

Inhaltsverzeichnis

| | | |
|----------|---|-----------|
| 1 | Einführung | 3 |
| 1.1 | Inhalt | 3 |
| 2 | Kryptologie | 4 |
| 2.1 | Grundbegriffe und einfache Verfahren | 4 |
| 2.1.1 | Verschlüsselung erfordert | 4 |
| 2.1.2 | Beispiel für (nicht sicheres) symm. Verfahren | 5 |
| 2.1.3 | Prinzip von Kerkhoffs (1835-1903) | 5 |
| 2.1.4 | Arten von Angriffen | 6 |
| 3 | One-Time-Pad und perfekte Sicherheit | 7 |
| 3.1 | One-Time-Pad | 7 |
| 3.2 | Perfekte Sicherheit | 8 |
| 4 | Symmetrische Blockchiffre | 9 |
| 4.1 | Blockchiffre | 9 |
| 5 | Affin-lineare Chiffre | 10 |
| 5.1 | Vorbemerkung | 10 |
| 5.1.1 | $n \times m$ -Matrix | 10 |
| 5.1.2 | Quadratische Matrix ($n \times n$) | 11 |
| 5.2 | Affin-lineare Chiffren | 11 |
| 6 | Der Advanced Encryption Standard (AES) | 14 |
| 6.1 | Mathematische Methoden gebraucht fuer AES | 14 |
| 6.2 | SubBytes-Transfer | 15 |
| 6.3 | Shift Rows Transformation | 16 |
| 6.4 | Mix Columns Transformation | 16 |
| 6.5 | Schlüsselerzeugung | 16 |
| 7 | Public-Key-Systeme | 18 |
| 7.1 | Grundidee | 18 |
| 7.2 | RSA-Verfahren | 18 |
| 7.2.1 | Schlüsselerzeugung | 19 |

| | | |
|-----------|--|-----------|
| 7.2.2 | Verschlüsselung | 19 |
| 7.2.3 | Entschlüsselung | 19 |
| 7.3 | Sicherheit vom RSA-Verfahren | 20 |
| 7.3.1 | Wie bestimmt man große Primzahlen? | 21 |
| 7.3.2 | Fermat-Test | 21 |
| 7.3.3 | Miller-Rabin-Test | 22 |
| 7.3.4 | Diffie-Hellman-Verfahren zur Schlüsselvereinbarung | 22 |
| 7.3.5 | Sicherheit | 23 |
| 7.3.6 | Man-in-the-Middle | 23 |
| 7.4 | ElGamal-Public Key Verfahren (1984) | 23 |
| 7.4.1 | Schlüsselerzeugung | 23 |
| 7.4.2 | Verschlüsselung | 23 |
| 7.4.3 | Entschlüsselung | 23 |
| 8 | Signaturen, Hashfunktionen, Authentifizierung | 24 |
| 8.1 | Anforderung an digitale Signaturen | 24 |
| 8.2 | RSA-Signatur (vereinfachte Version) | 24 |
| 8.2.1 | Wie lassen sich lange RSA-Signaturen vermeiden? | 24 |
| 8.3 | RSA-Signatur mit HASH-Funktion | 25 |
| 8.3.1 | Angriffsmöglichkeiten | 25 |
| 8.3.2 | Satz: Geburtstagsparadoxon | 25 |
| 8.3.3 | Hashfunktion | 26 |
| 8.4 | Authentifizierung | 26 |
| 8.5 | Challenge-Response-Authentifizierung | 27 |
| 9 | Secret Sharing Scheme | 28 |
| 9.1 | (k, n) - Schwellenwertsysteme | 28 |
| 9.1.1 | Konstruktion | 28 |
| 9.1.2 | Verteilung der Teilgeheimnisse | 28 |
| 9.1.3 | Rekonstruktion(versuch) des Geheimnisses | 29 |
| 10 | Codierungstheorie | 30 |
| 10.1 | Grundbegriffe und einfache Beispiele | 30 |
| 10.1.1 | Codierung | 30 |
| 10.1.2 | Ziele | 30 |
| 10.2 | Grundprinzip | 30 |
| 10.2.1 | FEC-Verfahren (Forward Error Correction) | 31 |
| 10.2.2 | ARQ-Verfahren (Automatic Repeat Request) | 31 |

