Kryptographie (IN 2197)

Thomas Stibor

thomas.stibor@in.tum.de

Sicherheit in der Informatik
Fakultät für Informatik
Technische Universität München

Organisatorisches

Vorlesung:

- 3+1 SWS, ECTS-Studium, ECTS-Credits: 5,0

Zeit und Ort der Vorlesung und Übung:

- Mittwoch 14:00c.t. - 16:00 Uhr, MI Hörsaal 2 (Vorlesung)
- Freitag 14:00s.t. - 15:00 Uhr, MI Hörsaal 2 (Vorlesung)
- Freitag 15:00 - 16:00 Uhr, MI Hörsaal 2 (Übung)

Klausur:

- Dienstag 10:00 Uhr, 28.07.2009 (Ort wird noch bekannt gegeben)

Übung:

- Vier Aufgaben pro Übungsblatt
- Übungsaufgaben werden am Freitag in der Übungsstunde mit den Studenten durchgerechnet

Organisatorisches (Fort.)

Übung:

- Übungen werden nicht für die Klausur angerechnet
- Teilnahme und Mitarbeit in der Übung ist sehr wünschenswert und hilft sehr die Klausur erfolgreich zu bestehen

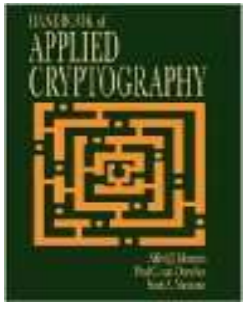
Sprechstunde:

Thomas Stibor

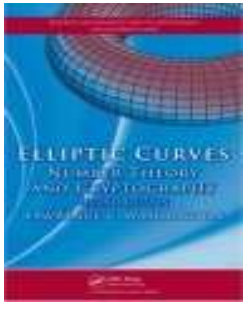
- Freitags, 16:00 - 17:00 Uhr, Raum: 01.08.055

Literatur

Einführung in die Kryptographie
Johannes Buchmann
Springer Verlag
4. erweitere Auflage, 2007



Handbook of Applied Cryptography
Alfred J. Menezes, Paul C.
van Oorschot and Scott A. Vanstone
CRC Press, 1996



Elliptic Curves:
Number Theory and Cryptography
Lawrence C. Washington
Chapman & Hall/CRC
2nd edition, 2003

Grundlage dieser Vorlesung ist das Buch: Einführung in die Kryptographie

Themenübersicht der Vorlesung

Kapitel 1: Organisatorisches und Einführung

Kapitel 2: Mathematische Grundlagen (1)

- Ganze Zahlen, Teilbarkeit, Darstellung, Euklidischer Algorithmus

Kapitel 3: Mathematische Grundlagen (2)

- Kongruenzen, Prime Restklassengruppen, Ordnung von Gruppenelementen, Satz von Fermat und Euler, Chinesischer Restsatz

Kapitel 4: Verschlüsselung

- Typen von Attacken, Blockchiffren, Stromchiffren, Verschlüsselungsmodi, Kryptoanalyse von affin linearen Blockchiffren

Themenübersicht der Vorlesung (Fort.)

Kapitel 5: DES-Algorithmus

- S-Boxen, Rundenschlüssel, Sicherheit von DES, Beispiel

Kapitel 6: AES-Algorithmus

- AES-Funktionen: Cipher, SubBytes, ShiftRows, KeyExpansion, Beispiel

Kapitel 7: Public-Key Verschlüsselung

- RSA, Rabin, ElGamal

Kapitel 8: Schlüsselaustausch und Authentisierung

- Diffie-Hellman, Fiat-Shamir

Kapitel 9: Kryptographische Hashfunktionen

- Kompressionsfunktionen, Geburtstagsattacke, SHA-1, Message Authentication Codes

Themenübersicht der Vorlesung (Fort.)

Kapitel 10: Kryptographische Signaturen

- RSA, ElGamal, Schnorr, DSA

Kapitel 11: Elliptische Kurven Kryptographie

- Grundlagen elliptische Kurven, Diffie-Hellman und ElGamal auf elliptischen Kurven

Einführung

Kryptographie (von griechisch *kryptos* "verborgen" und *graphein* "schreiben") ist die Wissenschaft der Verschlüsselung von Informationen.

- Häufig wird auch der Oberbegriff *Kryptologie* verwendet.
- Kryptologie umfasst die beiden Gebiete
  - Kryptographie: Entwicklung und Anwendung der einzelnen Verfahren.
  - Kryptoanalyse: Analyse der Stärken und Schwächen der Verfahren.

Grundsatz der modernen Kryptographie

Kerckhoffs' Prinzip:

> Sicherheit eines Kryptosystems darf nicht von der Geheimhaltung des Algorithmus abhängen.

- Sicherheit gründet auf Geheimhaltung frei wählbarer Eingangsgrößen des Algorithmus (z.B. Schlüssel).
- Negativ Beispiel: A5 Algorithmus in GSM ("security by obscurity").

Schutzziele

Kryptographie bietet mathematische Techniken zum Schutz von

- Vertraulichkeit
- Integrität
- Authentizität
- Verbindlichkeit
- Privatheit

Schutzziele (Fort.)

Vertraulichkeit:

- Schutz vor unautorisierter Informationsgewinnung

Kryptographisches Schutzkonzept:

- Symmetrische/asymmetrische Verschlüsselung

Integrität:

- Schutz vor unautorisierter und unbemerkter Daten-Modifikation

Kryptographisches Schutzkonzept:

- Berechnen von eindeutigen Prüfsummen von Objekten

Schutzziele (Fort.)

Authentizität:

- Nachweis der Echtheit und Glaubwürdigkeit der Identität eines Objekts/Subjekts

Kryptographisches Schutzkonzept:

- Challenge-Response Protokolle (z.B. Fiat-Shamir)

Verbindlichkeit:

- kein unzulässiges Abstreiten durchgeführter Handlungen
  Notwendig beispielsweise für:
  - Abschließen von elektronischen Kaufverträgen
  - Digital unterschriebene Mahnbriefe

Kryptographisches Schutzkonzept:

- Kryptographische Signaturen

Schutzziele (Fort.)

Privatheit:

- Schutz der personenbezogenen Daten, Schutz der Privatsphäre, Informationelles Selbstbestimmungsrecht

Kryptographisches Schutzkonzept:

- Verschlüsseln der Identitäten (Chaumsche Mixe)

Schutzziele im Kryptographischen Kontext

- Vertraulichkeit: Schlüsselaustausch (Diffie-Hellman, RSA, ...), Verschlüsselung (RSA, AES, DES, ...)
- Integrität: Hashfunktionen, SHA (Secure Hash Algorithm), ...
- Verbindlichkeit: Digitale Signatur, PKI, Zertifikate (RSA, DSS (Digital Signature Standard), ...)
- Authentizität: Message Authentication Code (MAC), Signaturen (HMAC-MD5, HMAC-SHA, ...)
- Privatheit: Verschlüsselung (AES, Triple-DES, ...)

Kryptographische Technologien werden in Protokollen/Diensten kombiniert z.B.

- Authentifizierte, vertrauliche und integere Kommunikation mit SSL: Zertifikate, RSA (oder Diffie-Hellman), MD5, SHA, Triple-DES

Mathematische Grundlagen

Kapitel 2 und 3 werden an der Tafel behandelt.

Verschlüsselungsverfahren

Ein Verschlüsselungsverfahren oder Kryptosystem ist ein Fünftupel $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ mit folgenden Eigenschaften:

1. $\mathcal{P}$ ist die Menge der Klartexte. Sie heißt Klartextraum.
2. $\mathcal{C}$ ist die Menge der Chiffretexte oder Schlüsseltexte. Sie heißt Chiffretextraum.
3. $\mathcal{K}$ ist die Menge der Schlüssel. Sie heißt Schlüsselraum.
4. $\mathcal{E} = \{E_k : k \in \mathcal{K}\}$ ist eine Familie von Funktionen $E_k : \mathcal{P} \to \mathcal{C}$. Ihre Elemente heißen Verschlüsselungsfunktionen.
5. $\mathcal{D} = \{D_k : k \in \mathcal{K}\}$ ist eine Familie von Funktionen $D_k : \mathcal{C} \to \mathcal{P}$. Ihre Elemente heißen Entschlüsselungsfunktionen.
6. Für jedes $e \in \mathcal{K}$ gibt es ein $d \in \mathcal{K}$, so daß für alle $p \in \mathcal{P}$ die Gleichung $D_d(E_e(p)) = p$ gilt.

Grundlagen Alphabete

Ein *Alphabet* ist eine endliche, nicht leere Menge Σ.

- *Länge* von Σ ist die Anzahl der Elemente in Σ.
- Elemente von Σ heißen *Zeichen*, *Buchstaben* oder *Symbole*.
- Als *Wort* oder *String* über Σ bezeichnet man eine endliche Folge von Zeichen aus Σ einschließlich des leeren Wortes ϵ.

Beispiel:

$$\begin{aligned} \Sigma &= \{0,1\}, \text{ Länge ist } 2. \\ \Sigma &= \{A,B,C,D,\ldots,X,Y,Z\}, \text{ Länge ist } 26. \end{aligned}$$

- Ist n eine nicht negative ganze Zahl, dann bezeichnet Σ^n die Menge aller Wörter der Länge n über Σ.

Beispiel: Verschiebungschiffre

Klartext-, Chiffretext- und Schlüsselraum ist $\Sigma = \{A, B, \ldots, Z\}$.

- Identifiziere Buchstaben $A, B, \ldots, Z$ als Zahlen $0, 1, \ldots, 25$, d.h. $A \leftrightarrow 0, B \leftrightarrow 1, \ldots, Z \leftrightarrow 25$.
- Für $e \in \mathbb{Z}_{26}$ ist Verschlüsselungsfunktion E_e definiert als

$$E_e : \Sigma \to \Sigma, x \mapsto (x + e) \mod 26.$$

- Entschlüsselungsfunktion ist für $d \in \mathbb{Z}_{26}$

$$D_d : \Sigma \to \Sigma, x \mapsto (x - d) \mod 26.$$

Beispiel: Schlüssel sei $e = d = 5$, das Wort "KRYPTOGRAPHIE" wird verschlüsselt zu Wort "PWDUYTLWFUMNJ".

Typen von Attacken

- *Ciphertext-Only-Attacke:* Der Angreifer kennt nur den Chiffretext. Dies ist der schwächste Angriff.
- *Known-Plaintext-Attacke:* Der Angreifer kennt andere Klartexte und die zugehörigen Chiffretexte.
- *Chosen-Plaintext-Attacke:* Der Angreifer kann Chiffretexte zu selbst gewählten Klartexten erzeugen.
- *Chosen-Ciphertext-Attacke:* Der Angreifer kann selbst gewählte Chiffretexte entschlüsseln ohne den Entschlüsselungsschlüssel zu kennen.

Grundlagen Permutationen

Sei X eine Menge. Eine Permutation von X ist eine bijektive Abbildung $f : X \to X$. Die Menge aller Permutationen von X wird mit $S(X)$ bezeichnet.

Beispiel: Sei $X = \{0, 1, 2, \ldots, 5\}$

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 4 & 3 & 5 & 0 \end{pmatrix}$$

Ist n eine natürliche Zahl dann bezeichnet man mit S_n die Gruppe der Permutationen der Menge $\{1, 2, \ldots, n\}$

Beispiel:

Die Gruppe S_2 hat die Elemente $\begin{pmatrix} 1 & 2 \\ 1 & 2 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$

Grundlagen Permutationen (Fort.)

Sei $X = \{0,1\}^n$ die Menge aller Bitstrings der Länge n. Eine Permutation von X in der nur die Positionen der Bits vertauscht werden, heißt *Bitpermutation*. Wählt man $\pi \in S_n$ so läßt sich Bitpermutation wie folgt beschreiben:

$$f : \{0,1\}^n \to \{0,1\}^n,\ b_1 b_2 \dots b_n \mapsto b_{\pi(1)} b_{\pi(2)} \dots b_{\pi(n)}$$

Die Gruppe S_n hat genau $n!$ Elemente, dementsprechend gibt es $n!$ Bitpermutationen von Bitstrings der Länge n.

Blockchiffren

- Blockchiffren sind Verschlüsselungsverfahren, die Blöcke fester Länge auf Blöcke derselben Länge abbilden.
- Allgemeinste Blockchiffre lässt sich wie folgt beschreiben: Fixiere Blocklänge n und ein Alphabet Σ. Verwende als Klartext- und Chiffretextraum $\mathcal{P} = \mathcal{C} = \Sigma^n$.
- Schlüsselraum ist die Menge $S(\Sigma^n)$ aller Permutationen von Σ^n.
- Zu einem Schlüssel $\pi \in S(\Sigma^n)$ gehört die Verschlüsselungsfunktion

$$E_\pi : \Sigma^n \to \Sigma^n, v \mapsto \pi(v).$$

- Die entsprechende Entschlüsselungsfunktion ist

$$D_\pi : \Sigma^n \to \Sigma^n, v \mapsto \pi^{-1}(v).$$

Permutationschiffre

Verwende nur solche Permutationen, die durch Vertauschen der Positionen der Zeichen entstehen.

- Falls $\Sigma = \{0, 1\}$ ist, sind das die Bitpermutationen.
- Schlüsselraum ist Permutationsgruppe S_n.

Für S_n setzt man

- $E_\pi : \Sigma^n \to \Sigma^n, (v_1, v_2, \ldots, v_n) \mapsto (v_{\pi(1)}, v_{\pi(2)}, \ldots, v_{\pi(n)})$.
- $D_\pi : \Sigma^n \to \Sigma^n, (x_1, x_2, \ldots, x_n) \mapsto (x_{\pi^{-1}(1)}, x_{\pi^{-2}(2)}, \ldots, x_{\pi^{-1}(n)})$.

Schlüsselraum hat $n!$ viele Elemente. Jeder einzelne Schlüssel läßt sich als Folge von n Zahlen kodieren.

Mehrfachverschlüsselung

- Sicherheit einer Blockchiffre kann gesteigert werden wenn man sie mehrmals hintereinander mit verschiedenen Schlüssel verwendet.
- Gebräuchlich ist die E-D-E-Dreifach-Verschlüsselung.
- Hierbei wird Klartext p wie folgt verschlüsselt:

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

dabei ist k_i der entsprechende Schlüssel und E_{k_i} die Verschlüsselungsfunktion und D_{k_i} die Entschlüsselungsfunktion zum Schlüssel k_i.

Man erreicht auf diese Weise eine erhebliche Vergrößerung des Schlüsselsraums. Will man die Schlüssellänge nur verdoppeln, benutzt man $k_1 = k_3$.

Verschlüsselungsmodi

Verschlüsselungsmodi ermöglichen Blockchiffren die Verschlüsselung von längeren Texten.

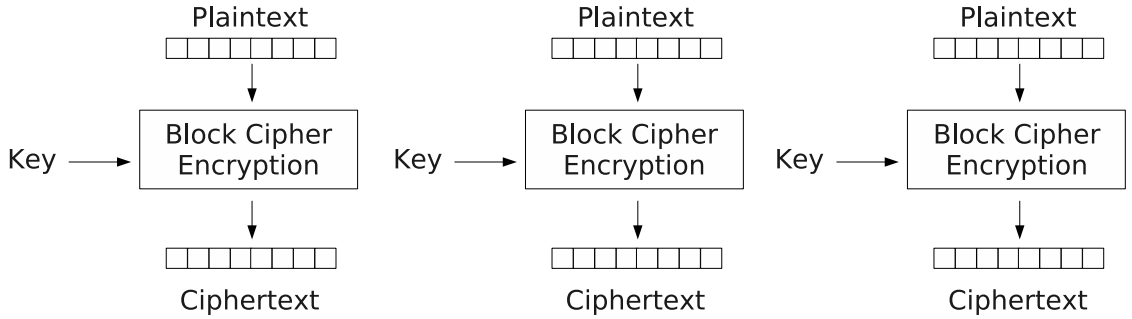
- Gegeben sei eine Blockchiffre mit Alphabet Σ und Blocklänge n.
- Der Schlüsselraum sei $\mathcal{K}$. Die Verschlüsselungsfunktionen seien E_k und die Entschlüsselfunktionen $D_k, k \in \mathcal{K}$.

Naheliegende Idee: Ein beliebig langer Klartext wird in Blöcke der Länge n aufgeteilt und gegebenenfalls so ergänzt, daß seine Länge durch n teilbar ist.

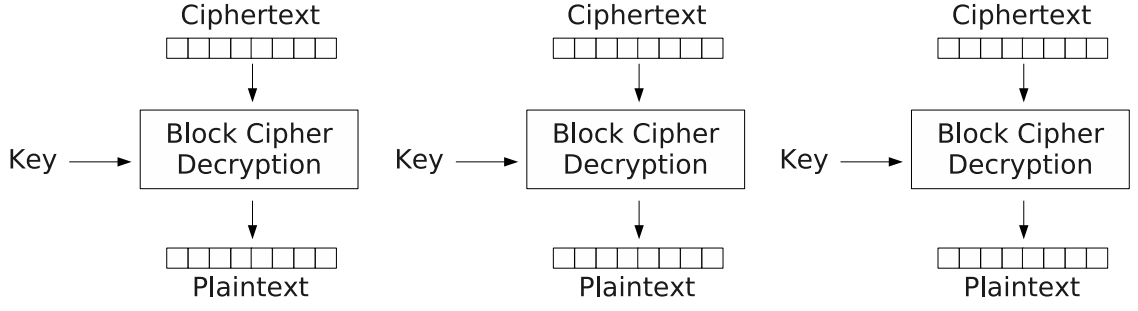
Diese Ergänzung erfolgt z.B. durch Anhängen von zufälligen Zeichen.

ECB-Mode Übersicht

- *Electronic Codebook* Mode (ECB-Mode) ist genau die Umsetzung unserer “naheliegender Idee”.



Electronic Codebook (ECB) mode encryption



Electronic Codebook (ECB) mode decryption

ECB-Mode Beispiel

Wir betrachten die Blockchiffre, die auf Bitvektoren der Länge vier Bitpermutationen durchführt, also die Permutationschiffre mit Alphabet $\Sigma = \{0, 1\}$ und Blocklänge 4. Es ist $\mathcal{K} = S_4$ und für $\pi \in S_4$ ist

$$E_\pi : \{0,1\}^4 \to \{0,1\}^4, b_1 b_2 b_3 b_4 \mapsto b_{\pi(1)} b_{\pi(2)} b_{\pi(3)} b_{\pi(4)}.$$

Der Klartext sei $m = 101100010100101$ und wird zerlegt in Teilblöcke m_i der Länge 4,

$$m_1 = 1011, m_2 = 0001, m_3 = 0100, m_4 = 1010$$

wobei letzte Teilblock auf Länge 4 ergänzt wird, indem eine 0 angehängt wird.

ECB-Mode Beispiel (Fort.)

Wir verwenden den Schlüssel

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix}$$

Die Blöcke werden nun einzeln verschlüsselt. Wir erhalten

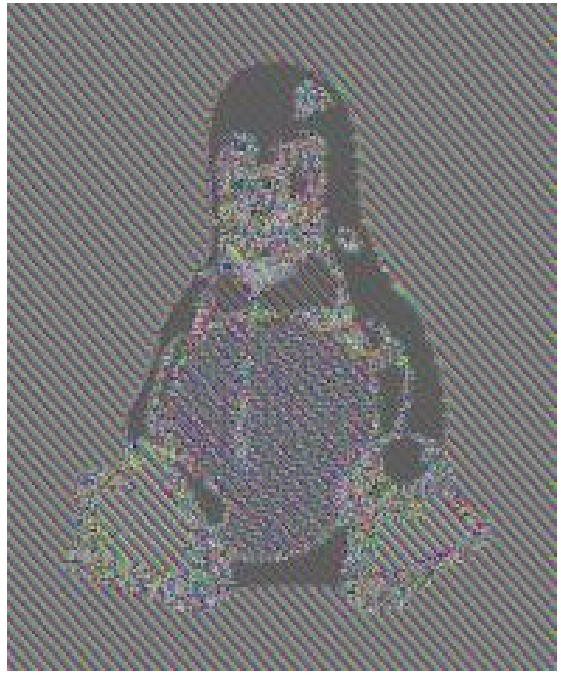
$$\begin{aligned} c_1 = & \quad E_\pi(m_1) & = 0111 \\ c_2 = & \quad E_\pi(m_2) & = 0010 \\ c_3 = & \quad E_\pi(m_3) & = 1000 \\ c_4 = & \quad E_\pi(m_4) & = 0101. \end{aligned}$$

Der Chiffretext lautet somit $c = 0111001010000101$.

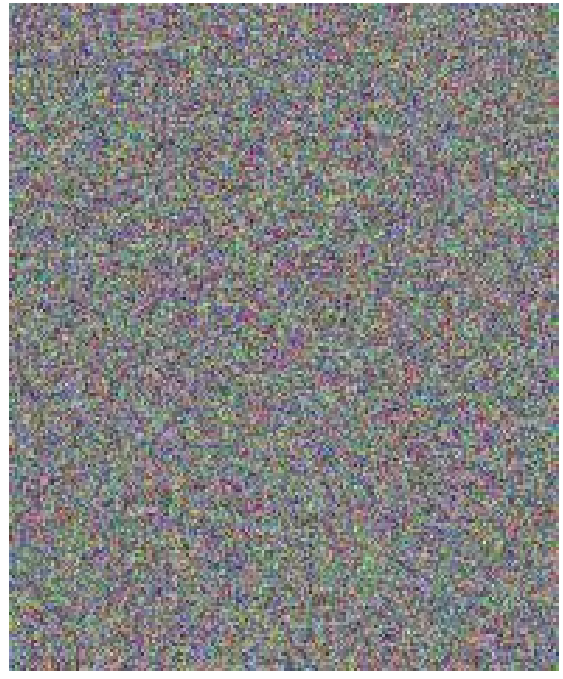
ECB Mode Probleme

Regelmäßigkeiten des Klartextes führen zu Regelmäßigkeiten des Schlüsseltextes. Siehe hervorstechendes Bildverschlüsselungs-Beispiel (Quelle: Wikipedia)

Original



Verschlüsselt im ECB-Mode



Verschlüsselt im anderen Mode

ECB-Mode Probleme (Fort.)

Beispiel verdeutlicht, dass man Informationen über den Klartext aus dem Schlüsseltext erhält, auch wenn man ihn nicht entschlüsseln kann. Weitere Nachteile sind:

- Angreifer kann Nachricht ändern, indem er Chiffretext einfügt, der mit dem gleichen Schlüssel verschlüsselt worden ist.
- Angreifer kann unbemerkt die Reihenfolge der Schlüsseltextblöcke verändern.

Aus diesen Gründen ist der ECB-Mode zur Verschlüsselung langer Nachrichten ungeeignet.

Man kann jedoch die Sicherheit des ECB-Mode steigern, wenn man die Blöcke nur teilweise aus dem Klartext und teilweise durch zufällige Zeichen bildet.

CBC-Mode

Um die Nachteile des ECB-Mode zu beseitigen, wurde der *Cipherblock Chaining* Mode (CBC-Mode) erfunden.

- Im CBC-Mode hängt die Verschlüsselung eines Klartextblocks nicht nur von diesem Block und dem Schlüssel, sondern auch von den vorhergegangenen Blöcken ab.
- Der CBC-Mode benötigt einen festen Initialisierungsvektor $IV \in \Sigma^n$.

Wie im ECB-Mode wird der Klartext in Blöcke der Länge n aufgeteilt. Will man eine Folge $m_1, m_2, \ldots, m_t$ von Klartextblöcken der Länge n mit dem Schlüssel e verschlüsseln, so setzt man

$$c_0 = IV, \qquad c_j = E_e(c_{j-1} \oplus m_j), \quad 1 \leq j \leq t.$$

CBC-Mode (Fort.)

Um Schlüsselblöcke $c_1, c_2, \ldots, c_t$ zu entschlüsseln wird Schlüssel d benötigt, der $D_d(E_e(w)) = w$ für alle Blöcke w erfüllt. Man setzt $c_0 = IV$ und berechnet:

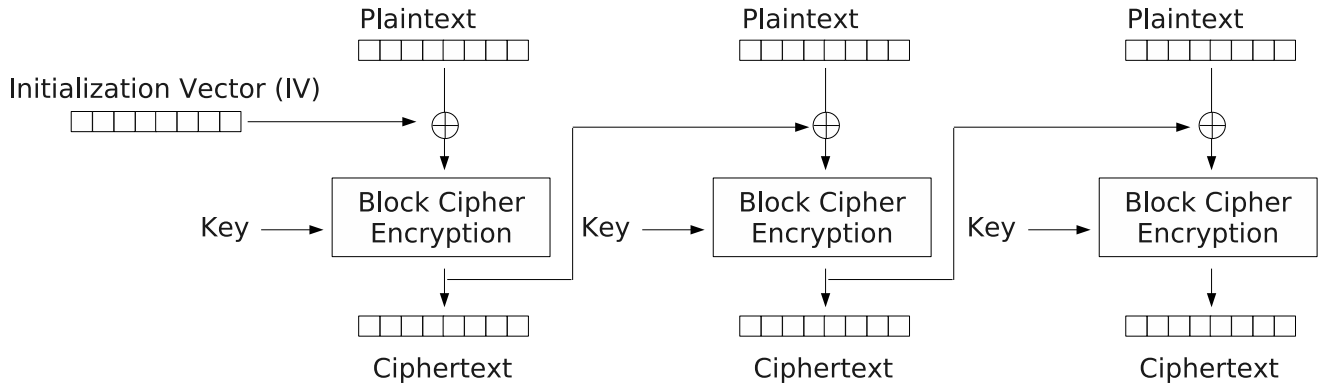
$$m_j = c_{j-1} \oplus D_d(c_j), \quad 1 \leq j \leq t.$$

Wie man sieht ist:

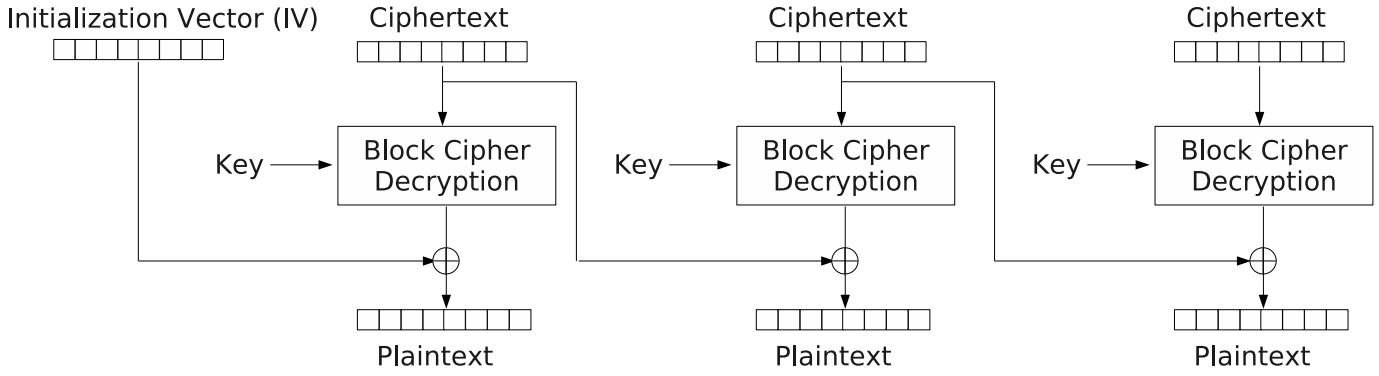
$$c_0 \oplus D_d(c_1) = c_0 \oplus c_0 \oplus m_1 = m_1$$

entsprechend verifiziert man das ganze Verfahren.

