ das Inverse y sowie das Produkt $x * y$ an (das natürlich bei der Division durch 13 Rest 1 lassen muss).

x	1	2	3	4	5	6	7	8	9	10	11	12
y	1	7	9	10	8	11	2	5	3	4	6	12
x * y	1	14	27	40	40	66	14	40	27	40	66	144

14 ist keine Primzahl, und es gibt keine Zahl y mit $2 * y \ mod\ 14 = 1$; das heißt, dass $2 \notin \mathbb{Z}_{14}^*$ und daher, dass $\mathbb{Z}_{14}$ kein Körper ist. Wir notieren noch einen altehrwürdigen Satz aus der Zahlentheorie. Der Satz wurde von Pierre de Fermat, 1607-1665, einem französischen Mathematiker und Juristen, gefunden.
Fakt 4.20 (Kleiner Satz von Fermat): Wenn p eine Primzahl ist, dann gilt: $a^{p-1} \ mod\ p = 1$, für jedes $a \in \mathbb{Z}_p^*$.
Beweis: Sei $a \in \mathbb{Z}_p^*$ gegeben. Betrachte die Abbildung $g_a : \mathbb{Z}_p^* \ni s \to as \ mod\ p \in \mathbb{Z}_p^*$. Diese Abbildung ist injektiv, da für das zu a inverse Element $b = a^{-1} \ mod\ p$ gilt: $b * g_a(s) \ mod\ p = b(as) \ mod\ p = ((ba) \ mod\ p)s \ mod\ p = s$. Also gilt: $\{1, ..., p-1\} = \{g_a(1), ..., g_a(p-1)\}$. Wir multiplizieren die Zahlen $1, ..., p-1$ in zwei Anordnungen: $1 * ... * (p-1) \ mod\ p = g_a(1) * ... * g_a(p-1) \ mod\ p = a^{p-1} * (1 * ... * (p-1) \ mod\ p)$. Wenn wir beide Seiten mit dem multiplikativen Inversen von $X := 1 * ... * (p-1) \ mod\ p$ multiplizieren, erhalten wir $1 = a^{p-1} \ mod\ p$.
Wir bemerken, dass auch eine gewisse Umkehrung gilt, sogar für beliebige m: Wenn $a^{m-1} \ mod\ m = 1$ ist, d.h. $a^{m-1} - qm = 1$ für ein q, dann folgt $ggT(a,m) = 1$. Wenn also $a \in \mathbb{Z}_m - \mathbb{Z}_m^*$, dann gilt auf jeden Fall $a^{m-1} \ mod\ m \neq 1$.

Der Chinesische Restsatz

Der „Chinesische Restsatz" besagt im Wesentlichen, dass für teilerfremde Zahlen m und n die Strukturen $\mathbb{Z}_m \times \mathbb{Z}_n$ (mit komponentenweisen Operationen) und $\mathbb{Z}_{mn}$ isomorph sind.
Wir beginnen mit einem Beispiel, nämlich $m = 3, n = 8$, also $mn = 24$. Die folgende Tabelle gibt die Reste der Zahlen $x \in \{0, 1, ..., 23\}$ $modulo$ 3 und $modulo$ 8 an. Die Restepaare wiederholen sich zyklisch für andere $x \in \mathbb{Z}$.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
x mod 3	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
x mod 8	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7

Wenn wir die Einträge in Zeilen 2 und 3 als 24 Paare in $\mathbb{Z}_3 \times \mathbb{Z}_8$ ansehen, erkennen wir, dass sie alle verschieden sind, also auch alle Möglichkeiten in $\{0, 1, 2\} \times \{0, 1, ..., 7\}$ abdecken. D.h.: Die Abbildung $x \to (x \ mod\ 3, x \ mod\ 8)$ ist eine Bijektion zwischen $\mathbb{Z}_{24}$ und $\mathbb{Z}_3 \times \mathbb{Z}_8$. Zudem spiegeln sich arithmetische Operationen auf den Elementen von $\mathbb{Z}_{24}$ in den Resten modulo 3 und 8 wider. Beispielsweise liefert die Addition von $(2, 7)$ und $(2, 1)$ das Resultat $(1, 0)$, das der Addition von 23 und 17 mit dem Resultat 40 $mod\ 24 = 16$ entspricht. Genauso ist $(2^5 \ mod\ 3, 3^5 \ mod\ 8) = (2, 3)$, was der Beobachtung $11^5 \ mod\ 24 = 11$ entspricht. Der Chinesische Restsatz sagt im wesentlichen, dass eine solche strukturelle Entsprechung zwischen den Resten modulo mn und Paaren von Resten modulo m bzw. n immer gilt, wenn m und n teilerfremd sind.
Fakt 4.21 Chinesischer Restsatz (*): m und n seien teilerfremd. Dann ist die Abbildung $\Phi : \mathbb{Z}_{mn} \ni x \to (x \ mod\ m, x \ mod\ n) \in \mathbb{Z}_m \times \mathbb{Z}_n$ bijektiv. Weiterhin: Wenn $\Phi(x) = (x_1, x_2)$ und $\Phi(y) = (y_1, y_2)$, dann gilt:

1. $\Phi(x +_{mn} y) = (x_1 +_m y_1, x_2 +_n y_2)$
2. $\Phi(x *_{mn} y) = (x_1 *_m y_1, x_2 *_n y_2)$
3. $\Phi(1) = (1, 1)$

(Dabei bezeichnen $+_j$ und $*_j$ die Addition und die Multiplikation modulo j.)
Für mathematisch-strukturell orientierte Leser/innen: Die Gleichungen 1. bis 3. kann man etwas abstrakter auch so fassen, dass die Abbildung Φ ein Ring-mit-1-Isomorphismus zwischen $\mathbb{Z}_{mn}$ und $\mathbb{Z}_m \times \mathbb{Z}_n$ ist.
Man kann sich noch fragen, wie man nötigenfalls zu gegebenen Zahlen $s \in \mathbb{Z}_m$ und $t \in \mathbb{Z}_n$ die Zahl $x \in \mathbb{Z}_{mn}$ berechnen kann, die $\Phi(x) = (s, t)$ erfullt. Dazu betrachtet man zunächst den Fall $s = 1$ und $t = 0$. Weil m und n teilerfremd sind, kann man mit dem erweiterten Euklidischen Algorithmus ein $u \in \mathbb{Z}_m$ mit $un \ mod\ m = 1$ finden. Wir setzen $y = un \in \mathbb{Z}_{mn}$. Dann gilt $y \ mod\ m = 1$ und $y \ mod\ n = 0$. Analog findet man ein $z \in \mathbb{Z}_{mn}$ mit $z \ mod\ m = 0$ und $z \ mod\ n = 1$. Nun setzen wir $x := (sy + tz) \ mod\ mn \in \mathbb{Z}_{mn}$. Wir haben, modulo m gerechnet: $x \equiv sy + tz \equiv s * 1 + t * 0 \equiv s \ (mod\ m)$. Analog ergibt sich $x \equiv sy + tz \equiv s * 0 + t * 1 \equiv t (mod\ n)$, wie gewünscht. Der Berechnungsaufwand für das Finden von x ist $O((log\ m)(log\ n))$ Ziffernoperationen, das geht also sehr schnell.
Beispiel: $m = 5, n = 8, s = 3, t = 7$. Wir finden $u = 2$ mit $u * 8 \ mod\ 5 = 1$ und $y = 2 * 8 = 16$ sowie $v = 5$ mit $v * 5 \ mod\ 8 = 1$ und $z = 5 * 5 = 25$. Nun setzen wir $x = (3 * 16 + 7 * 25) \ mod\ 40 = (48 + 175) \ mod\ 40 = (8 + 15) \ mod\ 40 = 23$.
Und tatsächlich: $23 \ mod\ 5 = 3$ und $23 \ mod\ 8 = 7$.
Wir wollen noch untersuchen, wie sich Zahlen, die zu m und n teilerfremd sind, in der Sichtweise des Chinesischen Restsatzes verhalten.
Proposition 4.22 (*): Wenn man die Abbildung Φ aus dem Chinesischen Restsatz auf $\mathbb{Z}_{mn}^*$ einschränkt, ergibt sich eine Bijektion zwischen $\mathbb{Z}_{mn}^*$ und $\mathbb{Z}_m^* \times \mathbb{Z}_n^*$.
Bemerkung: Der Chinesische Restsatz und die nachfolgenden Bemerkungen und Behauptungen lassen sich leicht auf $r > 2$ paarweise teilerfremde Faktoren $n_1, ..., n_r$ verallgemeinern. Die Aussagen lassen sich durch vollständige Induktion über r beweisen. Mit Prop. 4.22 können wir eine übersichtliche Formel für die Kardinalitäten der Mengen $\mathbb{Z}_m^*, m \geq 2$, entwickeln.
Definition 4.23 (Eulersche φ-Funktion): für $m \geq 2$ sei $\varphi(m) := |\mathbb{Z}_m^*| = |\{x | 0 < x < m, ggT(x,m) = 1\}|$.
Einige Beispielwerte, die man durch Aufzählen findet, sind in folgender Tabelle angegeben.
[Tabelle 2: Eulersche φ-Funktion für kleine m]

m	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\varphi(m)$	1	2	2	4	2	6	4	6	4	10	4	12	6	8

Folgendes ist eine unmittelbare Konsequenz aus Proposition 4.22:
Lemma 4.24: Für teilerfremde Zahlen n und m gilt $\varphi(mn) = \varphi(m) * \varphi(n)$.
(Man teste $20 = 4 * 5$ und $12 = 3 * 4$.) Wir können den kleinen Satz von Fermat (Fakt 4.20) auf den Fall beliebiger m verallgemeinern. Auch der Beweis ist sehr ähnlich.
Fakt 4.25 (Satz von Euler)(*): für $m \geq 2$ und x mit $ggT(m,x) = 1$ gilt: $x^{\varphi(m)} \ mod\ m = 1$.
Wenn m viele verschiedene kleine Primfaktoren hat, kann $\varphi(m)$ auch deutlich kleiner sein als m, z.B.
$\varphi(210) = \varphi(2 * 3 * 5 * 7) = 1 * 2 * 4 * 6 = 48$. Es gilt aber: $\varphi(m) = m - 1$, wenn m eine Primzahl ist, und $\varphi(m) \geq \pi(m) = |\{p \leq m | \text{p ist Primzahl}\}|$, wenn m zusammengesetzt ist.

Primzahlen

Jede positive ganze Zahl x ist durch 1 und durch x teilbar.
Definition 4.26:

1. Eine Zahl $p \geq 1$ heißt Primzahl, wenn p genau zwei positive Teiler hat. Diese Teiler sind dann 1 und p. (Die Zahl 1 hat nur einen positiven Teiler, nämlich 1. Also ist 1 keine Primzahl.)

2. Eine Zahl $x \geq 1$ heißt zusammengesetzt, wenn sie einen Teiler y mit $1 < y < x$ besitzt. (Die Zahl 1 besitzt keinen Teiler y mit $1 < y < 1$. Also ist 1 nicht zusammengesetzt.)

Bemerkung: Ein Blick auf die Teilbarkeitsbeziehung zeigt, welche besondere Rolle $1, -1, 0$ und die Primzahlen spielen. Die Zahlen 1 und -1 bilden das Minimum in der Teilbarkeitsrelation, die Primzahlen sitzen unmittelbar darüber, und die zusammengesetzten Elemente liegen strikt über den Primzahlen. Die 0, als Maximum in der Ordnung, sitzt strikt über allen zusammengesetzten Zahlen.
Fakt 4.27: Wenn p eine Primzahl ist und $p|xy$ gilt, dann gilt $p|x$ oder $p|y$.
Beweis: Wenn $p|x$, sind wir fertig. Also können wir $p \not| x$ annehmen. Das heißt, dass $ggT(p, x) = 1$ ist. Nach dem Lemma von Bezout können wir $1 = sp + tx$ schreiben, für ganze Zahlen s, t. Daraus folgt: $y = spy + txy$. Nun ist xy durch p teilbar, also auch $y = spy + txy$.
Satz 4.28 (Fundamentalsatz der Arithmetik) (*): Jede ganze Zahl $x \geq 1$ kann als Produkt von Primzahlen geschrieben werden. Die Faktoren sind eindeutig bestimmt (bis auf die Reihenfolge).
Mit Hilfe von Lemma 4.24 können wir nun eine Formel für $\varphi(m) = |\mathbb{Z}_m^*|$ angeben, die auf der Primzahlzerlegung beruht.
Lemma 4.29: für $m \geq 2$ gilt $\varphi(m) = m * \sum_{p\ prim, p|m}(1 - \frac{1}{p})$
Beweis: Wenn m eine Primzahlpotenz p^t ist, dann besteht $\mathbb{Z}_m^*$ aus den Zahlen in $\mathbb{Z}_m = \{0, 1, ..., p^t - 1\}$, die nicht durch p teilbar sind. Da es in $\mathbb{Z}_m$ insgesamt p^t Zahlen gibt und p^{t-1} Vielfache von p, gilt $\varphi(m) = p^t - p^{t-1} = m - m/p = m(1 - 1/p)$. Nun nehmen wir an, dass $m = p_1^{t_1}...p_s^{t_s}$ gilt, für verschiedene Primzahlen $p_1, ..., p_s$ und $t_1, ..., t_s \geq 1$. Die Faktoren $p_1^{t_1}, ..., p_s^{t_s}$ sind teilerfremd. denn wenn etwa $p_1^{t_1}$ und $p_2^{t_2}...p_s^{t_s}$ einen gemeinsamen Teiler > 1 hätten, dann wäre p_1 Teiler von $p_2^{t_2}...p_s^{t_s}$, was sofort einen Widerspruch zur Eindeutigkeit der Primfaktorzerlegung ergibt. Mit Lemma 4.24, $(s-1)$-mal angewendet, erhalten wir
$\varphi(m) = \prod_{i=1}^s \varphi(p_i^{t_i}) = \prod_{i=1}^s (p_i^{t_i}(1 - 1/p_i)) = m * \prod_{i=1}^s (1 - \frac{1}{p_i})$. Mit dieser Formel lassen sich die Werte in Tabelle 2 schnell verifizieren. (Beispiel: $\varphi(12) = 12(1 - 1/2)(1 - 1/3) = 12 * (1/2) * (2/3) = 4$) Man beachte als Spezialfall: Wenn $m = pq$ für verschiedene Primzahlen p und q, dann ist $\varphi(m) = pq(1 - 1/p)(1 - 1/q) = (p - 1)(q - 1)$. (Beispiel: $\varphi(15) = 2 * 4 = 8$)
Bemerkung: Die einfache Formel in Lemma 4.29 könnte zu dem Schluss verleiten, dass sich $\varphi(m)$ zu gegebenem m immer leicht berechnen lässt. Aber Achtung: Man muss dazu die Menge der Primfaktoren von m kennen. Dies läuft darauf hinaus, dass Faktorisierungsproblem für m zu lösen, also einen beliebigen Primfaktor für m zu finden, und hierfür kennt man keine effizienten Algorithmen. Tatsächlich ist auch kein effizienter Algorithmus bekannt, der es erlaubt, $\varphi(m)$ aus m zu berechnen.
Fakt 4.30: Jede zusammengesetzte Zahl x besitzt einen Primfaktor p mit $p \leq \sqrt{x}$.
Beweis: Man schreibt $x = yz$ für Zahlen y und z, die weder 1 noch x sind. Es ist nicht möglich, dass beide größer als $\sqrt{x}$ sind. Der kleinere Faktor enthält also einen Primfaktor, der nicht größer als $\sqrt{x}$ ist.
Bemerkung: Wir betrachten das resultierende naive Faktorisierungsverfahren: Teste die Zahlen in $\{2, ..., \lfloor\sqrt{x}\rfloor\}$ nacheinander darauf, ob sie x teilen; wenn ein Faktor p gefunden wurde, wende dasselbe Verfahren auf $x' = x/p$ an. Dieses Verfahren hat im schlechtesten Fall Rechenzeit mindestens $\Theta(\sqrt{x}) = \Theta(2^{(log\ x)/2})$, also exponentiell in der Bitlänge von x. Wie wir später genauer diskutieren werden, sind für das Auffinden der Primzahlzerlegung einer gegebenen Zahl x überhaupt keine effizienten Algorithmen bekannt (also Algorithmen mit Laufzeiten $O((log\ x)^c)$ für konstantes c). Aber es gibt effiziente Algorithmen, mit denen man feststellen kann, ob eine Zahl x eine Primzahl ist oder nicht. Dieser Unterschied in der Schwierigkeit des Faktorisierungsproblems und des Primzahlproblems liegt einer ganzen Reihe von kryptographischen Verfahren zugrunde.
Satz 4.31 (Euklid): Es gibt unendlich viele Primzahlen.
Beweis: Wenn $\{p_1, ..., p_k\}$, für $k \geq 1$, eine endliche Menge von (verschiedenen) Primzahlen ist, betrachten wir die Zahl $x = 1 + p_1...p_k$. Die Zahl x kann durch keine der Zahlen $p_1, ..., p_k$ teilbar sein, sonst wäre 1 durch diese Primzahl teilbar, was nicht möglich ist. Also sind alle Primfaktoren in der Primzahlzerlegung von x von $p_1, ..., p_k$ verschieden, es muss also außer $p_1, ..., p_k$ noch weitere Primzahlen geben.
Über die Verteilung der Primzahlen (ihre ,,Dichte") in N gibt der berühmte Primzahlsatz Auskunft. Mit $\pi(x)$ bezeichnen wir die Anzahl der Primzahlen, die nicht größer als x sind.
Satz 4.32 Primzahlsatz: $lim_{x\to\infty} \frac{\pi(x)}{x \backslash ln\ x} = 1$
Das heißt, dass für große x in $(x, 2x]$ etwa $\frac{2x}{ln(2x)} - \frac{x}{ln\ x} \approx \frac{x}{ln\ x}$ Primzahlen zu erwarten sind. Die n-Bit-Zahlen bilden das Intervall $[2^{n-1}, 2n)$. Der Anteil der Primzahlen in diesem Intervall ist näherungsweise $\frac{2^{n-1}\ ln(2^{n-1})}{2^{n-1}} \approx \frac{1}{(ln\ 2)(n-1)} \approx 1,44/n$. Für $n \approx 2000$ ist der relative Anteil von Primzahlen im interessanten Zahlenbereich also $\approx 1,44/2000 \approx 1/1400$. Er sinkt umgekehrt proportional zur Ziffernzahl. Eine schärfere Form des Primzahlsatzes ist folgende Aussage (wobei der Beweis Nichtspezialisten nicht zugänglich ist):
$\frac{x}{ln\ x}(1 + \frac{1}{ln\ x}) \leq \pi(x) \leq \frac{x}{ln\ x}(1 + \frac{1,2762}{ln\ x})$.
Für $x \geq 500000$ folgt daraus: $\frac{x}{ln\ x} < \pi(x) < 1, fracxln\ x$. Daraus folgt, dass für $n \geq 20$ der Anteil der Primzahlen unter den n-Bit-Zahlen folgende Ungleichung erfüllt:
$\frac{|\{p \in [2^{n-1}, 2^n) | \text{p is prime}\}|}{2^{n-1}} = \frac{\pi(2^n) - \pi(2^{n-1})}{2^{n-1}} \geq$
$\frac{2^n \backslash (n\ ln\ 2) - 1, 1 * 2^{n-1} \backslash ((n-1) ln\ 2)}{2^{n-1}} \geq \frac{2}{n\ ln\ 2} - \frac{1,1}{n\ ln\ 2} * \frac{n}{n-1} \geq \frac{6}{5n}$.
Mit Hilfe eines Computeralgebraprogramms findet man heraus, dass die Ungleichung $|\{p \in [2^{n-1}, 2^n) | \text{ p is prime}\}|/2^{n-1} \geq 6/(5n)$ auch für $9 \leq n \leq 20$ gilt. (Für $n = 8$ ist sie falsch.) Man kann sich also merken: für $n \geq 9$ ist der Anteil der Primzahlen an den n-Bit-Zahlen mindestens $\frac{6}{5n}$.

— Ziffernzahl n —	Dusart-Schranke $(\pi(2^n) - \pi(2^{n-1}))\backslash 2^{n-1}$ —	für numerische untere Schranke
— 256 —	$\frac{6}{5*256}$ —	$\geq \frac{1}{214}$
— 512 —	$\frac{6}{5*512}$ —	$\geq \frac{1}{427}$
— 1024 —	$\frac{6}{5*1024}$ —	$\geq \frac{1}{854}$
— 2048 —	$\frac{6}{5*2048}$ —	$\geq \frac{1}{1707}$

Eine leichter zu beweisende Aussage der Art
$\pi(2m) - \pi(m) = O(m/log\ m)$ ist die folgende:
Satz 4.33 Ungleichung von Finsler: Für jede ganze Zahl $m \geq 2$ liegen im Intervall $(m, 2m]$ mindestens $m/(3\ ln(2m))$ Primzahlen:
$\pi(2m) - \pi(m) \geq \frac{m}{3\ ln(2m)}$.
Ein vollständiger, vergleichsweise einfacher Beweis für Satz 4.33 findet sich zum Beispiel in dem Lehrbuch ,,Elemente der Diskreten Mathematik: Zahlen und Zählen, Graphen und Verbände" von Diekert, Kufleitner, Rosenberger (De Gruyter 2013).