Kapitel 1

Einführung

1.1 Inhalt

Übertragung (Speicherung) von Daten:

Schutz vor:

- zufälligen oder systematischen (physikalischen bedingten) Störungen
- Abhören, absichtliche Veränderung von Dritten (Kryptologie / Verschlüsselung)

Kryptologie:

- symmetrische Verfahren
- asymmetrische Verfahren (Public-Key Verfahren)
- Authentifizierung
- Signaturen

Codierungstheorie

- Fehlererkennung und Fehlerkorrektur
- lineare Blockcodes
- Decodierverfahren

Kapitel 2

Kryptologie

2.1 Grundbegriffe und einfache Verfahren

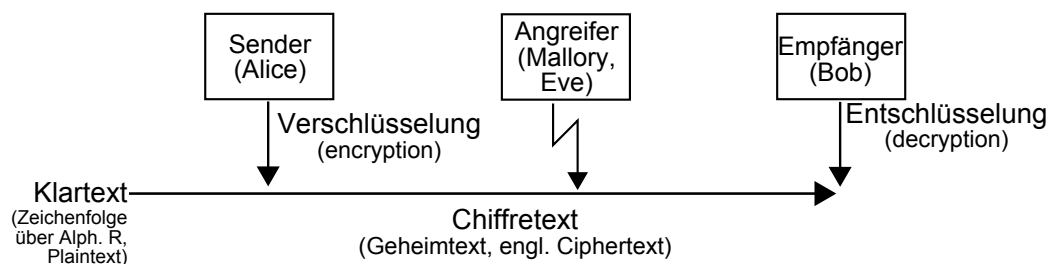


Abbildung 2.1: Schaubild der Kryptologie

2.1.1 Verschlüsselung erfordert

- Verschlüsselungsverfahren, Algorithmus (Funktion)
- Schlüssel k_e (encryption key)

$$E(m, k_e) = c$$

E =Verschlüsselungs Funktion, m =Klartext, c =Chiffretext

$$E(m_1, k_e) \neq E(m, k_e) \text{ fuer } m_1 \neq m_2$$

$$D(c, k_d) = m$$

(k_d zu k_e gehöriger Dechiffrierschlüssel!)

$k_d = k_e$ (oder k_d leicht aus k_e zu berechnen):

symmetrisches Verschl.verf., ansonsten **asymm. Verschl.verf.**. Ist k_d nur sehr schwer (oder garnicht) zu k_e berechenbar, so kann k_e veröffentl. werden:

Public-Key-Verfahren.

2.1.2 Beispiel für (nicht sicheres) symm. Verfahren

a) $R = S = \{0, 1, \dots, 25\}$

Verfahren: Verschiebechiffre

Schlüssel: $i \in \{0, 1, \dots, 25\}$

Verfahren $x \in \mathbb{R} \rightarrow x + i \bmod 26 = y$

$y \mapsto y - i \bmod 26 = x$

$m = x_1 \dots x_n \rightarrow c = (x_1 + i \bmod 26) \dots (x_n + i \bmod 26), E(m, i)$

Unsicher, weil Schlüsselmenge klein ist (Brute Force Angriff).

b) R, S , Schlüsselmenge=Menge aller Permutationen von $\{0, 1, \dots, 25\} = S_{26}$

Verschl.: Wähle Permutation π

$x \in \mathbb{R} \rightarrow \pi(x) = y$

Entschl.: $y \rightarrow \pi^{-1}(y) = x$

$m = x_1 \dots x_r \rightarrow c = \pi(x_1) \dots \pi(x_r)$

$\begin{pmatrix} 0 & 1 & 2 & \dots & 25 \\ 3 & 17 & 4 & \dots & 13 \end{pmatrix} \rightarrow \pi(0) = 3, \text{ u.s.w.}$

Anzahl der Permutationen: $|S_{26}| = 26! \approx 4 \cdot 10^{26} \rightarrow$ Brute-Force Angriff nicht mehr möglich!

Warum? Man muss im Schnitt 50% der Permutationen testen. Angenommen man könnte 10^6 Perm. pro Sekunde testen.

Aufwand: $2 \cdot 10^{14}$ Sekunden $\approx 6.000.000$ Jahre

Trotzdem unsicher!