CBC-Mode Übersicht



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

CBC-Mode Beispiel

Gegeben seien die (aufgefüllten) Klartextblöcke

$$m_1 = 1011, m_2 = 0001, m_3 = 0100, m_4 = 1010$$

und Schlüssel

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix}$$

Als Initialisierungsvektor verwenden wir $IV = 1010$. Man erhält:

$$\begin{aligned} c_0 &= 1010 \\ c_1 &= E_\pi(c_0 \oplus m_1) = E_\pi(0001) = 0010 \\ c_2 &= E_\pi(c_1 \oplus m_2) = E_\pi(0011) = 0110 \\ c_3 &= E_\pi(c_2 \oplus m_3) = E_\pi(0010) = 0100 \\ c_4 &= E_\pi(c_3 \oplus m_4) = E_\pi(1110) = 1101 \end{aligned}$$

CBC-Mode Beispiel (Fort.)

Chiffretext ist $c = 0010011001001101$. Entschlüsselung führt zu:

$$
\begin{aligned}
m_1 &= c_0 \oplus E_\pi^{-1}(c_1) = 1010 \oplus 0001 = 1011 \\
m_2 &= c_1 \oplus E_\pi^{-1}(c_2) = 0010 \oplus 0011 = 0001 \\
m_3 &= c_2 \oplus E_\pi^{-1}(c_3) = 0110 \oplus 0010 = 0100 \\
m_4 &= c_3 \oplus E_\pi^{-1}(c_4) = 0100 \oplus 1110 = 1010
\end{aligned}
$$

CBC-Mode Zusammenfassung

- Gleiche Text werden im CBC-Mode verschieden verschlüsselt, wenn man den Initialisierungsvektor ändert.
- Verschlüsselung eines Blockes hängt von den vergangenen Blöcken ab.
- Ändert man die Reihenfolge der Chiffretextblöcke oder ändert man einen Chiffretextblock, so läßt sich der Chiffretext nicht mehr entschlüsseln.
- Ein Übertragungsfehler im Chiffretext c_j bewirkt, daß man m_j und m_{j+1} falsch berechnet. Die Berechnung von $m_{j+2}, m_{j+3}, \ldots$ ist wieder korrekt, weil sie nicht von c_j abhängt.
- Sender und Empfänger können verschiedenen IV wählen. Empfänger kann ersten Block nicht entschlüsseln, aber dann alle weiteren Blöcke korrekt entschlüsseln.

CFB-Mode

- Im CBC-Mode muß Empfänger immer abwarten bis der Sender einen ganzen Chiffretextblock erzeugt und diesen versandt hat, bevor er mit der Entschlüsselung des Blocks beginnen kann.
- Im *Cipher Feedback* Mode (CFB-Mode) werden Blöcke, die eine kürzere Länge als n haben können, nicht direkt durch die Blockverschlüsselungsfunktion E_k, sondern durch Addition $\mod 2$ entsprechender Schlüsselblöcke verschlüsselt.

Benötigt wird $IV \in \{0,1\}^n$, natürliche Zahl $r, 1 \leq r \leq n$.

- Der Klartext wird in Blöcke der Länge r aufgeteilt.

CFB-Mode Verschlüsselung

Verschlüsselung der Blockfolge $m_1, m_2, \ldots, m_u$ erfolgt wie folgt
Setze $I_1 := IV$ und führe für $1 \leq j \leq u$ folgende Schritte aus

1. $O_j := E_k(I_j)$
2. Setze t_j auf den String, der aus den ersten r Bits von O_j gebildet wird
3. $c_j := m_j \oplus t_j$
4. $I_{j+1} := 2^r I_j + c_j \mod 2^n$. I_{j+1} entsteht also, indem in I_j die ersten r Bits gelöscht werden und c_j hinten angehängt wird.

Der Schlüsseltext ist die Folge $c_1, c_2, \ldots, c_u$.

CFB-Mode Entschlüsselung

Setze $I_1 := IV$ und führe für $1 \leq j \leq u$ folgende Schritte aus

1. $O_j := E_k(I_j)$
2. Setze t_j auf den String, der aus den ersten r Bits von O_j gebildet wird
3. $m_j := c_j \oplus t_j$
4. $I_{j+1} := 2^r I_j + c_j \mod 2^n$

- Sender und Empfänger können t_{j+1} bestimmen, sobald sie c_j kennen. Der Schlüssel t_1 kann also von Sender und Empfänger gleichzeitig berechnet werden.
- Dann erzeugt der Sender $c_1 = m_1 \oplus t_1$ und versendet c_1.
- Anschließend können Sender und Empfänger simultan t_2 berechnen usw.
- D.h. die Blockchiffre auf Sender- und Empfängerseite kann fast simultan berechnet werden.

CFB-Mode Beispiel

Gegeben sei Schlüssel π wie im CBC Beispiel und
$r = 3$, $m_1 = 101, m_2 = 100, m_3 = 010, m_4 = 100, m_5 = 101$,
sowie $IV = 1010$.
Man erhält für $1 \leq j \leq 5$ folgende Verschlüsselungsresultate

j	I_j	O_j	t_j	m_j	c_j
1	1010	0101	010	101	111
2	0111	1110	111	100	011
3	1011	0111	011	010	001
4	1001	0011	001	100	101
5	1101	1011	101	101	000

OFB-Mode

- Der *Output Feedback Mode* (OFB-Mode) ist dem CFB-Mode ähnlich.
- Voraussetzungen und Initialisierung sind genauso wie im CFB-Mode.

Setze $I_1 := IV$ und führe für $1 \leq j \leq u$ folgende Schritte aus

1. $O_j := E_k(I_j)$
2. Setze t_j auf den String, der aus den ersten r Bits von O_j gebildet wird
3. $c_j := m_j \oplus t_j$
4. $I_{j+1} := O_j$.

Entschlüsselung funktioniert wie Verschlüsselung, wobei $3.$ Schritt durch $m_j := c_j \oplus t_j$ ersetzt wird.

OFB-Mode (Fort.)

- Werden Bits bei der Übertragung eines Schlüsseltextwortes verändert, so entsteht bei der Entschlüsselung nur ein Fehler an genau derselben Position, aber sonst nirgends.
- Schlüsselstrings t_j hängen nur vom Initialisierungsvector I_1 und vom Schlüssel k ab und können vom Sender und Empfänger parallel berechnet werden.
- Verschlüsselung der Klartextblöcke hängt nicht von den vorherigen Klartextblöcken, sondern nur von der Position ab.
- Im OFB-Mode verschlüsselte Texte können daher leichter manipuliert werden wie Texte im CBC-Mode.

OFB-Mode Beispiel

Gegeben sei Schlüssel π wie im CBC Beispiel und
$r = 3$, $m_1 = 101, m_2 = 100, m_3 = 010, m_4 = 100, m_5 = 101$,
sowie $IV = 1010$.
Man erhält für $1 \leq j \leq 5$ folgende Verschlüsselungsresultate

j	I_j	O_j	t_j	m_j	c_j
1	1010	0101	010	101	111
2	0101	1010	101	100	001
3	1010	0101	010	010	000
4	0101	1010	101	100	001
5	1010	0101	010	101	111

OFB-Mode Mehrfachverwendung von k

- Wenn derselbe Schlüssel k mehrmals verwendet werden soll, muß der IV verändert werden.
- Man erhält sonst die selbe Folge von Schlüsselstrings t_j.
- Hat man zwei Schlüsseltextblöcke $c_j = m_j \oplus t_j$ und $c'_j = m'_j \oplus t_j$ erhält man $c_j \oplus c'_j = m_j \oplus m'_j$.
- Hieraus kann man m'_j ermitteln, wenn m_j bekannt ist.

Stromchiffren

- Alphabet: $\Sigma = \{0, 1\}$
- Klartext- und Chiffretext: Σ^*
- Schlüsselmenge ist Σ^n für eine natürliche Zahl n
- Wörter in Σ^* werden Zeichen für Zeichen verschlüsselt

Sei $k := (k_1, k_2, \ldots, k_n)$ ein Schlüssel und $w = \sigma_1 \ldots \sigma_m$ ein Wort der Länge m in Σ^*. Erzeuge Schlüsselstrom $z_1, z_2, \ldots, z_m$ wie folgt

- setze $z_i := k_i \quad 1 \leq i \leq n$ und wenn $m > n$ ist
- $z_i := \sum_{j=1}^{n} c_j z_{i-j} \mod 2, \quad n < i \leq m$

wobei $c_1, c_2, \ldots, c_n$ für das Verfahren fest gewählte Bits sind. Die Verschlüsselungsfunktion E_k und die Entschlüsselungsfunktion D_k sind dann gegeben als $E_k(w) = \sigma_1 \oplus z_1, \ldots, \sigma_m \oplus z_m, D_k(w) = \sigma_1 \oplus z_1, \ldots, \sigma_m \oplus z_m$.

Stromchiffren (Beispiel)

Sei $n = 4$. Schlüsselstrom wird generiert gemäß der Rekursion

$$z_{i+4} = z_i + z_{i+1} \mod 2.$$

Dann ist $c_1 = c_2 = 0, c_3 = c_4 = 1$. Schlüssel sei $k = (1, 0, 0, 0)$. Man erhält als Schlüsselstrom

$$1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, \ldots$$

Affine Chiffre

Sei m natürliche Zahl.

- Affine Chiffre mit Klartextalphabet $\mathbb{Z}_m$ ist eine Blockchiffre der Blocklänge $n = 1$.
- Schlüsselraum besteht aus allen Paaren $(a, b) \in \mathbb{Z}_m^2$ mit zu m primen a.
- Verschlüsselungsfunktion E_e zum Schlüssel $e = (a, b)$ ist

$$E_e : \Sigma \to \Sigma, \quad x \mapsto ax + b \mod m.$$

- Entschlüsselungsfunktion zum Schlüssel $d = (a', b)$ ist

$$D_d : \Sigma \to \Sigma, \quad x \mapsto a'(x - b) \mod m.$$

Um den zu (a, b) passenden Entschlüsselungsschlüssel zu berechnen, löst man $aa' \equiv 1 \mod m$ mit dem erweiterten euklidischen Algorithmus. Zu (a, b) passende Schlüssel ist (a', b).

Affine Chiffre (Beispiel)

Sei $(a, b) = (7, 3), m = 26$ und $\mathbb{Z}_{26}$ das verwendete Alphabet.

- Verschlüsselt man das Wort BALD mit der affinen Chiffre im ECB-Mode, so ergibt sich

$$\begin{array}{cccc} B & A & L & D \\ 1 & 0 & 11 & 3 \\ 10 & 3 & 2 & 24 \\ K & D & C & Y \end{array}$$

- Zur Berechnung der entsprechenden Entschlüsselungsfunktion bestimmt man a' mit

$$7a' \equiv 1 \mod 26.$$

- Erweiterte euklidische Algorithmus liefert $a' = 15$.

Affine Chiffre (Beispiel, Fort.)

Entschlüsselungsfunktion bildet also einen Buchstaben x auf

$$15(x-3) \mod 26$$

ab und man erhält

K	D	C	Y
10	3	2	24
1	0	11	3
B	A	L	D

- Der Schlüsselraum der affinen Chiffre mit $m = 26$ hat $\phi(26) \cdot 26 = 312$ Elemente. Somit kann die affine Chiffre mit einer Ciphertext-Only-Attacke gebrochen worden, indem man alle Schlüssel durchsucht.

Known-Plaintext Attacke auf affine Chiffre

- Kennt man zwei Zeichen und ihre Verschlüsselung, kann man den Schlüssel mit linearer Algebra ermitteln.

Beispiel: Kenntnis das Buchstabe E in R und S in H verschlüsselt wurde, dann gelten Kongruenzen

$$4a + b \equiv 17 \mod 26\,, \quad 18a + b \equiv 7 \mod 26.$$

Man erhält

$$18a + \underbrace{17 - 4a}_{b} \equiv 7 \mod 26$$

und somit $14a \equiv -10 \mod 26 \Leftrightarrow 14a \equiv 16 \mod 26$. Daraus folgt $7a \equiv 8 \mod 13$ und somit $a = 3$ und $b = 5$.

Affin lineare Blockchiffre

- Affin lineare Blockchiffren sind Verallgemeinerungen der affinen Chiffre.
- Solche Chiffren können leicht mit Known-Plaintext-Attacken angegriffen werden.
- Entwurf von Blockchiffren muß vermieden werden, daß sie affin linear sind.

Definition: Eine Blockchiffre mit Blöcklänge n und Klartext- und Schlüsselraum $\mathbb{Z}_m^n$ heißt *affin linear*, wenn ihre sämtliche Verschlüsselungsfunktionen affin linear sind. Sie heißt linear, wenn alle Verschlüsselungsfunktionen linear sind.

Affin lineare Blockchiffre (Fort.)

- Verschlüsselungsfunktion ist affin linear und von der Form

$$E : \mathbb{Z}_m^n \to \mathbb{Z}_m^n, \quad \mathbf{v} \mapsto A\mathbf{v} + \mathbf{b} \mod m,$$

  mit $A \in \mathbb{Z}_m^n$ und $\mathbf{b} \in \mathbb{Z}^n$
- E ist bijektiv und det A teilerfremd zu m.
- Verschlüsselungsfunktion ist durch das Paar $(A, \mathbf{b})$ eindeutig bestimmt und kann als Schlüssel genommen werden.
- Entschlüsselungsfunktion ist entsprechend

$$D : \mathbb{Z}_m^n \to \mathbb{Z}_m^n, \quad \mathbf{v} \mapsto A'(\mathbf{v} - \mathbf{b}) \mod m,$$

  wobei $A' = (a' \text{ adj } A) \mod m$ und a' das Inverse von det $A \mod m$ ist.

Vigenère Chiffre

- Benannt nach Blaise de Vigenère der im 16. Jahrhundert lebte.
- Schlüsselraum ist $\mathcal{K} = \mathbb{Z}_m^n$. Ist $\mathbf{k} \in \mathbb{Z}_m^n$, dann ist

$$E_{\mathbf{k}} : \mathbb{Z}_m^n \to \mathbb{Z}_m^n, \quad \mathbf{v} \mapsto \mathbf{v} + \mathbf{k} \mod m$$

  und entsprechend

$$D_{\mathbf{k}} : \mathbb{Z}_m^n \to \mathbb{Z}_m^n, \quad \mathbf{v} \mapsto \mathbf{v} - \mathbf{k} \mod m$$

- Abbildungen sind affin linear. Schlüsselraum hat m^n Elemente.

Hill Chiffre

- Benannt nach Lester S. Hill, die er 1929 erfunden hat.
- Schlüsselraum $\mathcal{K}$ Menge aller Matrizen $A \in \mathbb{Z}_m^{(n,n)}$ mit gcd(det $A, m) = 1$.
- Für $A \in \mathcal{K}$ ist $E_A : \mathbb{Z}_m^n \to \mathbb{Z}_m^n, \quad \mathbf{v} \mapsto A\mathbf{v} \mod m$ und entsprechend $D_A : \mathbb{Z}_m^n \to \mathbb{Z}_m^n, \quad \mathbf{v} \mapsto A'\mathbf{v} \mod m$

Kryptoanalyse affin linearer Blockchiffren

Wir zeigen wie eine affin lineare Blockchiffre mit Alphabet $\mathbb{Z}_m$ und Blocklänge n mittels einer Known-Plaintext-Attacke gebrochen werden kann.

- Ausgangssituation: Schlüssel ist fixiert worden, zugehörige Verschlüsselungsfunktion hat die Form

$$E : \mathbb{Z}_m^n \to \mathbb{Z}_m^n, \quad \mathbf{v} \mapsto A\mathbf{v} + \mathbf{b} \mod m$$

  mit $A \in \mathbb{Z}^{(n,n)}$ und $\mathbf{b} \in \mathbb{Z}^n$.
- Angreifer will Schlüssel $(A, \mathbf{b})$ bestimmen.
- Dazu verwendet er $n+1$ Klartexte $\mathbf{w}_i$, $0 \leq i \leq n$ und zugehörige Schlüsseltexte $\mathbf{c}_i = A\mathbf{w}_i + \mathbf{b} \mod m$, $0 \leq i \leq n$.

Kryptoanalyse (Fort.)

- Es gilt

$$\mathbf{c}_i - \mathbf{c}_0 \equiv A(\mathbf{w}_i - \mathbf{w}_0) \mod m.$$

- Ist W die Matrix

$$W = (\mathbf{w}_1 - \mathbf{w}_0, \ldots, \mathbf{w}_n - \mathbf{w}_0) \mod m$$

deren Spalten die Differenzen $(\mathbf{w}_i - \mathbf{w}_0)$ $\mod m$, $1 \leq i \leq n$, sind.

- C die Matrix

$$C = (\mathbf{c}_1 - \mathbf{c}_0, \ldots, \mathbf{c}_n - \mathbf{c}_0) \mod m$$

deren Spalten die Differenzen $(\mathbf{c}_i - \mathbf{c}_0)$ $\mod m$, $1 \leq i \leq n$, sind.

Kryptoanalyse (Fort.)

- Kompakt in Matrix Notation gilt

$$AW \equiv C \mod m.$$

- Ist det W teilerfremd zu m, so ist

$$A \equiv C(w' \text{ adj } W) \mod m,$$

  wobei w' das Inverse von det $W \mod m$ ist.
- Nachdem A bestimmt ist, folgt

$$\mathbf{b} = \mathbf{c}_0 - A\mathbf{w}_0.$$

- Man kann somit aus $n+1$ Paaren von Klar- und Schlüsseltexten den Schlüssel $(A, \mathbf{b})$ bestimmen.
- Ist Chiffre nur linear, so kann man $\mathbf{w}_0 = \mathbf{c}_0 = \mathbf{0}$ setzen, und es ist $\mathbf{b} = 0$.

Beispiel Kryptoanalyse von Hill-Chiffre

- Angenommen, man weiß, daß HAND in FOOT verschlüsselt wird und Blocklänge 2 benutzt wird.
- Dann wird $\mathbf{w}_1 = (7, 0)$ in $\mathbf{c}_1 = (5, 14)$ und $\mathbf{w}_2 = (13, 3)$ in $\mathbf{c}_2 = (14, 19)$ verschlüsselt.
- So erhält man

$$W = \begin{pmatrix} 7 & 13 \\ 0 & 3 \end{pmatrix} \text{ und } C = \begin{pmatrix} 5 & 14 \\ 14 & 19 \end{pmatrix}$$

- Es ist $\det W = 21$ teilerfremd zu 26.
- Das Inverse von $21 \mod 26$ ist 5. Somit ist $A = 5C(\text{adj } W) \mod 26$

$$= 5 \begin{pmatrix} 5 & 14 \\ 14 & 19 \end{pmatrix} \begin{pmatrix} 3 & -13 \\ 0 & 7 \end{pmatrix} \mod 26 = \begin{pmatrix} 23 & 9 \\ 2 & 15 \end{pmatrix}$$

Designziele in Blockchiffren

Konstruiere Verschlüsselungsfunktion die sich wie eine zufällige Funktion verhalten soll.

- Konfusion: Verschleiern des Zusammenhangs zwischen Klartext und Chiffretext.
- Diffusion: Verteilen der Information über den Chiffretext.
- Lawineneffekt: Jedes Bit im Chiffretext soll von jedem Bit des Klartexts und des Schlüssels abhängen. D.h. Änderungen im Klartext haben großen Einfluss auf den Chiffretext.
  - Im Idealfall wird durch die Änderung eines Bits des Klartextblockes jedes Bit des Geheimtextblockes geändert.

DES Algorithmus

- Data Encryption Standard (im Jahr 1977 als Standard veröffentlicht).
- Beruht auf dem Prinzip der Konfusion und Diffusion.
- Klartext- und Schlüsselraum des DES Verschlüsselungsverfahren ist $\mathcal{P} = \mathcal{C} = \{0,1\}^{64}$.
- DES Schlüssel hat Länge 64 Bits (8 Bytes), wobei das letzte Bit eines jeden Bytes so gesetzt wird, daß die Quersumme aller Bits im betreffenden Byte ungerade ist.
- In einem DES Schlüssel sind also nur 56 Bits frei wählbar und somit gibt es $2^{56} \sim 7.2 \cdot 10^{16}$ viele DES Schlüssel.
- DES Schlüssel für Ver- und Entschlüsselung ist derselbe.

Beispiel DES Schlüssel

Gültiger DES Schlüssel in hexadezimal Darstellung:

13 34 57 79 9B BC DF F1

Binärdarstellung:

0	0	0	1	0	0	1	1	13_{16}	$= 19_{10}$
0	0	1	1	0	1	0	0	34_{16}	$= 52_{10}$
0	1	0	1	0	1	1	1	.	
0	1	1	1	1	0	0	1	.	
1	0	0	1	1	0	1	1	.	
1	0	1	1	1	1	0	0	.	
1	1	0	1	1	1	1	1	DF_{16}	$= 223_{10}$
1	1	1	1	0	0	0	1	$F1_{16}$	$= 241_{10}$

(Horizontale) Quersumme ungerade

Position	1	2	3	4	5	6	7	8	9	10	11	...
Bitfolge	0	0	0	1	0	0	1	1	0	0	1	...

Initiale Permutation (IP)

Gegeben Klartextwort $p \in \{0,1\}^{64}$, $p = p_1 p_2 \dots p_{64}$.
Erster Schritt: Anwendung der initialen Permutation (IP) auf p, d.h. $\text{IP}(p) = p_{58} p_{50} p_{42} \dots p_7$

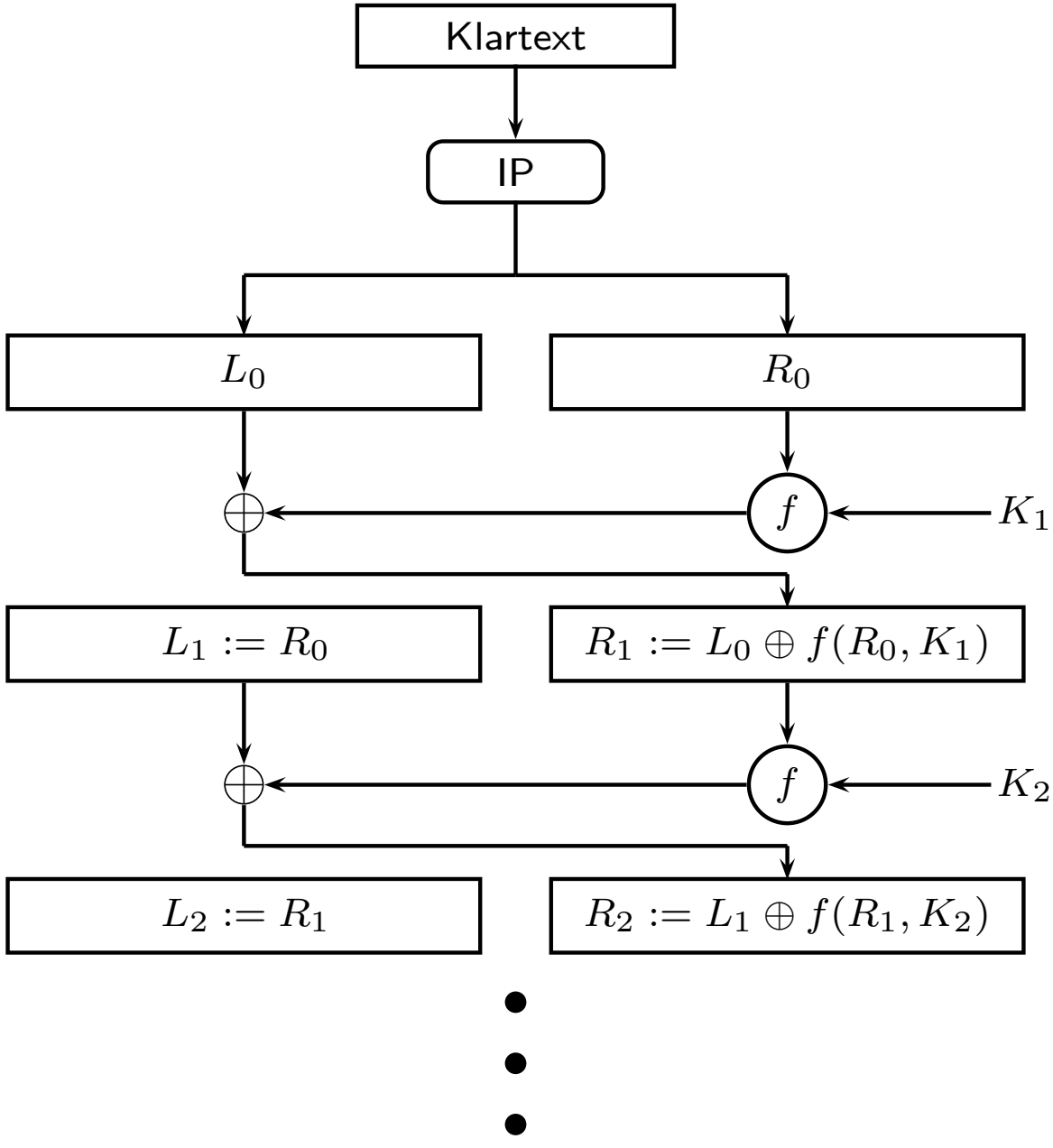
IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
⋮							
63	55	47	39	31	23	15	7

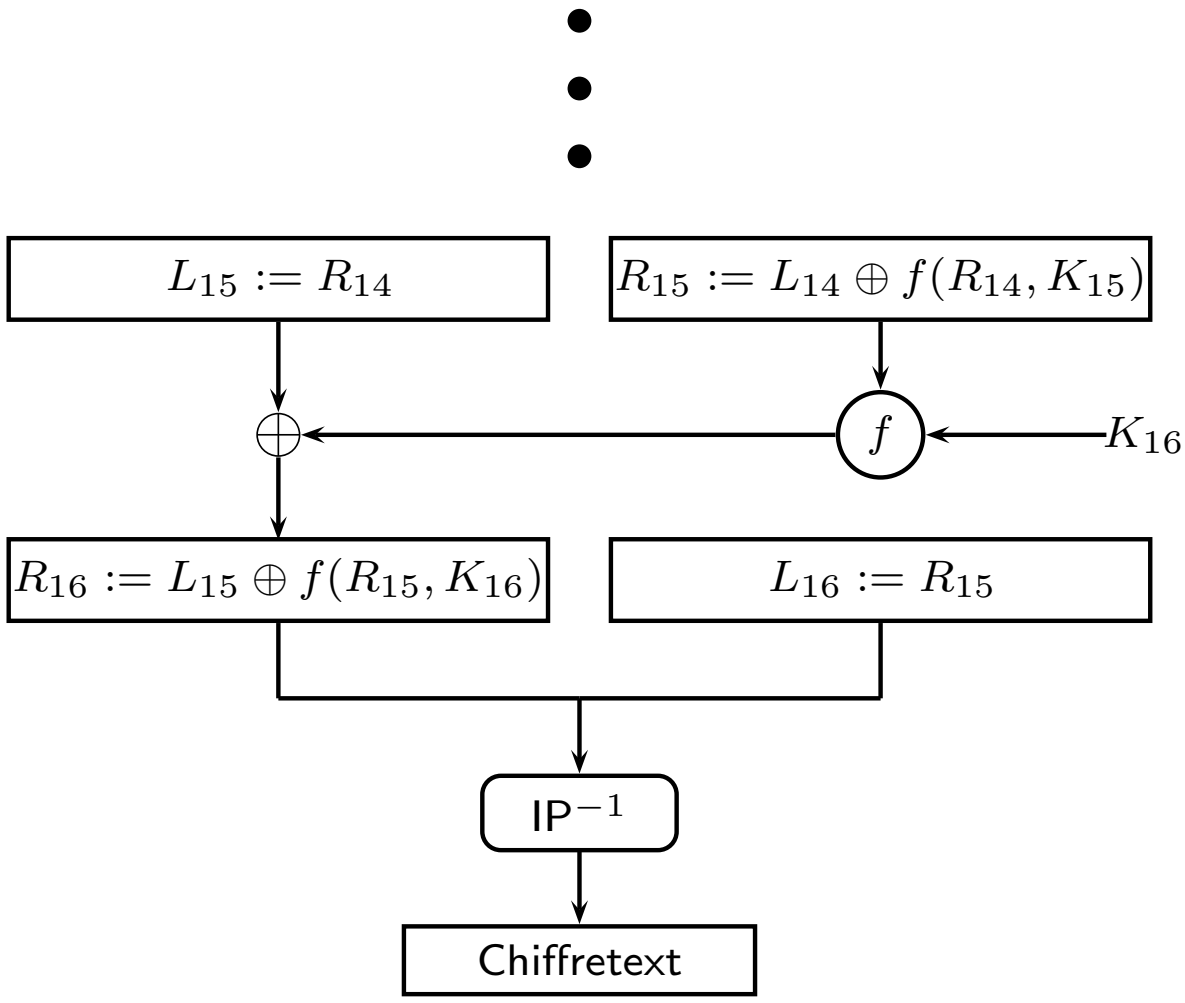
Auf $\text{IP}(p)$ wird eine 16-Runden Feistel-Chiffre angewendet.