Der Primzahltest von Miller und Rabin

In diesem Abschnitt lernen wir einen randomisierten Algorithmus kennen, der es erlaubt, zu einer gegebenen Zahl N zu entscheiden, ob N eine Primzahl ist oder nicht.
Ein idealer Primzahltest sieht so aus:

- Eingabe: Eine natürliche Zahl $N \geq 3$.
- Ausgabe: 0, falls N eine Primzahl ist; 1, falls N zusammengesetzt ist.

Wozu braucht man Primzahltests? Zunächst ist die Frage ,,Ist N eine Primzahl?" eine grundlegende mathematisch interessante Fragestellung. Spätestens mit dem Siegeszug des RSA-Kryptosystems hat sich die Situation jedoch dahin entwickelt, dass man Algorithmen benötigt, die immer wieder neue vielziffrige Primzahlen (etwa mit 1000 oder 1500 Bits bzw. 301 oder 452 Dezimalziffern) bereitstellen können. Den Kern dieser Primzahlerzeugungs-Verfahren bildet ein Verfahren, das eine gegebene Zahl N darauf testet, ob sie prim ist. Ein naiver Primzahltest (,,versuchsweise Division"), der dem brute-force-Paradigma folgt, findet durch direkte Division der Zahl N durch $2, 3, 4, ..., \lfloor\sqrt{N}\rfloor$ heraus, ob N einen nichttrivialen Teiler hat. Man kann dieses Verfahren durch einige Tricks beschleunigen, aber die Rechenzeit wächst dennoch mit $O(\sqrt{N})$. Dies macht es für Zahlen mit mehr als 40 Dezimalstellen praktisch undurchführbar, von Zahlen mit mehr als 100 Dezimalstellen ganz zu schweigen. (Achtung: Damit wird nichts über den Zeitaufwand bei anderen Faktorisierungsalgorithmen gesagt. Es gibt andere, sehr fortgeschrittene Faktorisierungsalgorithmen, die bei entsprechendem Zeitaufwand und mit sehr leistungsstarken Rechnern auch noch mit 200-stelligen Zahlen zurechtkommen. Für Information zu früheren und aktuelleren Faktorisierungserfolgen siehe z.B. Wikipedia RSA Numbers.
In diesem Abschnitt beschreiben wir den randomisierten Primzahltest von Miller-Rabin. Dabei handelt es sich um einen ,,Monte-Carlo-Algorithmus mit einseitigem Fehler". Das heißt: Auf Eingaben N, die Primzahlen sind, wird immer 0 ausgegeben; auf Eingaben N, die zusammengesetzt sind, gibt es eine gewisse (von N abhängige) Wahrscheinlichkeit, dass die Ausgabe 0, also falsch ist. Für kein zusammengesetztes N ist diese Wahrscheinlichkeit größer als die ,,Fehlerschranke" $\frac{1}{4}$. Wir beweisen nur die Fehlerschranke $\frac{1}{2}$. Im Beweis benutzen wir einfache zahlentheoretische Überlegungen. Eine herausragende Eigenschaft des Miller-Rabin-Tests ist seine Effizienz. Wir werden sehen, dass selbst bei Verwendung der Schulmethoden für Multiplikation und Division die Anzahl der Ziffernoperationen des Primzahltests nur $O((log\ N)^3)$ ist.
Bemerkung: Der Miller-Rabin-Algorithmus stammt aus dem Jahr 1977; er folgte einem kurz vorher vorgestellten anderen randomisierten Primzahltest (Solovay-Strassen-Test). Für diesen und andere randomisierte Primzahltests (z.B. der ,,Strong Lucas Probable Prime Test" oder der ,,Quadratic Frobenius Test" von Grantham) sei auf die Literatur verwiesen. Im Jahr 2002 stellten Agarwal, Kayal und Saxena einen deterministischen Primzahltest mit polynomieller Rechenzeit vor. (Die Rechenzeit ist z.B. durch $O((log\ N)^{7,5})$ beschränkt.) Dieser Algorithmus stellte insofern einen gewaltigen Durchbruch dar, als er ein Jahrhunderte altes offenes Problem löste, nämlich die Frage nach einem effizienten deterministischen Verfahren für das Entscheidungsproblem ,,ist N Primzahl oder zusammengesetzt"? Andererseits ist seine Laufzeit im Vergleich etwa zu dem hier diskutierten randomisierten Verfahren so hoch, dass nach wie vor die randomisierten Algorithmen benutzt werden, um für kryptographische Anwendungen Primzahlen zu erzeugen. Da gerade Zahlen leicht zu erkennen sind, beschränken wir im Folgenden unsere Überlegungen auf ungerade Zahlen $N \geq 3$.

Der Fermat-Test

Wir erinnern uns an Fakt 4.20, den Kleinen Satz von Fermat: Wenn p eine Primzahl ist und $1 \leq a < p$, dann gilt $a^{p-1} mod\ p = 1$. Wir können diese Aussage dazu benutzen, um ,,Belege" oder ,,Zertifikate" oder ,,Zeugen" dafür anzugeben, dass eine Zahl N zusammengesetzt ist: Wenn wir eine Zahl a mit $1 \leq a < N$ finden, für die $a^{N-1} mod\ N \neq 1$ gilt, dann ist N definitiv keine Primzahl.
Beispiel: Mit $N = 15$ und $a = 2$ rechnen wir:
$2^{14} \equiv (2^4)^3 * 2^2 \equiv 16^3 * 4 \equiv 1 * 4 \equiv 4 (mod\ 15)$. Also ist
$2^{14} mod\ 15 = 4 \neq 1$, also ist 15 definitiv keine Primzahl. (Man beachte, dass wir keinen Faktor angeben müssen, um zu diesem Schluss zu kommen.)
Definition 4.34: Sei $N \geq 3$ ungerade und zusammengesetzt. Eine Zahl $a \in \{1, ..., N-1\}$ heißt F-Zeuge für N, wenn $a^{N-1} mod\ N \neq 1$ gilt. Eine Zahl $a \in \{1, ..., N-1\}$ heißt F-Lügner für N, wenn $a^{N-1} mod\ N = 1$ gilt. Die Menge der F-Lügner nennen wir L_N^F.
Wir bemerken, dass ein F-Zeuge belegt, dass es Faktoren $k, l > 1$ mit $N = k * l$ gibt, dass aber ein F-Zeuge nicht auf solche Faktoren hinweist oder sie beinhaltet. Das Finden von Faktoren wird von Primzahltests auch nicht verlangt und normalerweise auch nicht geleistet. Man sieht sofort, dass 1 und $N - 1$ immer F-Lügner sind: Es gilt $1^{N-1} mod\ N = 1$ und $(N-1)^{N-1} \equiv (-1)^{N-1} = 1 (mod\ N)$, weil $N - 1$ gerade ist. Für jede zusammengesetzte Zahl N gibt es mindestens einen F-Zeugen. Nach Fakt 4.19 gilt $\{1, ..., N-1\} - \mathbb{Z}_N^* \neq \varnothing$, wenn N zusammengesetzt ist.
Lemma 4.35: Wenn N zusammengesetzt ist, ist jedes $a \in \{1, ..., N-1\} - \mathbb{Z}_N^*$ ein F-Zeuge.
Beweis: Sei $d = ggT(a, N) > 1$. Dann ist auch a^{N-1} durch d teilbar, also auch $a^{N-1}\ mod\ N = a^{N-1} - \lfloor a^{N-1} \backslash N \rfloor * N$. Daher ist $a^{N-1} mod\ N \neq 1$.
Beispiel: Für $N = 15$ und $a = 6$ gilt
$6^{14} \equiv 36^7 \equiv 6^7 \equiv 36^3 * 6 \equiv 6^4 \equiv 6^2 \equiv 6\ (mod\ 15)$. Der Rest 6 ist durch $ggT(6, 15) = 3$ teilbar.
Leider ist für manche zusammengesetzten Zahlen N die Menge $\{1, ..., N-1\} - \mathbb{Z}_N^*$ äußerst dünn. Wenn zum Beispiel $N = pq$ für zwei Primzahlen p und q ist, dann gilt $ggT(a, N) > 1$ genau dann wenn p

oder q ein Teiler von a ist. Es gibt genau $p+q-2$ solche Zahlen a in $\{1,...,N-1\}$, was gegenüber N sehr klein ist, wenn p und q annähernd gleich groß sind. Um eine gute Chance zu haben, F-Zeugen zu finden, sollte es also mehr als nur die in $\{1,...,N-1\}-\mathbb{Z}_N^*$ geben.
Beispiel: $N = 91 = 7*13$. Es gibt 18 Vielfache von 7 und 13 (für größere p und q wird der Anteil dieser offensichtlichen F-Zeugen noch kleiner sein), und daneben weitere 36 F-Zeugen und 36 F-Lügner in $\{1,2,...,90\}$. In diesem Beispiel gibt es um einiges mehr F-Zeugen als F-Lügner. Wenn dies für alle zusammengesetzten Zahlen N der Fall wäre, wäre es eine elegante randomisierte Strategie, einfach zufällig nach F-Zeugen zu suchen. Dies führt zu unserem ersten Versuch für einen randomisierten Primzahltest.
Tabelle 3: F-Zeugen und F-Lügner für $N = 91 = 7*13$. Es gibt 36 F-Lügner und 36 F-Zeugen in $\mathbb{Z}_{91}^*$. Wir wissen nach Lemma 4.35, dass alle 18 Vielfachen von 7 und 13 F-Zeugen sind.
— F-Zeugen in $\{1,...,90\}-\mathbb{Z}_{91}^*$: — 7, 14, 21, 28, 35, 42, 49, 56, 63, 70, 77, 84, 13, 26, 39, 52, 65, 78
— F-Lügner: — 1, 3, 4, 9, 10, 12, 16, 17, 22, 23, 25, 27, 29, 30, 36, 38, 40, 43, 48, 51, 53, 55, 61, 62, 64, 66, 68, 69, 74, 75, 79, 81, 82, 87, 88, 90
— F-Zeugen in $\mathbb{Z}_{91}^*$: — 2, 5, 6, 8, 11, 15, 18, 19, 20, 24, 31, 32, 33, 34, 37, 41, 44, 45, 46, 47, 50, 54, 57, 58, 59, 60, 67, 71, 72, 73, 76, 80, 83, 85, 86, 89
Algorithmus 4.4: Fermat-Test

- Eingabe: Ungerade Zahl $N \geq 3$
- Methode:
  1. Wähle a zufällig aus $\{1,...,N-1\}$
  2. if $a^{N-1}\ mod\ N \neq 1$ then return 1 else return 0

Die Laufzeitanalyse liegt auf der Hand: Der teuerste Teil ist die Berechnung der Potenz $a^{N-1}\ mod\ N$ durch schnelle Exponentiation, die nach den Ergebnissen von Lemma 4.14 $O(log\ N)$ arithmetische Operationen und $O((log\ N)^3)$ Ziffernoperationen benötigt. Weiter ist es klar, dass der Algorithmus einen F-Zeugen gefunden hat, wenn er ,,1" ausgibt, dass in diesem Fall also N zusammengesetzt sein muss. Umgekehrt ausgedrückt: Wenn N eine Primzahl ist, gibt der Fermat-Test garantiert ,,0" aus. Für $N = 91$ wird das falsche Ergebnis 0 ausgegeben, wenn als a einer der 36 F-Lügner gewählt wird. Die Wahrscheinlichkeit hierfür ist $\frac{36}{90} = \frac{2}{5} = 0,4$.
Für viele zusammengesetzte Zahlen N gibt es reichlich F-Zeugen, so dass der Fermat-Test für diese N mit konstanter Wahrscheinlichkeit das korrekte Ergebnis liefert. Wir analysieren das Verhalten des Fermat-Tests für solche ,,gutmütigen" Eingabezahlen N (für die $N = 91$ ein typisches Beispiel ist).
Satz 4.36: Sei $N \geq 9$ eine ungerade zusammengesetzte Zahl. Wenn es mindestens einen F-Zeugen $b \in \mathbb{Z}_N^*$ gibt, dann liefert der Fermat-Test auf Eingabe N mit Wahrscheinlichkeit größer als $\frac{1}{2}$ die korrekte Antwort ,,1".
Beweis: Sei $b \in \mathbb{Z}_N^*$ ein F-Zeuge. Betrachte die Funktiong $b : L_N^F \to \mathbb{Z}_N^*$, die den F-Lügner a auf $g_b(a) = ba\ mod\ N$ abbildet. Wie im Beweis von Fakt 4.20 sieht man, dass g_b injektiv ist. Weiter ist $g_b(a)$ für jedes $a \in L_N^F$ ein F-Zeuge: $(ba\ mod\ N)^{N-1}\ mod\ N =$
$(b^{N-1}\ mod\ N)(a^{N-1}\ mod\ N) = b^{N-1}\ mod\ N \neq 1$. Wir können also jedem F-Lügner a einen eigenen F-Zeugen $g_b(a)$ in $\mathbb{Z}_N^*$ zuordnen. Daraus folgt, dass es in $\mathbb{Z}_N^*$ mindestens so viele F-Zeugen wie F-Lügner gibt. Mit Lemma 4.35 ergibt sich, dass $\{1,...,N-1\}$ mehr F-Zeugen als F-Lügner enthält. Daher ist die Wahrscheinlichkeit, dass die im Fermat-Test zufällig gewählte Zahl a ein F-Lügner ist, kleiner als $\frac{1}{2}$. Eine Fehlerwahrscheinlichkeit in der Nähe von $\frac{1}{2}$ ist natürlich viel zu groß. Wir verringern die Fehlerschranke durch wiederholte Ausführung des Fermat-Tests.
Algorithmus 4.5: Iterierter Fermat-Test

- Eingabe: Ungerade Zahl $N \geq 3$, eine Zahl $l \geq 1$
- Methode:
  1. repeat l times
     - $a \leftarrow$ ein zufälliges Element von $\{1,...,N-1\}$
     - if $a^{N-1}\ mod\ N \neq 1$ then return 1
  2. return 0

Wenn die Ausgabe 1 ist, hat der Algorithmus einen F-Zeugen für N gefunden, also ist N zusammengesetzt. D.h.: Wenn N eine Primzahl ist, ist die Ausgabe 0. Andererseits: Wenn N zusammengesetzt ist, und es mindestens einen F-Zeugen $b \in \mathbb{Z}_N^*$ gibt, dann ist nach Satz 4.36 die Wahrscheinlichkeit für die falsche Ausgabe ,,0" höchstens $(\frac{1}{2})^l = 2^{-l}$. Indem wir l genügend groß wählen, können wir die Fehlerwahrscheinlichkeit so klein wie gewünscht einstellen.
Wenn es darum geht, aus einem genügend großen Bereich zufällig gewählte Zahlen darauf zu testen, ob es sich um eine Primzahl handelt, dann ist der Fermat-Test (in Kombination mit dem Testen auf kleine Teiler, etwa alle Primzahlen unter 1000) allem Anschein nach eine sehr effiziente und zuverlässige Methode. Dies wird durch empirische Resultate nahegelegt. Wenn man allerdingsüber die Herkunft der zu testenden Zahl N keine Information hat und eventuell damit rechnen muss, dass jemand (ein ,,Gegenspieler") absichtlich eine besonders schwierige Eingabe vorlegt, dann stößt der Fermat-Test an eine Grenze. Es gibt nämlich ,,widerspenstige" zusammengesetzte Zahlen, denen man mit diesem Test nicht beikommen kann, weil alle Elemente von $\mathbb{Z}_N^*$ F-Lügner sind. Mit diesen befasst sich der folgende Abschnitt.

Carmichael-Zahlen

Definition 4.37: Eine ungerade zusammengesetzte Zahl N heißt eine Carmichael-Zahl, wenn für alle $a \in \mathbb{Z}_N^*$ die Gleichung $a^{N-1}\ mod\ N = 1$ gilt.
Die kleinste Carmichael-Zahl ist $561 = 3*11*17$. Weitere kleine Carmichael-Zahlen sind $1105 = 5*13*17$ und $1729 = 7*13*19$. Erst im Jahr 1994 wurde bewiesen, dass es unendlich viele Carmichael-Zahlen gibt, genauer: Wenn x genügend groß ist, dann gibt es in $\{N \in \mathbb{N} | N \leq x\}$ mehr als $x^{2/7}$ Carmichael-Zahlen. Die aktuell beste bekannte untere Schranke ist $x^{1/3}$. Von Erdös (1956) stammt die obere Schranke $x * exp(\frac{-c\ ln\ x\ ln\ ln\ ln\ x}{ln\ ln\ x})$, für eine Konstante $c > 0$, die zeigt, dass Carmichael-Zahlen viel seltener als Primzahlen sind.
Wenn wir dem Fermat-Test eine Carmichael-Zahl N als Eingabe geben, ist die Wahrscheinlichkeit für die falsche Antwort 0 nach Lemma 4.29 genau $\frac{\varphi(N)}{N-1} > \frac{\varphi(N)}{N} = \prod_{p\ prim, p\ teilt\ N}(1-\frac{1}{p}) > 1 - \sum_{p\ prim, p\ teilt\ N}\frac{1}{p}$.
Diese Wahrscheinlichkeit liegt nahe an 1, wenn N nur wenige und relativ große Primfaktoren hat. An solchen Carmichael-Zahlen besteht etwa im Bereich der Zahlen im Bereich $[10^{17}, 10^{18}]$ kein Mangel, wie ein Blick in entsprechende Tabellen zeigt. Zum Beispiel ist
$N = 925619721362375041 = 425681*1277041*1702721$ eine 18-ziffrige Carmichael-Zahl mit $\varphi(N)/N > 0.999996$. Der Wiederholungstrick zur Wahrscheinlichkeitsverbesserung hilft hier leider auch nicht, denn wenn etwa p_0 der kleinste Primfaktor von N ist, und N nur 3 oder 4 Faktoren hat, dann sind $\Omega(p_0)$ Wiederholungen nötig, um die Fehlerwahrscheinlichkeit auf $\frac{1}{2}$ zu drücken. Sobald p_0 mehr als 30 Dezimalstellen hat, ist dies undurchführbar.
Für einen zuverlässigen, effizienten Primzahltest, der für alle Eingabezahlen funktioniert, müssen wir über den Fermat-Test hinausgehen. Interessanterweise ist dies praktisch ohne Effizienzverlust möglich. Für spätere Benutzung stellen wir noch eine Hilfsaussage über Carmichael-Zahlen bereit.
Lemma 4.38: Wenn N eine Carmichael-Zahl ist, dann ist N keine Primzahlpotenz.
Beweis: Wir beweisen die Kontraposition: Wenn $N = p^l$ für eine ungerade Primzahl p und einen Exponenten $l \geq 2$ ist, dann ist N keine Carmichael-Zahl. Dazu genügt es, eine Zahl $a \in \mathbb{Z}_N^*$ anzugeben, so dass $a^{N-1}\ mod\ N \neq 1$ ist. Wir definieren: $a := p^{l-1}+1$. (Wenn z.B. $p = 7$ und $l = 3$ ist, ist $N = 343$ und $a = 49+1 = 50$.) Man sieht sofort, dass $a < p^l = N$ ist, und dass a nicht von p geteilt wird, also a und N teilerfremd sind; also ist $a \in \mathbb{Z}_N^*$. Nun rechnen wir modulo N, mit der binomischen Formel: $a^{N-1} \equiv (p^{l-1}+1)^{N-1} \equiv$
$\sum_{0 \leq j \leq N-1}\binom{N-1}{j}(p^{l-1})^j \equiv 1 + (p^l-1)*p^{l-1}(mod\ N)$. (4.3)
(Die letzte Äquivalenz ergibt sich daraus, dass für $j \geq 2$ gilt, dass $(l-1)j \geq l$ ist, also der Faktor $(p^{l-1})j = p^{(l-1)j}$ durch $N = p^l$ teilbar ist, also modulo N wegfällt.) Nun ist p^{l-1} nicht durch p teilbar, also ist $(p^l-1)*p^{l-1}$ nicht durch $N = p^l$ teilbar. Damit folgt aus (4.3), dass $a^{N-1} \not\equiv 1(mod\ N)$ ist, also $a^{N-1}\ mod\ N \neq 1$. Folgerung: Jede Carmichael-Zahl N lässt sich als $N = N_1 * N_2$ schreiben, wo N_1 und N_2 teilerfremde ungerade Zahlen ≥ 3 sind. (Eine etwas genauere Untersuchung, die wir hier aber nicht benötigen, ergibt, dass die Primfaktoren einer Carmichael-ZahlN alle verschieden sein müssen, und dass N mindestens drei Primfaktoren haben muss. Auch aus dieser Tatsache kann man entnehmen, dass Carmichael-Zahlen eher selten sind.)