Grund: Charakteristische Häufigkeitsverteilung von Buchstaben in natürlichspr. Texten.

Verfahren beinhalten viele Verschlüsselungsmöglichkeiten, abhängig von der Auswahl des Schlüssels.

Verfahren bekannt, aber Schlüssel k_d geheim!

2.1.3 Prinzip von Kerkhoffs (1835-1903)

Sicherheit eines Verschlüsselungsverfahrens darf nicht von der Geheimhaltung des Verfahrens, sondern nur von der Geheimhaltung des verwendeten Schlüssels abhängen!

Kryptologie besteht aus Kryptographie (Entwurf) und der Kryptoanalyse (Angriff).
Angriffserfolge:

- Schlüssel k_d wird gefunden
- Eine zu der Dechiffrierfunktion $D(\cdot, k_d)$ äquivalente Funktion finden ohne Kenntnis von k_d
- gewisse Chiffretexte werden entschlüsselt

2.1.4 Arten von Angriffen

- Ciphertext-Only Angriff
- Known-Plaintext Angriff
- Chosen-Plaintext Angriff
- Chosen-Ciphertext Angriff

Kapitel 3

One-Time-Pad und perfekte Sicherheit

Lauftextverschlüsselung

Alphabet $\mathbb{Z}_k = \{0, 1, \dots, k-1\}$

In \mathbb{Z}_k kann man addieren und multiplizieren mit $\text{mod } k$.

Klartext x_1, x_2, \dots, x_n

Schlüsselwort k_1, k_2, \dots, k_n

$x_1 + k_1 \text{ mod } k, x_n + k_n \text{ mod } k \leftarrow$ Chiffretext

Mit natürlichsprachlichen Texten ist das Verfahren unsicher.

$\mathbb{Z}_2 = \{0, 1\}, 1 \oplus 1 = 0 = 0 \oplus 0, 0 \oplus 1 = 1 = 1 \oplus 0 \Rightarrow \text{XOR}$

Klartext in $\mathbb{Z}_2^n = \{(x_1, \dots, x_n) : x_i \in \mathbb{Z}_2\}$ Schlüssel: Zufallsfolge über \mathbb{Z}_2 der Länge n . m Klartext, k Zufallsfolge (beide Länge n)

$$c = m \oplus k, (x_1, \dots, x_n) \oplus (k_1, \dots, k_n) := (x_1 \oplus k_1, \dots, x_n \oplus k_n)$$

3.1 One-Time-Pad

Schlüssel k darf nur einmal verwendet werden!

$$m_1 \oplus k = c_1, m_2 \oplus k = c_2, c_1 \oplus c_2 = m_1 \oplus k \oplus m_2 \oplus k = m_1 \oplus m_2$$

Wieder nur Lauftext \rightarrow unsicher!

m_1 und m_2 lässt sich ermitteln.

Zufallsfolge der Länge n : eigentlich unsinniger Begriff. Da jedes Bit unabhängig von anderen mit Wahrscheinlichkeit $\frac{1}{2}$ erzeugt wird (Output einer binär symmetrischen Quelle)

Jede Folge der Länge n ist gleich wahrscheinlich (Wahrscheinlichkeit $\frac{1}{2^n}$)

One-Time-Pad ist perfekt sicher.

3.2 Perfekte Sicherheit

Ein Verschlüsselungsverfahren ist perfekt sicher, falls gilt: Für jeden Klartext m und jedem Chiffretext c (der festen Länge n)

$$pr(m|c) = pr(m)$$

$pr(m|c) \rightarrow$ A-posteriori-Wahrscheinlichkeit (Wahrscheinlichkeit, dass m Klartext, wenn c empfangen wurde)

$pr(m) \rightarrow$ A-priori-Wahrscheinlichkeit

Beispiel: Substitutionschiffre aus Kapitel 2.

$n = 5, m = HALLO, pr(m) > 0$

Ang: $c = QITUA$ wird empfangen, $LL \neq TU \rightarrow pr(m|c) = 0$
nicht perfekt sicher.

One-Time-Pad ist perfekt sicher.

(Bayes'sche Formel) $m \oplus k$

Jede Folge c lässt sich mit geeignetem k in der Form $c = m \oplus k$ erhalten.