Hierzu wird $\text{IP}(p)$ in zwei Bitfolgen (L_0, R_0) (linke Hälfte, rechte Hälfte) der Länge von jeweils 32 Bit zerlegt und rundenweise folgende Berechnung durchgeführt:
$L_i := R_{i-1}$, $R_i := L_{i-1} \oplus f(R_{i-1}, K_i), 1 \leq i \leq 16$

DES Runden Diagramm



DES Runden Diagramm (Fort.)



Beachte: Rechts- und Linkstausch in letzter Runde.

Berechnung der Rundenschlüssel

- Aus DES Schlüssel $k \in \{0,1\}^{64}$ werden Rundenschlüssel $K_i, 1 \leq i \leq 16$ der Länge 48 generiert
- Definiere $v_i, 1 \leq i \leq 16$ wie folgt:

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

- Permutationsfunktionen $PC1 : \{0,1\}^{64} \rightarrow \{0,1\}^{28} \times \{0,1\}^{28}, PC2 : \{0,1\}^{28} \times \{0,1\}^{28} \rightarrow \{0,1\}^{48}$

PC1

57	49	41	33	25	17	9
1	58	50	42	34	26	18
			⋮			
63	55	47	39	31	23	15
7	62	54	46	38	30	22
			⋮			

PC2

14	17	11	24	1	5
3	28	15	6	21	10
		⋮			
44	49	39	56	34	53
46	42	50	36	29	32

Berechnung der Rundenschlüssel (Fort.)

- Setze $(C_0, D_0) := PC1(k)$
- Für $1 \leq i \leq 16$ führe zirkulären Linksshift um v_i Stellen auf C_{i-1} (bzw. D_{i-1}) durch und weise das Resultat C_i (bzw. D_i) zu. Berechne $K_i := PC2(C_i, D_i)$.

Ist $k = k_1 k_2 \ldots k_{64}$, dann ist $C = k_{57} k_{49} \ldots k_{36}$ (obere Hälfte der PC1 Tabelle) und $D = k_{63} k_{55} \ldots k_4$ (untere Hälfte der PC1 Tabelle).

Der Wert PC2$(b_1 b_2 \ldots b_{56})$ ist entsprechend $b_{14} b_{17} \ldots b_{32}$.

Interne DES Blockchiffre $f(\cdot)$

Wir wissen wie der 48 Bit Schlüssel K berechnet wird und erläutern nun wie die Verschlüsselungsfunktion $f(R, K)$ funktioniert.

Argument $R \in \{0,1\}^{32}$ wird mittels Expansionsfunktion $E : \{0,1\}^{32} \to \{0,1\}^{48}$ verlängert. Ist $R = R_1 R_2 \ldots R_{32}$, dann ist $E(R) = R_{32} R_1 R_2 \ldots R_{32} R_1$.

E

32	1	2	3	4	5
4	5	6	7	8	9
			$\vdots$		
24	25	26	27	28	29
28	29	30	31	32	1

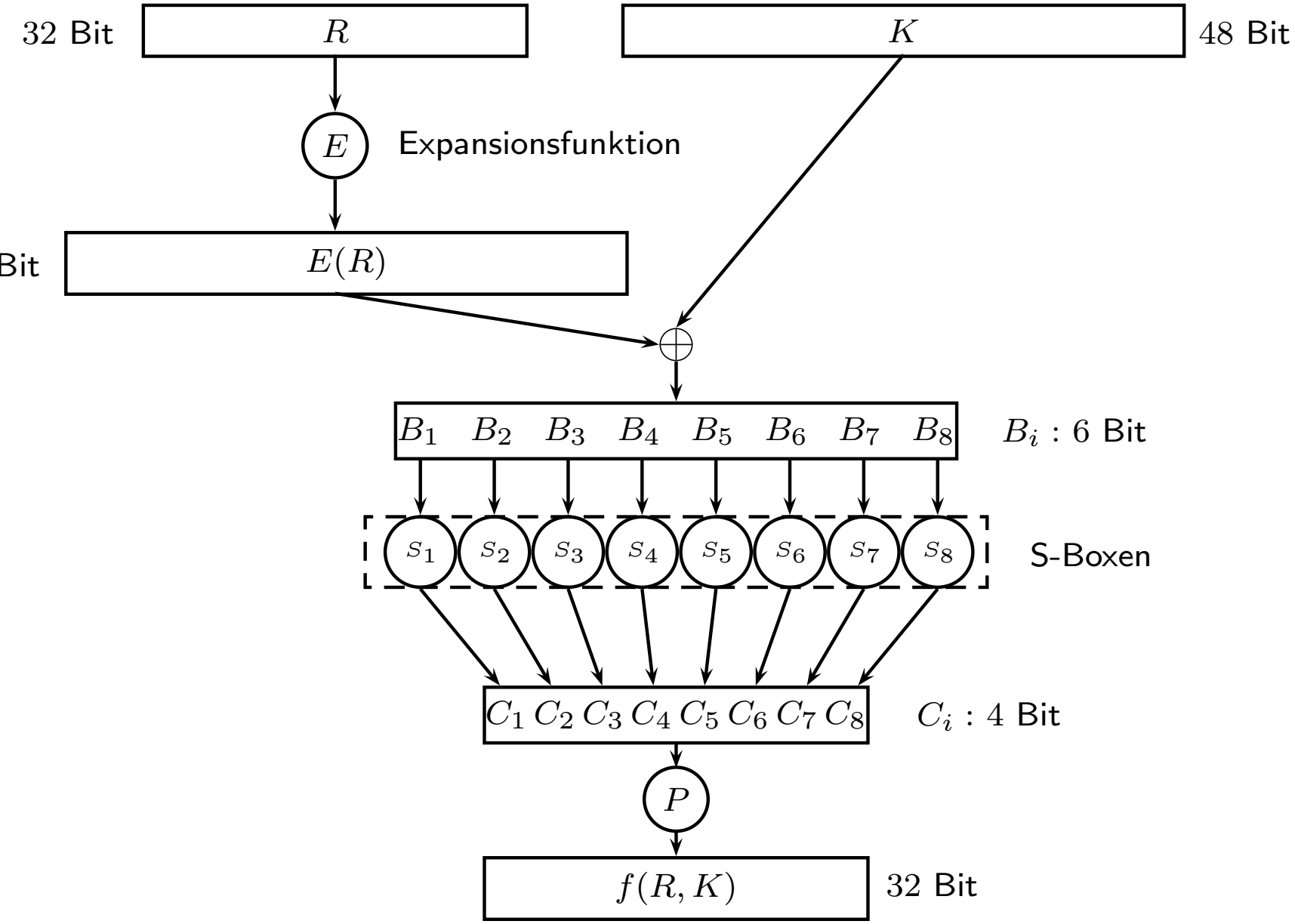
Interne DES Blockchiffre $f(\cdot)$ (Fort.)

- Blockaufteilung $B_1B_2B_3B_4B_5B_6B_7B_8 := E(R) \oplus K$ wobei $B_i \in \{0,1\}^6, 1 \leq i \leq 8$
- Berechnung von $C := C_1C_2C_3C_4C_5C_6C_7C_8$ wobei $C_i := S_i(B_i)$ und $S_i : \{0,1\}^6 \rightarrow \{0,1\}^4, 1 \leq i \leq 8$. Die Funktionen S_i bezeichnen S-Boxen und werden nachfolgend erklärt.
- Das Result C hat Länge 32 Bit und wird abschließend gemäß P wie folgt permutiert:

$$P$$

16	7	20	21
29	12	28	17
		$\vdots$	
22	11	4	25

Interne DES Blockchiffre Diagramm



S-Boxen

Jede S-Box wird durch eine Tabelle aus 4 Zeilen und 16 Spalten beschrieben (Bsp. S_1 Box).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	**2**	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

$S_i(b)$, wobei $b = b_1b_2b_3b_4b_5b_6$ wird wie folgt berechnet:

- Zeilenindex ist natürliche Zahldarstellung der Binärdarstellung b_1b_6.
- Spaltenindex ist natürliche Zahldarstellung der Binärdarstellung $b_2b_3b_4b_5$.

S-Boxen (Fort.)

- Eintrag in Zeile und Spalte wird binär dargestellt. Diese Binärdarstellung wird zusätzlich mit Nullen aufgefüllt, so daß Gesamtlänge 4 Bit beträgt.

Beispiel: $S_1(001011)$

$$\begin{aligned} b_1b_6 &= 01 &&= 1_{10} \\ b_2b_3b_4b_5 &= 0101 &&= 5_{10} \end{aligned}$$

Zeile 1, Spalte 5 enthält Wert 2 und somit

$S_1(001011) = \underbrace{00}_{\text{Nullen auffüllen}} \underbrace{10}_{2_{10}}$

Entschlüsselung im DES

Der Klartext wird nach der initialen Permutation in (L_0, R_0) zerlegt und rundenweise die Berechnung:

$$(L_i, R_i) \quad = \quad (R_{i-1}, L_{i-1} \oplus f(R_{i-1}, K_i))$$

durchgeführt. Aus $L_i = R_{i-1}$ und

$$(L_i, R_i \oplus f(L_i, K_i)) = (R_{i-1}, L_{i-1} \oplus f(L_i, K_i) \oplus f(L_i, K_i))$$

folgt $(R_{i-1}, L_{i-1}) = (L_i, R_i \oplus f(L_i, K_i))$.

Unter Verwendung der Schlüsselfolge $(K_{16}, K_{15}, \ldots, K_1)$ kann in 16 Runden das Paar (R_0, L_0) aus (R_{16}, L_{16}) deshalb zurückgerechnet werden.

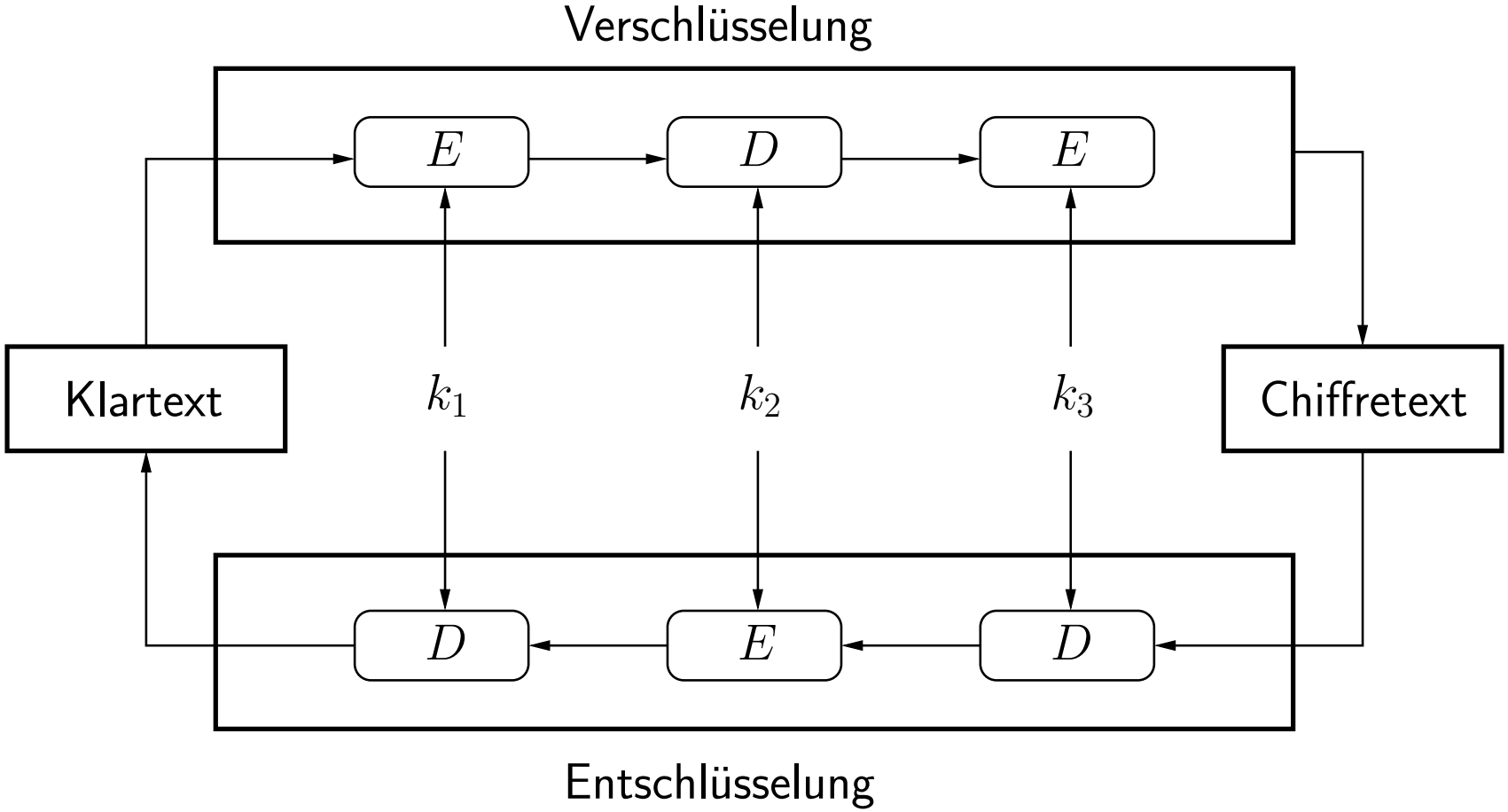
Mehrfachverschlüsselung

Sicherheit einer Blockchiffre kann gesteigert werden, indem man sie mehrmals hintereinander mit verschiedenen Schlüsseln verwendet.

- Gebräuchlich ist E-D-E-Dreifach Verschlüsselung (Triple Encryption).
- E-D-E mit zwei Schlüsseln
  - Klartext p verschlüsselt man in $c = E_{k_1}(D_{k_2}(E_{k_1}(p)))$.
  - Chiffretext c entschlüsselt man in $p = D_{k_1}(E_{k_2}(D_{k_1}(c)))$
- E-D-E mit drei Schlüsseln
  - $c = E_{k_3}(D_{k_2}(E_{k_1}(p)))$
  - $p = D_{k_1}(E_{k_2}(D_{k_3}(c)))$

Mittels Mehrfachverschlüsselung erreicht man eine erhebliche Vergrößerung des Schlüsselraums.

Triple DES



DES Verschlüsselungsfunktionen bilden keine Untergruppe der Permutationsgruppe $S_{64!}$, somit bietet Mehrfachverschlüsselung mehr Sicherheit (statt 2^{56}, sind 2^{112} Versuche in einer erschöpfende Suche nötig).

Sicherheit DES

- Wegen der geringen Schlüssellänge von 56 Bit lässt sich mittels einer Brute-Force Attacke der Schlüssel berechnen.
  - Im Jahr 1998 wurde Deep Crack gebaut (speziell auf DES abgestimmte Prozessoren). Preis der Maschine war 250.000 Dollar und es dauerte 50 Stunden um Schlüssel zu finden.
  - Kurz darauf wurde Deep Crack und verteiltes Rechnen (distributed.net) benutzt um ein Schlüssel in ca. 22 Stunden zu finden.
  - Einige Jahre später wurde die COPACOBANA (cost-optimized parallel code breaker) Maschine gebaut. Preis ca. 10.000 Dollar. Dauer ca. 6.4 Tage um Schlüssel zu finden.

Differentielle Kryptoanalyse

- Beruht auf den Arbeiten von Eli Biham und Adi Shamir aus den neunziger Jahren (Differential Cryptanalysis of DES-like Cryptosystems).
- Differentielle Kryptoanalyse ist eine *Chosen-Plaintext-Attacke* gegen DES.
- Idee: betrachte gewisse Paare von Chiffretexten, so daß deren zugehörige Klartexte bestimmte Differenzen aufweisen.
- Untersuche die Entwicklung dieser Differenzen, während die beiden Klartexte die DES-Runden durchlaufen und mit dem gleichen Schlüssel verschlüsselt werden.
- Differenz ist definiert über XOR Operator.
- Wähle Klartextpaare mit fester Differenz, wobei die Auswahl der Paare zufällig erfolgen kann, solange die Differenz nur bestimmte Bedingungen erfüllt.

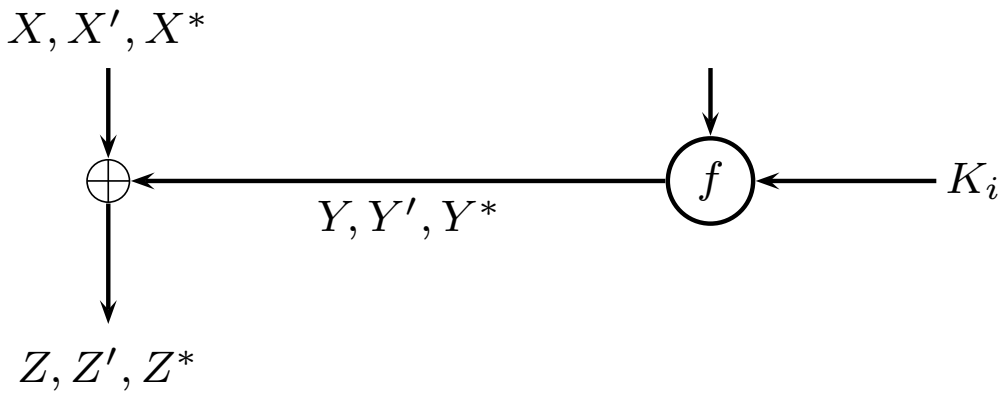
Differentielle Kryptoanalyse (Fort.)

- Mit Hilfe der Differenzen der entstandenen Chiffretexte weist man verschiedenen Schlüsseln unterschiedliche Wahrscheinlichkeiten zu.
- Analysiert man eine genügend hohe Anzahl von Chiffretext-Paaren, so kristallisiert sich schließlich ein Schlüssel als der wahrscheinlichste heraus — dies ist der korrekte Schlüssel.

Differentielle Kryptoanalyse Notation

- n_x: Tiefgestelltes x einer Zahl n bezeichnet die hexadezimale Darstellung (z.B. $10_x = 16$)
- P, P': Klartextpaar nach der Eigangspermutation IP. Die IP spielt bei der differentiellen Analyse von DES keine Rolle.
- $P^* = P \oplus P'$: Differenz der Klartexte P und P' ($\oplus$ bezeichnet XOR Operator).
- C, C': Chiffretextpaar vor der Ausgangspermutation IP^{-1}. Diese Permutation spielt ebenfalls keine Rolle bei der differentiellen Analyse.

XOR Differenzen in DES Runde



Wie wirken sich die Differenzen von X, X' bzw. Y, Y' auf das Resultat Z, Z' aus?

$$X \oplus X' = X^*, \quad Y \oplus Y' = Y^*$$

und

$$X \oplus Y = Z, \quad X' \oplus Y' = Z'.$$

Man erhält

$$Z^* = Z \oplus Z' = (X \oplus Y) \oplus (X' \oplus Y') = (X \oplus X') \oplus (Y \oplus Y') = X^* \oplus Y^*$$

XOR Differenzen in Funktion f

- Expansionsfunktion E

$$X, X', X^* \longrightarrow E \longrightarrow Z, Z', Z^*$$

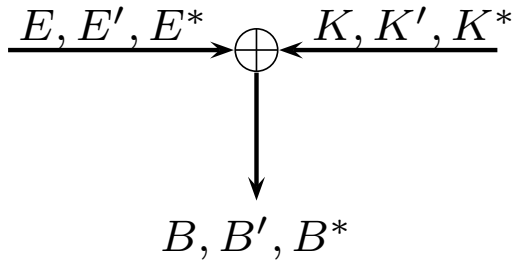
- Es gilt

$$X^* = X \oplus X',\ Z = E(X),\ Z' = E(X')$$

- Resultierende Differenz

$$Z^* = Z \oplus Z' = E(X) \oplus E(X') = E(X \oplus X')$$

XOR Differenzen in $E \oplus K$



- Es gilt

$$E^* = E \oplus E' \text{ und } K = K'$$

  und somit $K^* = K \oplus K' = 0$.

- Außerdem ist $B = E \oplus K, \quad B' = E' \oplus K.$
- Als Resultat erhält man

$$B^* = B \oplus B' = (E \oplus K) \oplus (E' \oplus K) = E \oplus E' = E^*.$$

Die Eingabedifferenz B^* für die S-Boxen (nächster Schritt) ist *unabhängig vom Schlüssel.*

XOR Differenzen in Permutation P

- Permutation P

$$X, X', X^* \longrightarrow P \longrightarrow Z, Z', Z^*.$$

- Es gilt

$$X^* = X \oplus X' \text{ sowie } Z = P(X) \text{ und } Z' = P(X').$$

- Man erhält

$$Z^* = Z \oplus Z' = P(X) \oplus P(X') = P(X \oplus X') = P(X^*).$$

Die einzigen *nichtlinearen* Teile von DES sind die S-Boxen.

XOR Differenzen in S-Boxen

- Jede S-Box hat 64×64 mögliche Eingabe Paare, wobei jedes Paar eine XOR-Eingabe und eine XOR-Ausgabe hat.
- Es gibt aber 64×16 mögliche XOR-Eingabe und XOR-Ausgabe Tupel.
- Jedes Tupel kann im Durchschnitt von $(64 \cdot 64)/(64 \cdot 16) = 4$ möglichen Paaren gebildet werden.
- Man stellt jedoch fest, daß nicht alle Tupel aus möglichen Paaren gebildet werden können. Die gebildeten Tupel sind außerdem nicht uniform verteilt.

Paar-XOR Verteilungstabelle einer S-BOX

Eing. XOR	Ausgabe XOR															
	0_x	1_x	2_x	3_x	4_x	5_x	6_x	7_x	8_x	9_x	A_x	B_x	C_x	D_x	E_x	F_x
0_x	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1_x	0	0	0	6	0	2	4	4	0	10	12	4	10	6	2	4
2_x	0	0	0	8	0	4	4	4	0	6	8	6	12	6	4	2
⋮									⋮							
34_x	0	8	16	6	2	0	0	12	6	0	0	0	0	8	0	6
⋮									⋮							
$3F_x$	4	8	4	2	4	0	2	4	4	2	4	8	8	6	2	2

Beispiel:

- Die XOR-Eingabe 34_x führt in 0 Fällen zur XOR-Ausgabe $0_x, 5_x, 6_x, \ldots$. Jedoch in 16 Fällen zur XOR-Ausgabe 2_x usw.
- Man sieht also, daß die XOR-Ausgabe Werte für jeweils einen XOR-Eingabe Wert *nicht gleichverteilt* sind.
- Einige Werte wie z.B. $0_x, 5_x, 6_x$ sind gar nicht möglich.
- Diese Tatsache ist die Basis für die differentielle Analyse von DES.

Beispiel einer differentiellen Analyse

- Gegeben sei $E_1 = 01_x$, $E_1' = 35_x$, wobei E_1 die ersten 6 Bits von E sind (bzw. von E' sind). Bekannt sei die Ausgabe der S-Box S_1 mit $C_1^* = D_x$.
- Gesucht sind die ersten 6 Bits von K_1 (bezeichnet als K_1^6).
- Für die Eingabe der S-Box S_1 gilt

$$B_1^* = E_1^* = 01_x \oplus 35_x = 34_x$$

- Analyse von S_1 ergibt, daß für Eingabe 34_x und Ausgabe D_x nur folgende Paare (B_1, B_1') für die Eingabe von S_1 möglich sind

$$(06_x, 32_x), (32_x, 06_x), \quad (10_x, 24_x), (24_x, 10_x)$$
$$(16_x, 22_x), (22_x, 16_x), \quad (1C_x, 28_x), (28_x, 1C_x)$$

Beispiel einer differentiellen Analyse (Fort.)

- Zu jedem Paar gehört genau ein bestimmter Schlüssel K_1^6, nämlich

$$E_1 \oplus K_1^6 = B_1 \text{ und folglich } K_1^6 = B_1 \oplus E_1.$$

- Zu den 8 Paaren gehören folgende Teilschlüssel

$$06_x \oplus 01_x = 07_x, 32_x \oplus 01_x = 33_x, 10_x \oplus 01_x = 11_x,$$

$$24_x \oplus 01_x = 25_x, 16_x \oplus 01_x = 17_x, 22_x \oplus 01_x = 23_x,$$

$$1C_x \oplus 01_x = 1D_x, 28_x \oplus 01_x = 29_x.$$

- Der gesuchte Schlüssel muß sich unter diesen 8 Schlüsseln befinden.
- Andere Werte für die Eingaben führen im Allgemeinen zu anderen Kandidaten für K_1^6. Richtige Schlüssel befindet sich im Durchschnitt der Kandidatenmenge.

Beispiel einer differentiellen Analyse (Fort.)

- Analyse wird fortgesetzt, bis schließlich ein einziger Schlüssel K_1^6 übrig bleibt.
- Nach der gleichen Vorgehensweise können auch die restlichen Teilschlüssel $K_2^6, K_3^6, \ldots, K_8^6$ bestimmt werden.