Nichttriviale Quadratwurzeln der 1

Beispiel: Betrachte $N = 7*13$. Es gilt $1^2 = 1$ und $90^2 \equiv (-1)^2(mod\ 91)$. Aber es gilt auch $27^2 = 81*9 \equiv (-10)*9 = -90 \equiv 1(mod\ 91)$. Daraus folgt auch $642 = (91-27)^2 \equiv (-27)^2 = 27^2 \equiv 1(mod\ 91)$. Wir nennen eine Zahl $b \in \{2,...,N-2\}$ mit $b^2\ mod\ N = 1$ eine nicht triviale Quadratwurzel der 1 modulo N. Bei Primzahlen gibt es solche Zahlen nicht.
Lemma 4.39: Wenn p eine ungerade Primzahl ist, dann gilt $b^2\ mod\ p = 1$, $b \in \mathbb{Z}_p$, genau für $b \in \{1, p-1\}$.
Beweis: Offensichtlich gilt für jedes beliebige $m \geq 2$, dass $1^2\ mod\ m = 1$ und $(m-1)^2\ mod\ m = (m(m-2)+1)\ mod\ m = 1$ ist. Nun sei $b \in \{0,...,p-1\}$ beliebig mit $b^2 \equiv 1(mod\ p)$. Dann gilt $b^2 - 1 \equiv 0(mod\ p)$, also ist p ein Teiler von $b^2-1 = (b+1)(b-1)$. Nach Fakt 4.27 ist p Teiler von $b+1$ oder von $b-1$. Im ersten Fall ist $b \equiv -1(mod\ p)$, im zweiten Fall ist $b \equiv 1(mod\ p)$.
Die im Lemma angegebene Eigenschaft lässt sich also in ein weiteres Zertifikat für zusammengesetzte Zahlen ummünzen: Wenn es eine nichttriviale Quadratwurzel der 1 modulo N gibt, dann ist N zusammengesetzt.
Die vier Zahlen $1, 27, 64$ und 90 sind genau die Quadratwurzeln der 1 modulo 91; davon sind 27 und $64 = 91-27$ nichttrivial. Beachte, dass $1 \equiv 63\ (mod\ 7)$ und $27 \equiv 90 \equiv -1(mod\ 7)$, und dass $1 \equiv 27(mod\ 13)$ und $64 \equiv 90 \equiv -1(mod\ 13)$. Allgemeiner sieht man mit der Verallgemeinerung von Fakt 4.21 (Chinesischer Restsatz) auf r Faktoren leicht ein, dass es für ein Produkt $N = p_1...p_r$ aus verschiedenen ungeraden Primzahlen $p_1,...,p_r$ genau 2^r Quadratwurzeln der 1 modulo N gibt, nämlich die Zahlen $b, 0 \leq b < N$, die $b\ mod\ p_j \in \{1, p_j-1\}, 1 \leq j \leq r$, erfüllen. Wenn N nicht sehr viele verschiedene Primfaktoren hat, ist es also aussichtslos, einfach zufällig gewählte b's darauf zu testen, ob sie vielleicht nichttriviale Quadratwurzeln der 1 sind. Dennoch wird uns dieser Begriff bei der Formulierung eines effizienten Primzahltests helfen.

Der Miller-Rabin-Test

Wir kehren nochmals zum Fermat-Test zurück und sehen uns die dort durchgeführte Exponentiation $a^{N-1}\ mod\ N$ etwas genauer an. Die Zahl $N-1$ ist gerade, daher kann man sie als $N-1 = u*2^k$ schreiben, für eine ungerade Zahl u und ein $k \geq 1$. Dann gilt
$a^{N-1} \equiv (a^u\ mod\ N)^{2^k}\ mod\ N$, und wir können $a^{N-1}\ mod\ N$ mit $k+1$ Zwischenschritten berechnen: Mit $b_0 = a^u\ mod\ N$
$b_1 = b_0^2\ mod\ N = a^{u*2}\ mod\ N$, $b_2 = b_1^2\ mod\ N = a^{u*2^2}\ mod\ N$, ...
$b_i = b_{i-1}^2\ mod\ N = a^{u*2^i}\ mod\ N$, ... $b_k = b_{k-1}^2\ mod\ N = a^{u*2^k}\ mod\ N$
ist $b_k = a^{N-1}\ mod\ N$. Beispielsweise erhalten wir für $N = 325 = 5^2*13$ den Wert $N-1 = 324 = 81*22$. In Tabelle 4 berechnen wir a^{81}, a^{162} und a^{324}, alle modulo 325, für verschiedene a.
Tabelle 4: Potenzen $a^{N-1}\ mod\ N$ mit Zwischenschritten, $N = 325$. (Die ,,Fälle" werden weiter unten erklärt.

a	$b_0 = a^{81}$	$b_1 = a^{162}$	$b_2 = a^{324}$	F-Z.?	MR-Z.?	Fall
126	1	1	1	—	—	1
49	324	1	1	—	—	2a
7	307	324	1	—	—	2b
2	252	129	66	—	×	3
15	200	25	300	—	×	3
224	274	1	1	—	×	4a
201	226	51	1	—	×	4b

Die Grundidee des Miller-Rabin-Tests ist nun, diese verlangsamte Berechnung der Potenz $a^{N-1}\ mod\ N$ auszuführen und dabei nach nichttrivialen Quadratwurzeln der 1 Ausschau zu halten.
Im Beispiel sehen wir, dass 2 ein F-Zeuge für 325 ist, der in $\mathbb{Z}_{325}^*$ liegt, und dass 15 ein F-Zeuge ist, der nicht in $\mathbb{Z}_{325}^*$ liegt. Dagegen sind 126, 49, 7, 224 und 201 F-Lügner für 325. Wenn wir aber $224^{324}\ mod\ 325$ mit zwei Zwischenschritten berechnen, dann entdecken wir, dass 274 eine nichttriviale Quadratwurzel der 1 ist, was beweist, dass 325 keine

Primzahl ist. Ähnlich liefert die Berechnung mit Basis 201, dass 51 eine nichttriviale Quadratwurzel der 1 ist. Die entsprechenden Berechnungen mit 49 und 7 dagegen liefern keine weiteren Informationen, weil $49^{81} \equiv -1 (mod\ 325)$ und $7^{162} \equiv 32^{162} \equiv -1 (mod\ 325)$ gilt. Auch die Berechnung der Potenzen von 126 liefert keine nichttriviale Quadratwurzel der 1, weil $126^{81}\ mod\ 325 = 1$ gilt.
Wie kann die Folge $b_0, ..., b_k$ überhaupt aussehen? Wenn $b_i = 1$ oder $b_i = N-1$ gilt, dann sind $b_{i+1}, ..., b_k$ alle gleich 1. Daher beginnt die Folge $b_0, ..., b_k$ mit einer (eventuell leeren) Folge von Elementen $\notin \{1, N-1\}$ und endet mit einer (eventuell leeren) Folge von Einsen. Diese beiden Teile können von einem Eintrag $N-1$ getrennt sein oder nicht.
Die möglichen Muster sind in Tabelle 5 angegeben. Dabei steht ,,*" für ein Element $\notin \{1, N-1\}$. Wir unterscheiden vier Fälle (mit Unterfällen).

- Fall 1: $b_0 = 1$. - Dann ist $b_1 = ... = b_k = 1$.
- Fall 2: Die Folge endet mit $b_k = 1$, und vor der ersten 1 in der Folge steht eine $N-1$.
- Fall 2a: $b_0 = N-1$
- Fall 2b: $b_i = N-1$ für ein $i \in \{1, ..., k-1\}$
- Fall 3: Die Folge endet mit $b_k \neq 1$. - Dann ist N zusammengesetzt, weil a ein F-Zeuge für N ist. In $b_0, ..., b_{k-1}$ können 1 und $N-1$ nicht vorkommen.
- Fall 4: Die Folge endet mit $b_k = 1$, aber $b_0 \neq 1$ und in $b_0, ..., b_{k-1}$ kommt $N-1$ nicht vor. - Dann gibt es ein $i \in \{1, ..., k\}$ mit $b_{i-1} \notin \{1, N-1\}$ und $b_i = 1$. Das heißt, dass b_{i-1} eine nichttriviale Quadratwurzel der 1 ist. Also ist N zusammengesetzt.
- Fall 4a: $i < k$
- Fall 4b: $i = k$

In Tabelle 4 findet man konkrete Beispiele für das Eintreten aller Fälle für die zusammengesetzte Zahl $N = 325$.
Tabelle 5: Potenzen $a^{N-1}\ mod\ N$ berechnet mit Zwischenschritten, mögliche Fälle.

b_0	b_1	...				...	b_{k-1}	b_k	Fall	F-Z.?	MR-Z.?
1	1	...	1	1	1	...	1	1			
N-1	1	...	1	1	1	...	1	1	2a		
*	*	...	*	N-1	1	...	1	1	2b		
*	*	...	*	*	...	*	$\neq 1$	3		×	×
*	*	...	*	1	1	...	1	1	4a		×
*	*	...	*	*	...	*	1	4b			×

Wir beobachten: Wenn N eine Primzahl ist, dann gilt nach dem kleinen Satz von Fermat für jedes a die Gleichung $b_k = a^{N-1}\ mod\ N = 1$.
Weiter kann es nach Lemma 4.39 nicht passieren, dass $b_{i-1} \neq N-1$ und $b_i = 1$ ist. Also können für eine Primzahl N nur Fälle 1 und 2 vorkommen. Umgekehrt findet man im Beispiel $N = 17$, dass $a = 1$ zu Fall 1, $a = 16$ zu Fall 2a, und $a = 2$ bzw. $a = 3$ zu Fall 2b mit $b_2 = 16$ bzw. $b_3 = 16$ führt. Um bei Primzahlen keine falschen Ergebnisse zu erzeugen, müssen wir also in den Fällen 1 und 2 den Wert 0 ausgeben. In den Fällen 3 und 4 hingegen stellt die Zahl a (mit ihren Potenzen $b_0, ..., b_k$) einen Beleg dafür dar, dass N keine Primzahl ist: Im Fall 3 ist a ein F-Zeuge, im Fall 4 ist b_{i-1} eine nichttriviale Quadratwurzel der 1, und das kann nur passieren, wenn N zusammengesetzt ist. Hier können wir also 1 ausgeben.
Das ist auch schon der ganze Algorithmus von Miller und Rabin, abstrakt formuliert: Wähle a aus $\{1, ..., N-1\}$ zufällig. Finde heraus, ob Fall 1 oder 2 eintritt (dann ist die Ausgabe 0) oder Fall 3 oder 4 eintritt (dann ist die Ausgabe 1).
Betrachten von Tabelle 5 zeigt sofort, dass Fall 1 oder 2 genau dann eintritt, wenn Folgendes gilt: $b_0 = 1$ oder in der Folge $b_0, ..., b_{k-1}$ zu a kommt $N-1$ vor. (4.4) (Das letzte Folgenglied b_k spielt keine Rolle für die Unterscheidung.) Dies führt zu folgender Definition in Analogie zu der der F-Zeugen und F-Lügner.
Definition 4.40: Sei $N \geq 3$ ungerade und zusammengesetzt. Wir schreiben $N-1 = u * 2^k$, für u ungerade, $k \geq 1$. Eine Zahl $a, 1 \leq a < N$, heißt ein MR-Zeuge für N, wenn (4.4) nicht gilt, d. h. $a^u \neq 1$ und $a^{u*2^i} \not\equiv N-1 (mod\ N)$ für alle i mit $0 \leq i < k$ (Fälle 3 und 4). Eine Zahl $a, 1 \leq a < N$, heißt ein MR-Lügner für N, wenn (4.4) gilt, $a^u \equiv 1$ oder $a^{u*2^i} \equiv N-1 (mod\ N)$ für ein i mit $0 \leq i < k$ (Fälle 1 und 2). Die Menge der MR-Lügner nennen wir L_N^{MR}.
Aus der obigen Diskussion der möglichen Fälle folgt sofort:
Lemma 4.41: Sei $N \geq 3$ ungerade.

1. a ist F-Zeuge für $N \Rightarrow a$ ist MR-Zeuge für N (Fall 3).
2. Wenn N eine Primzahl ist, dann gilt (4.4) für alle $a < N$.

Im Jahr 1976 stellte Gary M. Miller einen deterministischen Algorithmus vor, der auf der Idee beruhte, die Folge $b_0, ..., b_k$ zu betrachten. Er testete die kleinsten $O((log\ N)^2)$ Elemente $a \in \{2, 3, ..., N-1\}$ auf die Eigenschaft, MR-Zeugen zu sein. Der Algorithmus hat polynomielle Laufzeit und liefert das korrekte Ergebnis, wenn die ,,erweiterte Riemannsche Vermutung (ERH)" stimmt, eine berühmte Vermutung aus der Zahlentheorie, die bis heute unbewiesen ist. Um 1980 schlugen Michael Rabin und (unabhängig) Louis Monier vor, Millers Algorithmus so zu modifizieren, dass die zu testende Zahl a zufällig gewählt wird, und bewiesen, dass dieser Algorithmus konstante Fehlerwahrscheinlichkeit hat. Der Algorithmus heißt heute allgemein der Miller-Rabin-Test.
Der Test selbst ist leicht und sehr effizient durchführbar: Man wählt a zufällig und berechnet zunächst $b_0 = a^u\ mod\ N$ und testet, ob $b_0 \in \{1, N-1\}$ ist. Falls ja, trifft Fall 1 bzw. Fall 2a zu, die Ausgabe ist 0. Falls nein, berechnet man durch iteriertes Quadrieren nacheinander die Werte $b_1, b_2, ...,$ bis

- entweder der Wert $N-1$ auftaucht (Fall 2b, Ausgabe 0)
- oder der Wert 1 auftaucht (Fall 4a, a ist MR-Zeuge, Ausgabe 1)
- oder $b_1, ..., b_{k-1}$ berechnet worden sind, ohne dass Werte $N-1$ oder 1 vorgekommen sind (Fall 3, a ist F-Zeuge, oder 4b, a ist MR-Zeuge, Ausgabe 1).

In Pseudocode sieht das Verfahren dann folgendermaßen aus. Im Unterschied zur bisherigen Beschreibung wird nur eine Variable b benutzt, die nacheinander die Werte $b_0, b_1, ...$ aufnimmt.
Algorithmus 4.6 Der Miller-Rabin-Primzahltest

- Eingabe: Eine ungerade Zahl $N \geq 3$
- Methode:
  - Bestimme ??? ungerade und $k \geq 1$ mit $N-1 = u * 2^k$
  - wähle zufällig ein a aus $\{1, ..., N-1\}$
  - $b \leftarrow a^u\ mod\ N$ // mit schnellem Potenzieren
  - if $b \in \{1, N-1\}$ then return 0; // Fall 1 oder 2a
  - for j from 1 to $k-1$ do //,,wiederhole (k-1)-mal"
  - $b \leftarrow b^2\ mod\ N$
  - if $b = N-1$ then return 0 // Fall 2b
  - if $b = 1$ then return 1 // Fall 4a
  - return 1 // Fall 3 oder 4b

Wie steht es mit der Rechenzeit des Algorithmus? Man benötigt höchstens $log\ N$ Divisionen durch 2, um u und k zu finden (Zeile 1). Wenn man benutzt, dass N normalerweise in Binärdarstellung gegeben sein wird, ist die Sache noch einfacher: k ist die Zahl der Nullen, mit der die Binärdarstellung von $N-1$ aufhört, u ist die Zahl, die aus der Binärdarstellung von N durch Weglassen dieser letzten Nullen entsteht. Die Berechnung von $a^u\ mod\ N$ mit schneller Exponentiation in Zeile 3 benötigt $O(log\ N)$ arithmetische Operationen und $O((log\ N)^3)$ Ziffernoperationen (mit der Schulmethode für Multiplikation und Division). Die Schleife in Zeilen 5-8 wird höchstens $(k-1)$-mal durchlaufen; offenbar ist $k \leq log\ N$. In jedem Durchlauf ist die Multiplikation modulo N die teuerste Operation. Insgesamt benutzt der Algorithmus $O(log\ N)$ arithmetische Operationen auf Zahlen, die kleiner als N^2 sind, und $O((log\ N)^3)$ Ziffernoperationen. Wenn man die Rechenzeit mit der des Fermat-Tests vergleicht, stellt man fest, dass die Rechenwege etwas unterschiedlich sind, aber dieselbe Anzahl von Multiplikationen modulo N anfallen. Nur werden beim Miller-Rabin-Test ,,unterwegs" noch einige Zwischenergebnisse darauf getestet, ob sie gleich 1 oder gleich $N-1$ sind. Dies verursacht aber kaum Mehraufwand. Nun wenden wir uns dem Ein-/Ausgabeverhalten des Miller-Rabin-Tests zu.
Lemma 4.42: Wenn N eine Primzahl ist, gibt der MR-Test 0 aus.
Beweis: Wir haben oben schon festgestellt, dass Ausgabe 1 genau dann erfolgt, wenn die zufällig gewählte Zahl a ein MR-Zeuge ist. Nach Lemma 4.41(b) gibt es nur dann MR-Zeugen, wenn N zusammengesetzt ist.

Fehlerschranke für den Miller-Rabin-Test

Wir müssen nun noch die Fehlerwahrscheinlichkeit des Miller-Rabin-Tests für den Fall analysieren, dassN eine zusammengesetzte (ungerade) Zahl ist. Dies wird also für diesen Abschnitt angenommen.
Wir beweisen, dass die Wahrscheinlichkeit, dass der Miller-Rabin-Test die (unerwünschte) Antwort 0 gibt, kleiner als $\frac{1}{2}$ ist. Dazu wollen wir ähnlich vorgehen wie bei der Analyse des Fermat-Tests für Nicht-Carmichael-Zahlen (Beweis von Satz 4.36). Dort haben wir gezeigt, dass die F-Lügner für solche N höchstens die Hälfte von $\mathbb{Z}_N^*$ ausmachen. Wenn N keine Carmichael-Zahl ist, ist dies leicht: Nach Lemma 4.41(a) gilt $L_N^{MR} \supseteq L_N^F$, und wir können den Beweis von Satz 4.36 verwenden. Der folgende Beweis ist nicht prüfungsrelevant.
Es bleibt der Fall, dass N eine Carmichael-Zahl ist. Unser Plan ist, eine Menge B_N mit $L_N^{MR} \supseteq B_N \supseteq \mathbb{Z}_N^*$ zu definieren, die höchstens die Hälfte der Elemente von $\mathbb{Z}_N^*$ enthält. Weil u ungerade ist, gilt $(N-1)^{u*2^0} \equiv (N-1)^u \equiv (-1)^u = -1 \equiv N-1 (mod\ N)$. Sei i_0 das größte $i \in \{0, 1, ..., k\}$ mit der Eigenschaft, dass es einen MR-Lügner $a_\#$ mit $a_\#^{u*2^i} mod\ N = N-1$ gibt. (Wir werden dieses $a_\#$ weiter unten nochmals benutzen.) Weil N eine Carmichael-Zahl ist, gilt $a^{u*2^k} mod\ N = a^{N-1} mod\ N = 1$ für alle $a \in \mathbb{Z}_N^*$, und daher $0 \leq i_0 < k$.
Wir definieren: $B_N := \{a \in \mathbb{Z}_N^* | a^{u*2^{i_0}}\ mod\ N \in \{1, N-1\}\}$. (4.5) Zur Veranschaulichung betrachte man Tabelle 6. In der dort angegebenen Matrix entspricht jede Zeile einem der $\varphi(N)$ Elemente von $\mathbb{Z}_N^*$, beginnend mit $a_1 = 1$ und $a_2 = -1 \equiv N-1$. In der Zeile für a ist die von a erzeugte Folge $b_0, ..., b_k$ eingetragen. Dass N eine Carmichael-Zahl ist, drückt sich dadurch aus, dass alle Einträge in der b_k-Spalte gleich 1 sind. Der Eintrag für $a_2 = -1$ in der b_0-Spalte ist $N-1$. Spalte i_0 ist die am weitesten rechts stehende Spalte, in der es einen Eintrag $-1 \equiv N-1$ gibt, und $a_\#$ ist ein Element von $\mathbb{Z}_N^*$, das zu diesem Eintrag führt. Die Menge B_N besteht aus den Elementen von $\mathbb{Z}_N^*$, die in Spalte i_0 Eintrag 1 oder $N-1$ haben.
Tabelle 6: Sicht auf B_N als Teilmenge von $\mathbb{Z}_N^*$

a	b_0	b_1	...		b_{i_0}		...
$a_1 = 1$	1	1	...	1	1	1	...
$a_2 = N-1$	$N-1$	1	...	1	1	1	...
a_3	*	*	...	*	$N-1$	1	...
a_4	*	*	...	$N-1$	1	1	...
a	a^u	a^{u*2^1}	...	$a^{u*2^{i_0-1}}$	$a^{u*2^{i_0}}$	$a^{u*2^{i_0+1}}$	...
$a_\#$	*	*	...	*	$N-1$	1	...
$a_?$	*?	...	*?	*?	...	...	...
$a_{\varphi(N)}$	?	?	...	?	?	?	...