Wähle $k = m \oplus c, m \oplus k = m \oplus m \oplus c = c$

Bei gegebenem m und zufällige gewählten Schlüssel k ist jeder Chiffretext gleichwertig.

Kapitel 4

Symmetrische Blockchiffre

4.1 Blockchiffre

Zerlege Klartext in Blöcke (Strings) der Länge n . Jeder Block wird einzeln verschlüsselt (in der Regel wieder in einem Block der Länge n). Gleiche Blöcke werden gleich verschlüsselt.

Wieviele Blockchiffren der Länge n gibt es?

Alphabet $\mathbb{Z}_2 = \{0, 1\}$

$$|\underbrace{\{(0, \dots, 0), (0, \dots, 1), \dots, (1, \dots, 1)\}}_{\text{Block}}| = 2^n$$

Blockchiffre = Permutation der 2^n Blöcke.

$(2^n)!$ Blockchiffre

Wenn alle verwendet werden:

Schlüssel = Permutation der 2^n Blöcke

$(x_{1,1}, \dots, x_{1,n}, x_{2,1}, \dots, x_{2,n}, \dots)$ $n \cdot 2^n$ Bit

Zur Speicherung eines Schlüssels werden $n \cdot 2^n$ Bit benötigt.

Zum Beispiel:

$$n = 64, 64 \cdot 2^{64} = 2^{70} \approx 1 \text{ ZetaByte} \approx 1 \text{ Milliarde Festplatten à 1 TB}$$

Illusional!

Konsequenz: Verwende Verfahren, wo nur ein kleiner Teil der Permutation als Schlüssel verwendet wird und so sich die Schlüssel dann in kürzerer Form darstellt.

Kapitel 5

Affin-lineare Chiffre

5.1 Vorbemerkung

5.1.1 $n \times m$ -Matrix

$$\begin{pmatrix} a_{11} & \dots & a_{1m} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nm} \end{pmatrix}$$

$1 \times n$ = Zeilenvektor $= (a_1, \dots, a_n)$

$n \times 1$ = Spaltenvektor $= \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$

z.B. $a_{ij} \in \mathbb{R}$, $a_{ij} \in \mathbb{Z}$ oder $a_{ij} \in R$, R Ring

$n \times m$ -Matrix A, B

$$\begin{pmatrix} a_{11} & \dots & a_{1m} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nm} \end{pmatrix} + \begin{pmatrix} b_{11} & \dots & b_{1m} \\ \vdots & & \vdots \\ b_{n1} & \dots & b_{nm} \end{pmatrix} := \begin{pmatrix} a_{11} + b_{11} & \dots & a_{1m} + b_{1m} \\ \vdots & & \vdots \\ a_{n1} + b_{n1} & \dots & a_{nm} + b_{nm} \end{pmatrix}$$

$$A = n \times m, B = m \times k,$$

$$A \cdot B = \begin{pmatrix} c_{11} & \dots & c_{1k} \\ \vdots & & \vdots \\ c_{m1} & \dots & c_{mk} \end{pmatrix} = n \times k$$

$$c_{1l} = (a_{i1} \cdot b_{ij}) + (a_{i2} \cdot b_{2j}) + \dots + (a_{im} \cdot b_{mj})$$

$$(A + B) \cdot C = A \cdot B + B \cdot C$$

Im Allgemeinen: $A \cdot B \neq B \cdot A$

5.1.2 Quadratische Matrix ($n \times n$)

$$E_n = \begin{pmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{pmatrix}$$

$$A = n \times n, A \cdot E_n = E_n \cdot A = A$$

A $n \times n$ -Matrix über kommutativen Ring R mit Eins.

Wann existiert Matrix A^{-1} (Inverse Matrix) mit $A^{-1} \cdot A = A \cdot A^{-1} = E_n$?