Für jede weitere Runde muss man jedoch mehr und mehr Paare suchen. Insgesamt

Rundenzahl	Anzahl ausgewählter Klartexte	Anzahl ausgewählter Klartexte
	bei unabhängigen Schlüsseln	bei abhängigen Schlüsseln
4	16	8
6	256	256
8	2^{16}	2^{14}
16	2^{60}	2^{47}

Sicherheit DES gegen diff. Analyse

- DES ist gegen eine differentielle Analyse robust.
- Den Erfindern von DES soll die differentielle Analyse schon bekannt gewesen sein.
- Um die vollen 16 Runden im DES mittels differentieller Analyse zu brechen, braucht man ca. 2^{47} Chiffretexte zu selbst gewählten Klartexten (Chosen-Plaintexts).

AES Historie

- 1997 wurde vom NIST ein Auswahlprozess gestartet um unsicheren DES Algorithmus zu ersetzen.
- Endauswahl viel auf fünf Kandidaten: MARS, RC6, Rijndael, Serpent und Twofish.
- 2000 wurde Rijndael (entwickelt von V. Rijmen und J. Daemen aus Belgien) als DES Nachfolger ausgewählt.
- AES ist eine Blockchiffre mit Alphabet $\mathbb{Z}_2$. Sie ist ein Spezialfall der Rijndael-Chiffre.

Rijndael-Chiffre Bezeichnungen

`Nb` Klartext- und Chiffretextblöcke bestehen aus `Nb` vielen 32-Bit Wörtern, $4 \leq \texttt{Nb} \leq 8$. Rijndael Blocklänge ist also $32 \cdot \texttt{Nb}$. Für AES ist $\texttt{Nb} = 4$, somit ist Blocklänge 128.

`Nk` Schlüssel bestehen aus `Nk` vielen 32-Bit Wörtern, $4 \leq \texttt{Nk} \leq 8$. Rijndael Schlüsselraum ist also $\mathbb{Z}_2^{32 \cdot \texttt{Nk}}$. Für AES ist ist $\texttt{Nk} = 4, 6$ oder 8. AES-Schlüsselraum ist also $\mathbb{Z}_2^{128}, \mathbb{Z}_2^{192}$, oder $\mathbb{Z}_2^{256}$.

`Nr` Anzahl der Runden.

$$\text{Für AES ist } \texttt{Nr} = \begin{cases} 10 & \text{für } \texttt{Nk} = 4, \\ 12 & \text{für } \texttt{Nk} = 6, \\ 14 & \text{für } \texttt{Nk} = 8. \end{cases}$$

Datentypen: `byte` : Bitvektor der Länge 8, `word` : Bitvektor der Länge 32.

Rijndael-Chiffre Bezeichnungen (Fort.)

- Klartext und Chiffretext werden als 2-dimensionale `byte`-Arrays dargestellt.
- Arrays haben 4 Zeilen und `Nb` Spalten.
- Klartext oder Chiffretext wird also wie folgt im AES dargestellt

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix}$$

- Jedes $s_{i,j}$ entspricht einem Byte also einem Bitvektor der Länge 8.
- Wie konstruiert man einen Körper mit 2^8 Elementen, so daß Bytes additiv und multiplikativ verarbeitet werden können?

Bytes als Elemente von GF(2^8)

- Bytes werden in der Rijndael-Chiffre mit Elementen des endlichen Körpers $GF(2^8)$ identifiziert.
- Erzeugendes Polynom ist das irreduzible Polynom $X^8 + X^4 + X^3 + X + 1$.
- Ein Byte $(b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)$ entspricht dem Elemente $\sum_{i=0}^{7} b_i X^i$.
- Bytes können somit multipliziert und addiert werden. Falls von Null verschieden ebenfalls invertiert werden.

Beispiel: Byte $b = (0, 0, 0, 0, 0, 0, 1, 1)$ entspricht dem Körperelement $X + 1$.
$(X + 1)^{-1} = X^7 + X^6 + X^5 + X^4 + X^2 + X$. Somit ist $b^{-1} = (1, 1, 1, 1, 0, 1, 1, 0)$.

Funktion Cipher

```
Cipher(byte in[4,Nb], byte out[4,Nb], word w[Nb * (Nr+1)]) {
  byte state[4,Nb]
  state = in // initialisiere den Zustand state als Klartext in
  AddRoundKey(state, w[0, Nb-1]) // berechne das XOR vom Zustand
                                 // mit dem Rundenschlüssel
  for round = 1 step 1 to Nr-1
    SubBytes(state) // Substitutionsoperation auf Zustand (S-Box)
    ShiftRows(state) // Permutation
    MixColumns(state) // Spaltenvertauschung
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
  end for

  SubBytes(state)
  ShiftRows(state)
  AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
  out = state
}
```

Funktion `SubBytes`

- `SubBytes` ist eine nicht-lineare Funktion die einzelne Bytes transformiert.
- Transformation wird S-Box genannt, diese garantiert Nicht-Linearität von AES.
- Aus jedem Byte b von `state` macht die S-Box das neue Byte

$$b \leftarrow Ab^{-1} \oplus c$$

mit

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}, \quad c = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Beispiel `SubBytes`

- Gegeben sei Vektor $b = (0, 0, 0, 0, 0, 0, 1, 1)$.
- Das Inverse von b ist $b^{-1} = (1, 1, 1, 1, 0, 1, 1, 0)$.
- Somit ist $Ab^{-1} \oplus c = (0, 1, 1, 0, 0, 1, 1, 1)$

Die S-Box enthält 2^8 mögliche Einträge und somit kann `SubBytes` z.B. durch Table-Lookups realisiert werden.

- Hierbei wird die Byte Eingabe in das erste und zweite Zeichen zerlegt.
- Das erste Zeichen bestimmt die Zeile, das zweite Zeichen die Spalte in der S-Box.
- Beispiel: Die Zeichenfolge 19 wird aufgeteilt in 1 und 9. Anschließend wird in der S-Box "nachgeschlagen" (Zeile 1, Spalte 9) und somit wird aus 19 der Substitutionswert $d4$.

Funktion `ShiftRows`

- Sei s ein `state` (ein durch AES teiltransformierter Klartext).
- Die Matrix hat hat 4 Zeilen und `Nb` Zeilen.
- `ShiftRows` wendet auf die Zeilen der Matrix einen zyklischen Linksshift an.
- Die i-te Teile wird um c_i Positionen nach links verschoben, wobei $c_0 = 0, c_1 = 1, c_2 = 2, c_3 = 3$ für $\texttt{Nb} = 4$ ist (siehe Tabelle 7.1 im Buch für $4 \leq \texttt{Nb} \leq 8$).

$$\begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix} \overset{\texttt{ShiftRows}}{\Longrightarrow} \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{pmatrix}$$

Funktion `MixColumns`

- Für $0 \leq j < \texttt{Nb}$ wird die Spalte

$$s_j = (s_{0,j}, s_{1,j}, s_{2,j}, s_{3,j})$$

von `state` mit dem Polynom

$$s_{0,j} + s_{1,j}x + s_{2,j}x^2 + s_{3,j}x^3 \in GF(2^8)[x]$$

identifiziert.

- Transformation `MixColumns` setzt

$$s_j \leftarrow (s_j \cdot a(x)) \mod (x^4 + 1), \quad 0 \leq j < \texttt{Nb}$$

wobei

$$a(x) = \{03\}\, x^3 + \{01\}\, x^2 + \{01\}\, x + \{02\}$$

Dies sorgt für eine Diffusion innerhalb der Spalten von `state`.

Funktion `MixColumns` (Fort.)

Funktion `MixColumns` kann auch als lineare Transformation beschrieben werden.

$$s_j \leftarrow \begin{pmatrix} \{02\} & \{03\} & \{01\} & \{01\} \\ \{01\} & \{02\} & \{03\} & \{01\} \\ \{01\} & \{01\} & \{02\} & \{03\} \\ \{03\} & \{01\} & \{01\} & \{02\} \end{pmatrix} s_j, \quad 0 \leq j \leq \texttt{Nb}.$$

Funktion `AddRoundKey`

- Sind $s_0, s_1, \ldots, s_{\mathtt{Nb}-1}$ die Spalten von state, dann setzt Aufruf der Funktion `AddRoundKey`

$$s_j \leftarrow s_j \oplus w[l \cdot \mathtt{Nb} + j], \quad 0 \leq j < \mathtt{Nb}$$

  wobei l die aktuelle Rundenzahl ist und w der expandierte Schlüssel ist.

- Der expandierte Schlüssel w besteht (AES, $\mathtt{Nb} = 4$) aus dem ersten Rundenschlüssel $w[0], w[1], w[2], w[3]$, dem zweiten Rundenschlüssel $w[4], w[5], w[6], w[7]$, usw. und wird mit der Funktion `KeyExpansion` erstellt.

Funktion `KeyExpansion`

- `KeyExpansion` macht aus dem Rijndael-Schlüssel `key` der ein `byte`-array der Länge $4 \cdot$ `Nk` ist, einen expandierten Schlüssel w.
- Schlüssel w ist ein `word`-array der Länge $\texttt{Nb} \cdot (\texttt{Nr} + 1)$
- Zuerst werden die ersten `Nk` Wörter im expandierten Schlüssel w mit den Bytes des Schlüssels `key` gefüllt.
- Anschließend werden die Funktionen `RotWord` und `SubBytes` angewendet.
- `RotWord` bewirkt eine zyklische Drehung der 4 Bytes (b_0, b_1, b_2, b_3), d.h.

$$\texttt{RotWord}(b_0, b_1, b_2, b_3) = (b_1, b_2, b_3, b_0).$$

- Das Resultat von `RotWord` und `SubBytes` wird anschließend mit dem konstanten Array `Rcon` (round constant) mit einer XOR-Operation verknüpft.

Funktion InvCipher

```
InvCipher(byte in[4,Nb], byte out[4,Nb], word w[Nb * (Nr+1)]) {
  byte state[4,Nb]
  state = in
  AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

  for round = Nr-1 step -1 downto 1
    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    InvMixColumns(state)
  end for

  InvShiftRows(state)
  InvSubBytes(state)
  AddRoundKey(state, w[0, Nb-1])
  out = state
}
```

`Inv` **Funktionen in** `InvCipher`

- `InvShiftRows`: Zyklischer Rechtsshift um c_i Positionen mit den Werten c_i aus Tabelle 7.1.
- `InvSubBytes`: Substitution wird mittels einer anderen S-Box rückgängig gemacht ($b \rightarrow (A_{-1}(b \oplus c))^{-1}$).
- `InvMixColumns`: Rückgängige lineare Transformation

$$s_j \leftarrow \begin{pmatrix} \{0e\} & \{0b\} & \{0d\} & \{09\} \\ \{09\} & \{0e\} & \{0b\} & \{0d\} \\ \{0d\} & \{09\} & \{0e\} & \{0b\} \\ \{0b\} & \{0d\} & \{09\} & \{0e\} \end{pmatrix} s_j, \quad 0 \leq j \leq \mathtt{Nb}.$$

RSA Verfahren

- RSA benannt nach den Erfindern Ron Rivest, Adi Shamir und Leonard Adleman war das erste Public-Key Verschlüsselungsverfahren.
- Sicherheit hängt eng mit der Schwierigkeit zusammen, große Zahlen in ihre Primfaktoren zu zerlegen.
- RSA Verfahren ist heute eines der wichtigsten Public-Key Verschlüsselungsverfahren.

RSA Schlüsselerzeugung

RSA-Schlüssel besteht aus zwei Komponenten, dem privaten und dem öffentlichen Schlüssel.

- Wähle nacheinander zufällig zwei Primzahlen p und q und berechne Produkt $n = pq$. Die Zahlen p, q müssen geheim gehalten und werden im eigentlich Ablauf des RSA-Verfahrens nicht mehr gebraucht.
- Wähle natürliche Zahl e mit

  $1 < e < \phi(n) = (p-1)(q-1)$ und $\gcd(e, (p-1)(q-1)) = 1$.

- Berechne natürliche Zahl d mit

  $1 < d < (p-1)(q-1)$ und $de \equiv 1 \mod (p-1)(q-1)$.

Wir wir bereits gesehen haben, gibt es eine solche Zahl d weil $\gcd(e, (p-1)(q-1)) = 1$ ist.

RSA Schlüsselerzeugung (Fort.)

- Öffentliche Schlüssel besteht aus dem Paar (n, e).
- Private Schlüssel ist d.
- Zahl n heißt RSA-Modul,
  e heißt Verschlüsselungsexponent,
  d heißt Entschlüsselungsexponent.

Beispiel:
Sei $p = 11$ und $q = 23$. Somit ist $n = 253$ und
$(p-1)(q-1) = 10 \cdot 22$. Das kleinstmögliche e ist $e = 3$.
Erweiterte euklidische Algorithmus liefert $d = 147$.

RSA Verschlüsselung

- Der Klartextraum bestehe aus allen natürlichen Zahlen m mit $0 \leq m < n$.
- Klartext m wird verschlüsselt zu

$$c = m^e \mod n.$$

- Jeder, der öffentlichen Schlüssel (n, e) kennt, kann Verschlüsselung durchführen.

Beispiel:
Sei $n = 253$ und $e = 3$. Klartextraum ist $\{0, 1, \ldots, 252\}$. Zahl $m = 165$ wird zu $165^3 \mod 253 = 110$ verschlüsselt.

RSA Verschlüsselung (Blockchiffre)

Man kann mit der RSA-Verschlüsselungsmethode auch eine Art Blockchiffre realisieren.

- Das verwendete Alphabet Σ habe genau N Zeichen, wobei die Zeichen die Zahlen $0, 1, \ldots, N-1$ sind.
- Man setzt
$$k = \lfloor \log_N n \rfloor.$$
- Block $m_1 m_2 \ldots m_k$, $m_i \in \Sigma, 1 \leq i \leq k$, wird in die Zahl
$$m = \sum_{i=1}^{k} m_i N^{k-i}$$
verwandelt.

Block m wird verschlüsselt, indem $c = m^e \mod m$ bestimmt wird. Zahl c kann wieder zur Basis N geschrieben werden.

RSA Verschlüsselung (Blockchiffre) (Fort.)

- N-adische Entwicklung von c kann aber die Länge $k+1$ haben.
- Man schreibt

$$c = \sum_{i=0}^{k} c_i N^{k-i}, \quad c_i \in \Sigma,\ 0 \leq i \leq k.$$

- Chiffretext ist dann $c = c_0 c_1 \ldots c_k$.

Beispiel: Sei $\Sigma = \{0, a, b, c\}$ mit Entsprechung

0	a	b	c
0	1	2	3

bei Verwendung von $n = 253$ ist $k = \lfloor \log_4 253 \rfloor = 3$. Das ist die Klartextblocklänge.

RSA Verschlüsselung (Blockchiffre) (Fort.)

Länge der Chiffretextblöcke ist demnach 4.

- Es soll Klartextblock abb verschlüsselt werden.
- Dieser entspricht dem String 122 und damit der Zahl

$$m = 1 \cdot 4^2 + 2 \cdot 4^1 + 2 \cdot 4^0 = 26.$$

- Zahl m wird zu

$$c = 26^3 \mod 253 = 119$$

  verschlüsselt.
- Schreibt man diese zur Basis 4 erhält man

$$c = 1 \cdot 4^3 + 3 \cdot 4^2 + 1 \cdot 4^1 + 3 \cdot 4^0$$

  und somit Chiffretextblock $acac$.

RSA Entschlüsselung

Sei (n, e) ein öffentlicher Schlüssel und d der entsprechende private Schlüssel im RSA-Verfahren. Dann gilt

$$(m^e)^d \mod n = m$$

für jede natürliche Zahl m mit $0 \leq m < n$.

Da $ed \equiv 1 \mod (p-1)(q-1)$ ist, gibt es eine ganze Zahl k, so daß

$$ed = 1 + k(p-1)(q-1)$$

ist. Falls p kein Teiler von m ist ($m \not\equiv 0 \mod p$) gilt

$$\begin{aligned} (m^e)^d &\equiv m(m^{p-1})^{k(q-1)} \mod p \\ &\equiv m\,1^{k(q-1)} \mod p \\ &\equiv m \mod p \end{aligned}$$

RSA Entschlüsselung (Fort.)

Falls p Teiler von m ist ($m \equiv 0 \mod p$) gilt offensichtlich

$$m^{ed} \equiv m \mod p$$

für all m mit $0 \leq m < n$.

Analog zeigt man

$$m^{ed} \equiv m \mod q$$

Aus dem chinesischen Restsatz folgt dann

$$m^{ed} \equiv m \mod n.$$

Wurde $c = m^e \mod m$ berechnet, kann man m mittels $m = c^d \mod n$ rekonstruieren.

Sicherheit des geheimen Schlüssels

- Gegeben der öffentliche Schlüssel (n, e), ist es "möglich" den geheimen Schlüssel d zu berechnen?
- Wir werden sehen, daß die Bestimmung des geheimen Schlüssels d aus dem öffentlichen Schlüssel (n, e) genauso schwierig ist, wie die Zerlegung des RSA Moduls n in seine Primfaktoren.
- D.h. finden des geheimen Schlüssels lässt sich reduzieren auf das Faktorisierungsproblem für natürliche Zahlen und umgekehrt.

Faktorisierungsproblem für ganze Zahlen

Gegeben eine positive ganze Zahl n. Finde Primfaktoren $p_1^{e_1} p_2^{e_2} \dots p_k^{e_k} = n$ wobei p_i paarweise verschieden sind und $e_i \geq 1$.

Sicherheit von RSA

- Kennt ein Angreifer Primfaktoren p und q des RSA-Moduls n, kann er geheimen Schlüssel d durch lösen der Kongruenz

$$de \equiv 1 \mod (p-1)(q-1)$$

  berechnen.

- Umkehrung gilt auch. Kennt man n, e, d so kann man Faktoren p und q berechnen. Hierzu setzt man

$$s = \max\{t \in \mathbb{N} : 2^t \text{ teilt } ed-1\} \text{ und } k = (ed-1)/2^s.$$

Sicherheit von RSA (Fort.)

- Der Algorithmus, der n faktorisiert beruht dann auf folgender Aussage

Sei a eine zu n teilerfremde ganze Zahl. Wenn die Ordnung von $a^k \mod p$ und $\mod q$ verschieden ist, so ist $1 < \mathsf{gcd}(a^{2^t k} - 1, n) < n$ für ein $t \in \{0, 1, 2, \ldots, s-1\}$.

Um n zu faktorisieren, wählt man zufällig und gleichverteilt eine Zahl a aus $\mathbb{Z}_n^*$.

- Berechne $g = \mathsf{gcd}(a, n)$. Falls $g > 1$, dann ist g ein echter Teiler von n.
- Falls $g = 1$, dann berechne

$$g = \mathsf{gcd}(a^{2^t k}, n), \quad t = s-1, s-2, \ldots, 0.$$

Sicherheit von RSA (Fort.)

- Findet man einen Teiler im vorherigen Schritt der kleiner n ist, dann terminiere und gebe diesen Teiler aus.
- Andernfalls wird ein neues a gewählt und die Schritte wiederholt.
- Die Wahrscheinlich das ein Primteiler von n gefunden wird, ist in jeder Iteration wenigstens $1/2$.

Beispiel: Sei $n = 253, e = 3$ und $d = 147$. Somit ist $ed - 1 = 440$.

$2^0, 2^1, 2^2, 2^3$ teilt $ed - 1 = 440$ und somit ist $s = 3$ und $k = 440/8 = 55$.

Wenn man $a = 2$ verwendet, so erhält man $\gcd(2, 253) = 1$ und somit $t = 2, 1, 0$ so das $\gcd(2^{220} - 1, 253) = 253, \gcd(2^{110} - 1, 253) = 253$. Aber $\gcd(2^{55} - 1, 253) = 23$ und somit $253 = 23 \cdot 11$.

Auswahl von Verschlüsselungsexponent e

- Wahl von $e = 2$ wird ausgeschlossen, weil $\phi(n) = (p-1)(q-1)$ gerade ist und $\gcd(e, (p-1)(q-1)) = 1$ gelten muß.
- Kleinst mögliche Exponent ist also $e = 3$.
- Verwendet man $e = 3$, so kann man mit einer Quadrierung und einer Multiplikation $\mod n$ verschlüsseln.

Beispiel: Sei $n = 253, e = 3, m = 165$. Man berechnet $m^3 \mod n = ((m^2 \mod n) \cdot m) \mod n = 154 \cdot 165 \mod 253 = 110$.

Low-Exponent-Attacke

Low-Exponent-Attacke beruht auf folgendem Satz:

Sei $e \in \mathbb{N}$ und $n_1, n_2, \ldots, n_e \in \mathbb{N}$ paarweise teilerfremd und $m \in \mathbb{N}$ mit $0 \leq m < n_i$, $1 \leq i \leq e$. Sei $c \in \mathbb{N}$ mit $c \equiv m^e \mod n_i$, $1 \leq i \leq e$, und $0 \leq c < \prod_{i=1}^{e} n_i$. Dann folgt $c = m^e$.

- Denkbares Szenario: Bank schickt an e verschiedene Kunden dieselbe verschlüsselte Nachricht.
- Hierbei werden die verschiedenen öffentlichen Schlüssel n_i, $1 \leq i \leq e$ der Kunden benutzt.
- Angreifer kennt Chiffretext $c_i = m^e \mod n_i$, $1 \leq i \leq n$. Mittels chinesischen Restsatz berechnet er c mit $c \equiv c_i \mod n_i$, $1 \leq i \leq e$ und $0 \leq c < \prod_{i=1}^{e} n_i$.

Low-Exponent-Attacke (Beispiel)

Sei $e = 3, n_1 = 143, n_2 = 391, n_3 = 899, m = 135$. Dann ist $c_1 = 60, c_2 = 203, c_3 = 711$.

Angreifer berechnet

$$\begin{array}{rcl} c & \equiv & 60 \mod 143 \\ c & \equiv & 203 \mod 391 \\ c & \equiv & 711 \mod 899 \end{array} \quad \text{und erhält} \quad \begin{array}{rcl} 2460375 & \equiv & 60 \mod 143 \\ 2460375 & \equiv & 203 \mod 391 \\ 2460375 & \equiv & 711 \mod 899 \end{array}$$

mit $0 \leq 2460375 < n_1 \cdot n_2 \cdot n_3 = 50265787$. Nach Satz gilt $c = m^e$ und somit $m = 2460375^{1/3} = 135$.

- Möglichkeit um Low-Exponent-Attacke zu verhindern ist größere Verschlüsselungsexponenten zu wählen. Üblich ist $e = 2^{16} + 1$.

Rabin Verschlüsselung

Schlüsselerzeugung:

- Alice wählt zufällig zwei Primzahlen p und q mit $p \equiv q \equiv 3 \mod 4$.
- Kongruenzbedingung vereinfacht und beschleunigt die Entschlüsselung. Aber auch ohne diese Kongruenzbedingung funktioniert das Verfahren.
- Alice berechnet $n = pq$. Öffentlicher Schlüssel ist n. Geheimer Schlüssel ist (p, q).

Verschlüsselung:

- Klartextraum ist Menge $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$.
- Bob benutzt öffentlichen Schlüssel von Alice und verschlüsselt Nachricht m wie folgt

$$c = m^2 \mod n.$$

Rabin Entschlüsselung

Alice berechnet Klartext m aus Chiffretext c durch Wurzelziehen wie folgt

$$m_p = c^{(p+1)/4} \mod p, \quad m_q = c^{(q+1)/4} \mod q.$$

- Dann sind $\pm m_p + p\mathbb{Z}$ die beiden Quadratwurzel von $c + p\mathbb{Z}$ in $\mathbb{Z}/p\mathbb{Z}$, sowie $\pm m_p + p\mathbb{Z}$ die beiden Quadratwurzel von $c + q\mathbb{Z}$ in $\mathbb{Z}/q\mathbb{Z}$.
- Die vier gesuchten Quadratwurzeln von $c + n\mathbb{Z}$ in $\mathbb{Z}/n\mathbb{Z}$ kann Alice nun mittels des chinesischen Restsatzes berechnen.
- Hierzu bestimmt Alice die Koeffizienten $y_p, y_q \in \mathbb{Z}$ mit $y_p p + y_q q = 1$.

Rabin Entschlüsselung (Fort.)

- Alice berechnet dann

$$r = (y_p\, p\, m_q + y_q\, q\, m_p) \mod n,$$

$$s = (y_p\, p\, m_q - y_q\, q\, m_p) \mod n.$$

- Die Quadratwurzeln von $c \mod n$ sind dann

$$(r, -r \mod n, s, -s \mod n)$$

  Einer dieser Quadratwurzeln ist dann der Klartext.

Rabin Verfahren Beispiel

- Alice wählt geheimen Schlüssel $p = 271, q = 331$ und somit öffentlichen Schlüssel $n = 89701$.
- Ferner gilt $p \equiv q \equiv 3 \mod 4$.
- Nachricht sei $\overline{m} = 1001111001$. Bob repliziert die letzten 6 Bits von $\overline{m}$ und erhält Nachricht

$$m = 1001111001111001 = 40569_{10}.$$

- Bob berechnet $c = m^2 \mod n = 40569^2 \mod 89701 = 9813$ und schickt c an Alice.
- Alice berechnet

$$m_p = c^{(p+1)/4} \mod p = 9813^{272/4} \mod 271 = 81$$
$$m_q = c^{(q+1)/4} \mod q = 9813^{332/4} \mod 331 = 144$$

und erhält die vier Quadratwurzel $(81, -81), (144, -144)$.

Rabin Verfahren Beispiel (Fort.)

- Mittels des erweiterten euklidischen Algorithmus bestimmt Alice die Koeffizienten $y_p, y_q \in \mathbb{Z}$ mit

$$y_p\, p + y_q\, q = 1 = -160 \cdot 271 + 131 \cdot 331.$$

- Anschließend berechnet Alice r und s

$$\begin{aligned} r &= (y_p\, p\, m_q + y_q\, q\, m_p) \mod n \\ &= (-160 \cdot 271 \cdot 144 + 131 \cdot 331 \cdot 81) \mod 89701 = 49132 \end{aligned}$$

$$\begin{aligned} s &= (y_p\, p\, m_q - y_q\, q\, m_p) \mod n \\ &= (-160 \cdot 271 \cdot 144 - 131 \cdot 331 \cdot 81) \mod 89701 = 21328. \end{aligned}$$

Rabin Verfahren Beispiel (Fort.)

- Die vier Quadratwurzeln von $9813 \mod 89701$ sind somit

$$\begin{aligned} 49132, 40569 &= -49132 \mod 89701 \\ 21328, 68373 &= -21328 \mod 89701 \end{aligned}$$

$$\begin{aligned} 49132^2 \mod 89701 &= 9813 \\ 40569^2 \mod 89701 &= 9813 \\ 21328^2 \mod 89701 &= 9813 \\ 68373^2 \mod 89701 &= 9813 \end{aligned}$$

Rabin Verfahren Beispiel (Fort.)

$$
\begin{aligned}
49132_{10} &= 1011111111101100 = m_1 \\
40569_{10} &= 1001111001111001 = m_2 \\
21328_{10} &= 1010011010100000 = m_3 \\
68373_{10} &= 1000010110001010 1 = m_4
\end{aligned}
$$

- Nur Nachricht m_2 hat die 6-Bit Redundanz und ist somit die gesuchte Nachricht.