Wir werden zeigen, dass diese Menge B_N die gewünschten Eigenschaften hat.
Lemma 4.43:

1. $L_N^{MR} \supseteq B_N$
2. $B_N \not\subseteq \mathbb{Z}_N^*$
3. $|B_N| \leq \frac{1}{2}|\mathbb{Z}_N^*|$, also $Pr_{a \in \{1, ..., N-1\}}$ (a ist MR-Lügner) $< \frac{1}{2}$

Beweis: Teil 1. ist nur die Überprüfung, dass die Definition technisch sinnvoll ist. Der eigentlich interessante Teil ist 2.; Teil 3. ist dann Routine.

1. Sei a ein beliebiger MR-Lügner.
   - Fall 1: $a^u\ mod\ N = 1$. - Dann ist auch $a^{u*2^{i_0}} mod\ N = 1$, und daher gilt $a \in B_N$. (Die Zeile zu a in Abb. 6 sieht aus wie die von $a_1 = 1$.)
   - Fall2: $a^{u*2^i} mod\ N = N-1$ für ein $i < k$ - Dann ist $0 \leq i \leq i_0$ nach der Definition von i_0. Wenn $i = i_0$ ist, haben wir direkt $a \in B_N$ (Beispiel in Tab. 6: a_3 oder $a_\#$); wenn $i < i_0$ ist, dann gilt $a^{u*2^{i_0}} mod\ N = (a^{u*2^i} mod\ N)^{2^{i_0-i}} mod\ N = (-1)^{2^{i_0-i}} mod\ N = 1$, also ebenfalls $a \in B_N$ (Beispiel in Tab. 6: a_2 oder a_4).

2. Dass $B_N \supseteq \mathbb{Z}_N^*$ gilt, folgt aus der Definition. Wir müssen also nur zeigen, dass $\mathbb{Z}_N^* - B_N \neq \varnothing$ gilt. Wir benutzen die in Lemma 4.38 beobachtete Eigenschaft von Carmichael-Zahlen, keine Primzahlpotenz zu sein. Nach diesem Lemma können wir $N = N_1 * N_2$ schreiben, für teilerfremde ungerade Zahlen $N_1, N_2 \geq 3$. Im Folgenden werden die Eigenschaften aus dem Chinesischen Restsatz (Fakt 4.21) benutzen.
   - Die grobe Idee der Konstruktion ist folgende: Wir haben das Element 1 mit $1^{u*2^{i_0}} mod\ N = 1$ und das Element $a_\#$ mit $a^{u*2^{i_0}} \equiv -1(mod\ N)$. Aus diesen beiden Elementen ,,basteln" wir mit Hilfe des Chinesischen Restsatzes ein $b \in \mathbb{Z}_N^*$, das sich modulo N_1 wie 1 und modulo N_2 wie $a_\#$ verhält. Es wird sich zeigen, dass $b^{u*2^{i_0}} mod\ N \notin \{1, N-1\}$ gilt, also $b \notin B_N$ ist.
   - Wir führen diese Idee jetzt formal durch. Sei $x_2 = a_\#\ mod\ N_2$. Nach dem Chinesischen Restsatz (Fakt 4.21) gibt es eine eindeutig bestimmte Zahl $b \in \{0, 1, ..., N-1\}$, die $b \equiv 1(mod\ N_1)$ und $b \equiv x_2(mod\ N_2)$ (4.6) erfüllt. (Es folgt $b \equiv a_\#(mod\ N_2)$.) Wir zeigen, dass b in $\mathbb{Z}_N^* - B_N$ liegt.
   - Wir notieren, dass für beliebige $x, y \in \mathbb{Z}$ gilt: $x \equiv x'(mod\ N) \Rightarrow x \equiv x'(mod\ N_i)$, für $i = 1, 2$. (4.7)
   - (Wenn $x - x'$ durch N teilbar ist, dann auch durch N_1 und N_2.)
   - Beh. 1: $b \in \mathbb{Z}_N^*$.
   - Bew.: Offensichtlich gilt $b^{N-1} \equiv 1^{N-1} \equiv 1(mod\ N_1)$. Weiter haben wir, modulo N_2 gerechnet: $b^{N-1} \equiv a_\#^{N-1} \equiv ((a_\#^{N-1}) mod\ N) \equiv 1(mod\ N_2)$.
   - Wegen der Eindeutigkeitsaussage im Chinesischen Restsatz folgt daraus $b^{N-1} \equiv 1(mod\ N)$. Nach Lemma 4.35 folgt $b \in \mathbb{Z}_N^*$.
   - Beh. 2: $b \notin B_N$.
   - Bew.: Indirekt. Annahme: $b \in B_N$, d.h. $b^{u*2^{i_0}} \equiv 1(mod\ N)$ oder $b^{u*2^{i_0}} \equiv -1(mod\ N)$.
   - 1. Fall: $b^{u*2^{i_0}} \equiv 1(mod\ N)$. - Mit (4.7) folgt $b^{u*2^{i_0}} \equiv 1(mod\ N_2)$. Andererseits gilt $b^{u*2^{i_0}} \equiv x_2^{u*2^{i_0}} \equiv a_\#^{u*2^{i_0}} \equiv (a_\#^{u*2^{i_0}} mod\ N) \equiv N - 1 \equiv -1(mod\ N_2)$, also $2 \equiv 0(mod\ N_2)$, ein Widerspruch, weil $N_2 \geq 3$ ist.
   - 2. Fall: $b^{u*2^{i_0}} \equiv -1(mod\ N)$. - Mit (4.7) folgt $b^{u*2^{i_0}} \equiv -1(mod\ N_1)$. Andererseits gilt $b^{u*2^{i_0}} \equiv 1^{u*2^{i_0}} \equiv 1(mod\ N_1)$, also $2 \equiv 0(mod\ N_1)$, ebenfalls ein Widerspruch.
3. Wir verwenden die in 2. konstruierte Zahl $b \in \mathbb{Z}_N^* - B_N$. Wie im Beweis von Satz 4.36 betrachtet man die injektive Funktion $g_b : B_N \ni a \to ba\ mod\ N \in \mathbb{Z}_N^*$. Es gilt $g_b(a) \notin B_N$ für jedes $a \in B_N$, denn $g_b(a)^{u*2^{i_0}} mod\ N = (b^{u*2^{i_0}} mod\ N)(a^{u*2^{i_0}}) mod\ N \in \{b^{u*2^{i_0}} mod\ N, (N - b^{u*2^{i_0}}) mod\ N\}$, und die letzte Menge ist disjunkt zu $\{1, N-1\}$. Man erhält sofort $|B_N| \leq \frac{1}{2}|\mathbb{Z}_N^*|$.

Ab hier wieder prüfungsrelevant.
Wir haben also eine Schranke von $\frac{1}{2}$ für die Irrtumswahrscheinlichkeit im Miller-Rabin-Algorithmus bewiesen. Eine etwas kompliziertere Analyse zeigt, dass sogar die Fehlerschranke $\frac{1}{4}$ gilt; man kann auch zeigen, dass es zusammengesetzte Zahlen N gibt (zum Beispiel $703 = 19 * 37$), bei denen eine Fehlerwahrscheinlichkeit von fast $\frac{1}{4}$ tatsächlich auftritt.
Durchl-fache Wiederholung, ebenso wie in Algorithmus 4.5, kann man die Fehlerschranke auf 4^{-l} reduzieren. Wir kürzen den Miller-Rabin-Test mit ,,MiRa-Test" ab. Man beachte, dass bei jedem neuen Aufruf eine neue Zufallszahlagewählt wird.

Algorithmus 4.7 Iterierter MR-Test

- Eingabe: Ungerade Zahl $N \geq 3$, eine Zahl $l \geq 1$.
- Methode:
- repeat l times
- if MiRa-Test(N) = 1 then return 1;
- return 0;

Diesen Test wollen wir kurz IterMiRa(N,l) nennen. - Wir erhalten zusammenfassend:
Proposition 4.44: Algorithmus 4.7 benötigt $O(l * log\ N)$ arithmetische Operationen auf Zahlen, die kleiner als N^2 sind, und $O(l * (log\ N)^3)$ Ziffernoperationen. Wenn N eine Primzahl ist, ist die Ausgabe 0, wenn N zusammengesetzt ist, ist die Wahrscheinlichkeit, dass 0 ausgegeben wird, kleiner als 4^{-l}.

Die Erzeugung von (zufälligen) Primzahlen

Für kryptographische Anwendungen (zum Beispiel für die Erzeugung von Schlüsselpaaren für das RSA-Public-Key-Kryptosystem) werden vielziffrige Primzahlen benötigt. Eine typische Aufgabe in diesem Zusammenhang lautet: ,,Finde eine zufällige Primzahl mit n Bits!"
Das heißt: Gesucht ist eine zufällige Primzahl in $[2^{n-1}, 2^n)$. Dabei hängt n von der Anwendung ab; wir können uns z.B. $n = 1024$ oder $n = 2048$ vorstellen.
Man erinnere sich an den Primzahlsatz und an die Dusart-Schranke am Ende von Abschnitt 4.5, die besagt, dass es für $n \geq 9$ in diesem Intervall mindestens $(6/5) * 2^{n-1}/n$ Primzahlen gibt.
Ein naheliegender Ansatz zur Erzeugung einer zufälligen n-Bit-Primzahl ist dann folgender: Man wählt wiederholt eine (ungerade) Zahl N aus $[2^{n-1}, 2^n)$ zufällig und wendet auf sie den (iterierten) Miller-Rabin-Test an. Dies wiederholt man, bis eine Zahl gefunden wurde, die die Ausgabe 0 liefert.
Algorithmus 4.8: GetPrime - Randomisierte Primzahlerzeugung

- Eingabe: Bitanzahl $n \geq 9$, Zahl $l \geq 1$ // Zuverlässigkeitsparameter
- Methode:
- repeat
- $N \leftarrow$ zufällige ungerade Zahl in $[2^{n-1}, 2^n)$;
- until IterMiRa(N,l) = 0; // Algorithmus 4.7
- return N.

Die Ausgabe ist korrekt, wenn der Algorithmus eine Primzahl zurückgibt, ein Fehler tritt auf, wenn eine zusammengesetzte Zahl zurückgegeben wird. Auf den ersten Blick könnte man meinen (aufgrund von Proposition 4.44), dass die Fehlerwahrscheinlichkeit höchstens $1/4^l$ ist - eben die Fehlerwahrscheinlichkeit des iterierten MiRa-Tests. Wir werden sehen, dass wir auf der Basis der Analyse des Miller-Rabin-Tests nur ein etwas schwächeres Ergebnis bekommen. (Tatsächlich ist die Fehlerwahrscheinlichkeit durch $1/4^l$ beschränkt, für (von l abhängig) genügend großen. Dies kann man aber nur durch fortgeschrittene zahlentheoretische Untersuchungen über die erwartete Wahrscheinlichkeit, dass eine zufällige ungerade zusammengesetzte Zahl den l-fach iterierten MiRa-Test übersteht, beweisen [P. Beauchemin, G. Brassard, C. Crèpeau, C. Goutier, C. Pomerance: The generation of random numbers that are probably prime.J. Cryptology 1 (1): 53-64 (1988)].)
Wir definieren: $Prim_n := \{p \in [2^{n-1}, 2^n) | \text{p ist Primzahl}\}$.
Satz 4.45: Bei der Anwendung von Algorithmus 4.8 auf $n \geq 9$ gilt:

1. Wenn das Ergebnis eine Primzahl ist, hat jede Primzahl in $[2^{n-1}, 2^n)$ dieselbe Wahrscheinlichkeit, als Ergebnis zu erscheinen.
2. $Pr(GetPrime(n, l) \notin Prim_n) \leq \frac{5n}{12*4^l} = O(\frac{n}{4^l})$. (Nicht $1/4^l$, wie naiv vermutet.)
3. Die erwartete Rundenzahl ist $O(n)$, der erwartete Rechenaufwand ist $O(n^2)$ arithmetische Operationen und $O(n^4)$ Ziffernoperationen.

Beweis:

1. Eine Primzahl wird als Resultat genau dann geliefert, wenn keine zusammengesetzte Zahl fälschlicherweise vom Miller-Rabin-Test akzeptiert wird, bevor (in Zeile 2) die erste echte Primzahl gewählt wird. Jede der Primzahlen in $[2^{n-1}, 2^n)$ hat dieselbe Wahrscheinlichkeit, diese erste gewählte Primzahl zu sein.
2. $Pr(GetPrime(n, l) \notin Prim_n) = Pr(\exists i \geq 1 :$ in Runden j=1,...,i-1 wird eine zusammengesetzte Zahl N gewählt und erl in Runde i wird zusammengesetzte Zahl N gewählt und der iterierte MiRa $\leq \sum_{i \geq 1}(1 - \frac{|Prim_n|}{2^{n-2}})^{i-1} * \frac{1}{4^l} = \frac{2^{n-2}}{|Prim_n|} * \frac{1}{4^l} \leq \frac{2^{n-2}}{\frac{6}{5} * 2^{n-1}/n} * \frac{1}{4^l} = \frac{5n}{12*4^l}$. Wir haben benutzt, dass es in $[2^{n-1}, 2^n)$ genau 2^{n-2} ungerade Zahlen gibt.
3. Da man in jeder Runde mit Wahrscheinlichkeit mindestens $\frac{|Prim_n|}{2^{n-2}}$ eine Primzahl wählt, ist die erwartete Rundenzahl nicht größer als $\frac{2^{n-2}}{|Prim_n|} \leq \frac{5n}{12}$.

Bemerkung: Bei der Erzeugung zufälliger n-Bit-Primzahlen für kryptographische Zwecke wird man aus Effizienzgründen nicht unseren Algorithmus anwenden, der $\Theta(n^2)$ Multiplikationen benötigt, sondern eine Kombination zweier verschiedener Primzahltests (z. B. Miller-Rabin und ,,Lucas Strong Probable Prime Test") mit sehr wenigen Iterationen und einem Test auf Teilbarkeit durch sehr kleine Primteiler. Dies erfordert nur $O(n)$ Multiplikationen. Die Dichte der zusammengesetzten Zahlen, die von einem solchen Test nicht erkannt werden, ist als sehr gering einzuschätzen. Über eine interessante experimentelle Untersuchung hierzu berichtet die kurze Notiz http://people.csail.mit.edu/rivest/Rivest-FindingFourMillionLargeRandomPrimes.ps von Ron Rivest (bekannt von ,,RSA" und von ,,Cormen, Leiserson, Rivest und Stein").

Beweise und Bemerkungen zu Kapitel 4

Beweis der Existenz und der Eindeutigkeit des größten gemeinsamen Teilers $ggT(x, y)$ zweier Zahlen (Definition 4.3.2):

- *Eindeutigkeit*: Wenn in 2. d und d' beide (i) und (ii) erfüllen und nichtnegativ sind, dann folgt $d|d'$ und $d'|d$, also $d = d'$ nach Fakt 4.2.5.
- *Existenz*: Weil t gemeinsamer Teiler von x und y ist genau dann wenn t gemeinsamer Teiler von $a = |x|$ und $b = |y|$ ist, und weil offenbar das Vertauschen von x und y nichts ändert, können wir uns auf den Fall $x = a \geq y = b \geq 0$ beschränken. Wir zeigen durch Induktion über $b = min\{a, b\}$, dass $ggT(a, b)$ existiert.
- *Induktionsanfang*: $b = 0$.
- 1. Fall: $a = 0$. Dann ist jede Zahl t ein gemeinsamer Teiler von a und b. Wir wählen $d = 0$. Dann gilt (i), weil 0 Teiler von 0 ist, und (ii), weil jede Zahl t die Zahl 0 teilt. (0 ist ,,größter" gemeinsamer Teiler von 0 und 0 im Sinn der Quasiordnung ,,Teilbarkeit". Hier ist 0 das größte Element überhaupt.)
- 2. Fall: $a > 0$. Wir wählen $d = a$. Dann gilt (i), weil a Teiler von a und von 0 ist, und es gilt (ii), weil jeder gemeinsame Teiler von a und 0 auf jeden Fall Teiler von a ist.
- *Induktionsschritt*: $b > 0$. Setze $q := a\ div\ b$ und $r := a - qb = a\ mod\ b$ und $(a', b') := (b, r)$. Dann ist $b' = r < b = a'$. Nun haben a und b genau dieselben gemeinsamen Teiler wie a' und b'. (Aus $t|a$ und $t|b$ folgt $t|(a - qb)$, also $t|a'$, und aus $t|a'$ und $t|b'$ folgt $t|r + qb$, also $t|a$.) Nach I.V. existiert $d = ggT(a', b')$, und dieses d ist dann auch größter gemeinsamer Teiler von a und b.

Asymmetrische Verschlüsselung: RSA & Co.

Das RSA-Kryptosystem (,,Vorlesungs-Version")

RSA: Erfunden von Ronald L. Rivest, Adi Shamir und Leonard Adleman, 1977. Ein ähnliches Verfahren wurde (von J. H. Ellis, C. C. Cocks und M. J. Williamson) bereits Anfang der 1970er Jahre in den

Government Communications Headquarters, der Kryptologie-Abteilung des britischen Geheimdienstes, entwickelt, aber nicht veröffentlicht.
Vorsicht: In der hier zunächst vorgestellten einfachen/naiven Version ist RSA unsicher. Es müssen Modifikationen und Einschränkungen vorgenommen werden, um ein sicheres Verfahren zu erhalten. Gute Referenz hierfür: Baumann, Franz, Pfitzmann, Kryptographische Systeme, Springer Vieweg 2014, S. 231ff.
Wir betrachten ein einfaches Kommunikationsszenario. Die Teilnehmer sind Alice und Bob. Alice möchte an Bob eine Nachricht schicken. Dies soll über einen offen zugänglichen Kanal (E-Mail, ein Webportal) geschehen. Dieser Kanal wird von Eva mitgelesen. Alice und Bob wollen vermeiden, dass Eva den Inhalt der Nachricht erfährt. Wenn auch Bob an Alice Nachrichten schicken möchte, müssen die Aktionen auch mit vertauschten Rollen ausgeführt werden. Die Menge X der möglichen Nachrichten kann als Teilmenge von $\{0,1\}^*$ aufgefasst werden. Wir möchten vermeiden, dass sich Alice und Bob vor der Kommunikation auf einen gemeinsamen Schlüssel einigen müssen. Stattdessen wird ein Schlüsselpaar $(k, \hat{k})$ verwendet. Die erste Komponente k heißt der öffentliche Schlüssel von Bob und wird von Bob allen zugänglich gemacht, etwa über seine Webseite oder als Anhang von E-Mails. Die zweite Komponente $\hat{k}$ heißt der private Schlüssel von Bob und ist nur ihm bekannt. Die Menge der Schlüsselpaare $(k, \hat{k})$, die zusammengehören, nennen wir die Schlüsselmenge K.
Definition 5.1: Ein Public-Key-Kryptosystem (X, Y, K, E, D) hat 5 Komponenten:

- Klartextmenge X (endlich),
- Chiffretextmenge Y (endlich),
- Schlüsselmenge K, wobei $K \supseteq K_{pub} \times K_{priv}$ für Mengen K_{pub} und K_{priv},
- Verschlüsselungsfunktion $E : X \times K_{pub} \to Y$,
- Entschlüsselungsfunktion $D : Y \times K_{priv} \to X$,
- wobei die folgende Dechiffrierbedingung gilt: $D(E(x,k), \hat{k}) = x$, für alle $x \in X, (k, \hat{k}) \in K$.