$\det(A) \in R$ Determinante von A

$$2 \times 2\text{-Matrix: } \det \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = a_{11} \cdot a_{22} - a_{12} \cdot a_{21}$$

A besitzt inverse Matrix $\Leftrightarrow \det(A)$ in R ein inverses besitzt

(z.B. R Körper, $\mathbb{Z}, \mathbb{Q}, \mathbb{Z}_p, \det(A) \neq 0$)

$$A^{-1} = \begin{pmatrix} \frac{1}{\det(A)} \cdot b_{11} & \dots & \frac{1}{\det(A)} \cdot b_{1m} \\ \vdots & & \vdots \\ \frac{1}{\det(A)} \cdot b_{n1} & \dots & \frac{1}{\det(A)} \cdot b_{nm} \end{pmatrix}$$

$$b_{ij} = (-1)^{i+j} \det(A_{ji})$$

$A_{ji} = (n-1) \times (n-1)$ -Matrix, die aus A durchstreichen der j -ten Zeile und i -ten Spalte entsteht.

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad A^{-1} = \begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix}$$

$$R = \mathbb{Z}_k \{0, 1, \dots, k\}$$

Addition und Multiplikation in $\mathbb{Z}_k(\oplus, \odot)$

normale Add. und Mult. mit *mod* k

5.2 Affin-lineare Chiffren

Klartextalphabet = Chiffretextalphabet = \mathbb{Z}_k ($k = 2, k = 26$)

Wähle $n \times n$ -Matrix A über \mathbb{Z}_k und Zeilenvektor b der Länge n über \mathbb{Z}_k . Dies wird der Schlüssel sein für die Chiffrierung.

Blockchiffre der Länge n . Block = Zeilenvektor der Länge n über \mathbb{Z}_k . Klartextblock v

Chiffretextblock $v \cdot A + b =: w$

$v \rightarrow v \cdot A + b =: w \quad w - b = v \cdot A$ benötigen: A^{-1} existiert (d.h. $\text{ggT}(\det(A), k) = 1$)

Dechiffrierung: $(w - b) \cdot A^{-1} = v \cdot A \cdot A^{-1} = v \cdot E_n = v$

(wenn immer $b=0$ gewählt wird, dann lineare Chiffren, Hill-Chiffren)
 Beispiel:

$$A = \begin{pmatrix} 1 & 3 \\ 3 & 2 \end{pmatrix} \mathbb{Z}_6$$

Blockchiffre der Länge n $\det(A) = 1 \cdot 2 - 3 \cdot 3 = -7 = 5$ inverse in \mathbb{Z}_6

$$\frac{1}{\det(A)} = \det(A)^{-1} = 5$$

$$A^{-1} = 5 \cdot \begin{pmatrix} 2 & -3 \\ -3 & 1 \end{pmatrix} = \begin{pmatrix} 10 & -15 \\ -15 & 5 \end{pmatrix} = \begin{pmatrix} 4 & 3 \\ 3 & 5 \end{pmatrix}$$

Test:

$$A \cdot A^{-1} = \begin{pmatrix} 1 & 3 \\ 3 & 2 \end{pmatrix} \cdot \begin{pmatrix} 4 & 3 \\ 3 & 5 \end{pmatrix} = \begin{pmatrix} 4+9 & 3+15 \\ 12+6 & 9+10 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Verschlüsselung:

$$\text{Schlüssel: } A = \begin{pmatrix} 1 & 3 \\ 3 & 2 \end{pmatrix} b = (3, 5)$$

Klartextblock: $(1, 2)$

Chiffretextblock:

$$w = (1, 2) \cdot \begin{pmatrix} 1 & 3 \\ 3 & 2 \end{pmatrix} + (3, 5) = (1, 1) + (3, 5) = (4, 0)$$

Entschlüsselung:

$$(w - b) \cdot A^{-1} = (1, 1) \cdot \begin{pmatrix} 4 & 3 \\ 3 & 5 \end{pmatrix} = (1, 2)$$

\mathbb{Z}_2 : $n^2 + n$ Bit zur Speicherung eines Schlüssels.

Wieviele inverse Matrizen über \mathbb{Z}_2 mit $n = 64$?

$$(2^{64} - 1) \cdot (2^{64} - 2) \cdot \dots \cdot (2^{64} - 2^{63}) \approx 0.29 \cdot 2^{4096}$$

Verfahren ist unsicher gegenüber Known-Plaintext-Angriffe.