Sicherheit des Rabin Verfahrens

- Sicherheit des Rabins Verfahren beruht auf der Schwierigkeit quadratische Wurzeln modulo n zu berechnen.

Quadratwurzeln Modulo n Problem

Gegeben eine zusammengesetzte ganze Zahl n und $a \in Q_n$ (Q_n ist Menge der quadratischen Reste modulo n). Finde Quadratwurzel von $a \mod n$, d.h. eine ganze Zahl x mit $x^2 \equiv a \mod n$.

Sicherheit des Rabin Verfahrens (Fort.)

- Sicherheit des Rabin-Verfahrens gegen Ciphertext-Only-Angriffe ist äquivalent zur Schwierigkeit des Faktorisierungsproblems.
- Jeder den öffentlichen Modul n faktorisieren kann, kann auch das Rabin-Verfahren brechen
- Die Umkehrung gilt auch.

Angenommen, mann kann Rabin-Verfahren brechen, d.h. zu jedem Quadrat $c + n\mathbb{Z}$ kann man eine Quadratwurzel $m + n\mathbb{Z}$ bestimmen, sagen wir mit einem Algorithmus R.

Algorithmus R liefert bei Eingabe von $c \in \{0, 1, \ldots, n-1\}$ eine ganze Zahl $m = R(c) \in \{0, 1, \ldots, n-1\}$ für die die Restklasse $m + n\mathbb{Z}$ eine Quadratwurzel von $c + n\mathbb{Z}$ ist.

Sicherheit des Rabin Verfahrens (Fort.)

- Um n zu faktorisieren, wählt man zufällig und gleichverteilt eine Zahl $x \in \{1, 2, \ldots, n-1\}$.
- Wenn $\gcd(x, n) \neq 1$, dann hat man Modul n faktorisiert.
- Andernfalls berechnet man

$$c = x^2 \mod n \text{ und } m = R(c).$$

- Restklasse $m + n\mathbb{Z}$ ist eine Quadratwurzel von $c + n\mathbb{Z}$, diese muß aber nicht mit $x + n\mathbb{Z}$ übereinstimmen.
- Jedoch erfüllt m eine der folgenden Bedingungspaare

$$\begin{aligned}
m &\equiv x \mod p \quad \text{und} \quad m \equiv x \mod q, && (1)\\
m &\equiv -x \mod p \quad \text{und} \quad m \equiv -x \mod q, && (2)\\
m &\equiv x \mod p \quad \text{und} \quad m \equiv -x \mod q, && (3)\\
m &\equiv -x \mod p \quad \text{und} \quad m \equiv x \mod q, && (4)
\end{aligned}$$

Sicherheit des Rabin Verfahrens (Fort.)

- Im Fall (1) ist $m = x$ und gcd$(m - x, n) = n$.
- Im Fall (2) ist $m = n - x$ und gcd$(m - x, n) = 1$.
- Im Fall (3) ist gcd$(m - x, n) = p$.
- Im Fall (4) ist gcd$(m - x, n) = q$.

Da x zufällig und gleichverteilt gewählt wurde ist bei einem Durchlauf des Verfahrens die Zahl n mit der Wahrscheinlichkeit $\geq 1/2$ faktorisierbar.

In k Durchläufen des Verfahrens wird n mit Wahrscheinlichkeit $\geq 1 - (1/2)^k$ zerlegt.

Beispiel Rabin Verfahren und Faktorisierung

- Sei $n = 253$, wir wählen zufällig und gleichverteilt z.B. $x = 17$. Es ist $\gcd(17, 253) = 1$.
- Wir bestimmen $c = 17^2 \mod 253 = 36$ und lassen von unserem Algorithmus die Werte $R(c)$ bestimmen.
- Die Quadratwurzeln von $36 \mod 253$ sind $6, 17, 236, 247$.
- Es ist $\gcd(6 - 17, 253) = 11$ und $\gcd(247 - 17, 253) = 23$.

Diskrete Logarithmen

- Sei p eine Primzahl. Wir wissen, daß $\mathbb{Z}_p^*$ zyklisch ist und Ordnung $p-1$ hat.
- Sei g eine Primitivwurzel $\mod p$. Dann gibt es für jede Zahl $A \in \{1, 2, \ldots, p-1\}$ einen Exponenten $a \in \{0, 1, 2, \ldots, p-2\}$ mit

$$A \equiv g^a \mod p.$$

- Exponent a heißt *diskreter Logarithmus* von A zur Basis g.
- Man schreibt $a = \log_g A$.
- Berechnung solcher diskreten Logarithmen gilt als schwieriges Problem. Keine effizienten Algorithmen sind bis heute bekannt.

ElGamal Verfahren (Schlüsselerzeugung)

Schlüsselerzeugung:

- Alice wählt Primzahl p und eine Primitivwurzel $g \mod p$. Dann wählt sie zufällig und gleichverteilt einen Exponenten $a \in \{0, 1, \ldots, p-2\}$ und berechnet

$$A = g^a \mod p.$$

- Öffentlicher Schlüssel ist (p, g, A). Geheimer Schlüssel ist Exponent a.

ElGamal Verfahren (Verschlüsselung)

- Klartextraum ist Menge $\{0, 1, \ldots, p-1\}$.
- Bob besorgt sich öffentlichen Schlüssel von Alice (p, g, A) und verschlüsselt Klartext m wie folgt
  - Zuerst wählt er eine Zufallszahl $b \in \{1, 2, \ldots, p-2\}$ und berechnet

$$B = g^b \mod p.$$

  - Dann bestimmt Bob

$$c = A^b m \mod p.$$

  - Der Chiffretext ist (B, c).

ElGamal Verfahren (Entschlüsselung)

- Alice erhält den Chiffretext (B, c) und kennt ihren geheimen Schlüssel a und entschlüsselt wie folgt
- Alice bestimmt Exponenten $x = p - 1 - a$. Weil $1 \leq a \leq p - 2$ ist, gilt $1 \leq x \leq p - 2$.
- Dann berechnet Alice $B^x c \mod = m$. Dies ist der Klartext weil

$$\begin{aligned} B^x c \equiv g^{b(p-1-a)} A^b m &\equiv (g^{p-1})^b (g^a)^{-b} A^b m \\ &\equiv A^{-b} A^b m \equiv m \mod p. \end{aligned}$$

ElGamal Verfahren (Sicherheit)

- Sicherheit des ElGamals Verfahren beruht auf der Schwierigkeit diskrete Logarithmen $\mod p$ zu berechnen.

Diskrete Logarithmus Problem (DLP)

Gegeben eine Primzahl p, eine Primitivwurzel α von $\mathbb{Z}_p^*$ und eine Zahl $\beta \in \mathbb{Z}_p^*$. Finde Zahl x, $0 \leq x \leq p-2$ mit $\alpha^x \equiv \beta \mod p$.

- Könnte man effizient den diskreten Logarithmus berechnen, dann könnte man aus A den geheimen Exponenten a ermitteln und

$$m = B^{p-1-a}c \mod p$$

berechnen.

Diffie-Hellman Schlüsselaustausch

Problemstellung:

- Bob und Alice wollen sich auf gemeinsamen Schlüssel K einigen, haben aber nur unsicheren Kommunikationskanal zur Verfügung.

Lösung des Problems:

- Bob und Alice einigen sich auf eine Primzahl p und eine Primitivwurzel $g \mod p$ mit $2 \leq g \leq p-2$.
- Primzahl p und Primitivwurzel g können öffentlich bekannt sein.
- Alice wählt natürliche Zahl $a \in \{0, 1, \ldots, p-2\}$ zufällig und berechnet

$$A = g^a \mod p$$

und schickt A an Bob, hält aber ihren Exponenten a geheim.

Diffie-Hellman Schlüsselaustausch (Fort.)

- Bob wählt natürliche Zahl $b \in \{0, 1, \ldots, p-2\}$ zufällig und berechnet

$$B = g^b \mod p$$

  und schickt B an Alice, hält aber seinen Exponenten b geheim.

- Alice berechnet

$$B^a \mod p = g^{ab} \mod p.$$

- Bob berechnet

$$A^b \mod p = g^{ab} \mod p.$$

- Gemeinsamer Schlüssel ist

$$K = g^{ab} \mod p.$$

Sicherheit des DH-Schlüsselaustauschs

- Angreifer erfährt die Zahlen p, g, A, B, aber nicht a und nicht b.

Diffie-Hellman Problem (DHP)

Gegeben eine Primzahl p, eine Primitivwurzel g von $\mathbb{Z}_p^*$ und die Zahlen $g^a \mod p$ und $g^b \mod p$. Finde $g^{ab} \mod p$.

- Falls Angreifer das DLP lösen kann, dann kann er auch das DHP lösen.
- Diffie-Hellman Schlüsselaustausch ist angreifbar mit der "Man-In-The-Middle-Attacke".
  - Angreifer (in der Mitte) tauscht sowohl mit Alice als auch mit Bob einen geheimen Schlüssel aus.
  - Alice glaubt, der Schlüssel komme von Bob und Bob glaubt, der Schlüssel komme von Alice.

Zero-Knowledge-Beweise

Problem: Beweiser Bob will Verifizierer(in) Alice nachweisen, daß er ein Geheimnis kennt ohne es ihr zu verraten.

- In Zero-Knowledge-Beweisen kann man mathematisch beweisen, daß der Verifizierer nichts über das Geheimnis erfährt, obwohl er am Ende des Protokolls von der Identität des Beweisers überzeugt ist.

In dem nachfolgenden Verfahren, überzeugt Beweiser Bob Alice davon, daß er eine Quadratwurzel kennt, ohne jedoch diese zu verraten.

Fiat-Shamir Verfahren

- Beweiser Bob wählt zwei Primzahlen p und q und berechnet $n = pq$.
- Er wählt außerdem eine Zahl s zufällig und gleichverteilt aus $\mathbb{Z}_n^*$ und berechnet $v = s^2 \mod n$.
  - Bobs öffentlicher Schlüssel ist (v, n). Sein geheimer Schlüssel ist s.

1. *Commitment:* Bob wählt zufällig und gleichverteilt $r \in \{1, 2, \ldots, n-1\}$ und berechnet $x = r^2 \mod n$. Ergebnis x sendet er an Alice.
2. *Challenge:* Alice wählt zufällig und gleichverteilt $e \in \{0, 1\}$ und sendet e an Bob.
3. *Response:* Bob schickt $y = r \cdot s^e \mod n$ an Alice.

Alice verifiziert, ob $y^2 \equiv x \cdot v^e \mod n$ ist.

Fiat-Shamir Verfahren (Beispiel)

Sei $n = 391 = 17 \cdot 23$. Bobs geheimer Schlüssel ist $s = 123$, sein öffentlicher Schlüssel ist $(271, 391)$.

- Bob will also Alice nachweisen, daß er eine Quadratwurzel von $v \mod n$ kennt, aber ohne diese zu verraten.
- Bob wählt $r = 271$ und schickt $x = r^2 \mod n = 324$ an Alice.
- Alice schickt "challenge" $e = 1$ an Bob.
- Bob schickt "response" $y = r \cdot s \mod n = 98$ an Alice.
- Alice verifiziert $220 = y^2 \equiv x \cdot v \mod n$.

Analyse des Fiat-Shamir Verfahrens

- Wenn Bob das Geheimnis s kennt, dann kann er beide möglichen Fragen $(e = 0, e = 1)$ richtig beantworten.
- Betrüger Oskar führt Protokoll mit Alice durch.
  - Er kann jedoch nicht *beide* Fragen richtig beantworten.
  - Würde Oskar Zahlen r und $r \cdot s$ kennen (beide Fragen richtig beantworten), für die $r^2 \equiv x \mod n$ und $y^2 \equiv x \cdot v \mod n$ gilt, dann könnte er Quadratwurzel $s = r \cdot s \cdot r^{-1} \mod n$ berechnen.

Analyse des Fiat-Shamir Verfahrens (Fort.)

Mit Wahrscheinlich $1/2$ kann Oskar die Challenge richtig beantworten.

- Oskar vermutet, daß $e = 0$ gesendet wird und wählt

$$x = r^2 \mod n \quad \text{und} \quad y = r.$$

  Alice akzeptiert weil $y^2 = r^2 \equiv \underbrace{x \cdot v^0}_{r^2} \mod n$.

- Diesmal vermutet Oskar $e = 1$ und wählt

$$x = r^2 v^{-1} \mod n \quad \text{und} \quad y = r.$$

  Alice akzeptiert weil $y^2 = r^2 \equiv r^2 v^{-1} v \mod n$.

Beachten sie: bevor die Challenge e von Alice kommt, muß sich Oskar schon festgelegt haben.

Analyse des Fiat-Shamir Verfahrens (Fort.)

- Wahrscheinlichkeit das Oskar in k aufeinander folgenden Iterationen des Protokolls erfolgreich betrügt ist $1/2^k$.
- Somit ist Alice nach k erfolgreichen Versuchen mit Wahrscheinlichkeit $1 - 1/2^k$ davon überzeugt, daß Bob tatsächlich Bob ist.

ZK-Eigenschaft des Fiat-Shamir Verfahrens

ZK-Eigenschaft garantiert, daß Alice die im Protokoll ausgetauschten Nachrichten auch ohne Beteiligung von Bob so simulieren kann, daß die Verteilung der simulierten Nachrichten nicht von der Verteilung der echten Nachrichten unterschieden werden kann.

- Im Fiat-Shamir Protokoll entsteht in jedem Durchlauf ein Nachrichtentripel (x, e, y).
  - x : Gleichverteilt zufälliges Quadrat modulo n aus $\{0, 1, \ldots, n-1\}$.
  - e : Zahl aus $\{0, 1\}$, die gemäß einer von Alice bestimmten Verteilung ausgewählt wird.
  - y : Gleichverteilt zufällige Quadratwurzel von xs^e aus $\{0, 1, \ldots, n-1\}$.

ZK-Eigen. des Fiat-Shamir Verfahrens (Fort.)

- Tripel (x, e, y) mit derselben Verteilung können wie folgt simuliert werden.
- Simulator wählt eine gleichverteilt zufällige Zahl $\tilde{e}$ aus $\{0, 1\}$ sowie gleichverteilt zufällige Zahl $y \in \{0, 1, \ldots, n-1\}$.
- Simulator berechnet dann $x = y^2 v^{-\tilde{e}} \mod n$, wobei $v^{-\tilde{e}}$ das Inverse von $v^{\tilde{e}}$ modulo n ist.
- Simulator wählt gemäß der von Alice ausgesuchten Verteilung eine Zahl $e \in \{0, 1\}$.
- Falls $e = \tilde{e}$ dann gibt Simulator Tripel (x, e, y) aus. Dann gilt $x = y^2 v^{-\tilde{e}} \mod n$. Andernfalls verwirft der Simulator das Tripel (x, e, y).

Feige-Fiat-Shamir Verfahren

- Beweiser Bob wählt zwei Primzahlen p und q und berechnet $n = pq$.
- Er wählt außerdem $s_1, s_2, \ldots, s_k$ zufällig und gleichverteilt aus $\{1, 2, \ldots, n-1\}^k$ und berechnet $v_i = s_i^2 \mod n$, $1 \leq i \leq k$.
  - Bobs öffentlicher Schlüssel ist $(n, v_1, v_2, \ldots, n_k)$. Sein geheimer Schlüssel ist $(s_1, s_2, \ldots, s_k)$.

1. *Commitment:* Bob wählt zufällig und gleichverteilt $r \in \{1, 2, \ldots, n-1\}$ und berechnet $x = r^2 \mod n$. Ergebnis x sendet er an Alice.
2. *Challenge:* Alice wählt zufällig und gleichverteilt $\mathbf{e} = (e_1, e_2, \ldots, e_k) \in \{0, 1\}^k$ und sendet $\mathbf{e}$ an Bob.
3. *Response:* Bob schickt $y = r \prod_{i=1}^{k} s_i^{e_i} \mod n$ an Alice. Alice verifiziert, ob $y^2 \equiv x \prod_{i=1}^{k} v_i^{e_i} \mod n$ ist.

Hashfunktionen und Kompressionsfunktionen

- Hashfunktion ist eine Abbildung

$$h : \Sigma^* \to \Sigma^n, \quad n \in \mathbb{N}.$$

- Hashfunktionen bilden beliebig lange Strings auf Strings fester Länge ab und sind nie *injektiv*.

Beispiel: Abbildung die jedem Wort $b_1 b_2 \ldots b_k$ aus $\{0,1\}^*$ die Zahl $b_1 \oplus b_2 \oplus \ldots \oplus b_k$ zuordnet.

- Eine Kompressionsfunktion ist eine Abbildung

$$h : \Sigma^m \to \Sigma^n, \quad n, m \in \mathbb{N}, \quad m > n.$$

- Kompressionsfunktion bildet Strings einer festen Länge auf Strings fester kürzerer Längen ab.

Hashfkt. und Kompressionsfkt. (Fort.)

Beispiel: Abbildung die jedem Wort $b_1 b_2 \ldots b_m$ aus $\{0,1\}^m$ die Zahl $b_1 \oplus b_2 \oplus \ldots \oplus b_m$ zuordnet, ist Kompressionsfunktion, wenn $m > 1$ ist.

- Definitionsbereich von h ist D, somit ist $D = \Sigma^*$, wenn h eine Hashfunktion ist, und $D = \Sigma^m$, wenn h eine Kompressionsfunktion ist.
- Man verlangt, daß Wert $h(x)$ für alle $x \in D$ effizient berechenbar ist.
- Funktion h heißt *Einwegfunktion*, wenn es "praktisch unmöglich" ist, zu einem $s \in \Sigma^n$ ein $x \in D$ mit $h(x) = s$ zu finden.

Hashfkt. und Kompressionsfkt. (Fort.)

Beispiel einer Einwegfunktion: Ist p eine zufällig gewählte 1024-Bit-Primzahl und g eine Primitivwurzel $\mod p$, dann ist Funktion

$$f : \{0, 2, 3, \ldots, p-1\} \to \{1, 2, \ldots, p-1\}, x \mapsto g^x \mod p$$

eine Einwegfunktion, weil kein effizientes Verfahren zur Berechnung diskreter Logarithmen bekannt ist.

- Eine Kollision von h ist ein Paar $(x, x') \in D^2$ von Strings, für die $x \neq x'$ und $h(x) = h(x')$ gilt.
- Alle Hashfunktionen und Kompressionsfunktionen besitzen Kollisionen, weil sie nicht injektiv sind.

Kollisionsresistente Funktionen

Die Funktion h heißt *schwach kollisionsresistent*, wenn es praktisch unmöglich ist, für ein *vorgegebenes* $x \in D$ ein Kollision (x, x') zu finden.

Die Funktion h heißt *(stark) kollisionsresistent*, wenn es praktisch unmöglich ist, *irgendeine* Kollision (x, x') von h zu finden.

Geburtstagsattacke

- Angriff gilt der starken Kollisionsresistenz von h. Hierbei versucht man eine Kollision von $h : \Sigma^* \to \Sigma^n$ zu finden.

Hierbei nutzt man das sog. Geburtstagsparadoxon.

Wieviele Leute müssen in einem Raum sein, damit man eine gute Chance hat, daß wenigstens zwei von ihnen am gleichen Tag Geburtstag haben.

- Angenommen es gibt n verschiedene Geburtstage und k sein die Anzahl der Personen in einem Raum.
- Ein ähnliches aber dennoch anderes Problem ist das folgende:
  - Wie groß ist Wahrscheinlichkeit, daß unter k Personen mindestens eine dabei ist, die am gleichen Tag Geburtstag hat wie man selbst.

Geburtstagsattacke (Fort.)

- In einem n-Tag Jahr ist die Wahrscheinlich das eine Person an einem bestimmten Tag *nicht* Geburtstag hat $(n-1)/n$.
- Betrachtet man die Geburtstage von k Personen als unabhängig, so ist die Wahrscheinlichkeit $((n-1)/n)^k$.
- Somit ist die Wahrscheinlichkeit mit der mindestens eine Person unter k Personen an einem bestimmten Tag Geburtstag hat

$$\widetilde{P}_n(k) = 1 - \left(\frac{n-1}{n}\right)^k = 1 - \left(1 - \frac{1}{n}\right)^k$$

- Nun kann man ausrechnen wie viele Personen k man braucht, um eine bestimmte Wahrscheinlichkeit zu erreichen, daß mindestens eine Person an *einem bestimmten* Tag (z.B. wie man selbst) Geburtstag hat.

Geburtstagsattacke (Fort.)

Für eine vorgegebene Wahrscheinlich von p erhält man

$$p = 1 - \left(\frac{n-1}{n}\right)^k \Longleftrightarrow k = \frac{\ln(1-p)}{\ln\left(\frac{n-1}{n}\right)}.$$

- Für eine Wahrscheinlich von $p = 1/2$ benötigt man für $n = 365$ also

$$k = \frac{\ln\left(\frac{1}{2}\right)}{\ln\left(\frac{364}{365}\right)} = 252.65\ldots \approx 253 \text{ Personen.}$$

- Die richtige Interpretation unseres Ursprungsproblems lautet jedoch: Wie groß ist die Wahrscheinlichkeit, daß mindestens zwei Personen an einem Tag Geburtstag haben.
- D.h. wir fixieren nicht einen bestimmten Tag.

Geburtstagsattacke (Fort.)

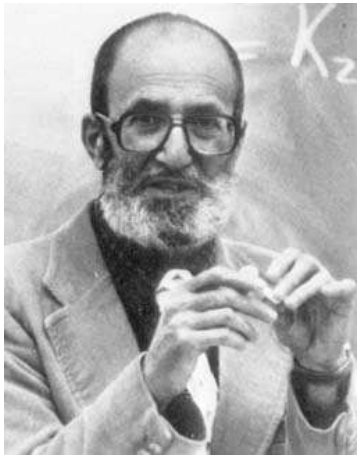
- Wahrscheinlichkeit das mindestens zwei Personen an einem Tag Geburtstag haben $\equiv 1-$ Wahrscheinlichkeit das alle Personen an verschiedenen Tagen Geburtstag haben.
- Geburtstag der ersten Person kann in n von n möglichen Tagen vorkommen. Von der zweiten Person in $n-1$ von n möglichen Tagen, usw. somit

$$\begin{aligned} P_n(k) &= 1 - \frac{n}{n} \cdot \frac{n-1}{n} \cdot \frac{n-2}{n} \cdots \frac{n-(k-1)}{n} \\ &= 1 - \frac{n!}{n^k (n-k)!} \\ &= 1 - \frac{k!}{n^k} \binom{n}{k} \end{aligned}$$

Geburtstagsattacke (Paul Halmos)

In seiner Autobiographie schrieb der berühmte ungarische Mathematiker Paul Halmos folgendes

> *A good way to attack the problem is to pose it in reverse: what's the largest number of people for which the probability is less than $1/2$ that they all have different birthdays?*



und zeigte eine *schöne* analytische Herleitung des Problems.

- Wir suchen das größte k für gegebenes n, so daß

$$\frac{n}{n} \cdot \frac{n-1}{n} \cdot \frac{n-2}{n} \cdots \frac{n-(k-1)}{n} < \frac{1}{2} \Longleftrightarrow$$

$$1 \cdot \left(1 - \frac{1}{n}\right) \cdot \left(1 - \frac{2}{n}\right) \cdots \left(1 - \frac{k-1}{n}\right) < \frac{1}{2}.$$

Geburtstagsattacke (Paul Halmos) (Fort.)

Unter Ausnutzung der Ungleichung

$$\sqrt[k]{a_1 a_2 a_3 \cdots a_k} \leq \frac{a_1 + a_2 + a_3 + \cdots + a_k}{k}$$

$$\text{mit } a_r = \left(1 - \frac{r-1}{n}\right) \text{ für } r = 1, 2, \ldots, k$$

erhält man

$$\sqrt[k]{1 \cdot \left(1 - \frac{1}{n}\right) \cdot \left(1 - \frac{2}{n}\right) \cdots \left(1 - \frac{k-1}{n}\right)} \leq$$

$$\frac{1 + \left(1 - \frac{1}{n}\right) + \left(1 - \frac{2}{n}\right) + \cdots + \left(1 - \frac{k-1}{n}\right)}{k} = \frac{1}{k} \sum_{i=0}^{k-1} \left(1 - \frac{i}{n}\right)$$

Geburtstagsattacke (Paul Halmos) (Fort.)

$$
\begin{aligned}
&= \frac{1}{k}\sum_{i=0}^{k-1}\left(1-\frac{i}{n}\right)\\
&= \frac{1}{k}\left(\sum_{i=0}^{k-1}1-\sum_{i=0}^{k-1}\frac{i}{n}\right)\\
&= \frac{1}{k}\left(k-\frac{1}{n}\,\frac{k-1}{2}\,k\right)\\
&= \left(1-\frac{k-1}{2n}\right) \qquad \text{und somit}
\end{aligned}
$$

$$
1\cdot\left(1-\frac{1}{n}\right)\cdot\left(1-\frac{2}{n}\right)\cdots\left(1-\frac{k-1}{n}\right)\leq\left(1-\frac{k-1}{2n}\right)^{k}.
$$

Geburtstagsattacke (Paul Halmos) (Fort.)