Um ein solches System benutzen zu können, benötigt man noch ein Verfahren, um Schlüsselpaare $(k, \hat{k})$ zu erzeugen. Dieses Verfahren heißt G, ist randomisiert und wird von Bob angestoßen, dem auch das Ergebnis mitgeteilt wird. Dies ist notwendig, da die Ausgabe von G den geheimen Schlüsselteil $\hat{k}$ enthält. Nach Erzeugung des Paars $(k, \hat{k})$ gibt Bob k bekannt und speichert $\hat{k}$ geheim ab. Wenn Alice Bob eine Nachricht $x \in X$ schicken will, berechnet sie $y = E(x,k)$ und schickt y. Wenn Bob einen Chiffretext y empfängt, berechnet er $z = D(y, \hat{k})$. Nach der Dechiffrierbedingung ist dies wieder x.
Das RSA-System in seiner puren Form benutzt Klartext- und Chiffretextmenge $X = Y = [N]$, für eine feste Zahl N. Ab hier nehmen wir stets an, dass der Klartext x selbst eine Zahl in $X = X_N = [N]$ ist. Dann ist $Y = Y_N = [N]$ die Menge der möglichen Chiffretexte.
Das (randomisierte) Verfahren G zur Erzeugung von Schlüsselpaaren $(k, \hat{k})$ aus $K_{pub} \times K_{priv}$ hat üblicherweise einen Parameter l, die Schlüssellänge. Den Wertebereich von G, also die Menge aller möglichen Schlüsselpaare $(k, \hat{k})$, wollen wir K nennen. Dabei ist k Bobs ,,öffentlicher Schlüssel", der öffentlich zugänglich ist (zum Beispiel in einem allgemein zugänglichen ,,Schlüsselbuch" oder auf Bobs Webseite) und $\hat{k}$ ist sein ,,privater Schlüssel", den nur er kennt. Alice (und andere Teilnehmer, die Bob eine Nachricht schicken wollen) verwendet k zur Verschlüsselung, Bob verwendet $\hat{k}$ zur Entschlüsselung.
Es gibt einen (eventuell randomisierten) Verschlüsselungsalgorithmus E, der aus $x \in X$ und dem öffentlichen Schlüssel k den Chiffretext $y \in Y$ berechnet, sowie einen (deterministischen) Entschlüsselungsalgorithmus D, der aus $y \in Y$ und dem privaten Schlüssel $\hat{k}$ den entschlüsselten Text $z \in X$ berechnet. Abstrakt haben wir zwei Funktionen
$E : (x,k) \to y \in Y$, wobei $x \in X$ und k erste Komponente eines Schlüsselpaars $(k, \hat{k}) \in K$ mit zugeordneter Zahl N ist, und
$D : (y, \hat{k}) \to z \in X$, wobei $y \in Y$ und $\hat{k}$ zweite Komponente eines Schlüsselpaars $(k, \hat{k}) \in K$ mit zugeordneter Zahl N ist.

Typischerweise sind die Funktionen E und D allgemein bekannt; das einzige Geheimnis im Verfahren steckt im privaten Schlüssel $\hat{k}$.
Korrektheit/Dechiffrierbedingung: Es gilt: $D(E(x,k), \hat{k}) = x$, für $x \in X$ und $(k, \hat{k}) \in K$.
Sicherheit: Es soll verhindert werden, dass Eva aus dem Chiffretext y und dem öffentlichen Schlüssel k ,,relevante Informationen" über den Klartext x gewinnen kann. Hierbei wird üblicherweise angenommen, dass sie nur ,,begrenzten Rechenaufwand" betreiben kann.

Schlüsselerzeugung

Wir beschreiben den Erzeugungsprozess G. Dieser Vorgang wird von Bob selbst oder einer Sicherheitsagentur (,,trust center") durchgefuhrt. Die Schlüssellänge ist festzulegen (etwa $l = 1024, 2048$ oder 4096 Bits). Danach werden zwei (zufällige, verschiedene) Primzahlen p und q bestimmt, deren Bitlänge die Hälfte der Schlüssellänge ist. Hierzu kann man im Prinzip wie in Abschnitt 4.7 beschrieben vorgehen. Nun wird das Produkt $N = pq$ berechnet. Die Zahl N hat l oder $l-1$ Bits. Weiter wird $\varphi(N) = (p-1)(q-1)$ berechnet. Es wird eine Zahl $e \in \{3, ..., \varphi(N)-1\}$ mit $ggT(e, \varphi(N)) = 1$ gewählt. (Überprüfung mittels des erweiterten Euklidischen Algorithmus (Algorithmus 4.2), Rechenzeit $O(l^3)$.) Dann wird das multiplikative Inverse $d < \varphi(N) modulo\ \varphi(N)$ von e bestimmt, so dass also $ed\ mod\ \varphi(N) = 1$ gilt. (Man beachte, dass nur ungerade e in Frage kommen, weil $\varphi(N)$ gerade ist. Man weiß, dass die Anzahl der geeigneten Werte e mindestens $\frac{\varphi(N)}{log(\varphi(N))}$ ist, so dass die erwartete Anzahl von Versuchen $O(log(\varphi(N))) = O(logN)$ ist.)
Bob erhält aus seiner Rechnung N, e und d. Aus diesen wird das Schlüsselpaar $(k, \hat{k})$ gebildet:

- Der öffentliche Schlüssel k ist das Paar (N, e). Dieser wird bekanntgegeben.
- Der geheime Schlüssel $\hat{k}$ ist (N, d). (Natürlich ist nur der Teil d wirklich geheim.)

Der erwartete Berechnungsaufwand für die Schlüsselerzeugung ist $O((log\ N)^4) = O(l^4)$, weil dies die Kosten der Primzahlerzeugung sind (siehe Abschnitt 4.7); die Kosten für das Finden von e sind geringer.

Verschlüsselung

- Gegeben: Klartext x.
- Benötigt: Öffentlicher Schlüssel $k = (N, e)$.

Verschlüsselung von $x \in X = [N] : y = E(x, (N, e)) := x^e\ mod\ N$. (Zu berechnen mit schneller Exponentiation, Rechenzeit $O((log\ N)^3) = O(l^3)$.)

Entschlüsselung

- Gegeben: Chiffretext y.
- Benötigt: Privater Schlüssel $\hat{k} = (N, d)$.

$z = D(y, (N, d)) := y^d\ mod\ N$. (Zu berechnen mit schneller Exponentiation, Rechenzeit $O((log\ N)^3) = O(l^3)$.)
Nach den Potenzrechenregeln, die auch für die modulare Arithmetik gelten, ist $z = (x^e\ mod\ N)^d\ mod\ N = x^{ed}\ mod\ N$. Die Korrektheit der Entschlüsselung beruht auf folgendem Satz.
Satz 5.2: Korrektheit/Dechiffrierbedingung von RSA: Wenn $ed\ mod\ \varphi(N) = 1$ gilt, dann haben wir $x^{ed}\ mod\ N = x$, für alle $x \in [N]$.
Beweis: Weil x und $x^{ed}\ mod\ N$ beide in $[N]$ liegen, also der Betrag ihrer Differenz kleiner als N ist, genügt es zu zeigen, dass $x^{ed} \equiv x (mod\ N)$ gilt, d.h. dass N Teiler von $x^{ed} - x$ ist. Dies weisen wir im Folgenden nach. Betrachte die Primfaktoren p und q von N. Da $ed \equiv 1 (mod\ \varphi(N))$ und $\varphi(N) = (p-1)(q-1)$ gilt, haben wir $ed = k(p-1)(q-1) + 1$ für eine Zahl $k \in \mathbb{N}$, also $x^{ed} - x = (x^{k(p-1)(q-1)} - 1) * x$. Behauptung: p teilt $x^{ed} - x$. - Wenn x durch p teilbar ist, ist dies klar. Wir können also annehmen, dass p kein Teiler von x ist, dass also x und p teilerfremd sind. Nach dem kleinen Satz von Fermat (Fakt 4.27) gilt $x^{p-1}\ mod\ p = 1$, also auch $x^{k(p-1)(q-1)}\ mod\ p = (x^{(p-1)})^{k(q-1)}\ mod\ p = 1$. Dies bedeutet, dass p ein Teiler von $x^{k(p-1)(q-1)} - 1$ ist, also auch ein Teiler von $x^{ed} - x$. Die Behauptung gilt naturlich genauso für q: Auch q ist ein Teiler von $x^{ed} - x$. Da p und q teilerfremd sind, folgt aus Fakt 4.8, dass $N = pq$ Teiler von $x^{ed} - x$ ist, wie gewünscht.
Beispiel:

- Schlüsselerzeugung: $N = 55 = 5 * 11$.
- $\varphi(N) = 4 * 10 = 40$.
- $e = 3, ggT(3, 40) = 1, d = e^{-1}\ mod\ 40 = 27$, weil $3 * 27 = 81 \equiv 1 (mod\ 40)$ gilt.
- Also: $k = (55, 3)$, $\hat{k} = (55, 27)$.
- Sei der Klartext $x = 9$ gegeben.
- Verschlüsselung: $y = 9^3\ mod\ 55 = 729\ mod\ 55 = 14$.
- Entschlüsselung: Wir rechnen modulo 55: $y^{27} \equiv (14^2 * 14)^9 \equiv (31 * 14)^9 \equiv 434^9 \equiv ((-6)^3)^3 \equiv 4^3 \equiv 64 \equiv 9\ (mod\ 55)$.

RSA, unsichere Version, insgesamt

- Bob erzeugt einen Schlüsselsatz $(k, \hat{k}) = ((N, e), (N, d))$.
- Er veröffentlicht (N, e) und hält d geheim.
- Wenn Alice $(x_1, ..., x_r) \in [N]^r$ an Bob schicken will, verschlüsselt sie $y_i := x_i^e\ mod\ N$, für $i = 1, ..., r$, und sendet $(y_1, ..., y_r)$ an Bob.
- Dieser entschlüsselt $z_i := y_i^d\ mod\ N$, für $i = 1, ..., r$, und erhält $(z_1, ..., z_r) = (x_1, ..., x_r)$.

Asymmetrische Kryptoschemen, Sicherheitsbegriff

Wir betrachten die Situation asymmetrisch verschlüsselter Kommunikation allgemein, also für beliebig lange Klartexte, und formulieren ein Sicherheitskonzept. Der Einfachheit halber nehmen wir als Menge Y der Chiffretexte die Menge aller Binärstrings an.
Szenarium 4: Alice möchte Bob vertrauliche Nachrichten beliebiger Länge über einen abhörbaren Übertragungsweg zukommen lassen. Alice und Bob besitzen keinen gemeinsamen Schlüssel.

Asymmetrische Kryptoschemen

Definition 5.3: Ein asymmetrisches Kryptoschema ist ein Tupel $S = (X, K, G, E, D)$, wobei

- $X, K \supseteq K_{pub} \times K_{priv}$ Mengen,
- $G() : K_{pub} \times K_{priv}$ ein randomisierter Algorithmus,
- $E(x : X, k : K_{pub}) : \{0,1\}^*$ ein randomisierter Algorithmus und
- $D(y : \{0,1\}^*, k : K_{priv}) : \{0,1\}^*$ ein deterministischer Algorithmus sind,
- so dass gelten
- die (erwartete) Laufzeit von ? ist beschränkt durch eine Konstante,
- die Laufzeiten von ? und D sind polynomiell beschränkt in der Länge von x bzw. y,
- für jedes $x \in X, k \in K_{pub}$, jede Ausgabe $(k, \hat{k})$ von ? und jedes $m \in \{0,1\}^{p(|x|)}$ (die Ausgänge der flip-Anweisungen in E, für ein Polynom $p(n)$) gilt: $D(E^m(x,k), \hat{k}) = x$.

Begriffe

- Die Elemente von X heißen ,,Klartexte", die Menge der ,,Chiffretexte" ist $\{0,1\}^*$.
- Die Elemente von K_{pub} heißen ,,Öffentliche Schlüssel", die von K_{priv} ,,private Schlüssel", mit $K \supseteq K_{pub} \times K_{priv}$ bezeichnen wir die Menge der Schlüsselpaare, die G möglicherweise ausgibt.
- G ist der ,,Schlüsselgenerierungsalgorithmus". Er hat im Prinzip kein Argument, eventuell gibt es verschiedene Varianten, die von einem ,,Sicherheitsparameter" l abhängen.
- E ist der ,,Verschlüsselungs-" und D der ,,Entschlüsselungsalgorithmus".

Beispiel: Die Erweiterung des RSA-Schemas auf Strings beliebiger Länge durch Anwendung auf Blöcke der Länge $w \leq log\ N$, wie oben beschrieben, liefert ein Kryptoschema. Die Frage ist, wie gut dieses Kryptoschema ist.

Sicherheit von asymmetrischen Kryptoschemen

Definition 5.4: Ein Angreifer A auf ein asymmetrisches Kryptoschema $S = (X, K, G, E, D)$ ist ein Paar zufallsgesteuerter Algorithmen
$AF(k : K_{pub}) : (\{0,1\} \times \{0,1\}) + \times V$,
$AG(v : V, k : K_{pub}, y : \{0,1\}^*) : \{0,1\}$
Die Idee ist wie folgt: Der Finder AF bekommt einen öffentlichen Schlüssel k. Daraus berechnet er zwei verschiedene Klartexte (z_0, z_1) gleicher Länge und ,,Notizen" $v \in V$. Danach wird zufällig z_0 oder z_1 zu y verschlüsselt. Im zweiten Schritt verwendet der Rater AG die Zwischeninformation v, den öffentlichen Schlüssel k und die ,,Probe" y, um zu bestimmen, ob z_0 oder z_1 verschlüsselt wurde.
Definition 5.5: Sei $S = (X, K, G, E, D)$ ein asymmetrisches Kryptoschema und $A = (AF, AG)$ ein Angreifer. Das zugehörige Experiment oder Spiel ist der folgende Algorithmus $G_A^S : \{0,1\}$:

1. $(k, \hat{k}) \leftarrow G()$ (ein Schlüsselpaar des Kryptoschemas S wird gewählt)
2. $(z_0, z_1, v) \leftarrow AF(k)$ (der Finder berechnet ein Paar von Klartexten gleicher Länge, von denen er annimmt, ihre Chiffretexte unterscheiden zu können)
3. $b \leftarrow flip()$ und $y \leftarrow E(z_b, k)$ (einer der Klartexte wird zuyverschlüsselt)
4. $b' \leftarrow AG(v, k, y)$ (der Rater versucht herauszubekommen, ob z_0 oder z_1 verschlüsselt wurde)
5. falls $b = b'$, so gib 1 zurück, sonst 0.

- Das verkürzte Experiment oder Spiel S_A^S gibt im 5. Schritt einfach b' aus.

Dann ist $Pr(G_A^S = 1)$ die Wahrscheinlichkeit dafür, dass der Angreifer A den korrekten Klartext erkennt. Man kann jetzt wie in Abschnitt 2.4 (Sicherheit von l-Blockkryptosystemen) den Vorteil
$adv(A, S) = 2Pr(G_A^S = 1) - 1$, den ,,Erfolg"
$suc(A, S) = Pr(G_A^S \langle b = 1 \rangle = 1)$ und den ,,Misserfolg"
$fail(A, S) = Pr(G_A^S \langle b = 0 \rangle = 1)$ definieren. Lemma 2.16 gilt dann wörtlich.
Beispiel 5.6: Sei $S = (X, K, G, E, D)$ ein asymmetrisches Kryptoschema, in dem der Verschlüsselungsalgorithmus E deterministisch ist. Wir betrachten den folgenden Angreifer A mit $V = \{0,1\}^*$. Seien z_0 und z_1 verschiedene Elemente von X:

- $AF(k : K_{pub}) : (\{0,1\} \times \{0,1\})^+ \times V$
- $v \leftarrow E(z_0, k); return(z_0, z_1, v)$
- $AG(v : V, k : K_{pub}, y : \{0,1\}^*) : \{0,1\}$
- if $v = y$ then return 0 else return 1.
- Im Ablauf des Spiels G_A^S wird der Rater AG also mit $E(z_0, k)$ oder mit $E(z_1, k)$ gestartet. Wegen $E(z_0, k) \neq E(z_1, k)$ gilt $Pr(G_A^S = 1) = 1$, d.h. $adv(A, S) = 1$.

Dieses Beispiel lässt sich verallgemeinern:
Lemma 5.7: Für jedes deterministische asymmetrische Kryptoschema S gibt es einen Angreifer A mit $adv(A, S) = 1$.
Definition 5.8: Sei $t \in \mathbb{N}$, A ein Angreifer auf ein asymmetrisches Kryptoschema S. Dann heißt A t-beschränkt, wenn die Laufzeit des Experiments G_A^S durch t beschränkt ist. Sei $\epsilon > 0$. Dann heißt $S(t, \epsilon)$-sicher, wenn für jeden t-beschränkten Angreifer A gilt $adv(A, S) \leq \epsilon$.
Nach obigem Lemma gibt es für jedes deterministische asymmetrische Kryptoschema S eine kleine Konstante t, so dass S für kein $\epsilon > 0 (t, \epsilon)$-sicher ist. Da die Verschlüsselung von RSA deterministisch ist, ist dieses Verfahren nach Lemma 5.7 nicht sicher.
Zur Übung entwerfe man einen Angreifer A gegen das RSA-System für die Situation, wo die Klartexte (x_1, x_2) nur aus zwei Blöcken in $[N]$ bestehen. Dabei soll A das Spiel mit Wahrscheinlichkeit 1 gewinnen und die Notizenmenge V soll trivial sein, also $V = \{0\}$, so dass A sich gar keine Notizen machen kann.
Betrachten wir nun konkret das RSA-System mit nur einem Block, also $X = [N]$. Kann Eva aus Kenntnis von $y = x^e\ mod\ N$, e und N irgendeine konkrete Information über x ermitteln?
Definition 5.9:

1. Das ,,Legendre-Symbol" gibt an, ob $a \in \mathbb{Z}$ modulo einer Primzahl p ein Quadrat ist. Für $p > 2$ prim und $a \in \mathbb{Z}$ setze
$L_p(a) = \begin{cases} 0 & \text{falls } p|a \\ 1 & \text{falls } p \not| a, \exists b : b^2 \equiv a (mod\ p) \\ -1 & \text{falls } p \not| a, \neg\exists b : b^2 \equiv a (mod\ p) \end{cases}$.
2. Das ,,Jacobi-Symbol" verallgemeinert dies auf Moduli N, die nicht notwendig Primzahlen sind. Für $N > 2$ ungerade mit Primzahlzerlegung $N = \prod_{1 \leq i \leq r} p_i^{alpha_i}$ und $a \in \mathbb{Z}$ setze
$J_N(a) = \prod_{1 \leq i \leq r} L_{p_i}(a)^{\alpha_i}$

Beobachtungen und Beispiele:

- Weil $4^2\ mod\ 7 = 2$, gilt $L_7(2) = 1$. Weiter gilt $L_7(21) = 0$ und $L_7(5) = -1$, weil $1, 2, 4$ die einzigen Quadrate modulo 7 sind.
- $J_N(a) = 0$ genau dann, wenn $ggT(a, N) > 1$.
- Wenn $p \not| a$ und p ist ein Primfaktor, der in N mit geradem Exponenten vorkommt, dann ist der Beitrag von p zu $J_N(a)$ der Faktor 1, spielt also keine Rolle.
- Wenn $ggT(a, N) = 1$, dann ist
$J_N(a) = (-1)^{\#}\{i \leq r | \text{a ist Nichtquadrat modulo } p_i\}$.

Das Legendre-Symbol ist leicht mit schneller modularer Exponentiation zu berechnen.
Fakt 5.10 Euler-Kriterium: $L_p(a) = a^{(p-1)/2} mod\ p$ für alle $a \in \mathbb{Z}$ und Primzahlen $p > 2$.
Überraschenderweise lässt sich auch das Jacobi-Symbol $J_N(a)$ effizient berechnen, selbst wenn man die Primzahlzerlegung von N nicht kennt.
Fakt 5.11: Für $a \geq 1$ und ungerade $N \geq 3$ lässt sich $J_N(a)$ in Zeit $O((log\ N + log\ a)^3)$ berechnen (ohne N zu faktorisieren!).
(Man benutzt einen klassischen Satz aus der Zahlentheorie, nämlich das quadratische Reziprozitätsgesetz, das auf Gauß zurückgeht. Damit erhält man eine Berechnung, deren Rekursionsschema dem des Euklidischen Algorithmus ähnelt. Details: Buch von Küsters/Wilke.)
Lemma 5.12: Seien $p, q > 2prim, N = pq, e \in \mathbb{Z}^*_{\varphi(N)}$. Dann gilt $J_N(x^e\ mod\ N) = J_N(x)$ für alle $x \in \mathbb{Z}$.