(A, b) Schlüssel, A inverse $n \times n$ -Matrix über \mathbb{Z}_k , $b \in \mathbb{Z}_k^n$

Angenommen Angreifer kennt $n+1$ Klartext/Chiffretextpaare verschlüsselt mit

$(A, b), v_0, v_1, \dots, v_n w_0, \dots, w_n$

Dann kann er häufig (A, b) bestimmen.

$$V = \begin{pmatrix} v_1 - v_0 \\ v_2 - v_0 \\ \vdots \\ v_n - v_0 \end{pmatrix} n \times n\text{-Matrix}$$

Angenommen: V ist invertierbar. Setze $W = \begin{pmatrix} w_1 - w_0 \\ \vdots \\ w_n - w_0 \end{pmatrix}$

$$V \cdot A = \begin{pmatrix} (v_1 - v_0) \cdot A \\ \vdots \\ (v_n - v_0) \cdot A \end{pmatrix} = \begin{pmatrix} v_1 \cdot A + b - v_0 \cdot A + b \\ \vdots \\ v_n \cdot A + b - v_0 \cdot A + b \end{pmatrix} = \begin{pmatrix} w_1 - w_0 \\ \vdots \\ w_n - w_0 \end{pmatrix} = W$$

$V \cdot A$ bekannt, also auch V^{-1} :

$$A = V^{-1} \cdot w$$

$$b = w_0 - v_0 \cdot A$$

Beispiel: $n = 2, k = 25 \quad \{A, \dots, Z\} = \{0, \dots, 25\}$

| HERBST | | | NEBLIG | | |
|--------|----|-------|--------|----|-------|
| H | 7 | v_0 | N | 13 | w_0 |
| E | 4 | | E | 4 | |
| R | 17 | v_1 | B | 1 | w_1 |
| B | 1 | | L | 11 | |
| S | 18 | v_2 | I | 8 | w_2 |
| T | 19 | | G | 6 | |

$$V = \begin{pmatrix} 10 & -3 \\ 11 & 15 \end{pmatrix} = \begin{pmatrix} 10 & 23 \\ 11 & 15 \end{pmatrix}, \quad W = \begin{pmatrix} 14 & 7 \\ 21 & 2 \end{pmatrix}$$

$$\det(V) = 10 \cdot 15 + 33 = 183 \equiv 1 \pmod{26}$$

$$V^{-1} = \begin{pmatrix} 15 & 3 \\ -11 & 10 \end{pmatrix} = \begin{pmatrix} 15 & 3 \\ 15 & 10 \end{pmatrix}$$

$$A = V^{-1} \cdot W = \begin{pmatrix} 15 & 3 \\ 15 & 10 \end{pmatrix} \cdot \begin{pmatrix} 14 & 7 \\ 21 & 2 \end{pmatrix} = \begin{pmatrix} 210 + 63 & 105 + 6 \\ 210 + 210 & 105 + 20 \end{pmatrix} = \begin{pmatrix} 13 & 7 \\ 4 & 21 \end{pmatrix}$$

$$b = w_0 - v_0 \cdot A = (13, 4) - (7, 4) \cdot \begin{pmatrix} 13 & 7 \\ 4 & 21 \end{pmatrix} = (10, 1)$$

Test:

$$v_1 \cdot A + b = w_1, v_2 \cdot A + b = w_2$$

Kapitel 6

Der Advanced Encryption Standard (AES)

6.1 Mathematische Methoden gebraucht fuer AES

Seit 70er Jahren gab es DES (Blocklänge 64 Bit, Schlüssellänge 56 Bit)

Nachfolger des DES: Daemen, Rijmen (Belgier)
Rijndael-Verfahren → AES (2002 FIPS 197)

Iterierte Blockchiffre
Version mit 128 Bit Block und Schlüssellänge.

<BILD VON EINER RUNDE VON AES KOMMT HIER HIN>

Vorbemerkung: 128-Bit Blöcke werden dargestellt als:

$$\begin{pmatrix} a_{01} & a_{02} & \dots & a_{03} \\ a_{10} & a_{11} & \dots & a_{13} \\ \vdots & \vdots & \vdots & \vdots \\ a_{30} & \dots & \dots & a_{33} \end{pmatrix}$$

Jedes a_{ij} = Byte

128er Block $\hat{=}$ $a_{00}a_{10}a_{20} \dots a_{01}a_{11} \dots a_{33}$ (spaltenweise gelesen)

endlicher Körper: einfachste Möglichkeit \mathbb{Z}_p (p Primzahl)