Unter Ausnutzung der Ungleichung $1 - x \leq e^{-x}$ für $x \geq 0$ erhält man $1 - (k-1)/2n \leq e^{-(k-1)/2n}$ und somit

$$1 \cdot \left(1 - \frac{1}{n}\right) \cdot \left(1 - \frac{2}{n}\right) \cdots \left(1 - \frac{k-1}{n}\right) \leq \left(1 - \frac{k-1}{2n}\right)^k \leq \left(e^{-(k-1)/2n}\right)^k = e^{-k(k-1)/2n}.$$

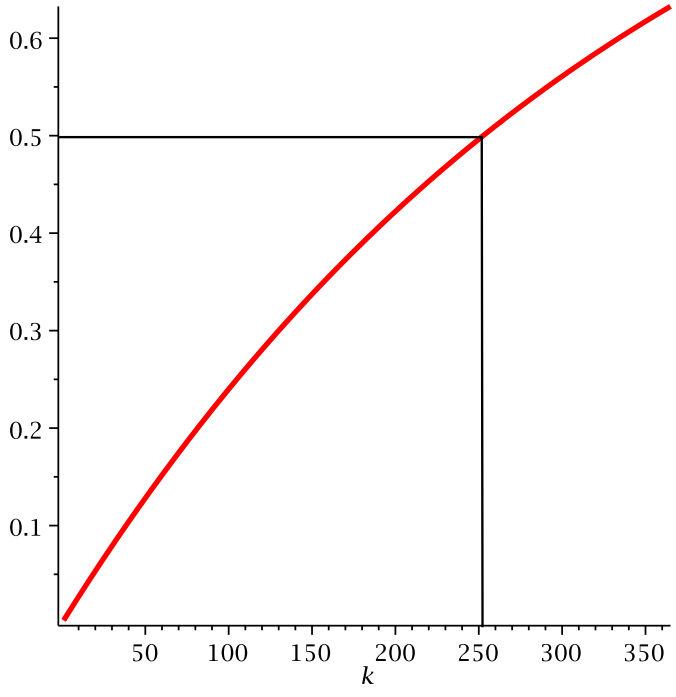
Das kleinste k für das $e^{-k(k-1)/2n} \leq 1/2$ gilt, ist somit eine obere Schranke für das kleinste k für das gilt

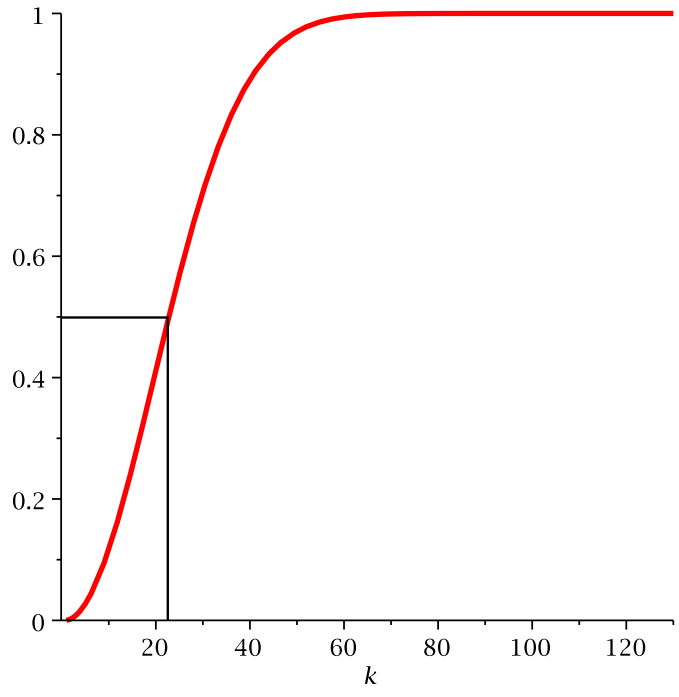
$$1 \cdot \left(1 - \frac{1}{n}\right) \cdot \left(1 - \frac{2}{n}\right) \cdots \left(1 - \frac{k-1}{n}\right) < \frac{1}{2}.$$

Geburtstagsattacke Finales Resultat

- Ist $k \geq (1 + \sqrt{1 + 8n \ln 2})/2$ dann ist $e^{-k(k-1)/2n} \leq 1/2$.
- Für $n = 365$ genügt $k = 23$, damit die Wahrscheinlichkeit $\geq 1/2$ ist, daß wenigstens zwei Personen am gleichen Tag Geburtstag haben.
- Allgemein genügen ungefähr $\sqrt{n}$ viele Personen, damit zwei am gleichen Tag Geburtstag haben.

Geburtstagsattacke Visualisierung





$$\widetilde{P}_{365}(k) = 1 - \left(1 - \frac{1}{365}\right)^k$$

$$P_{365}(k) = 1 - \frac{k!}{365^k}\binom{365}{k}$$

Geburtstagsattacke auf eine Hashfunktion

- Geburtstagsattacke gilt der starken Kollisionsresistenz von h. Man versucht also eine Kollision von h zu finden.
- Idee: so viele Hashwerte wie möglich zu berechnen und im vorhandenen Speicher abzulegen. Anschließend sortiert man diese Werte und sucht nach Kollisionen.
- Geburstagsparadoxon erlaubt eine Analyse dieses Verfahrens.
- Annahme: Strings aus Σ^* können zufällig gewählt werden und Hashwerte sind gleichverteilt.
- Wählt man k Argumente $x \in \Sigma^*$ zufällig aus, wobei

$$k \geq (1 + \sqrt{1 + (8 \ln 2)|\Sigma|^n})/2$$

dann ist die Wahrscheinlichkeit größer $1/2$, daß zwei von ihnen denselben Hashwert haben.

Geburtstagsattacke auf eine Hashfkt (Fort.)

Annahme: $\Sigma = \{0, 1\}$, dann braucht man

$$k \geq f(n) = (1 + \sqrt{1 + (8 \ln 2) 2^n})/2$$

viele Hashwerte.

Tabelle zeigt Werte von $\log_2(f(n))$

n	50	100	150	200
$\log_2(f(n))$	25.24	50.24	100.24	

- Wenn man etwas mehr als $2^{n/2}$ viele Hashwerte bildet, findet die Geburtstagsattacke mit Wahrscheinlichkeit $\geq 1/2$ eine Kollision.
- Um Geburtstagsattacke zu verhindern, muß man n groß wählen, z.B. $n \geq 128$. Für dig. Signaturen wird sogar $n \geq 160$ verlangt.

Kompressionsfkt. aus Verschlüsselungsfkt.

- Benötigt wird eine Verschlüsselungsfunktion $e_k : \{0,1\}^n \to \{0,1\}^n, k \in \{0,1\}^n$.
- Hashwerte habe die Länge n.
- Um Geburtstagsattacke zu verhindern, muß $n \geq 128$ sein. DES könnte man z.B. nicht benutzen.

Kompressionsfunktionen $h : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ kann man wie folgt definieren

$$\begin{aligned} h(k,x) &= e_k(x) \oplus x \\ h(k,x) &= e_k(x) \oplus x \oplus k \\ h(k,x) &= e_k(x \oplus k) \oplus x \\ h(k,x) &= e_k(x \oplus k) \oplus x \oplus k. \end{aligned}$$

Annahme: Ist das Verschlüsselungsverfahren sicher, so scheinen diese Kompressionsfunktionen auch sicher zu sein.

SHA-1 Hashfunktion

- SHA-1 ist eine häufig benutze Hashfunktion.

Sei $x \in \{0,1\}^*$ und bezeichne $|x|$ die Länge von x. Im ersten Schritt wird x ergänzt, so daß die Länge von x ein Vielfaches von 512 ist.

1. An x wird eine 1 angehängt, d.h. $x \leftarrow x \,\|\, 1$.
2. An x werden so viele Nullen angehängt, daß $|x| = k \cdot 512 - 64$ ist.
3. Länge des originalen x wird als 64-Bit-Zahl geschrieben und an x angehängt.

SHA-1 Hashfunktion (Fort.)

Beispiel: Nachricht x sei

$$01100001 \quad 01100010 \quad 01100011 \quad 01100100 \quad 01100101.$$

Nach dem ersten Schritt ist x

$$01100001 \quad 01100010 \quad 01100011 \quad 01100100 \quad 01100101 \quad 1.$$

Länge von x ist 41, somit werden 407 Nullen an x angehängt.
Länge ist dann $448 = 512 - 64$. In Hexadezimalschreibweise

$$\begin{array}{cccc} 61626364 & 65800000 & 00000000 & 00000000 \\ 00000000 & 00000000 & 00000000 & 00000000 \\ 00000000 & 00000000 & 00000000 & 00000000 \\ 00000000 & 00000000 & & \end{array}$$

SHA-1 Hashfunktion (Fort.)

Länge des originalen x war 40. Als 64-Bit-Zahl in Hexadezimalschreibweise $00000000 \quad 00000028$.

Somit ist x schließlich

$$\begin{array}{cccc} 61626364 & 65800000 & 00000000 & 00000000 \\ 00000000 & 00000000 & 00000000 & 00000000 \\ 00000000 & 00000000 & 00000000 & 00000000 \\ 00000000 & 00000000 & 00000000 & 00000028 \end{array}$$

- Hashwert wird wie folgt berechnet. Sei x ein nach den beschriebenen Regeln erweiterter Bitstring, dessen Länge durch 512 teilbar ist.
- Folge der 512-Bit-Wörter sei $x = M_1 M_2 \ldots M_n$.

SHA-1 Hashfunktion (Fort.)

- Es werden folgende 32-Bit-Wörter initialisiert: $H_0 = 67452301, H_1 = EFCDAB89, H_2 = 98BADCFE, H_3 = 10325476, H_4 = C3D2E1F0$.
- Führe folgende Prozedur für $i = 1, 2, \ldots, n$ aus.

1. Schreibe M_i als Folge $M_i = W_0 W_1 \ldots W_{15}$ von 32-Bit-Wörtern.
2. Für $t = 16, 17, \ldots, 79$ setze $W_t = S^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$.
3. Setze $A = H_0, B = H_1, C = H_2, D = H_3$ und $E = H_4$.
4. Für $t = 0, 1, \ldots, 79$ setze $T = S^5(A) + f_t(B, C, D) + E + W_t + K_t, E = D, D = C, C = S^{36}(B), B = A, A = T$.
5. Setze $H_0 = H_0 + A, H_1 = H_1 + B, H_2 = H_2 + C, H_3 = H_3 + D, H_4 = H_4 + E$.

SHA-1 Hashfunktion (Fort.)

Der errechnete Hashwert ist dann

$$\text{SHA-1}(x) = H_0 H_1 H_2 H_3 H_4.$$

Wobei folgende Funktionen in den 5 Schritten benutzt werden

$$f_t : \{0,1\}^{32} \times \{0,1\}^{32} \times \{0,1\}^{32} \to \{0,1\}^{32} \quad \text{mit}$$

$$f_t(B,C,D) = \begin{cases} (B \wedge C) \vee (\neg B \wedge D) & \text{für } 0 \leq t \leq 19 \\ B \oplus C \oplus D & \text{für } 20 \leq t \leq 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & \text{für } 40 \leq t \leq 59 \\ B \oplus C \oplus D & \text{für } 60 \leq t \leq 79 \end{cases}$$

SHA-1 Hashfunktion (Fort.)

Des weiteren werden folgende Konstanten benutzt

$$K_t = \begin{cases} 5A827999 & \text{für } 0 \leq t \leq 19 \\ 6ED9EBA1 & \text{für } 20 \leq t \leq 39 \\ 8F1BBCDC & \text{für } 40 \leq t \leq 59 \\ CA62C1D6 & \text{für } 60 \leq t \leq 79 \end{cases}$$

und $S^k(w)$ bezeichnet den zirkulären Links-Shift eines 16-Bitwortes w um k Bits. Die arithmetische $+$ Operation zweier 16-Bit Wörter dargestellter Zahlen wird $\mod 2^{16}$ gerechnet.

Andere Hashfunktionen

Hashfunktion	Blocklänge	Relative Geschwindigkeit
MD4	128	1.00
MD5	128	0.68
RIPEMD-128	128	0.39
SHA-1	160	0.28
RIPEMD-160	160	0.24

- MD4 kann nicht mehr als kollisionsresistent gelten. Mittels Berechnung von 2^{20} Hashwerten kann Kollision gefunden werden.

Message Authentication Codes

- Mittels Hashfunktionen kann man überprüfen, ob Nachrichten verändert wurden.
- Man kann aber nicht feststellen, von *wem* eine Nachricht kommt.
- Parametrisiert man Hashfunktionen, dann kann man die Authentizität einer Nachricht überprüfen.

Eine parametrisierte Hashfunktion ist eine Familie $\{h_k : k \in \mathcal{K}\}$ von Hashfunktionen. Hierbei ist $\mathcal{K}$ eine Menge. Sie heißt Schlüsselraum von h.

Eine parametrisierte Hashfunktion heißt auch *Message Authentication Code*, kurz MAC.

Message Authentication Codes (Fort.)

Beispiel: Sei

$$g : \{0,1\}^* \to \{0,1\}^4$$

eine Hashfunktion. Dann kann man daraus folgendermaßen einen MAC mit Schlüsselraum $\{0,1\}^4$ machen:

$$h_k : \{0,1\}^* \to \{0,1\}^4, \quad x \mapsto g(x) \oplus k.$$

- Wichtig hierbei ist, daß der MAC *fälschungsresistent* ist.
- Es muß also "unmöglich" sein, ohne Kenntnis von k ein Paar $(x, h_k(x))$ zu generieren.
- Ist das in diesem Beispiel wirklich gelungen?

CBC-MAC

- Konstruiere MACs aus Blockchiffren im CBC-Mode.
- Sei $E_k : \{0,1\}^n \to \{0,1\}^n$ eine Blockchiffre mit Schlüssel k und $m \in \{0,1\}^*$ eine Nachricht.
  - Schreibe $m = m_1 \| m_2 \| \ldots \| m_l$ mit $m_i \in \{0,1\}^n$ (und Padding).
  - $y_0 := IV = 0^n$.
  - Für $i := 1, 2, \ldots, l$ berechne $y_i := E_k(y_{i-1} \oplus m_i)$.
  - Ausgabe y_l.

Der CBC-MAC ist somit der letzte Chiffretext der Verschlüsselung unter Verwendung des Initialisierungsvektors 0^n.

CBC-MAC Sicherheit

- Ist die Blockchiffre sicher und ist die Nachrichtenlänge konstant, dann gilt CBC-MAC als sicher.

Angenommen die Nachrichtenlänge ist nicht konstant:

1. Erfrage $y_1 := h_k(m_1)$.
2. Erfrage $y_2 := h_k(y_1 \| m_2)$.
3. Konstruiere Fälschung $y_2 = h_k(m_1 \| 0^n \| m_2)$.
   1. $y_1 := E_k(00\ldots0 \oplus m_1) = E_k(m_1)$.
   2. $\overline{y}_1 := E_k(00\ldots0 \oplus y_1) = E_k(y_1)$,
      $\overline{y}_2 := E_k(\overline{y}_1 \oplus m_2)$.
   3. $\widetilde{y}_1 := E_k(00\ldots0 \oplus m_1) = E_k(m_1) = y_1$.
      $\widetilde{y}_2 := E_k(y_1 \oplus 00\ldots0) = E_k(y_1) = \overline{y}_1$.
      $\widetilde{y}_3 := E_k(\overline{y}_1 \oplus m_2) = \overline{y}_2$.

Problem kann gelöst werden, indem man z.B. $h_k(|m| \, \|m)$ verwendet.

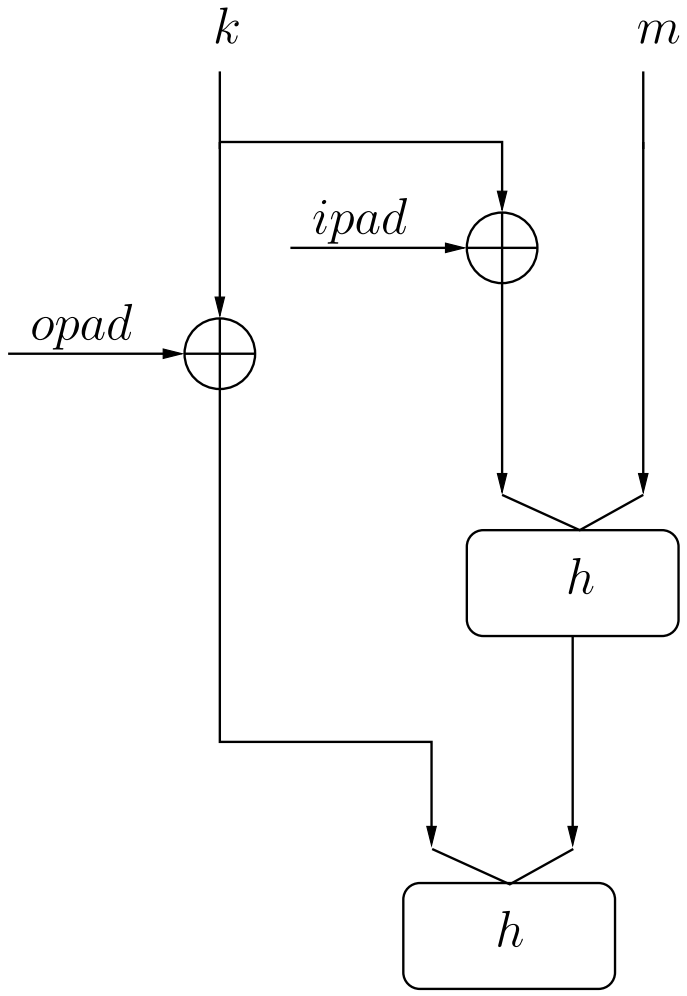
HMAC

- Keyed-Hash Message Authentication Code (HMAC).
- HMAC benutzt als Basis eine beliebige Hashfunktion $h()$ und einen symmetrischen Schlüssels k, und liefert und liefert einen MAC.
- Länge des erzeugten HMACs entspricht der Ausgabelänge der Hashfunktion.
- HMAC ist beruht auf Anwendung einer "inneren" Hashfunktion und einer "äußeren" Hashfunktion.

$$\text{HMAC}_k(m) = h((k \oplus opad) \,||\, h((k \oplus ipad) \,||\, m))$$

wobei $ipad = 0x3636\ldots36$ und $opad = 0x5c5c\ldots5c$ ist.

HMAC (Fort.)



$$\mathsf{HMAC}_k(m) = h((k \oplus opad) \,||\, h((k \oplus ipad) \,||\, m))$$

Digitale Signaturen

- Realisierung der digitalen Signaturen ist eng verwandt mit der Public-Key-Verschlüsselung.
- Idee: Alice will Dokument m signieren. Sie berechnet mit dem privaten Schlüssel d die digitale Signatur $s(d, m)$.
- Unter Verwendung des öffentlichen Schlüssels kann jeder verifizieren, daß die Signatur korrekt ist.

Sicherheit des privaten Schlüssels:

- Signaturverfahren kann nur sicher sein, wenn es unmöglich ist, in vertretbarer Zeit aus dem öffentlichen Schlüssel den privaten Schlüssel zu berechnen.

RSA-Signaturen

- Idee: Alice signiert ein Dokument m indem sie $s = m^d \mod n$ berechnet.
- Bob verifiziert Signatur, indem er darauf das Verschlüsselungsverfahren anwendet, also $s^e \mod n$ berechnet.
- Bob hat m aus der Signatur s gewonnen.

Schlüsselerzeugung läuft analog wie in der RSA-Verschlüsselung. D.h. öffentl. Schlüssel ist (n, e), geheimer Schlüssel ist d.

Signatur: Alice will Zahl $m \in \{0, 1, \ldots, n-1\}$ signieren. Sie berechnet den Wert $s = m^d \mod n$. Signatur ist s.

RSA-Signaturen (Verifikation)

- Bob will Signatur s verifizieren. Er besorgt sich öffentlichen Schlüssel (n, e) von Alice und berechnet $m = s^e \mod n$.
- Mit der Verifikation hat Bob m aus der Signatur s gewonnen (er braucht m vorher nicht zu kennen). Bob ist sich auch sicher, daß Alice die Signatur erzeugt hat.
- Diese Verifikation kann jeder durchführen, der den öffentlichen Schlüssel von Alice kennt.

Angriff auf RSA-Signaturen

Vortäuschen einer falschen Identität (Maskierung):

- Vor Beginn der Verifikation besorgt sich Bob den öffentlichen Schlüssel (n, e) von Alice.
- Wenn es einem Angreifer in dieser Phase gelingt, seinen eignen Schlüssel Bob "unterzuschieben", kann Angreifer danach Signaturen erzeugen, die Bob als Signaturen von Alice anerkennt.
- Lösung: Authentizität des öffentlichen Schlüssels (z.B. durch Trustcenter).

Angriff auf RSA-Signaturen (Fort.)

No-Message-Attack:

- Angreifer wählt eine Zahl $s \in \{0, 1, \ldots, n-1\}$ und behauptet, s sei eine RSA-Signatur von Alice.
- Derjenige der diese Signatur verifizieren will, berechnet $m = s^e \mod n$ und glaubt, Alice habe m signiert.
- Falls m ein sinnvoller Text ist, wurde Alice damit die Signatur dieses sinnvollen Textes "untergeschoben".

Angriff auf RSA-Signaturen (Fort.)

RSA-Multiplikativität:

- Sind $m_1, m_2 \in \{0, 1, \ldots, n-1\}$ und sind $s_1 = m_1^d \mod n$ und $s_2 = m_2^d \mod n$ die Signaturen von m_1 und m_2, dann ist

$$s = s_1 s_2 \mod n = (m_1 m_2)^d \mod n$$

  die Signatur von $m = m_1 m_2$.
- Man kann also aus zwei gültigen Signaturen leicht eine dritte *gültige* Signatur generieren.
- Nutzt ein Angreifer die Multiplikativität aus, kann er mit einer Chosen-Message-Attacke jede Signatur fälschen.

Angriff auf RSA-Signaturen (Fort.)

Chosen-Message-Attacke:

- Angreifer will sich Nachricht $m \in \{0, 1, \ldots, n-1\}$ signieren lassen. Er wählt eine von m verschiedene Nachricht $m_1 \in \{0, 1, \ldots, n-1\}$ mit gcd$(m_1, n) = 1$. Angreifer berechnet dann

$$m_2 = m\, m_1^{-1} \mod n$$

- Angreifer lässt sich nun beide Nachrichten m_1 und m_2 signieren. D.h.

$s_1 = m_1^d \mod n$ und $s_2 = m_2^d \mod n = (m\, m_1^{-1})^d \mod n.$

- Hieraus berechnet Angreifer
$s = s_1\, s_2 = m_1^d (m\, m_1^{-1})^d = (m\, m_1^{-1} m_1)^d = m^d \mod n.$

RSA-Signatur mit Hashwert

- Will man beliebig lange Texte signieren, verwendet man öffentlich bekannte, kollisionsresistente Hashfunktionen

$$h : \{0,1\}^* \to \{0,1,\ldots,n-1\}.$$

- Beachte: weil h kollisionsresistent ist, muß h auch Einwegfunktion sein.

Signatur eines Textes x ist

$$s = (h(x))^d \mod n.$$

- Aus Signatur s lässt sich nur der Hashwert des Textes rekonstruieren, aber nicht der Text x selbst.
- Zur Verifikation wird also auch Text x benötigt.

RSA-Signatur mit Hashwert (Fort.)

- Alice erzeugt Signatur $s = (h(x))^d \mod n$ und sendet diese zusammen mit x an Bob.
- Bob berechnet $m = s^e \mod n$ und vergleicht das Resultat mit dem Hashwert $h(x)$.
- Weil $h(x)$ öffentlich bekannt ist, kann Bob $h(x)$ berechnen.
- Stimmen m und $h(x)$ überein, so ist Signatur gültig, andernfalls nicht.

Resistent gegen No-Message-Attack:

- Angreifer wählt Signatur s. Beachte, Angreifer muß auch Text x produzieren, und schickt somit an Bob (x, s).
- Bob berechnet $m = s^e \mod n$ und prüft, ob $m = h(x)$ ist.

RSA-Signatur mit Hashwert (Fort.)

- Angreifer muß also ein x so bestimmen, daß $h(x) = m$ gilt.
- Das ist "praktisch unmöglich" weil h eine Einwegfunktion ist.

RSA-Multiplikativität Problem:

- Weil h eine Einwegfunktion ist, ist es "praktisch unmöglich" einen Text x zu finden mit $h(x) = m = m_1 m_2 \mod n$.
- Das Problem der Multiplikativität besteht also nicht, wenn sichere Hashfunktionen benutzt werden.

Signaturen aus Public-Key-Verfahren

- Jedes Public-Key-Verschlüsselungsverfahren kann als Signatur-Verfahren benutzt werden, falls folgende Zusatzbedingung erfüllt ist:
  - Seien E und D Ver- und Entschlüsselungsfunktionen und e und d der öffentl. und der geheime Schlüssel.
  - Dann muß für jeden Klartext m die Gleichung

$$m = E(D(m, d), e)$$

    gelten.
  - D.h. die Rolle von Ver- und Entschlüsselung muß vertauschbar sein.
  - Im Beispiel von RSA ist das erfüllt, weil

$$(m^d)^e \equiv (m^e)^d \equiv m \mod n$$ gilt.

ElGamal-Signatur

- Im ElGamal-Verfahren ist die Verschlüsselung und die Entschlüsselung nicht "einfach" austauschbar.
- Die ElGamal-Signatur muß deshalb anders konstruiert werden.

Schlüsselerzeugung:

- Schlüsselerzeugung läuft analog wie in der ElGamal-Verschlüsselung.
- Alice wählt Primzahl p und eine Primitivwurzel $g \mod p$. Dann wählt sie zufällig und gleichverteilt einen Exponenten $a \in \{0, 1, \ldots, p-2\}$ und berechnet

$$A = g^a \mod p.$$

- Öffentlicher Schlüssel ist (p, g, A). Geheimer Schlüssel ist Exponent a.

ElGamal-Signatur (Fort.)

- Alice signiert einen Text $x \in \{0,1\}^*$, indem sie eine öffentl. bekannte kollisionsresistente Hashfunktion

$$h : \{0,1\}^* \to \{1,2,\ldots,p-2\}$$

  benutzt.
- Alice wählt eine Zufallszahl $k \in \{1,2,\ldots,p-2\}$, die zu $p-1$ teilerfremd ist.
- Alice berechnet das Signatur-Paar (r,s) wobei

$$r = g^k \mod p, \qquad s = k^{-1}(h(x) - ar) \mod (p-1).$$

ElGamal-Signatur (Fort.)

- Bob der Verifizierer besorgt sich den authentischen Schlüssel (p, g, A) von Alice.
- Er verifiziert, daß $1 \leq r \leq p-1$ ist. Falls diese Bedingung nicht erfüllt ist, wird Signatur in diesem Schritt zurück gewiesen.
- Andernfalls prüft Bob, ob die Kongruenz

$$A^r r^s \equiv g^{h(x)} \mod p$$ erfüllt ist.

- Falls ja, dann akzeptiert Bob die Signatur, sonst nicht.

ElGamal-Signatur (Fort.)

Falls Signatur s ordnungsgemäß konstruiert wurde, folgt

$$A^r r^s \equiv g^{ar} g^{kk^{-1}(h(x)-ar)} \equiv g^{h(x)} \mod p,$$

wobei k^{-1} das Inverse von $k \mod p-1$.

- Ist

  $$A^r r^s \equiv g^{h(x)} \mod p$$

  für ein Paar (r,s) erfüllt, und ist k der diskrete Logarithmus von r zur Basis g, so gilt

  $$g^{ar+ks} \equiv g^{h(x)} \mod p.$$

- Weil g eine Primitivwurzel $\mod p$ ist, folgt

  $$ar + ks \equiv h(x) \mod p-1.$$

ElGamal-Signatur (Fort.)

- Ist k teilerfremd zu $p-1$, so folgt daraus

$$r = g^k \mod p, \qquad s = k^{-1}(h(x) - ar) \mod (p-1).$$

ElGamal-Signatur gleicher Exponent k

- Werden zwei Signaturen s_1 und s_2 mit dem gleichen Exponent k erzeugt, kann der geheime Schlüssel a berechnet werden.
- Angenommen die beiden Signaturen s_1 und s_2 sind das Resultat von x_1 und x_2, wobei der gleiche Exponent k benutzt wurde.
- Es gilt somit

$$s_1 - s_2 \equiv k^{-1}(h(x_1) - h(x_2)) \mod (p-1)$$

  und $r = g^k \mod p$ ist gleich für beide Signaturen.
- Hieraus kann man k bestimmen, wenn $(h(x_1) - h(x_2))$ invertierbar modulo $p-1$ ist.

ElGamal-Signatur gleicher Exponent k (Fort.)