Beweis: $L_p(x^e\ mod\ N) \equiv_p (x^e\ mod\ N)^{\frac{p-1}{2}} \equiv_p x^{e\frac{p-1}{2}} \equiv_p$
$(x^{\frac{p-1}{2}}\ mod\ p)^e \equiv_p (L_p(x)^e = L_p(x)$ denn e ist ungerade und damit $L_p(x^e\ mod\ N) = L_p(x)$. Analog gilt $L_q(x^e\ mod\ N) = L_q(x)$. Also: $J_N(x^e\ mod\ N) = L_p(x^e\ mod\ N) * L_q(x^e\ mod\ N) = L_p(x) * L_q(x) = J_N(x)$.
Das heißt: $J_N(E(x, (N, e))) = J_N(x)$. Der Chiffretext zu x erbt eine nichttriviale Eigenschaft von x, nämlich den Wert des Jacobi-Symbols. Da sich das Jacobi-Symbol effizient berechnen lässt, erhält man sofort einen effizienten Angreifer. Wie? Der Finder berechnet zwei Klartexte z_0 und z_1 mit $J_N(z_0) = 1$ und $J_N(z_1) = -1$. Aus dem Jacobisymbol $J_N(y)$ der Probe kann er schließen, welcher der beiden Klartexte verschlüsselt wurde. Es folgt erneut: RSA ist nicht sicher im Sinn des Tests in Abschnitt 5.2.2. Man könnte versuchen, diesem ,,kleinen" Problem zu entkommen, indem man nur Klartexte $x \in [N]$ mit $J_N(x) = 1$ zulässt. Allerdings wird dadurch die Menge der Bitstrings, die Klartexte sein dürfen, auf etwas unübersichtliche Weise eingeschränkt.

Weitere Bemerkungen zur Sicherheit und effizienten Verwendung von RSA

Der Einwand, dass man das Jacobisymbol des Klartextes ermitteln kann, ist vielleicht eine Schwäche von RSA, die man als praktisch geringfügig einschätzen kann. Viel schlimmer ist es, wenn man aus y und (e, N) den kompletten Klartext x ermitteln kann. Es ist kein Verfahren bekannt, das dies in der allgemeinen Situation effizient bewerkstelligt. Wir beschreiben hier (informal) Begründungen dafür, dass die naheliegendsten Angriffe vermutlich nicht effizient durchführbar sind, und Maßnahmen, die RSA zu einem (praktisch, vermutet) sicheren System machen.

Äquivalenz ,,d bekannt" zu ,,N faktorisieren"

Wenn Eva die Zahl N effizient in ihre Faktoren p und q zerlegen kann, kann sie natürlich sofort $\varphi(N)$ und d berechnen und damit sämtliche mit $k = (e, N)$ verschlüsselten Nachrichten lesen. Nach aktuellem Stand sind Faktorisierungsverfahren nicht effizient genug, um Zahlen mit einigen hundert Dezimalstellen zuverlässig effizient zu faktorisieren. Eine scheinbar schwächere Anforderung ist, dass Eva den Wert $\varphi(N)$ berechnen kann, denn dann kann sie auch d berechnen (mit dem erweiterten Euklidischen Algorithmus) und direkt entschlüsseln. Ganz allgemein ist der RSA-Schlüssel $(k, \hat{k}) = ((e, N), (d, N))$ vollständig gebrochen, wenn Eva d ermitteln kann, nur aus der Kenntnis von (e, N). Ist dies vielleicht ,,leichter" als die Faktorisierung von N? Man kann Folgendes zeigen: Aus N, e und d lassen sich mit einem randomisierten Verfahren effizient die Faktoren p und q berechnen. Das heißt: Das vollständige Brechen eines RSA-Schlüssels durch Ermittlung von d ist in etwa genauso schwierig wie die Faktorisierung von N.
Wir skizzieren die entsprechenden überlegungen.
Satz: Es gibt einen randomisierten Polynomialzeitalgorithmus, der aus e, d und N die Primfaktoren p und q von N berechnet.
Skizze des Vorgehens in diesem Algorithmus: Wir schreiben $ed - 1 = 2^k u$ für eine ungerade Zahl u und $k \geq 1$. (Beachte, dass e und d ungerade sein müssen, weil sie teilerfremd zu $\varphi(N) = (p-1)(q-1)$ sind, einer geraden Zahl.) Definiere: $ord_p(b) = min\{w | b^w\ mod\ p = 1\}$, für $b\ mod\ p \neq 0$, analog $ord_q(b)$, für $b\ mod\ q \neq 0$. Dies ist die ,,Ordnung" von $b\ mod\ p$ in der multiplikativen Gruppe $\mathbb{Z}_p^*$ bzw. $b\ mod\ q$ in $\mathbb{Z}_q^*$.
Lemma 1: $p \not| a \Rightarrow ord_p(a^u) \in \{2^0, 2^1, ..., 2^k\}$. Beweis des Lemmas: Wenn $p|a$, ist nichts zu zeigen. Sonst benutze, dass es ein s mit $s\varphi(N) = ed - 1$ gibt. Dann gilt $(a^u)^{2^k} \equiv a^{s(p-1)(q-1)} \equiv (a^{p-1})^{s(q-1)} \equiv 1 (mod\ p)$ nach dem kleinen Satz von Fermat. Also ist (das gilt immer in Gruppen) die Ordnung von a^u ein Teiler von 2^k. Dies beweist das Lemma.
Lemma 2: Wenn $p \not| a$ und $q \not| a$ und $ord_p(a^u) \neq ord_q(a^u)$, dann gibt es ein $i \leq k$ mit $ggT(N, (a^{u2^i} - 1) mod\ N) \in \{p, q\}$. Beweis des Lemmas:
O.B.d.A.: $ord_p(a^u) > ord_q(a^u) = 2^i$, mit $i < k$. Dann:
$a^{u2^i} = (a^u)^{ord_q(a^u)} \equiv 1 (mod\ q)$, also $q | a^{u2^i} - 1$, aber
$a^{u2^i} = (a^u)^{2^i} \not\equiv 1 (mod\ p)$, also $p \not| a^{u2^i} - 1$. Daraus folgt sofort
$ggT(N, (a^{u2^i} - 1) mod\ N) = ggT(N, (a^u)^{2^i} - 1) = q$. Das beweist das Lemma.
Algorithmus: Wähle a aus $\{1, ..., N-1\}$ zufällig. Prüfe, ob $ggT(N, a) > 1$. Falls ja, ist $ggT(a, N)$ ein Primfaktor von N. Sonst berechne $ggT(N, a^{u2^i} - 1\ mod\ N)$, für $i = 0, 1, ..., k$. Falls einer dieser Werte > 1 ist, ist er ein Primfaktor von N. Was noch fehlt: Satz $|\{a \in \{1, ..., N-1\} | ord_p(a^u) \neq ord_q(a^u)\}| \geq \frac{1}{2}\varphi(N)$. Der Satz, für dessen Beweis wir auf das Buch von Baumann, Franz, Pfitzmann verweisen, besagt, dass der Algorithmus mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ erfolgreich einen Primfaktor von N findet. Konsequenz aus diesen Überlegungen: Wenn d bekannt wird, darf man den Modulus N keinesfalls weiter verwenden. Im Wesentlichen ist also das vollständige Brechen des Systemsäquivalent dazu, N in seine Faktoren zu zerlegen. Das Faktorisierungsproblem gilt bislang als im Allgemeinen schwierig und für das Produkt N von zwei zufälligen Primzahlen mit 512 Bits (oder 1024 Bits) als praktisch nicht lösbar. Man beachte aber: Den geheimen Schlüssel d zu finden muss nicht die einzige Möglichkeit sein, aus N, e und y partielle Information über x zu gewinnen, wie man am Jacobi-Symbol sieht.

Bemerkungen zum Faktorisierungsproblem, allgemein

Nach wie vor gibt es keinen Polynomialzeitalgorithmus, der beliebige zusammengesetzte Zahlen N in ihre Primfaktoren zerlegen kann (äquivalent: in polynomieller Zeit einen nichttrivialen Faktor von N bestimmen kann). Allerdings gab es in den letzten Jahrzehnten auch gewaltige Fortschritte bei der Entwicklung immer besserer Verfahren.

- Pollards $(p-1)$-Methode (klassisch): Führt zu schneller Ermittlung eines Faktors von N, wenn für einen Primfaktor p von N gilt, dass $p-1$ nur kleine Primfaktoren hat.
- Pollards ρ-Methode (klassisch): Führt zur Ermittlung eines Faktors von N in Zeit $O(\sqrt{p})$, wo p ein Primfaktor von N ist. Da N immer einen Primfaktor in $O(\sqrt{n})$ hat, ist die Rechenzeit im schlechtesten Fall in $O(\sqrt[4]{n}) = O(2^{(log\ N)/4})$. Schnell zum Ziel kommt man also, wenn N einen kleinen Primfaktor p hat. (Die oben angegebenen Vorschriften zur Wahl von p und q vermeiden diese Situation.)
- Quadratisches Sieb (Pomerance, 1981): Rechenzeit $O(e^{sqrt(ln\ N)(ln\ ln\ N)})$.
- Faktorisierung mit Elliptischen Kurven (Hendrik W. Lenstra, 1987): Rechenzeit $O(e^{(1+o(1))\sqrt{(ln\ p)(ln\ ln\ p)}})$, wo p der kleinste Primteiler von N ist.
- Zahlkörpersieb (um 1990, zwei Varianten): Die schnellsten heute bekannten Faktorisierungsverfahren. Die anfallenden Rechnungen können auf vielen Rechnern parallel durchgeführt werden. Die Gesamtrechenzeit ist beschränkt durch $O(e^{C(ln\ N)^{\frac{1}{3}}(ln\ ln\ N)^{\frac{2}{3}}})$, für eine Konstante C. An der Verbesserung der Verfahren wird laufend gearbeitet.

Der Rechenaufwand für die Faktorisierung auch nur 260-stelliger Zahlen in Dezimaldarstellung ist heute noch so immens, dass diese Art von Angriff auf das RSA-System noch nicht als ernsthafte Bedrohung angesehen wird. Das BSI (Bundesamt für Sicherheit in der Informationstechnik) empfiehlt für RSA-Anwendungen Schlüssellängen von 2000 Bits oder etwa 600 Dezimalziffern (bzw. 3000 Bits oder 900 Dezimalziffern, wenn auch Entwicklungen der nächsten Jahre mit einkalkuliert werden sollen.) Wenn sich p und q nur geringfügig unterscheiden, lässt sich N effizient faktorisieren. Man sollte also darauf achten, dass p und q in nicht mehr als etwa den ersten 20 Binärstellen übereinstimmen. Bei zufälliger Wahl wird dies mit sehr hoher Wahrscheinlichkeit eintreten.
Es gibt ein RSA-spezifisches Angriffsverfahren („Iteration"), das effizient funktioniert, wenn $p-1$ und $q-1$ kleine Primfaktoren haben. Bei der Wahl von p und q ist also auch darauf zu achten, dass dies nicht der Fall ist (s. Buch von Buchmann).

Zufallskomponente

In der Praxis wird der Klartext $x \in \{0,1\}^w$ mit $w < \lfloor log\ N \rfloor$ durch Anhängen eines zufälligen Bitstrings r auf Länge $\lfloor log\ N \rfloor$ gebracht und das Wort $x \circ r$ (Konkatenation von x und r) wie beschrieben verschlüsselt. Die Zufallskomponentermacht den Verschlüsselungsalgorithmus E zu einem randomisierten Verfahren. Der Empfänger muss natürlich ebenfalls die Länge w kennen, um nach der Entschlüsselung aus $x \circ r$ die eigentliche Botschaft x zu ermitteln. Dieses Vorgehen hat auch den Vorteil, dass selbst bei mehrmaligem Versenden derselben Botschaft (oder Wiederholung von Blöcken) unterschiedliche Chiffretexte entstehen und zum Beispiel die Parität der Anzahl der 1-Bits in $x \circ r$ sowie das letzte Bit von $x \circ r$ rein zufällig sind und keine Information über x liefern. Auch das Problem mit dem Jacobisymbol wird damit entschärft, da $J_N(x \circ r)$ vermutlich einigermaßen zufällig sein wird. Zudem wird in realen Anwendungen noch eine „kryptographische Hashfunktion" h auf $x \circ r$ angewendet und der neue Klartext $x' = x \circ r \circ h(x \circ r)$ mit dem RSA-System verschlüsselt. Bob entschlüsselt zu $z \circ t \circ w$ und akzeptiert die Nachricht z nur, wenn $h(z \circ t) = w$ ist. Vorteil hiervon: Manipulationen an der Nachricht werden erkannt, die Nachrichten-Integrität ist gesichert. (Es bleibt das Problem sicherzustellen, dass die Nachricht tatsächlich von Alice gesendet worden ist.)

Effizienzverbesserungen

1. Wahl des Verschlüsselungsschlüssels e: Mitunter wird vorgeschlagen, $e = 3$ zu wählen, weil dann die Verschlüsselung besonders effizient vor sich geht. (Man benötigt nur zwei Multiplikationen modulo N.) Es gibt aber einen Angriff, der den Klartextxeffizient ermitteln kann, wenn drei Chiffretexte für x mit verschiedenenöffentlichen Schlüsseln $(N_1,3),(N_2,3),(N_3,3)$ vorliegen. (Dies kann passieren, wenn Alice dieselbe Nachricht ohne Zufallskomponente an drei verschiedene Empfänger mit verschiedenen RSA-Schlüsseln schickt.) Diesem Problem entgeht man, indem man etwa $e = 2^{16}+1$ wählt. Auch in diesem Fall ist die Verschlüsselung billiger als im allgemeinen Fall, da man mit 17 Multiplikationen modulo N auskommt.
2. Empfänger rechnet modulo p und q, um Rechenzeit zu sparen: Da Bob d kennt, können wir auch annehmen, dass er sogar die Faktoren p und q kennt. Dies kann zu einer Beschleunigung der Berechnung von $z = y^d\ mod\ N$ benutzt werden, wie folgt: Bob berechnet $z_p = y^d\ mod\ p$ und $z_q = y^d\ mod\ q$. Aus diesen beiden Zahlen bestimmt er mit dem Chinesischen Restsatz $z \in [N]$ mit $z \equiv z_p (mod\ p)$ und $z \equiv z_q (mod\ q)$. Dieses Vorgehen spart Rechenaufwand, da die Exponentiationen nur mit Zahlen der halben Länge durchgeführt werden müssen und die Anwendung des Chinesischen Restsatzes auf den erweiterten Euklidischen Algorithmus hinauslaufen, der eine deutlich kleinere Rechenzeit als eine Exponentiation hat. Eine Überschlagsrechnung ergibt, dass sich durch dieses Vorgehen der Rechenaufwand für die Entschlüsselung etwa um den Faktor 4 verringert. (Man beachte, dass dies die im vorherigen Abschnitt diskutierte Erweiterung des Klartextes um einen Zufallsstring und einen Hashwert kompensiert.)

Das Rabin-Kryptosystem

Es ist nicht bekannt, ob das Problem, RSA-verSchlüsselte Botschaften unberechtigterweise zu entschlüsseln, äquivalent zum Faktorisierungsproblem ist. Wir betrachten hier noch das Rabin-Verschlüsselungsverfahren, bei dem dies der Fall ist. Auch beim Rabin-Verfahren handelt es sich um ein Public-Key-Kryptosystem.
Schlüsselerzeugung (Bob oder Trust Center): Wähle zwei verschiedene zufällige große Primzahlen p und q mit $p \equiv q \equiv 3 (mod\ 4)$, also Primzahlen, die um 1 kleiner als ein Vielfaches von 4 sind. Berechne $N = pq$. Der öffentliche Schlüssel ist $k = N$; der geheime Schlüssel ist $\hat{k} = (p,q)$.
Verschlüsselung eines Blocks, der eine Zahl $x < N$ ist: $y := x^2\ mod\ N$.
Entschlüsselung eines Chiffretextes $y < N$: Wir müssen Quadratwurzeln von y modulo N berechnen, das sind Zahlen b mit $b^2 mod\ N = y$. Wir kennen die Faktoren p und q. Berechne Quadratwurzeln getrennt modulo p und modulo q: $r := y^{(p+1)/4} mod\ p$ und $s := y^{(q+1)/4} mod\ q$. Weil $r^2\ mod\ p = ((x^2\ mod\ N)^{(p+1)/4})^2\ mod\ p = x^{p+1}\ mod\ p = (x^p * x) mod\ p = x^2\ mod\ p$ (wir haben den kleinen Satz von Fermat in der Version „$x^p \equiv x (mod\ p)$ für alle x" benutzt), gilt $r^2 - x^2 \equiv (r-x)(r+x) \equiv 0 (mod\ p)$. Das heißt, dass entweder $r \equiv x (mod\ p)$ oder $p - r \equiv x (mod\ p)$ gilt. Genauso sieht man, dass $s \equiv x (mod\ q)$ oder $q - s \equiv x (mod\ q)$ gilt. Mit der konstruktiven Variante des chinesischen Restsatzes (Bemerkung nach Fakt 4.28) können wir nun vier Zahlen $z_1, ..., z_4 \in [N]$ berechnen, die die folgenden Kongruenzen erfullen:

- $z_1 \equiv r (mod\ p)$ und $z_1 \equiv s (mod\ q)$
- $z_2 \equiv r (mod\ p)$ und $z_2 \equiv q - s (mod\ q)$
- $z_3 \equiv p - r (mod\ p)$ und $z_3 \equiv s (mod\ q)$
- $z_4 \equiv p - r (mod\ p)$ und $z_4 \equiv q - s (mod\ q)$

Wegen der obigen Überlegung ist $x \in \{z_1, ..., z_4\}$. Wir wählen eine dieser vier Möglichkeiten. (Man kann Vorkehrungen treffen, dass „sinnvolle" Blöcke x leicht zu erkennen sind. Beispielsweise könnte man den Block x mit einer bestimmten Bitfolge wie 10000 abschließen. Es ist dann nicht anzunehmen, dass die Binärdarstellung einer der anderen Möglichkeiten zufällig ebenso endet.) Welcher Rechenaufwand ist nötig? Für die Verschlüsselung muss nur eine Quadrierung modulo N durchgeführt werden; sie kostet nur Zeit $O((log\ N)^2)$. Die Entschlüsselung erfordert eine Exponentiation modulo p und eine modulo q und mehrere Anwendungen des erweiterten Euklidischen Algorithmus - insgesamt Zeit $O((log\ N)^3)$.
Sicherheit: Wir nehmen an, Eva hätte ein effizientes Verfahren B, mit dem sie alle Chiffretexte zum öffentlichen Schlüssel N entschlüsseln kann. Wir zeigen, dass sie dann auch N faktorisieren kann. (Solange man annimmt, dass dies ein schwieriges Problem ist, kann auch die Annahme, dass Eva Verfahren B hat, als unwahrscheinlich gelten.) Eva geht so vor: Sie wählt eine Zahl x aus $[N]$ zufällig. Wenn $ggT(x,N) > 1$, ist sie fertig, denn dieser größte gemeinsame Teiler ist entweder p oder q. Andernfalls berechnet sie $y = x^2\ mod\ N$. Dann wendet sie ihr Entschlüsselungsverfahren an und berechnet ein $z = B(y)$ mit $z^2 \equiv y\ mod\ N$. Dieses z hängt wohlgemerkt nicht von x, sondern nur von y ab. Es gilt $x^2 \equiv z^2 (mod\ N)$. Wie oben gesehen gibt es vier Möglichkeiten:

1. $x \equiv z (mod\ p)$ und $x \equiv z (mod\ q)$
2. $x \equiv z (mod\ p)$ und $x \equiv -z (mod\ q)$
3. $x \equiv -z (mod\ p)$ und $x \equiv z (mod\ q)$
4. $x \equiv -z (mod\ p)$ und $x \equiv -z (mod\ q)$

Welche dieser Möglichkeiten die richtige ist, hängt vom Zufall ab, der die Auswahl von x steuert. Jede der 4 Quadratwurzeln von y hat dieselbe Wahrscheinlichkeit 1/4, als x gewählt worden zu sein.

1. Fall: $x = z$, Misserfolg.
2. Fall: $0 < |x-z| < N$ und $x - z$ durch p teilbar, woraus $ggT(x-z,N) = p$ folgt: Erfolg!
3. Fall: $0 < |x-z| < N$ und durch q teilbar, woraus $ggT(x-z,N) = q$ folgt: Erfolg!
4. Fall: $x + z = N$, also $x - z \equiv 2x (mod\ N)$. Weil $2x$ teilerfremd zu N ist, ergibt sich $ggT(x-z,N) = 1$, Misserfolg.

Eva muss also nur $ggT(x-z,N)$ berechnen! Damit gelingt es ihr mit Wahrscheinlichkeit 1/2, die Faktoren von N zu ermitteln. Durch l-fache Wiederholung desselben Experiments lässt sich die Erfolgswahrscheinlichkeit auf $1 - \frac{1}{2^l}$ erhöhen.

Diskrete Logarithmen und Anwendungen

Diskrete Logarithmen

Wir betrachten eine endliche zyklische Gruppe $(G, \circ, e)$ (multiplikativ geschrieben) mit einem erzeugenden Element g. Das bedeutet : $G = \{g^0 = e, g^1 = g, g^2, ..., g^{|G|-1}\}$. Sei $N = |G|$ die Größe („Ordnung") dieser Gruppe. Dann haben wir:

- Die Exponentiationsabbildung $exp_g : \{0,1,...,N-1\} \to G, a \to g^a$, ist eine Bijektion (sogar ein Gruppenisomorphismus zwischen $(\mathbb{Z}_N, +, 0)$ und $(G, \circ, e)$).
- Die Umkehrfunktion $log_g : G \to \{0,1,...,N-1\}, g^a \to a$, heißt der diskrete Logarithmus zur Basis g.
- Das DL-Problem für G und g bezeichnet die Aufgabe, zu gegebenem $h \in G$ den Exponenten $a = log_g(h) \in \{0,1,...,N-1\}$ zu berechnen, also den Exponenten a mit $g^a = h$.