\mathbb{F}_{2^8} Körper mit $2^8 = 256$ Elementen

Menge: Polynome vom Grad < 8 über \mathbb{Z}_2

$$b_7x^7 + \dots + b_1x + b_0, b_i \in \mathbb{Z}_2$$

(b_7, b_6, \dots, b_0) Byte

Addition = normale Addition von Polynomen

Multiplikation = normale Multiplikation von Polynomen + Reduktion modulo irreduzibler Polynom vom Grad 8. $(x^8 + x^4 + x^3 + x + 1)$

Bsp.

$$(x^7 + x + 1) \odot (x^3 + x) = x^{10} + x^8 + x^4 + x^3 + x^2 + x$$

$$x^{10} + x^8 + x^4 + x^3 + x^2 + x \bmod x^8 + x^4 + x^3 + x + 1$$

$$x^{10} + x^8 + x^4 + x^3 + x^2 + x \div x^8 + x^4 + x^3 + x + 1 = x^2 + 1$$

$$\begin{array}{r} x^{10} + x^6 + x^5 + x^3 + x^2 \\ \underline{x^8 + x^6 + x^5 + x^4 + x} \\ x^8 + x^4 + x^3 + x + 1 \\ \underline{x^6 + x^5 + x^3 + 1} \leftarrow \end{array}$$

$$(x^7 + x + 1) \odot (x^3 + x) = x^6 + x^5 + x^3 + 1$$

In \mathbb{F}_{2^8} hat jedes Element $\neq 0$ ein Inverses bzgl. \odot :

$$g \neq 0. \exists x. g^{-1} \in \mathbb{F}_{2^8} : g \odot g^{-1} = 1$$

Erweiterte Euklid. Algo. für Polynome:

$$g \neq 0 \text{ (Grad} \leq 7) \quad h = x^8 + x^4 + x^3 + x + 1 \text{ irred.} \quad \text{ggT}(g, h) = 1$$

$$\text{EEA: } u, v \in \mathbb{Z}_2[x] : u \cdot g + v \cdot h = 1$$

$$u \bmod h =: g^{-1}$$

$$g^{-1} \odot g = ((u \bmod h) \cdot g) \bmod h = u \cdot g \bmod h = (1 - vh) \bmod h = 1 \bmod h = 1$$

6.2 SubBytes-Transfer

$$S_{i-1} = \begin{pmatrix} a_{01} & a_{02} & \dots & a_{03} \\ a_{10} & a_{11} & \dots & a_{13} \\ \vdots & \vdots & \ddots & \vdots \\ a_{30} & \dots & \dots & a_{33} \end{pmatrix}, a_{ij} \text{ Bytes}$$

Sei g eines dieser Bytes, $g = (b_7 b_6 \dots b_0)$, $b_i \in \mathbb{Z}_2$

1. Schritt: Fasse g als Element in \mathbb{F}_{2^8} auf.

Ist $g = (0, \dots, 0)$, so lasse g unverändert.

Ist $g \neq (0, \dots, 0)$, so ersetze g durch g^{-1} .

2. Schritt: Ergebnis nach Schritt 1: \tilde{g} wird folgenderm. transformiert

$\tilde{g} \cdot A + b = \tilde{\tilde{g}}$ (affin-lin. Transformation) (\tilde{g} : g-schlange, $\tilde{\tilde{g}}$: g-doppel-schlange)

A wird durch zyklischer Shift der vorherigen Zeile um 1 Stelle nach rechts erzeugt.

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

Schritt 1 und 2 werden kombiniert, nicht jedes mal berechnet. Alle möglichen Sub-Bytes (2^8 viele) sind in einer 16x16 Matrix und wird per Table-Lookup nachgeschlagen.

$g = (b_7b_6b_5b_4b_3b_2b_1b_0)$ $b_7b_6b_5b_4 = 0$ bis 15 (Zeile) $b_3b_2b_1b_0$ (Spalte)

6.3 Shift Rows Transformation

4x4-Matrix von Bytes: $\begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix}$ $\begin{matrix} \leftarrow \text{Erste Zeile unverändert} \\ \leftarrow 1 \text{ Stelle nach links zykl.} \\ \leftarrow 2 \text{ Stellen nach links zykl.} \\ \leftarrow 3 \text{ Stellen nach links zykl.} \end{matrix}$

6.4 Mix Columns Transformation

4x4-Matrix, Einträge als Elemente in \mathbb{F}_{2^8} auffassen.