- Aus den Zahlen $k, s_1, r, h(x_1)$ kann man nun den geheimen Schlüssel a von Alice berechnen, weil:

$$\begin{aligned} & s_1 = k^{-1}(h(x_1) - ar) \mod (p-1) \\ \Leftrightarrow \quad & a \equiv r^{-1}(h(x_1) - ks_1) \mod (p-1) \end{aligned}$$

ElGamal-Signatur ohne Hashfunktion

- Wird der Text x direkt signiert, dann lautet die Verifikation

$$A^r r^s \equiv g^x \mod p.$$

- Wählt man passend die Werte r, s, x, so kann man diese Kongruenz erfüllen. D.h. beliebige Signaturen erstellen.
- Wähle zwei ganze Zahlen u, v mit $\gcd(v, p-1) = 1$ und setze dann

$$\begin{aligned} r &= g^u A^v \mod p, \quad s = -rv^{-1} \mod (p-1), \\ x &= su \mod (p-1). \end{aligned}$$

- Mit diesen Werten gilt die Kongruenz

$$A^r r^s \equiv A^r g^{su} A^{sv} \equiv A^r g^{su} A^{-r} \equiv g^x \mod p.$$

Digital Signature Algorithm (DSA)

- Wurde 1991 vom NIST vorgeschlagen und später vom NIST zum Standard erklärt.
- Orientiert sich am ElGamal-Verfahren und ist verwandt mit der Schnorr-Signatur.

Schlüsselerzeugung:

- Alice wählt eine Primzahl q mit $2^{159} < q < 2^{160}$ (Binärentwicklung hat genau die Länge von 160).
- Alice wählt als nächstes Primzahl p mit
  - $2^{511+64t} < p < 2^{512+64t}$ für ein $t \in \{0, 1, \ldots, 8\}$, wobei gewählte Primzahl q ein Teiler von $p-1$ ist.
  - Bedingung $q \,|\, (p-1)$ impliziert, daß die Gruppe $\mathbb{Z}_p^*$, Elemente der Ordnung q besitzt.
- Alice wählt eine Primitivwurzel $x \in \mathbb{Z}_p^*$ und berechnet $g = x^{(p-1)/q} \mod p$.

Digital Signature Algorithm (DSA) (Fort.)

- Zuletzt wählt Alice eine Zahl a zufällig aus der Menge $\{1, 2, \ldots, q-1\}$ und berechnet

$$A = g^a \mod p.$$

- Öffentlicher Schlüssel ist (p, q, g, A). Geheimer Schlüssel ist a.

Erzeugung der Signatur:

- Alice signiert einen Text $x \in \{0,1\}^*$, indem sie eine öffentl. bekannte kollisionsresistente Hashfunktion

$$h : \{0,1\}^* \to \{1, 2, \ldots, q-1\}$$

benutzt.

Digital Signature Algorithm (DSA) (Fort.)

- Alice wählt eine Zufallszahl $k \in \{1, 2, \ldots, q-1\}$, die zu $p-1$ teilerfremd ist.
- Alice berechnet das Signatur-Paar (r, s) wobei

$$r = (g^k \mod p) \mod q, \qquad s = k^{-1}(h(x) - ar) \mod q.$$

- Hierbei ist k^{-1} das Inverse von k modulo q.
- Die Signatur ist (r, s).

Digital Signature Algorithm (DSA) (Fort.)

Verifikation

- Bob will Signatur (r, s) des Textes x verifizieren.
- Er besorgt sich authentischen Schlüssel (p, q, g, A) von Alice und verifiziert, daß

$$1 \leq r \leq q - 1 \text{ und } 1 \leq s \leq q - 1 \text{ gilt.}$$

- Ist Bedingung erfüllt, verifiziert Bob ob

$$r = ((g^{(s^{-1}h(x)) \mod q} A^{(rs^{-1}) \mod q}) \mod p) \mod q.$$

- Ist die Signatur ordnungsgemäß konstruiert, dann gilt

$$g^{(s^{-1}h(x)) \mod q} A^{(rs^{-1}) \mod q} \equiv g^{s^{-1}(h(x)+ra)} \equiv g^k \mod p.$$

Digital Signature Algorithm (DSA) (Fort.)

- Wie in der ElGamal-Signatur, muß für jede neue Signatur ein neues k gewählt werden.
- Die Verwendung einer Hashfunktion ist ebenfalls wie in der ElGamal-Signatur unbedingt notwendig.

Wesentlicher Vorteil von DSA gegenüber der ElGamal-Signatur ist, daß die Berechnung nicht mehr in der gesamten Gruppe $\mathbb{Z}_p^*$ ausgeführt werden, sondern in der wesentlich kleineren Untergruppe.

Schnorr-Signatur

Schlüsselerzeugung läuft analog wie im DSA-Verfahren, jedoch ohne Längenbeschränkung von p und q.
Erzeugung der Signatur:

- Alice signiert einen Text $m \in \{0,1\}^*$, indem sie eine öffentl. bekannte kollisionsresistente Hashfunktion

$$h : \{0,1\}^* \to \{1,2,\ldots,q-1\}$$

  benutzt.
- Alice wählt eine Zufallszahl $k \in \{1,2,\ldots,q-1\}$.
- Alice berechnet das Signatur-Paar (r,s) wobei

$$r = g^k \mod p, \qquad e = h(m\|r), \text{ und } s = ae+k \mod q.$$

- Die Signatur ist (s,e).

Schnorr-Signatur (Fort.)

Verifikation

- Bob will Signatur (s, e) des Textes m verifizieren.
- Er besorgt sich authentischen Schlüssel (p, q, g, A) von Alice und berechnet

$$v = g^s A^{-e} \mod p \text{ und } e' = h(m\|v).$$

- Bob akzeptiert die Signatur wenn $e' = e$.
- Ist die Signatur ordnungsgemäß konstruiert, dann gilt

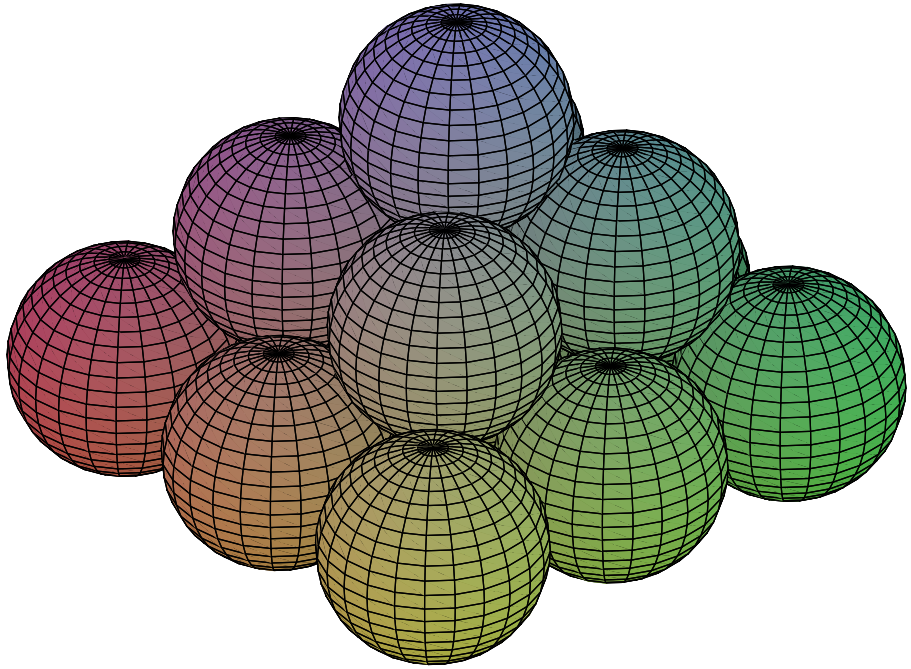
$$v \equiv g^s A^{-e} \equiv g^{ae+k} g^{-ae} \equiv g^k \equiv r \mod p.$$

Somit, $h(m\|v) = h(m\|r)$ und $e' = e$.

Elliptische Kurven Einführendes Bsp.

Eine Menge von Kugeln wird als eine quadratische Pyramide angeordnet. Mit 1 Kugel oben, 4 weiteren darunter, dann 9 weiteren darunter usw.

Wenn diese quadratische Kugelpyramide in sich zusammenfällt, ist es dann möglich die Kugeln in einer Ebene als *quadratisches* Feld anzuordnen?



Elliptische Kurven Einführendes Bsp. (Fort.)

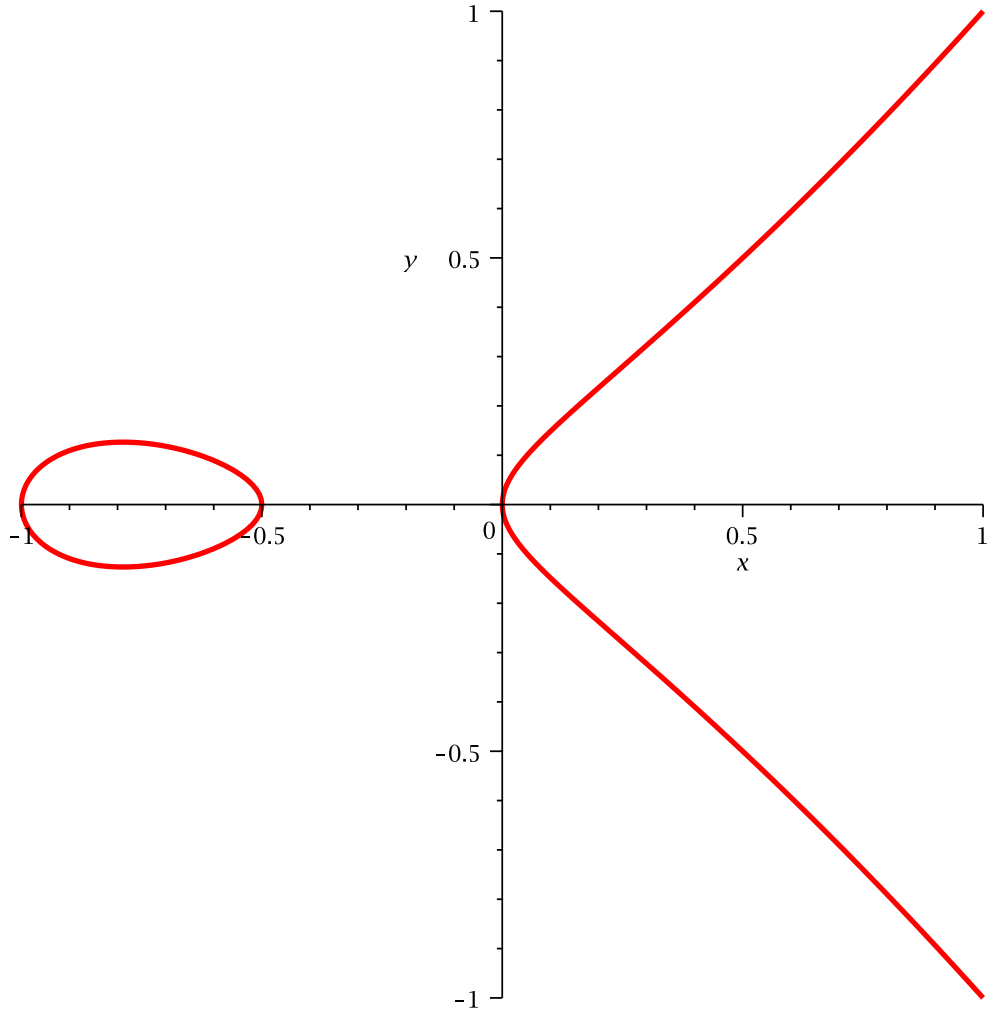
- Die Kugelpyramide hat Höhe (Ebenen) 3 und besteht somit aus insgesamt $1+4+9=14$ Kugeln. Somit lassen sich diese *nicht* in einer Ebene als *quadratisches* Feld anzuordnen.
- Wenn die Kugelpyramide die Höhe x hat, dann besteht sie insgesamt aus

$$1^2+2^2+3^2+\cdots+x^2=\frac{x(x+1)(2x+1)}{6} \text{ Kugeln.}$$

- Damit sich die Kugeln in einer Ebene als *quadratisches* Feld anzuordnen lassen, muß gelten

$$y^2=\frac{x(x+1)(2x+1)}{6}.$$

Elliptische Kurven Einführendes Bsp. (Fort.)



$$y^2 = \frac{x(x+1)(2x+1)}{6}$$

Elliptische Kurven Einführendes Bsp. (Fort.)

- Lösungen sind also alle Zahlentupel (x, y) die die Gleichung erfüllen.
- Triviale Lösungen sind die Zahlentupel $(0, 0)$ und $(1, 1)$. Aus diesen lassen sich jedoch neue Lösungen konstruieren.
- Die Gerade die diese beiden Punkte schneidet ist $y = x$. Schnittpunkt mit der Kurve gibt

$$x^2 = \frac{x(x+1)(2x+1)}{6} = \frac{1}{3}x^3 + \frac{1}{2}x^2 + \frac{1}{6}x.$$

- Umformung der Gleichung ergibt

$$x^3 - \frac{3}{2}x^2 + \frac{1}{2}x = 0.$$

Elliptische Kurven Einführendes Bsp. (Fort.)

Für beliebige Zahlen a, b, c gilt

$$(x-a)(x-b)(x-c) = x^3 - (a+b+c)x^2 + (ab+ac+bc)x - abc.$$

- Der Koeffizient von x^2 kann benutzt werden um ein weiteres Lösungstupel zu finden, wenn bereits zwei Lösungen bekannt sind $(a = 0, b = 1)$.
- Somit $a + b + c = 3/2$, und $c = 1/2$. Eine weitere Lösung ist also $x = 1/2$ und weil $x = y$ also $y = 1/2$.
- Aufgrund der symmetrischen Eigenschaften der elliptischen Kurve ist $(1/2, -1/2)$ ebenfalls eine Lösung.
- Hinsichtlich unseres Problems der Kugelpyramide sind diese neuen Lösungtupel jedoch bedingt hilfreich.

Elliptische Kurven Einführendes Bsp. (Fort.)

- Wiederholt man die Schritte für die beiden Punkte $(1/2, -1/2)$ und $(1, 1)$ erhält man Schnittgerade $y = 3x - 2$.
- Schnittpunkt der Schnittgerade mit der Kurve ist somit

$$(3x - 2)^2 = \frac{x(x+1)(2x+1)}{6}.$$

- Umformung der Gleichung ergibt

$$x^3 - \frac{51}{2}x^2 + \cdots = 0.$$

- Unter Ausnutzung des "Koeffiziententricks" erhält man $1/2 + 1 + c = 51/2$ und somit $x = 24$ und $y = 70$ (wegen $y = 3x - 2$).

Elliptische Kurven Einführendes Bsp. (Fort.)

Somit

$$1^2 + 2^2 + 3^2 + \cdots + 24^2 = 70^2.$$

- Wenn wir also 4900 Kugeln haben, können wir diese in einer quadratischen Kugelpyramide der Höhe 24 anordnen oder in einem 70×70 quadratischem Feld.
- Man kann zeigen, daß $(24, 70)$ neben der trivialen Lösung $(1, 1)$ die einzigen positiven ganzzahligen Lösungen sind
- Siehe: W. S. Anglin. The square pyramid puzzle. *Amer. Math. Monthly*, 97(2):120-124,1990.

Elliptische Kurven Grundlagen

Wir betrachten eine elliptische Kurve E der Form

$$y^2 = x^3 + Ax + B$$

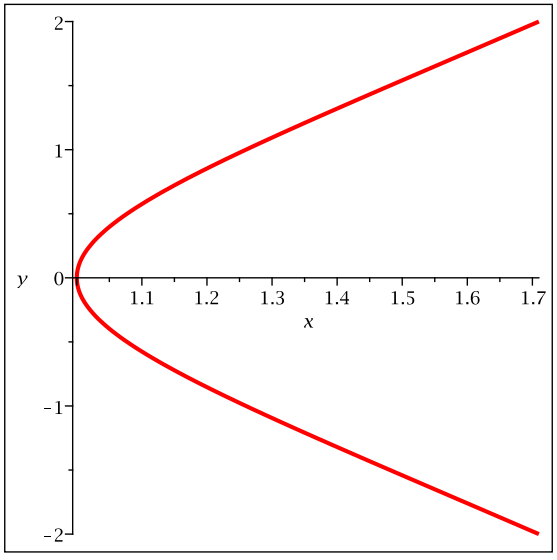
wobei A und B Konstanten sind und x, y, A, B Elemente eines Körpers K.

- Ist $A, B \in K$, dann sagen wir, E ist definiert über K.
- Die Menge aller Lösungstupel (Punktmenge) aus einem Körper $L \supseteq K$ ist definiert als

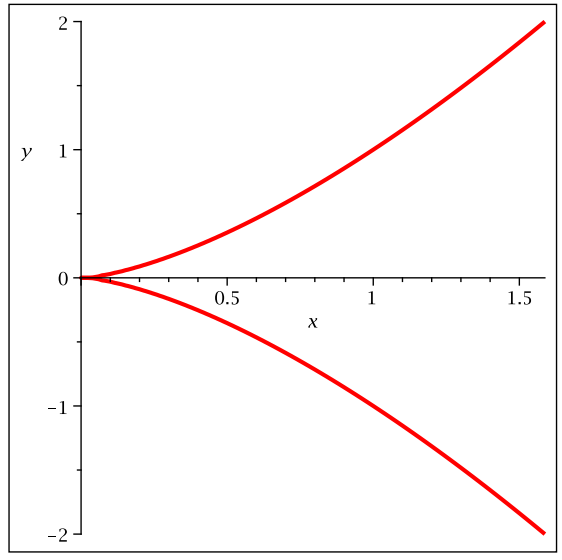
$$E(L) = \{\mathcal{O}\} \cup \{(x, y) \in L \times L \,|\, y^2 = x^3 + Ax + B\}.$$

Wir betrachten elliptische Kurven mit Diskriminante $-16(4A^3 + 27B^2) \neq 0$.

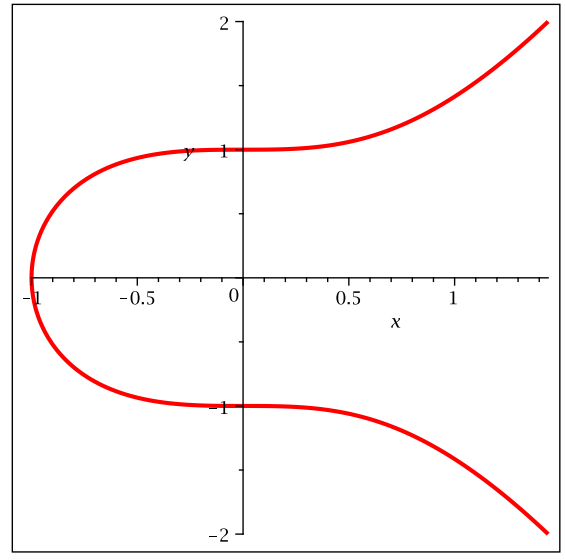
Elliptische Kurven für unterschiedliche A, B



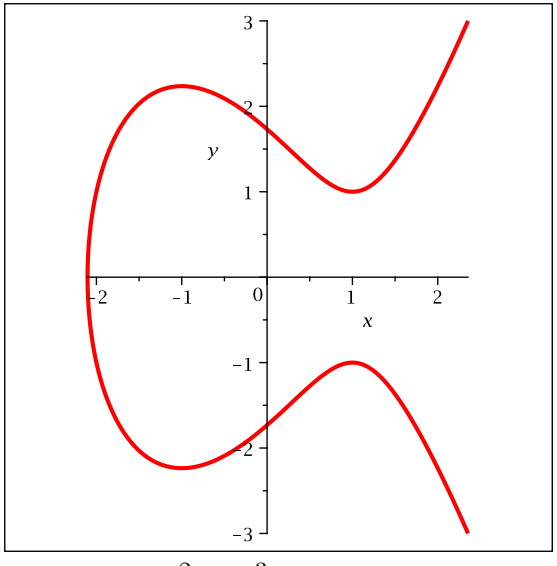
$y^2 = x^3 - 1$



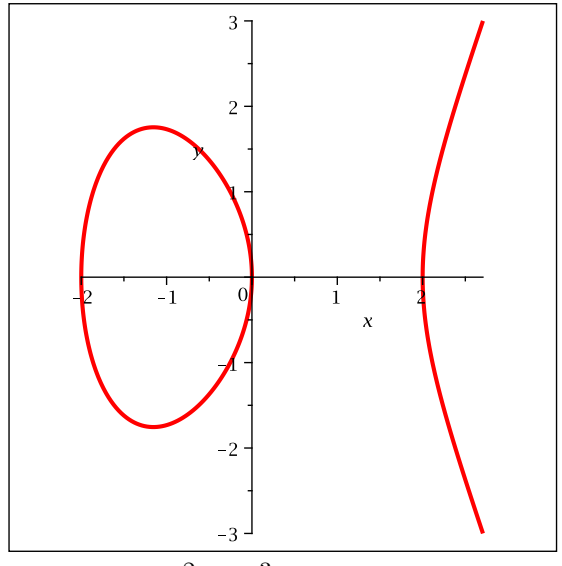
$y^2 = x^3$



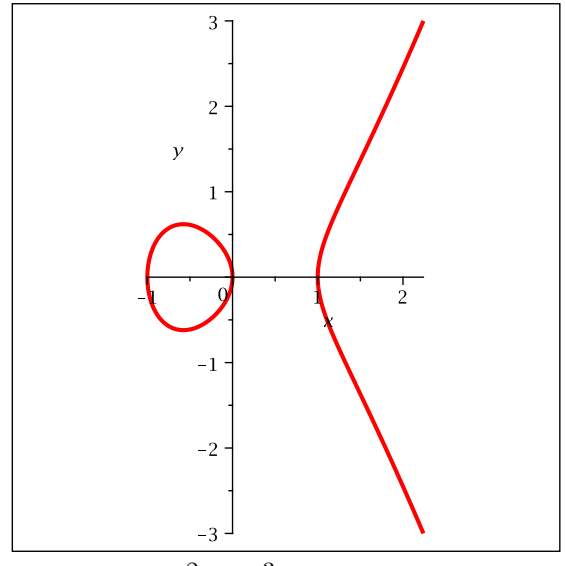
$y^2 = x^3 + 1$



$y^2 = x^3 - 3x + 3$



$y^2 = x^3 - 4x$



$y^2 = x^3 - x$

Addition von Punkten

Sei E eine elliptische Kurve der Form $y^2 = x^3 + Ax + B$.
Seien $P_1 = (x_1, y_1)$ und $P_2 = (x_2, y_2)$ Punkte auf E, wobei $P_1, P_2 \neq \mathcal{O}$. Wir definieren $P_1 + P_2 = P_3 = (x_3, y_3)$ wie folgt:

1. Wenn $x_1 \neq x_2$, dann

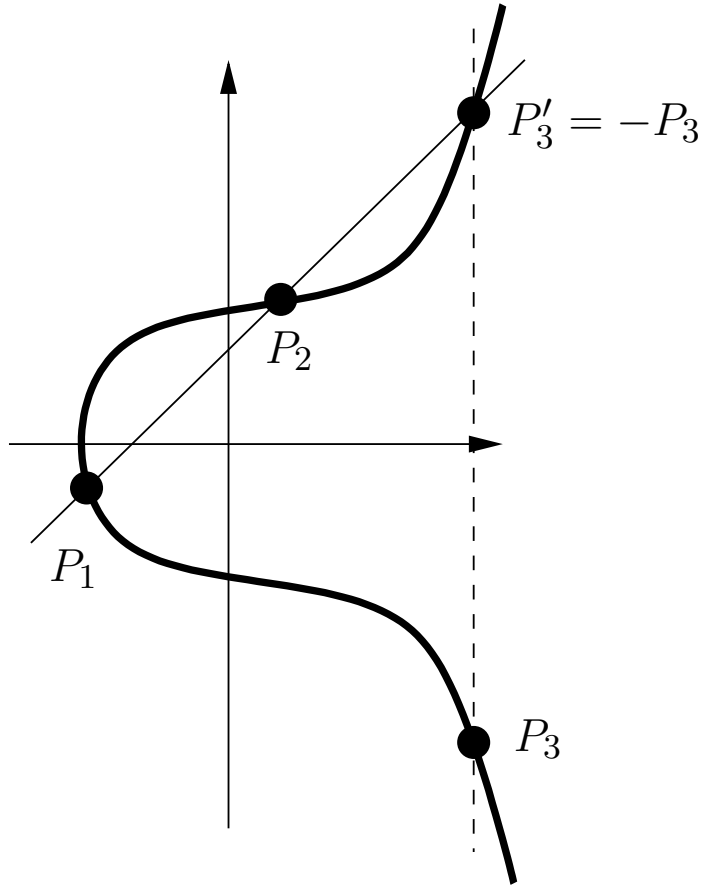
$$x_3 = m^2 - x_1 - x_2, \quad y_3 = m(x_1 - x_3) - y_1, \quad \text{wobei } m = \frac{y_2 - y_1}{x_2 - x_1}.$$

2. Wenn $x_1 = x_2$ aber $y_1 \neq y_2$, dann $P_1 + P_2 = \mathcal{O}$.
3. Wenn $P_1 = P_2$ und $y_1 \neq 0$, dann

$$x_3 = m^2 - 2x_1, \quad y_3 = m(x_1 - x_3) - y_1, \quad \text{wobei } m = \frac{3x_1^2 + A}{2y_1}.$$

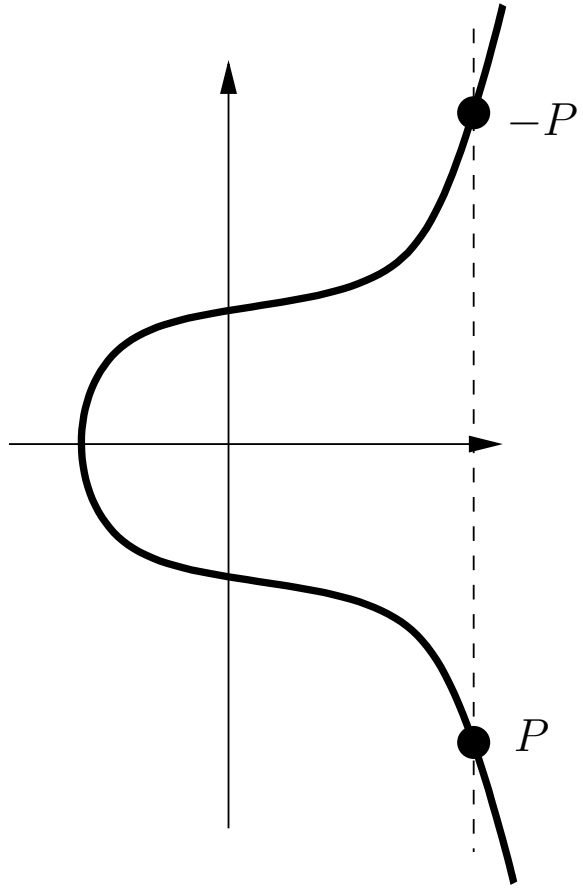
4. Wenn $P_1 = P_2$ und $y_1 = 0$, dann $P_1 + P_2 = \mathcal{O}$.
5. Für alle P über E gilt $P + \mathcal{O} = P$.