Für alle kryptographischen Verfahren, die mit zyklischen Gruppen arbeiten, müssen die Gruppenelemente eine explizite Darstellung haben, auf denen die Gruppenoperationen $\circ$ und $^{-1}$ effizient ausführbar sind. Dann ist auch die Exponentiation $a \to h^a$ effizient ausführbar (mit $\leq 2log\ a$ Gruppenoperationen), mittels einer Variante der schnellen modularen Exponentiation. Das DL-Problem darf jedoch keinen (bekannten) effizienten Algorithmus haben. Als Gruppen G kommen u.a. in Frage:

- Für Primzahlen p: Die multiplikative Gruppe $\mathbb{Z}_p^*$ mit Grundmenge $\{1,...,p-1\}$ und Multiplikation modulo p als Operation. Die Ordnung (d.h. die Größe) dieser Gruppe ist bekanntermaßen $N = p-1$. Siehe die folgende Tabelle. Es ist ein recht einfacher Fakt aus der Zahlentheorie, dass diese Gruppe für jede Primzahlpzyklisch ist, also ein erzeugendes Element g hat. Die Gruppenoperationen sind effizient ausführbar: Die Gruppenoperation $\circ$ ist die Multiplikation modulo p; das neutrale Element ist die 1; zu gegebenem $h \in G$ kann man h^{-1} mit dem erweiterten Euklidischen Algorithmus berechnen. Für p mit 2048 Bits oder mehr, wobei $p-1$ einen „großen" Primteiler enthält, gilt das DL-Problem als praktisch nicht lösbar (für „allgemeine" a).

- Die multiplikative Gruppe eines beliebigen endlichen Körpers, z. B. $GF(2^k)$. Es muss nur sichergestellt sein, dass die Multiplikation im Körper effizient ausführbar ist. Hierfür benötigt man ein irreduzibles Polynom von Grad k und man muss Implementierungen von Polynommultiplikation und -Division haben. Für das Bilden von Inversen kann man den Potenzierungstrick benutzen oder den erweiterten Euklidischen Algorithmus für Polynome. Schließlich muss ein erzeugendes Element der multiplikativen Gruppe $GF(2^k)^*$ bekannt sein. (Die Kardinalität der Gruppe sollte mindestens 2^{2000} sein.)
- Zyklische Untergruppen von ,,elliptischen Kurven" über endlichen Körpern. (Hier genügen Körperelemente mit einer Bitlänge von 256 Bits.)

Tabelle: Die multiplikative Gruppe $\mathbb{Z}_{11}^*$ für $p = 11$ mit erzeugendem Element $g = 2$. Die erzeugenden Elemente sind mit $*$ markiert. Sie entsprechen den Potenzen g^a mit $ggT(a, 10) = 1$.

— a —	0	1	2	3	4	5	6	7	8	9
— 2^a —	1	2	4	8	5	10	9	7	3	6
		*	*				*		*	

Für kryptographische Anwendungen ungeeignetist dagegen die bekannteste zyklische Gruppe $(\mathbb{Z}_N, +, 0)$. Hier ist die Gruppenoperation die Addition modulo N; die Potenz g^a für $a \in \mathbb{Z}$ entspricht dem Element $a * g \ mod \ N$. Die erzeugenden Elemente sind gerade die Elemente von $\mathbb{Z}_N^*$. Wenn $g \in \mathbb{Z}_N^*$ ist, dann besteht das DL-Problem für g in folgendem: Gegeben $h = g^a$ (in der Gruppe gerechnet, das ist also $h = a * g \ mod \ N$), finde a. Dies ist mit dem erweiterten Euklidischen Algorithmus leicht möglich: finde $g - 1 \ mod \ N$, berechne $h * g^{-1} \ mod \ N$. Das DL-Problem in dieser Gruppe ist also effizient lösbar. Wenn eine zyklische Gruppe G mit effizienten Operationen gefunden worden ist, muss man immer noch ein erzeugendes Element finden. Das ist unter Umständen nicht ganz einfach. Wir wissen:

- Eine zyklische Gruppe $G = \langle g \rangle$ mit $|G| = N$ hat genau $\varphi(N) = |\mathbb{Z}_N^*|$ viele erzeugende Elemente, nämlich die Elemente g^a mit $a \in \mathbb{Z}_N^*$.

Das bedeutet, dass ein Anteil von $\varphi(N)/N$ der Elemente von G erzeugende Elemente sind. Aus Kapitel 4 wissen wir, dass $\varphi(N)/N = \prod_{p \ ist \ prim, p|N}(1 - \frac{1}{p})$ gilt. Man kann zeigen, dass dies $\Omega(1/log \ log \ N)$ ist. Wir können also analog zum Vorgehen bei der Primzahlerzeugung ein erzeugendes Element finden, wenn wir folgende Operationen zur Verfügung haben:

- zufälliges Wählen eines Elements von G
- Test, ob $h \in G$ ein erzeugendes Element ist oder nicht

Jedoch setzen alle bekannten effizienten Verfahren für den Test ,,Ist h erzeugendes Element?" voraus, dass man die Primfaktoren von $N = |G|$ kennt. Für den Fall $G = \mathbb{Z}_p^*$ ist die Situation so: Für große zufällige Primzahlen p (512, 1024 oder 2048 Bits) ist die Primfaktorzerlegung von $N = p - 1$ normalerweise nicht leicht zu ermitteln. Ein Ausweg ist, gezielt nach Primzahlen p zu suchen, für die $p - 1$ eine übersichtliche Primfaktorzerlegung hat. Besonders angenehm ist die Situation, wenn $p = 2q + 1$ für eine Primzahl q ist. In dieser Situation gibt es genau $q - 1 = (p - 3)/2$ erzeugende Elemente und man kann ein $x \in G$ durch schnelle Exponentiation darauf testen, ob es ein erzeugendes Element ist. Man erhält direkt ein randomisiertes Verfahren, das nach durchschnittlich zwei Tests ein erzeugendes Element gefunden hat. Alle im Folgenden beschriebenen kryptographischen Verfahren werden unsicher, wenn man in der verwendeten Gruppe das DL-Problem effizient lösen kann. Bei der Verwendung solcher Verfahren muss man also unbedingt darauf achten, keine Gruppen zu benutzen, bei denen praktikable Verfahren für das DL-Problem bekannt sind. Für das Folgende nehmen wir an, dass die verwendete Gruppe diesen Mindestanforderungen genügt.

Diffie-Hellman-Schlüsselaustausch

Sei G eine zyklische Gruppe mit Erzeuger g und sei (X, G, Y, e, d) ein symmetrisches Kryptosystem wie in Kapiteln 1 und 2 oder (G, E, D) ein symmetrisches Kryptoschema wie in Kapitel 3. (Das heißt, dass die Elemente von G die Schlüssel sind.) Nehmen wir an, Alice und Bob möchten per Kommunikationüber einen öffentlich (insbesondere von Eva) einsehbaren Kanal einen gemeinsamen, geheimen, zufälligen Schlüssel $k \in G$ festlegen. Das Diffie-Hellman-Schlüsselaustausch-Protokoll macht dies möglich! Die Idee dabei ist, dass $k = g^{ab}$ ist, wo nur Alice a kennt und nur Bob b. Über den öffentlichen Kanal laufen die Gruppenelemente g^a und g^b. Eva hat also das Problem, aus g^a und g^b den Wert g^{ab} zu berechnen. (Das ist das sogenannte DH-Problem, benannt nach W. Diffie und M. E. Hellman, die das Schlüsselaustauschverfahren 1976 vorgeschlagen haben.) Wenn Eva das DL-Problem lösen könnte, wäre dies leicht. Andere Möglichkeiten, das DH-Problem effizient zu lösen, ohne einen effizienten Algorithmus für das DL-Problem, sind prinzipiell denkbar, aber nicht bekannt.
Protokoll ,,Diffie-Hellman-Schlüsselaustausch"

- Voraussetzung: Alice and Bob kennen $G, |G|$ und g.

1. Alice wählt $a \in \{2, ..., |G| - 2\}$ zufällig, und sendet $A = g^a$ an Bob.
2. Bob wählt $b \in \{2, ..., |G| - 2\}$ zufällig, und sendet $B = g^b$ an Alice.
3. Alice berechnet $B^a = (g^b)^a = g^{ab} = k$.
4. Bob berechnet $A^b = (g^a)^b = g^{ab} = k$.

Dabei wird benutzt, dass die Multiplikation der Exponenten a und b kommutativ ist. Alice und Bob kennen nun k; Eva kennt nur A und B, sowie (nach dem Kerckhoffs-Prinzip) G und $|G|$.
Achtung! Das DH-Protokoll in der hier vorgestellten Form wird völlig unsicher, wenn Eva den Kommunikationskanal nicht nur abhört, sondern aktiv eingreifen kann.
,,Man-in-the-middle-attack": Eva fängt alle Nachrichten von Alice an Bob und umgekehrt ab und modifiziert sie.
Schlüsselvereinbarung: Alice sendet $A = g^a$. Eva fängt A ab, wählt selbst eine Zufallszahl a' und schickt $A' = g^{a'}$ an Bob. Dieser nimmt an, A' käme von Alice. Sein $B = g^b$ wird ebenfalls von Eva abgefangen und durch $B' = g^{b'}$ ersetzt, für ein von Eva gewähltes b'. Alice nimmt an, B' käme von Bob.
Senden eines Chiffretextes: Wenn Alice Nachricht x an Bob schicken möchte, verwendet sie Schlüssel $k' = (B')^a = g^{ab'}$ und sendet $y' = E(x, k')$. Eva fängt dies ab und entschlüsselt mittels $k' = A^{b'}$. Sie kennt nun x, berechnet $k'' = B^{a'}$ und schickt $E(x, k'')$ an Bob. Dieser entschlüsselt mittels $k'' = (A')^b$, und erhält wieder x.
Um das DH-Protokoll gegen solche Angriffe abzusichern, muss man andere Protokolle (sog. Authentifizierungsprotokolle) benutzen, die es ermöglichen herauszufinden, ob der Absender einer Nachricht tatsächlich der gewünschte Partner (Alice bzw. Bob) ist.

Das ElGamal-Kryptosystem

Das ElGamal-Kryptosystem steht in engem Zusammenhang mit dem DH-Schlüsselaustausch. Wie RSA ist auch dieses System ein Public-Key-Kryptosystem. Seine Sicherheit beruht nicht auf der (vermuteten) Schwierigkeit des Faktorisierungsproblems wie RSA, sondern auf der (vermuteten) Schwierigkeit des DL-Problems bzw. des DH-Problems.
Erzeugung des Schlüssels (Bob oder Trust-Center im Auftrag von Bob): Es wird eine zyklische Gruppe $(G, \circ, e)$ mit einem erzeugenden Element g benötigt, sowie $N = |G|$, so dass das zugehörige DH-Problem schwer zu lösen ist. Ein Element b wird zufällig aus $\{2, ..., |G| - 2\}$ gewählt, und es wird mittels schneller Exponentiation $B = g^b$ berechnet. Der öffentliche Schlüssel ist $k_{pub} = (G, g, B)$, der geheime Schlüssel ist b bzw. $k_{priv} = (G, g, b)$.
Verschlüsselung: Wir nehmen an, dass die Menge der möglichen Botschaften (Binärstrings) eine Teilmenge von G ist. Um eine Botschaft $x \in G$ zu verschlüsseln, wählt Alice eine Zufallszahl a aus $\{2, ..., |G| - 2\}$ und berechnet $A = g^a$. Weiter berechnet sie $y := B^a \circ x$. Der Chiffretext ist (A, y).
Kommentar: Der Rechenaufwand liegt im Wesentlichen in den zwei schnellen Exponentiationen. RSA benötigt eine schnelle Exponentiation. Man erkennt die Komponenten A und B aus dem Diffie-Hellman-Schlüsselaustausch wieder. Der Wert B^a ist $k = g^{ab}$, der Wert y ist $k \circ x$.
Entschlüsselung: Bob kennt die Gruppe G und g, sowie A und y (von Alice) sowie seinen geheimen Schlüssel b. Er berechnet $A^b = (g^a)^b = k$. Dann berechnet er das Gruppenelement $z = k^{-1} \circ y$, mit Hilfe der effizienten Invertierung und Gruppenoperation in G.
Beobachtung: $z = x$. (Das ist klar: $z = k^{-1} \circ y = k^{-1} \circ (k \circ x) = x$)
Kommentar: Der Rechenaufwand der Entschlüsselung liegt im Wesentlichen in der schnellen Exponentiation und der Invertierung von k.
Sicherheit: Man kann sich überlegen, dass für eine Angreiferin Eva und eine Gruppe G mit erzeugendem Element g folgende Situationen äquivalent sind:

1. Eva kann alle mit dem ElGamal-Verfahren bzgl. G und g verschlüsselten Nachrichten effizient entschlüsseln, also aus B, A und y die Nachricht x berechnen, die zum Chiffretext (A, y) geführt hat.
2. Eva kann das DH-Problem für G lösen.

Wenn Eva diskrete Logarithmen bezüglich G und g berechnen kann, gelten natürlich 1. und 2. Wir beweisen die Äquivalenz.

- ,,1.⇒2.": Eva hat $B = g^b$ und $A = g^a$ vorliegen und möchte $k = g^{ab}$ bestimmen. Sie wendet ihr Entschlüsselungsverfahren auf B, A und $y = 1$ an. Es ergibt sich ein Wert x mit $g^{ab} \circ x = k \circ x = y = 1$. Es gilt also $x = k^{-1}$, und Eva kann k durch Invertierung von x in G berechnen.
- ,,2.⇒1.": Eva hat $B = g^b$,$A = g^a$,$y = g^{ab} \circ x$ vorliegen. Weil sie das DH-Problem lösen kann, kann sie $k = g^{ab}$ berechnen und damit natürlich $x = k^{-1} \circ y$ bestimmen.

Unterschiede zwischen RSA und ElGamal:

- RSA in der puren Form benötigt einen Chiffretext $y = x^e \ mod \ N$, der die gleiche Bitlänge hat wie der Klartext x. ElGamal hat einen doppelt so langen Chiffretext (B, y).
- ElGamal ist erzwungenermaßen randomisiert. Daher führt die wiederholte Verschlüsselung desselben Klartextes x stets zu unterschiedlichen Chiffretexten, weil der Schlüssel k zufällig ist.

Allerdings gibt es die Empfehlung, beim Arbeiten mit RSA den Klartext x durch das Anhängen eines nicht ganz kurzen Zufallsstrings zu randomisieren. Wenn dieser angehängte Zufallsstring die gleiche Länge wie x hat, ist der Chiffretext genauso lang wie bei ElGamal.

Berechnung diskreter Logarithmen

Wie schwierig ist das ,,DL-Problem"?
Geg.: Zyklische Gruppe $(G, \circ, e), |G| = N$, mit erzeugendem Element g.
Input: $h \in G$
Gesucht: a mit $0 \leq a < N$, das $g^a = h$ erfüllt.
Trivialer Algorithmus: ,,Enumeration" Für $a = 0, 1, 2, ...$ teste, ob $g^a = h$ gilt. Man benötigt bis zu N schnelle Exponentiationen. In kryptographischen Anwendungen wird a zufällig gewählt. Die Wahrscheinlichkeit, dass $a \leq 2^{log \ N-k}$ ist etwa 2^{-k}. Für $k = 80$ ist dies $2^{-80} < 10^{-24}$, also winzig. Wenn $N \geq 2^{200}$, ist mit überwältigender Wahrscheinlichkeit $a \geq 2^{120} > 10^{36}$, und an eine Ausführung von ,,Enumeration" ist nicht zu denken.
Babystep-Giantstep-Algorithmus von Shanks: Wähle $m = \lceil \sqrt{N} \rceil$. Der gesuchte Exponent a hat eine Zerlegung $a = bm + c$, für $0 \leq c < m$ und passendes $b \leq a/m < N/m \leq \sqrt{N}$.

- Es gilt: $h = g^a = g^{bm+c} = g^{bm} \circ g^c$, also $g^{-c} = h^{-1} \circ g^{bm}$.
- Gesucht: b und c.
- Algorithmus: Berechne alle Potenzen g^{bm}, $0 \leq b < N/m$, und speichere $h^{-1} \circ g^{bm}$ (als Schlüssel) mit Wert b in einer Hashtabelle T, Umfang $2N/m \leq 2\sqrt{N}$. Berechne g^{-c}, für $c = 0, 1, ..., m - 1$ und suche g^{-c} in T. Wenn gefunden, gilt $h^{-1} \circ g^{bm} = g^{-c}$, also $h = g^{bm+c}$.

- Rechenzeit: $O(\sqrt{N})$ (erwartet) für Tabellenaufbau und $O(\sqrt{N})$ (erwartet) für die Suche, zusammen $O(\sqrt{N})$.
- Platzbedarf: $O(\sqrt{N})$
- Wenn $N = 2^{200}$, ist $\sqrt{N} = 2^{100} > 10^{30}$. Selbst wenn man nur in dem unwahrscheinlichen Fall $a \leq 2^{160}$ erfolgreich sein möchte, muss man Tabellengröße $2^{80} > 10^{24}$ veranschlagen. Auch dies ist nicht durchführbar.

Pollards ρ-Algorithmus für DL Idee: Definiere eine Folge $(x_i, a_i, b_i), i = 0, 1, 2, ...$, in $G \times \mathbb{Z}_N \times \mathbb{Z}_N$, so dass $x_i = F(x_{i-1})$ gilt, für eine als zufällig geltende Funktion $F : G \to G$. Dann verhalten sich die ersten Komponenten x_i wie zufällige Elemente in G, solange noch keine Wiederholung in der Folge $(Z_i)_{i=0,1,2,...}$ aufgetreten ist. Nach dem Geburtstagsparadoxon weiß man, dass für eine genügend große Konstante K die Wahrscheinlichkeit, dass $K\sqrt{N}$ zufällig gewählte Elemente von G alle verschieden sind, sehr klein ist. Wir erwarten also nach $O(\sqrt{N})$ Schritten die Wiederholung eines Elements, also $i_0 < j_0 = O(\sqrt{N})$ mit $x_{i_0} = x_{j_0}$. Danach wiederholt sich die Folge: $x_{i_0+1} = F(x_{i_0}) = F(x_{j_0}) = x_{j_0+1}$, usw., also gilt $x_i = x_{i+kl}$ für alle $i \geq i_0$ und alle $k \geq 1$, wenn man $l = j_0 - i_0$ definiert. Man kann das Verhalten der Folge wie folgt zeichnen: $x_0, ..., x_{i0}$ als gerade Linie ohne Wiederholung, daran angehängt $x_{i0}, ..., x_{j0} = x_{i0}$ als Kreis. Dies gibt die Form ,,ρ", wie beim griechischen Buchstaben ,,rho". Wenn man es nun noch schafft, aus zwei Indizes $i < j$ mit $x_i = x_j$ den gesuchten Exponenten a mit $h = g^a$ auszurechnen, ist man fertig. Um den Algorithmus auszuführen, muss man scheinbar $x_0, x_1, ..., x_{j0-1}$ speichern, was wieder zu Speicherplatzbedarf $\Theta(\sqrt{N})$ führen würde. Es gibt nun einen Trick, mit dessen Hilfe man nur Platz $O(1)$ benötigt. Man beobachtet die Doppelfolge $((x_i, x_{2i}))_{i=0,1,2,...}$ und wartet, bis $x_i = x_{2i}$ gilt. Gibt es solche i? Ja, denn für jedes $i \geq i_0$, für das $2i - i = i$ durch $l = j_0 - i_0$ teilbar ist, ist $x_{2i} = x_{i+kl}$ für ein $k > 0$, also $x_{2i} = x_i$. Zwischen i_0 und j_0 liegt auf jeden Fall ein solches i. Nun müssen wir noch F definieren. Dazu betrachten wir erweiterte Elemente $(x, b, c) \in G \times \mathbb{Z}_N \times \mathbb{Z}_N$, die die Gleichung $x = g^b \circ h^c$ erfüllen. Das Starttripel ist $(g^{b0}, b_0, 0)$ für ein zufällig gewähltes b_0. Die Gruppe G muss in etwa drei gleich große disjunkte Teilmengen S_1, S_2, S_3 aufgeteilt sein, etwa über die letzten 10 Bits der Darstellung der Elemente. Dann definieren wir die Schrittfunktion wie folgt:

$$f(x,b,c) = \begin{cases} (h \circ x, b, c+1) & \text{falls } x \in S_1 \\ (x^2, 2b, 2c), & \text{falls } x \in S_2 \\ (g \circ x, b+1, c), & \text{falls } x \in S_3 \end{cases}$$

Beachte, dass mit (x, b, c) auch $f(x, b, c)$ die geforderte Gleichung erfüllt. Die Funktion F ist einfach die Einschränkung von f auf die erste Komponente. Nun berechnen wir in Runden $Y_i = (x_i, b_i, c_i)$ und $Z_i = (x_{2i}, b_{2i}, c_{2i})$, für $i = 0, 1, 2, ...$ (Es ist $Y_0 = Z_0 = (g^{b0}, b_0, 0)$ für b_0 zufällig und $Y_i = f(Y_{i-1})$ und $Z_{2i} = f(f(Z_{i-1}))$.)
Dies wird so lange durchgeführt, bis zum ersten Mal $x_i = x_{2i}$ gilt. Dann haben wir: $g^{b_{2i}} h^{c_{2i}} = g^{b_i} h^{c_i}$, also $g^{b_{2i}+ac_{2i}} = g^{b_i+ac_i}$. Weil g Ordnung N hat, folgt $b_{2i} + ac_{2i} \equiv b_i + ac_i (mod\ N)$, das heißt $a(c_{2i} - c_i) \equiv b_i - b_{2i} (mod\ N)$. Falls nun $ggT(c_{2i} - c_i, N) = 1$ ist, können wir mit $a = (b_i - b_{2i})(c_{2i} - c_i)^{-1}\ mod\ N$ den gesuchten Exponenten berechnen. Die Rechenzeit ist $O(\sqrt{N})$, wenn man unterstellt, dass die Abbildung $F : x_{i-1} \to x_i$ rein zufällig ist. (In der Praxis bestätigt sich diese Vorstellung.) Weitere Algorithmen für das DL-Problem:

- Pohlig-Hellman-Algorithmus. Dieser Algorithmus benötigt die Primfaktorzerlegung von $N = |G|$. Seine Rechenzeit ist $O(\sum_{1 \leq i \leq k} e_i(log|G| + \sqrt{p_i}) + (log\ |G|)^2)$, wenn $|G|$ die Primfaktorzerlegung $p_1^{e_1} ... p_k^{e_k}$ hat. Dieser Algorithmus ist also effizient, wenn $|G|$ nur eher kleine Primfaktoren hat. Wenn man also mit G arbeiten will, muss $N = |G|$ mindestens einen ,,großen" Primfaktor enthalten, damit nicht der Pohlig-Hellman-Algorithmus das DL-Problem effizient löst.
- Indexkalkül. Dieser Algorithmus ist nur für die multiplikative Gruppe $GF(q)^*$ in endlichem Körper $GF(q)$ anwendbar.
- Zahlkörpersieb. Ebenso nur für $GF(q)^*$ (mit ähnlichen subexponentiellen Rechenzeiten wie bei dem gleichnamigen Algorithmus bei der Faktorisierung).

Letzteres ist eine allgemeine Beobachtung: DL in $GF(q)^*$ scheint nicht viel schwieriger zu sein als das Faktorisierungsproblem für Zahlen in der Größenordnung von q.

Elliptische Kurven über endlichen Körpern

Elliptische Kurven (,,elliptic curves", ,,EC") sind mathematische Strukturen, die eine moderne, attraktive Methode zur Erzeugung endlicher zyklischer Gruppen zur Verwendung in der Kryptographie liefern. Der Ansatz wurde 1985 unabhängig von den amerikanischen Mathematikern Neal Koblitz (1948) und Victor S. Miller (1947) vorgeschlagen. Der große Vorteil des Verfahrens ist, dass die unter gewissen Umständen schnellen DL-Verfahren für $GF(q)^*$, nämlich Indexkalkül und Zahlkörpersieb, nicht anwendbar sind. Die benötigte Gruppengröße, um ,,Sicherheit" im praktischen Sinn zu garantieren, ist deutlich kleiner als die bei der zyklischen Gruppe $GF(q)^*$. Dies führt zu Gruppenelementen mit kleinerer Darstellungsgröße und daher effizienterer Verfahren für Verschlüsselung und Entschlüsselung. (Übliche Längen im Jahr 2016: Bestehende Verfahren benutzen 160 Bits, Planung bis 2030: 224 oder 256 Bits. Ein System, das $\mathbb{Z}_p^*$ für eine 256-Bit-Primzahl p benutzt, gilt als äußerst sicher. Andere endliche Körper kommen ebenfalls in Frage. Vorsicht bei $GF(2^k)^*$! Man benötigt andere Formeln als die unten diskutierten!)
Wir geben nur eine Beschreibung der Verfahren. Für mathematischen Hintergrund sei auf die Literatur verwiesen, z.B. A. Werner, Elliptische Kurven in der Kryptographie, Springer 2002, oder Diekert, Kufleitner, Rosenberger, Diskrete algebraische Methoden, de Gruyter 2013.
Als anschaulichen Hintergrund, nicht zur Anwendung in der Kryptographie, betrachten wir zunächst Elliptische Kurven in $\mathbb{R}^2$. Gegeben seien Koeffizienten A, B in $\mathbb{R}$, die die Ungleichung $4A^3 + 27B^3 \neq 0$ erfüllen. (Diese Bedingung hat zur Folge, dass die Funktion $x \to x^3 + Ax + B$ keine Mehrfachnullstellen hat, weder im Reellen noch im Komplexen. Dabei heißt $a \in C$ eine Mehrfachnullstelle, wenn man $x^3 + Ax + B = (x - b)(x - a)^2$ schreiben kann, für ein $b \in C$.)
Betrachte die Menge $\{(x, y) \in \mathbb{R}^2 | y^2 = x^3 + Ax + B\}$. Diese bildet eine ,,Kurve" in $\mathbb{R}^2$. Verschiedene Formen sind möglich. Die ,,Kurve" kann auch mehrere Komponenten haben. Man beobachtet allgemein: Die Punktmenge ist symmetrisch bezüglich der x-Achse; die ,,Gestalt" hängt von der Lage der Nullstellen von $x \to x^3 + Ax + B$ ab.
Veranschaulichung für $(A, B) \in \{(-1, -1), (-1, 0), (-1, 1)\}$ später. Da Doppelnullstellen ausgeschlossen sind, gibt es eine oder drei Nullstellen. Solche Kurven, eventuell ergänzt um einen Punkt O, nennt man Elliptische Kurven in $\mathbb{R}^2$.
Unsere eigentliche Konstruktion benutzt nicht $\mathbb{R}$, sondern endliche Körper $\mathbb{Z}_p$ für eine Primzahl $p > 3$. Wir rechnen ab hier in einem solchen Körper, für festes p; die Operationen $+$ und $*$ sind immer als Addition und Multiplikation modulo p zu interpretieren.
Definition 5.13: Sei $p > 3$ eine Primzahl, seien $A, B \in \mathbb{Z}_p$ mit $4A^3 + 27B^3 \neq 0$. Die elliptische Kurve $E_{A,B}$ besteht aus der Menge aller Lösungen $(x, y) \in \mathbb{Z}_p^2$ der Gleichung $y^2 = x^3 + Ax + B$ sowie einem zusätzlichen Punkt O (genannt ,,der unendliche Punkt"). Wenn man für $\mathbb{Z}_p$ die Repräsentanten $-\frac{p-1}{2}, ..., 0, ..., \frac{p-1}{2}$ benutzt, beobachtet man wiederum die Symmetrie entlang der x-Achse, sonst gibt es kein erkennbares Muster.
Beispiel: $\mathbb{Z}_7$. Setze $f(x) = x^3 + 3x + 3$ (in $\mathbb{Z}_7$) und betrachte $E_{3,3} = \{(x, y) \in \mathbb{Z}_7^2 | f(x) = y^2\} \cup \{O\}$. Damit (x, y) zu $E_{3,3}$ gehört, muss $f(x)$ ein Quadrat sein. Allgemein weiß man, dass es in $\mathbb{Z}_p$ genau $\frac{p-1}{2}$ Quadrate gibt. Jedes $f(x)$, das von 0 verschieden und ein Quadrat ist, führt zu zwei Punkten auf der elliptischen Kurve. Weiter gibt es für jedes x mit $f(x) = 0$ einen Punkt auf der Kurve. Man rechnet aus:

x	0	1	2	3	4	5	6
f(x)	3	0	3	4	2	3	6
Quadrat?	-	X	-	X	X	-	-
Wurzeln		0		2, 5	3, 4		
Punkte		(1,0)		(3,2)	(4,3)		
				(3,5)	(4,4)		

Um auf dieser Menge eine Gruppenstruktur festzulegen, benötigen wir ,,Geraden" in $\mathbb{Z}_p^2$. Dies sind Punktmengen $\{(x, y) | y = ax + b\}$ für $a, b \in \mathbb{Z}_p$ oder $\{(x, y) | x = b\} \cup \{O\}$ (,,senkrechte Geraden"). Man stellt nun (durch Unterscheiden verschiedener Fälle) Folgendes fest: Wenn eine Gerade eine elliptische Kurve $E = E_{A,B}$ in mindestens einem Punkt schneidet, dann sogar in drei Punkten, die aber nicht notwendigerweise verschieden sein müssen. (Das liegt an der Anzahl der Nullstellen von $x^3 + Ax + B$. Veranschaulichung im Reellen!) Wenn zwei der betrachteten Schnittpunkte zusammenfallen, ist die Gerade eine Tangente in diesem Punkt; bei senkrechten Geraden gilt der unendliche Punkt als einer der Schnittpunkte oder gar als drei zusammenfallende Schnittpunkte. Die Operation $\circ$ kann dann anschaulich wie folgt beschrieben werden (man verwendet wieder für einen Moment die Bilder im $\mathbb{R}^2$): Gegeben seien Punkte P und Q auf der elliptischen Kurve, beide von O verschieden. Man legt eine Gerade durch P und Q (wenn sie identisch sind, eine Parallele zur Kurve in P) und bestimmt den dritten Punkt R im Schnitt von Kurve und Gerade. Wenn $R = O$ ist, ist $P \circ Q = O$, wenn $R = (x, y) \neq O$, dann ist $P \circ Q = \overline{R} = (x, -y)$ (Spiegelung an der x-Achse). Weiter definiert man: $P \circ O = O \circ P = P$ für alle Punkte P.
Es ergeben sich (mit einigem Rechnen) die folgenden Formeln. Diese werden dann wörtlich auch als Definition für eine Operation $\circ$ in einer elliptischen Kurve über $\mathbb{Z}_p$ benutzt.

- $O + O = O$,
- $O + (x, y) = (x, y) + O$ für alle $(x, y) \in \mathbb{Z}_p^2$,
- $(x_1, y_1) + (x_2, y_2) = \begin{cases} O, & \text{falls } x_1 = x_2 \text{ und } y_1 = -y_2 \\ (x_3, y_3), & \text{sonst,} \end{cases}$

wobei (x_3, y_3) folgendermaßen berechnet wird: $x_3 = \lambda^2 - x_1 - x_2$, $y_3 = \lambda(x_1 - x_3) - y_1$ mit $\lambda = \begin{cases} (y_2 - y_1)/(x_2 - x_1), & \text{falls } (x_1, y_1) \neq (x_2, y_2) \\ (3x_1^2 + A)/(2y_1), & \text{falls } (x_1, y_1) = (x_2, y_2) \end{cases}$. Der erste Fall bezieht sich auf den dritten Schnittpunkt einer Geraden durch zwei Punkte; der zweite auf den Tangentenfall.
Satz 5.14: Mit dieser Operation $\circ$ bildet $E_{A,B}$ eine kommutative Gruppe.
Man beweist dies durch Nachrechnen. Der Nachweis der Assoziativität ist etwas mühselig. Das zu (x, y) inverse Element ist $(x, -y)$, das zu O inverse Element ist O.
Notation: Die Gruppenoperation in Gruppen zu elliptischen Kurven wird additiv geschrieben, also mit $+$ bezeichnet. Das Inverse von P heißt $-P$. Die wiederholte Verknüpfung eines Elementes mit sich selbst ist dann eine Multiplikation mit $a \in \mathbb{Z}$, etwa $2P, 3P, ...$ Beispiel: Wenn $P = (x, 0)$, dann gilt $P = -P$ und $2P = O$.
Wir benötigen zyklische Untergruppen von $E_{A,B}$. Damit eine solche Gruppe ein schwieriges DL-Problem hat, muss sie natürlich groß sein, also muss auch $E_{A,B}$ groß sein. Wenn die Funktion $f(x) = x^3 + Ax + B$ wie eine zufällige Funktion wirkt, wird sie für etwa die Hälfte der x einen Wert haben, der ein Quadrat ist, und alle diese Werte (außer der 0) führen zu zwei Punkten auf der Kurve. Nullstellen ergeben einen Punkt. Wir erwarten daher, dass $E_{A,B}$ etwa p Elemente hat, und Folgendes sollte nicht zu sehr überraschen.
Fakt 5.15 Hasse-Schranke: Sei E elliptische Kurve über $\mathbb{Z}_p$. Dann gilt $p + 1 - 2\sqrt{p} \leq |E| \leq p + 1 + 2\sqrt{p}$.
Es gibt einen ,,effizienten" Algorithmen zur Ermittlung der Gruppenordnung $N = |E|$ der höchstens $O((log\ p)^6)$ Gruppenoperationen benötigt. Wenn wir Glück haben, ist N eine Primzahl; dann ist jedes Element von $E - \{O\}$ ein erzeugendes Element. Ein Standardverfahren ist, die Wahl von A und B so lange wiederholen, bis $N = |E_{A,B}|$ eine Primzahl q ist. Dann wird ein Element P aus $E - \{O\}$ zufällig gewählt und (p, A, B, N, P) an den Kunden abgeliefert. Mit p und A kann man die Gruppenoperationen implementieren, mit P und N zusätzlich kann man den Diffie-Hellman-Schlüsselaustausch und

das ElGamal-Kryptoschema umsetzen. Standardverfahren für das DL-Problem (Pollards ρ-Algorithmus, Pohlig-Hellman) funktionieren auch für Gruppen, die zu elliptischen Kurven gehören, nicht aber die viel schnelleren Verfahren wie Indexkalkül oder Zahlkörpersieb. Dies führt dazu, dass man annimmt, dass in EC-basierten Gruppen das DL-Problem (noch) schwieriger zu lösen ist als in $\mathbb{Z}_p^*$, so dass man mit kleineren Zahlbereichen arbeiten kann, was Verschlüsselung und Entschlüsselung wieder effizienter macht.
Effizienter Einsatz: Wenn man versucht, das ElGamal-Kryptosystem auf der Basis einer elliptischen Kurve umzusetzen, gibt es das Problem, dass die Elemente von E in $\mathbb{Z}_p^2$ eher dünn sind, so dass die Menge E oder auch die Menge der x- Koordinaten von Punkten in E als Klartextmenge schlecht geeignet ist. Wie soll man also gewöhnliche Nachrichten auf Punkte auf der Kurve abbilden? Es gibt ein reales kryptographisches Verfahren, das zeigt, wie man diese Schwierigkeit umgeht: ,,Elliptic Curve Integrated Encryption Scheme (ECIES)". Es beruht darauf, nur für die Manipulationen auf der Schlüsselseite die Gruppe E zu benutzen, und die eigentliche Verschlüsselung in $\mathbb{Z}_p^*$ auszuführen. Das reale ECIES-Verfahren integriert noch ein symmetrisches Verschlüsselungsverfahren und ,,message authentication" (ein ganz anderes kryptographisches Elementarwerkzeug). Wir geben hier nur den Kern an, der ein asymmetrisches Kryptosystem darstellt. Es gibt Anklänge an das ElGamal-Kryptosystem, aber Unterschiede im Detail.
Mit eingebaut ist ein Verfahren, das Elemente von $E - \{O\}$ kompakt darstellt: Anstelle von $(x, y) \in \mathbb{Z}_p^2$ speichern wir x und ein Bit b. Im Allgemeinen gibt es keinen oder zwei Punkte auf $E_{A,B}$ mit erster Koordinate x (außer bei den Nullstellen von f). Wenn $f(x) = x^3 + Ax + B \neq 0$ ein Quadrat in $\mathbb{Z}_p$ ist, gibt es zwei passende Werte y_1 und y_2 mit $y_1 + y_2 = p$, von denen einer gerade und einer ungerade ist. Diese beiden Situationen werden durch b unterschieden.

$$Point - Compress : E - \{O\} \to \mathbb{Z}_p \times \{0,1\}, (x, y) \to (x, y \ mod \ 2)$$

Dies ist eine injektive Funktion. Die Umkehrfunktion Point-Decompress benötigt die Funktion ,,Quadratwurzel modulo p", die effizient berechnet werden kann, wenn $p+1$ durch 4 teilbar ist, siehe ,,Entschlüsselung beim Rabin-Kryptosystem". Daher verwendet man in der EC-Kryptographie vorzugsweise solche Primzahlen. Wenn man eine (und damit beide) Quadratwurzeln von $f(x)$ berechnet hat, wählt man als y die gerade/ungerade davon, je nachdem ob $b = 0$ oder $b = 1$ ist, und gibt (x, y) aus.
Simplified ECIES Gegeben (und allen Beteiligten bekannt): Elliptische Kurve $E = E_{A,B}$ über $\mathbb{Z}_p$ (also p, A, B), zyklische Untergruppe $G = \langle P \rangle$ mit erzeugendem Element P, Kardinalität $N = |G|$, wobei N eine Primzahl ist. Wir unterdrücken diese Angaben im Folgenden. (Sie sind aber Teil des öffentlichen Schlüssels und natürlich auch dem Empfänger Bob bekannt.)
Klartextmenge: $X = \mathbb{Z}_p^*$.
Chiffretextmenge: $Y = (\mathbb{Z}_p \times \{0,1\}) \times \mathbb{Z}_p^*$.

- (Paare aus: (komprimiertes) Element von G und Element von $\mathbb{Z}_p^*$).
- Öffentliche Schlüssel: $K_{pub} = G$. Private Schlüssel: $K_{priv} = \mathbb{Z}_N$.
- Schlüsselmenge:
  $K = \{(Q, b) | Q \in K_{pub} = G, b \in K_{priv} = \mathbb{Z}_N, Q = bP\}$.

Schlüsselerzeugung: (Gegeben sind E [also p, A, B], P, N.)

- Wähle $b \in \mathbb{Z}_N$ zufällig und berechne $Q = bP$ (schnelle Exponentiation).
- Schlüssel:(Q, b).
- Der öffentliche Schlüssel ist $k_{pub} = Q$.
- Der private Schlüssel ist $k_{priv} = b$.

Verschlüsselungsfunktion $E : X \times G \to Y$, als randomisierter Algorithmus.

- Gegeben: Klartext $x \in \mathbb{Z}_p^*$.
- Öffentlicher Schlüssel $Q \in G$.
- Wähle zufällig $a \in \mathbb{Z}_N$ und berechne $(k, y) = aQ$ mit $k \in \mathbb{Z}_p^*$. //(Falls $k = 0$, wähle neues a.)
- Berechne
  $E^a(x, Q) \leftarrow (Point - Compress(aP), x * k \ mod \ p) =: (y', y'')$;
  das Paar $(y', y'') \in (\mathbb{Z}_p \times \{0,1\}) \times \mathbb{Z}_p^*$ ist der Chiffretext.

Bemerkung: $k \in \mathbb{Z}_p^*$, die erste Komponente eines Punktes in G, wird durch eine Operation in G erstellt, und dann wie beim One-Time-Pad (oder beim Vernam-System) benutzt, wobei diese Verschlüsselung durch Multiplikation in $\mathbb{Z}_p^*$ ausgeführt wird.
Entschlüsselungsfunktion $D : Y \times \mathbb{Z}_N \to X$, als (deterministischer) Algorithmus.

- Gegeben: Chiffretext $y = (y', y'')$ mit $y' \in \mathbb{Z}_p \times \{0,1\}$ und $y'' \in \mathbb{Z}_p^*$. Privater Schlüssel b.
- Berechne $(x_1, y_1) \leftarrow Point - Decompress(y')$ //nun gilt $(x_1, y_1) = aP$
- $(x_0, y_0) \leftarrow b(x_1, y_1)$ (in G, nun gilt $(x_0, y_0) = (ba)P = a(bP) = aQ = (k, y)$), und schließlich in $\mathbb{Z}_p^*$
- $D((y', y''), b) \leftarrow y'' * (x_0)^{-1} \ mod \ p$.

Behauptung: Wenn $Q = bP$, dann gilt $D(E^a(x, Q), b) = x$, für jedes $x \in X$ und jedes a, für das $(k, y) = aQ \in \mathbb{Z}_p^* \times \mathbb{Z}_N$. (Dies gilt nach den Anmerkungen in der Beschreibung von E und D.)