Geometrie der Addition von Punkten (1)



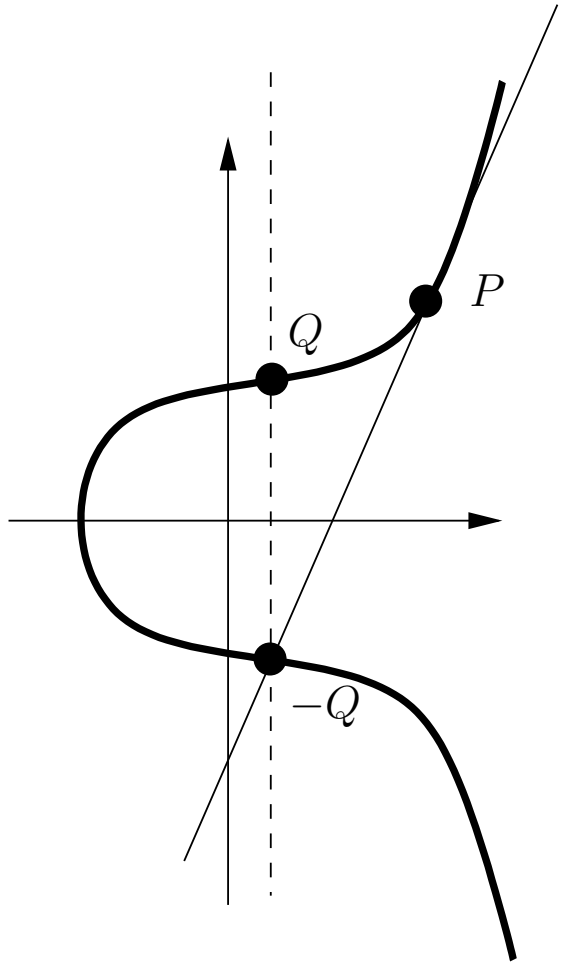
$$P_1 + P_2 = P_3.$$

Geometrie der Addition von Punkten (2)



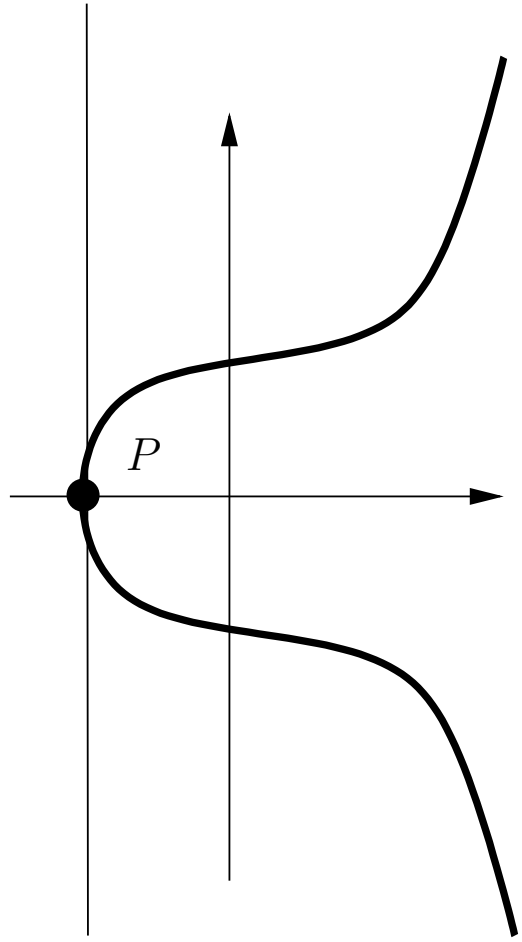
$$P + (-P) = \mathcal{O}.$$

Geometrie der Addition von Punkten (3)



$$2P = Q.$$

Geometrie der Addition von Punkten (4)



$2P = \mathcal{O}$, weil $y = 0$. P ist zu sich selbst invers ($P = (-P)$).

Eigenschaften der Addition von Punkten

1. Kommutativität: $P_1 + P_2 = P_2 + P_1$ für alle P_1, P_2 auf E.
2. Neutrales Element: $P + \mathcal{O} = P$ für alle P auf E.
3. Inverses Element: Gegeben P auf E, es existiert ein P' auf E mit $P + P' = \mathcal{O}$. Der Punkt P' wird häufig als $-P$ bezeichnet. Sei $P = (x, y)$ ein von $\mathcal{O}$ verschiedener Punkt der Kurve. Dann ist $-P = (x, -y)$ und man setzt $P + (-P) = \mathcal{O}$.
4. Assoziativität: $(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3)$ für alle P_1, P_2, P_3 auf E.

Punkte auf E bilden eine additive abelsche Gruppe mit $\mathcal{O}$ als neutrales Element.

Punktmultiplikation

- Sei P ein Punkt auf deiner elliptischen Kurven und k eine positive ganze Zahl.
- Die Operation kP ist definiert als $P + P + \cdots + P$ mit k Summanden.
- Wenn $k < 0$, dann ist $kP = (-P) + (-P) + \cdots (-P)$ mit $|k|$ Summanden.

Punktmultiplikation (Fort.)

Sei k eine positive ganze Zahl und P ein Punkt auf einer elliptischen Kurve. Der folgende Algorithmus berechnet kP.

1. Setze $a = k, B = \mathcal{O}, C = P$.
2. Wenn a gerade, dann $a = a/2$ und setze $B = B, C = 2C$.
3. Wenn a ungerade, dann $a = a - 1$ und setze $B = B + C, C = C$.
4. Wenn $a \neq 0$, gehe zu Schritt 2.
5. Geben B aus.

Die Ausgabe ist $B = kP$.

Elliptische Kurven über endlichen Körpern

- Sei $\mathbb{F}$ ein endlicher Körper und E eine elliptische Kurve definiert über $\mathbb{F}$.
- Weil es nur endliche viele Tupel (x, y) mit $x, y \in \mathbb{F}$ gibt, ist die Gruppe $E(\mathbb{F})$ endlich.

Beispiel:
Sei E die elliptische Kurve $y^2 = x^3 + x + 1$ definiert über $\mathbb{F}_5$.

- Wir zählen alle möglichen Tupel auf, d.h. zuerst für x, dann für $x^3 + x + 1 \mod 5$, dann alle Quadratwurzeln y.

x	$x^3 + x + 1$	y	Punkte
0	1	± 1	$(0,1), (0,4)$
1	3	–	–
2	1	± 1	$(2,1), (2,4)$
3	1	± 1	$(3,1), (3,4)$
4	4	± 2	$(4,2), (4,3)$
$\mathcal{O}$	$\mathcal{O}$	$\mathcal{O}$	

$E(\mathbb{F}_5)$ hat Ordnung 9.

Elliptische Kurven über endl. Körpern (Fort.)

Beispiel: $(3,1)+(2,4)=?$

- Steigung der Gerade die $(3,1)$ und $(2,4)$ schneidet

$$\frac{4-1}{2-3} \equiv 2 \mod 5.$$

- Geradengleichung $y = 2(x-3)+1 \equiv 2x$. Einsetzen ergibt

$$\begin{aligned} y^2 &= x^3+x+1 \mod 5 \\ (2x)^2 &= x^3+x+1 \mod 5 \\ 0 &= x^3-4x^2+x+1 \mod 5. \end{aligned}$$

- Wir kennen bereits $(3,1),(2,4)$, d.h. $3+2+c=4$ und somit $c=-1 \equiv 4 \mod 5$.

Elliptische Kurven über endl. Körpern (Fort.)

- Weil $y = 2x$, erhalten wir $y = 8 \equiv 3 \mod 5$.
- Spiegelung an der x-Achse ergibt $-3 \equiv 2 \mod 5$, somit $(3,1) + (2,4) = (4,2)$.

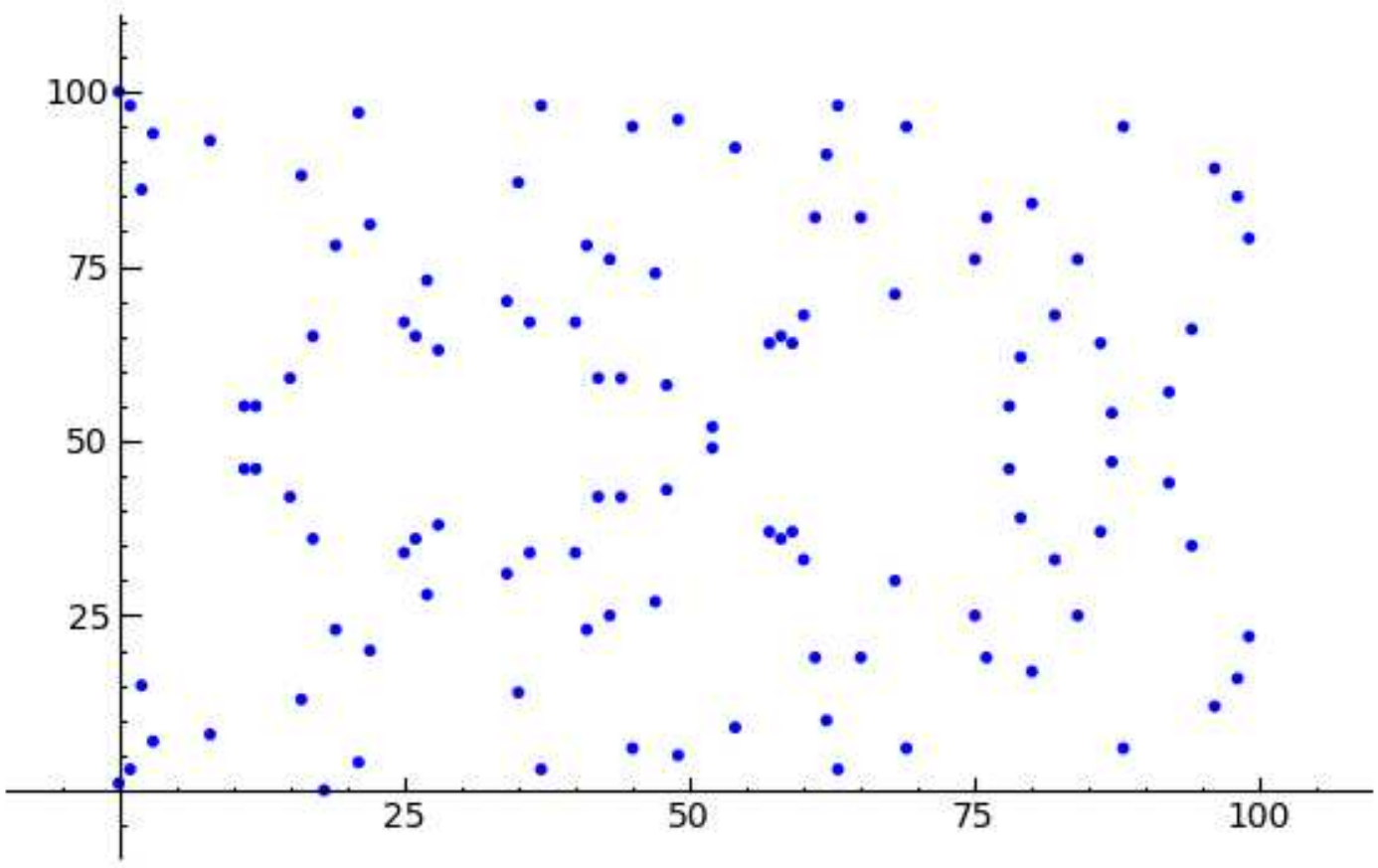
Theorem (Hasse):

Sei E eine elliptische Kurve über einem endlichen Körper $\mathbb{F}_q$. Für die Ordnung von $E(\mathbb{F}_q)$ gilt

$$|q + 1 - \#E(\mathbb{F}_q)| \leq 2\sqrt{q}.$$

Eine elliptische Kurve über $\mathbb{F}_q$ hat ungefähr q Punkte. D.h. um eine Kurve mit 2^{163} Punkten zu erhalten, braucht man $p \approx 2^{163}$.

Beispiel $y^2 = x^3 + 7x + 1$ über $\mathbb{F}_{101}$



$y^2 = x^3 + 7x + 1$ über $\mathbb{F}_{101}$

Elliptische Kurven über endl. Körpern (Fort.)

Theorem:

Sei E eine elliptische Kurve mit $y^2 = x^3 + Ax + b$ über $\mathbb{F}_q$.
Dann gilt

$$\#E(\mathbb{F}_q) = q + 1 + \sum_{x \in \mathbb{F}_q} \left(\frac{x^3 + Ax + B}{\mathbb{F}_q} \right).$$

Beispiel:

$$\begin{aligned} \#E(\mathbb{F}_5) &= 5 + 1 + \sum_{x=0}^{4} \left(\frac{x^3 + x + 1}{5} \right) \quad \text{Legendre-Symbol } \left(\frac{x}{p} \right) \\ &= 6 + \left(\frac{1}{5} \right) + \left(\frac{3}{5} \right) + \left(\frac{1}{5} \right) + \left(\frac{1}{5} \right) + \left(\frac{4}{5} \right) \\ &= 6 + 1 - 1 + 1 + 1 + 1 = 9 \end{aligned}$$

Elliptische Kurven über endl. Körpern (Fort.)

- Sei $P \in E(\mathbb{F}_q)$. Die Ordnung von P ist die kleinste positive ganze Zahl k mit $kP = \mathcal{O}$.
- Sei $E : y^2 = x^3 + x + 1 \mod 5$, die Ordnung von $(0, 1)$ ist 9 weil $9(0, 1) = \mathcal{O}$.

Für jeden Punkt G ist die Menge $(\mathcal{O}, G, G + G, G + G + G, \ldots)$ eine zyklische Untergruppe von $\mathbb{F}_q$.

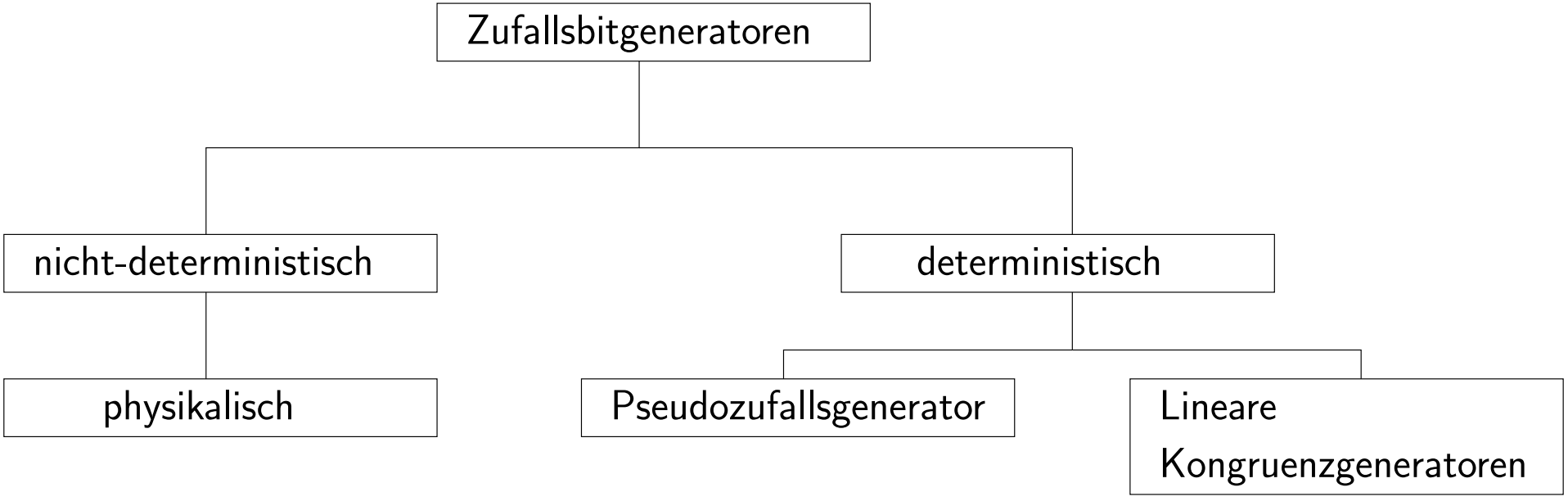
Pseudozufallsgeneratoren

- In welchen kryptographischen Verfahren werden *keine* Zufallszahlen benötigt?

Wie generiert man Zufallszahlen in einer deterministischen Maschine wie dem Computer?

- Wenn man eine Folge von Zufallszahlen erzeugt hat. Wie misst (bzw. quantifiziert) man die Güte des Zufalls?

Deterministisch und Nicht-Deterministisch



Zufallsbitgenerator

Ein *Zufallsbitgenerator* (engl. Random Bit Generator RBG) ist ein Gerät oder ein Verfahren, das eine Folge statistisch unabhängiger und gleich verteilter Bits ausgibt.

- Gleichverteilte Zahlen in einem vorgegebenen Intervall $[0, n]$ können mit einem Zufallsbitgenerator erzeugt werden.
- Hierzu wird eine Zufallsbitfolge der Länge $\lfloor \log_2 n \rfloor + 1$ erzeugt und in eine ganze Zahl umgewandelt.
- Ist die erhaltene Zahl größer als n, wird die Zahl verworfen und eine neue Zufallsbitfolge erzeugt.

Deterministischer Zufallsgenerator

- Ein deterministischer Zufallszahlengenerator liefert bei gleichen Ausgangsbedingungen immer die gleiche Folge von Zufallszahlen.

Ein *Pseudozufallsbitgenerator* (engl. Pseudo Random Bit Generator PRBG) ist ein deterministischer Algorithmus, der als Eingabe eine echt zufällige Bitfolge der Länge k erhält und als Ausgabe eine Bitfolge der Länge $l \gg k$ erzeugt, die den "Eindruck" der Zufälligkeit erweckt.

- Die Ausgabe eines PRBG ist nicht wirklich zufällig.
- Idee: Eingabe ist eine "echte" kurze Bitfolge. Diese wird benutzt um eine lange Bitfolge zu erzeugen.
- Ein Angreifer kann im Idealfall die expandierte Folge von einer echten Bitfolge der Länge l nicht unterscheiden.

Anforderungen an PRBG

- Länge k des Startwertes für einen PRBG sollte groß genug sein.
- Ausgabe eines PRBG sollte statistisch nicht von echt zufälligen Bitfolgen unterscheidbar sein.
- Die generierte Bitfolge sollte keine wiederholten Muster aufweisen.
- Die erzeugte Bitfolge soll effizient erzeugbar sein, d.h. in polynomieller Zeit bezüglich der Länge des Startwertes.

Zusammenfassend: Die Bitfolge sollte nicht vorhersagbar sein, d.h. sind der Generator und eine Anfangsbitfolge bekannt, nicht aber der *Startwert*, so soll es nicht effizient, d.h. in polynomieller Zeit, möglich sein, die Fortsetzung der Bitfolge vorherzusagen.

Anforderungen an PRBG (Fort.)

- Sind die ersten l Bits einer Bitfolge s bekannt, darf das Vorhersagen des $(l+1)$-ten Bits von s nicht signifikant größer sein als $1/2$ (next-bit Test).
- PRBG sollte "alle" statistischen Tests bestehen.
- PRBG der den next-bit Test besteht, jedoch möglicherweise basierend auf einer noch unbewiesenen mathematischen Annahme wie z.B. dem Faktorisierung großer Zahlen beruht wird kryptografisch sicherer Pseudozufallsbitgenerator (engl. CSPRBG) genannt.

Lineare Kongruenzgeneratoren

Eine linearer Kongruenzgenerator erzeugt eine Pseudozufallsfolge $x_1, x_2, x_3, \ldots$ wie folgt

$$x_n = a\, x_{n-1} + b \mod m, \quad n \geq 1,$$

wobei $m \in \mathbb{N}, a \in \{0, 1, \ldots, m-1\}$ und $b \in \{0, 1, \ldots, m-1\}$ (Modul, Faktor, Inkrement).

Der geheime Start-Parameter $x_0 \in \{0, 1, \ldots, m-1\}$ wird Saat (engl. seed) genannt.

- Solche lineare Kongruenzgeneratoren werden häufig in Simulationen oder probabilistischen Algorithmen verwendet.
- Die Ausgaben sind jedoch vorhersagbar und somit sind solche Generatoren nicht für kryptographische Verfahren geeignet.

Lineare Kongruenzgeneratoren (Beispiel)

```
# Beispiel für ungeeignete Parameterwahl
m <- 2^32;  a   <- 4095;
b <- 12794; x.0 <- 253;

x.max <- 10;
X     <- rep(0,x.max);
X[1]  <- x.0;

for (i in 2:x.max) { X[i] <- (a*X[i-1] + b) %% m }
```

Ausgabe:

$x_0 = 253, x_1 = 1048829, x_2 = 253, x_3 = 1048829, \ldots.$

- Schon nach dem zweiten Wert bildet sich eine Periode.
- Wünschenswertes Qualitätskriterium von linearen Kongruenzgeneratoren ist jedoch, dass die Länge der Periode möglichst nahe bei der Maximallänge m liegt.

CSPRNG aus Kryptografischen Verfahren

Ein CSPRNG kann aus unterschiedlichen kryptografischen Verfahren aufgebaut werden

- Stromverschlüsselung (RC4).
- Blockverschlüsselung (Triple-DES).
- Hash-Funktionen (SHA-1).
- Auf der Grundlage mathematisch harter Probleme (Faktorisierung).

ANSI X9.17 Generator (Triple-DES)

Eingabe: Zufällige gewählte und geheim gehaltene 64-Bit Saat s, $m \in \mathbb{N}$ und DES E-D-E Schlüssel k.
Ausgabe: m pseudozufällige 64-Bitfolgen $x_1, x_2, \ldots, x_m$.

- Berechne Zwischenwert $I = E_k(D)$, wobei D eine 64-Bit Darstellung von Datum/Uhrzeit mit möglichst genauer Auflösung ist.
- Für i von 1 bis m
  - $x_i \leftarrow E_k(I \oplus s)$.
  - $s \leftarrow E_k(x_i \oplus I)$.
- Ausgabe der Bitfolgen $x_1, x_2, \ldots, x_m$.

FIPS 186 Einwegfunktion mit SHA-1

Eingabe: Ein Bitfolge t der Länge 160-Bit und eine Bitfolge c der Länge b, $160 \leq b \leq 512$.
Ausgabe: Ein Bitfolge der Länge 160-Bit bezeichnet als $G(t, c)$.

- Zerlege t in fünf 32-Bit Blöcke $t = t_0 \| t_1 \| t_2 \| t_3 \| t_4$.
- Fülle c mit 0'en auf um den 512-Bit langen Nachrichtenblock $X \leftarrow c \| 0^{512-b}$ zu erzeugen.
- Zerlege X in 16 32-Bit Wörter $x_0 x_1 \ldots x_{15}$ und setze $m \leftarrow 1$.
- Führe Schritte 2-5 von SHA-1 aus (vergl. Seite 168)
- Ausgabe ist Konkatenation $G(t, c) = H_1 \| H_2 \| H_3 \| H_4 \| H_5$.

RSA Pseudozufalls-Bitgenerator

Ausgabe: Pseudozufällige Bitfolge $z_1, z_2, \ldots, z_l$ der Länge l.

1. Wähle zwei geheime RSA-Primzahlen p und q und berechne $n = pq$ und $\phi = (p-1)(q-1)$. Wähle zufällig eine ganze Zahl e, mit $1 < e < \phi$ und $\gcd(e, \phi) = 1$.
2. Wähle zufällig eine ganze Zahl x_o aus $\{1, 2, \ldots, n-1\}$ (seed).
3. Für i von 1 bis l
   - $x_i \leftarrow x_{i-1}^e \mod n$.
   - $z_i \leftarrow$ Bit von x_i mit dem niedrigsten Stellenwert.
4. Ausgabe der Folge $z_1, z_2, \ldots, z_l$.

Blum-Blum-Shup Pseudozufalls-Bitgenerator

Ausgabe: Pseudozufällige Bitfolge $z_1, z_2, \ldots, z_l$ der Länge l.

1. Wähle zwei geheime und unterschiedliche Primzahlen p und q, wobei jede kongruent zu 3 modulo 4 ist und berechne $n = pq$.
2. Wähle zufällig eine ganze Zahl s aus $\{1, 2, \ldots, n-1\}$ (seed), mit $\gcd(s, n) = 1$ und berechne $x_0 \leftarrow s^2 \mod n$.
3. Für i von 1 bis l
   - $x_i \leftarrow x_{i-1}^2 \mod n$.
   - $z_i \leftarrow$ Bit von x_i mit dem niedrigsten Stellenwert.
4. Ausgabe der Folge $z_1, z_2, \ldots, z_l$.

Blum-Blum-Shup Pseudozufalls-Bitgenerator wird auch als $x^2 \mod n$ Generator bezeichnet.

Statistische Tests

Wie misst (bzw. quantifiziert) man die Güte des Zufalls?

- 0 0 0 1 0 0 0 0 0 0 1 1 1 0 1 1 1 1 0 0 0 1 0 1 0 1 0 1 1
  1 0 1 1 1 0 0 1 0 1 0 1 1 1 1 0 1 0 0 0 1 0 1 0 1 0 1 0 1
  0 1 0 0 0 0 0 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 1 1 1 1
  1 1 1 0 0 0 1 1 0 0 1 0 1
- 1 1 0 1 0 0 1 1 1 1 1 1 0 1 0 1 1 0 0 1 1 1 1 0 0 1 1 0 0
  1 0 1 0 0 1 1 1 1 1 1 0 0 0 1 0 1 0 0 0 0 1 0 1 1 1 0 0 1
  0 1 1 0 1 0 1 1 0 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 0 0
  0 0 0 0 0 0 0 1 1 1 0 1 0 1
- Welche der beiden Bitfolgen erfüllt am besten das Kriterium der Nichtvorhersagbarkeit?

Frequenz-Test (Monobit-Test)

- Sei n_0 die Anzahl an Nullen und n_1 die Anzahl an Einsen in einer Bitfolge der Länge n.
- Die Prüfgröße für den Test ist

$$X_1 = \frac{(n_0 - n_1)^2}{n},$$

  wobei die Prüfgröße X_1 bei ausreichend großen n annähernd χ^2-verteilt mit einem Freiheitsgrad.
- Mittels des χ^2-Test (Verteilungstest) wollen wir prüfen, ob gegebene Bitfolgen auf eine bestimmte Weise verteilt sind.

Frequenz-Test (Monobit-Test) Beispiel

```
#####################################################
gen.rand.bitstring <- function(N = 10, p = c(0.5,0.5)) {
#####################################################
  X <- sample(c(0,1), N, replace = TRUE, prob = p);
  return(X);
}

#####################################################
monobit.test <- function(X) {
#####################################################

  n   <- length(X);
  n.1 <- sum(X);
  n.0 <- n - n.1;

  return( (n.0 - n.1)^2 / n );
}
```

Frequenz-Test (Monobit-Test) Beispiel (Fort.)

```
> N <- 1000;
> X.good <- gen.rand.bitstring(N,p=c(0.5,0.5));
> X.bad  <- gen.rand.bitstring(N,p=c(0.3,0.7));
> monobit.test(X.good);
[1] 0.4
> monobit.test(X.bad);
[1] 163.216
```

- Bei einem Signifikanzniveau von $\alpha = 0.05$ liegt der kritische Wert der Testprüfgröße bei $\chi^2(0.95; 1) = 3.84$.
- Weil $0.4 < 3.84$, besteht die Bitfolge `X.good` den Test.
- Wohingegen $163.21 > 3.84$ und somit besteht die Bitfolge `X.bad` den Test nicht.

Dies ist ein möglicher Test, weitere Tests sind z.B. Gap-Tests, Poker-Tests, Run-Tests (vergl. Chapter 5, Handbook of Applied Cryptography).