Multiplikation von links mit Matrix (Mult. der Eintr. in \mathbb{F}_{2^8}): $\begin{pmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{pmatrix}$

$x \triangleq (0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0)$

6.5 Schlüsselerzeugung

Ausgangsschlüssel hat 128 Bit. (16er String in Hexcode)

Schreibe als 4x4-Matrix von Bytes. 4 Spalten $w(0), w(1), w(2), w(4)$. Definiere weitere 40 Spalten à 4 Bytes.

$w(i-1)$ sei schon definiert.

$4 \nmid i : w(i) := w(i-4) \oplus w(i-1)$ (byteweise XOR)

$4 \mid i : w(i) := w(i-4) \oplus T(w(i-1))$ (T Transformation)

T?

$$w(i-1) = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}, a, \dots, d \text{ Bytes}$$

Wende auf b, c, d, a SubBytes-Transformation an $\rightarrow e, f, g, h$

$$r(i) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}^{\frac{(i-4)}{4}} \text{ Potenz. in } \mathbb{F}_{2^8}$$

$$T(w(i-1)) = \begin{pmatrix} e \oplus r(i) \\ f \\ g \\ h \end{pmatrix}$$

Rundenschlüssel K_i : 4x4-Matrix mit Spalten $w(4i), w(4i+1), w(4i+2), w(4i+3)$

(Nebenbemerkung: Linear heißt $f(x+y) = f(x) + f(y)$)

Kapitel 7

Public-Key-Systeme

7.1 Grundidee

Diffie, Hellman, 1976 Jeder Teilnehmer hat ein Paar von Schlüsseln:

- Öffentlichen Schlüssel P_A
- geheimen Schlüssel G_A

Zu P_A gehört öffentlich bekannte Verschlüsselungsfunktion $E_{P_A} (=E(\cdot, P_A))$

$B \xrightarrow{m} A : E_{P_A}(m) = c$

1. m darf mit "realistischen Aufwand" nicht aus $E_{P_A}(m)$ berechenbar sein. E_{P_A} ist **Einwegfunktion**
(E_{P_A} muss effizient berechenbar sein, aber $E_{P_A}^{-1}$ nicht!)
2. A muss mit Hilfe einer Zusatzinformation ($=G_A$) in der Lage sein, $E_{P_A}^{-1}$ effizient zu berechnen.

$$D_{G_A}(c) = m = E_{P_A}^{-1}(c)$$

Injektive Einwegfunktionen, die mit Zusatzinformation effizient invertierbar sind: **Geheimtürfunktion** (trapdoor function)

Aus 1) und 2) folgt:

- 3 G_A darf aus P_A nicht schnell berechenbar sein!

Es ist unbekannt ob Einwegfunktion existieren! Notwendig für die Existenz von Einwegfunktionen:

$$P \neq NP$$

Es gibt Kandidaten für Einwegfunktionen.

7.2 RSA-Verfahren

(Rivest, Shamir, Adleman, 1977) Beruht auf Schwierigkeit große Zahlen zu faktorisieren!

7.2.1 Schlüsselerzeugung

Wähle zwei große Primzahlen $p, q (p \neq q)$ (mindestens 500 Bit Länge)

Bilde $n = p \cdot q$

$$\varphi(n) = ||\{a \in \mathbb{N} : 1 \leq a < n, \text{ggT}(a, n) = 1\}||$$

$$n = p \cdot q : \varphi(n) = (p - 1) \cdot (q - 1)$$

[nicht teilerfremd zu n : $1 \cdot p, 2 \cdot p, \dots, (q - 1) \cdot p = n = 1 \cdot q, 2 \cdot q, \dots, (p - 1) \cdot q$

$(p - 1) + (q - 1) + 1$

$$\varphi(n) = n - (p - 1) - (q - 1) - 1 = n - p - q + 1 = p \cdot q - p - q + 1 = (p - 1) \cdot (q - 1)]$$

Wähle $e, 1 < e < \varphi(n)$ mit $\text{ggT}(e, \varphi(n)) = 1$

Zufallswahl, bestimme $\text{ggT}(e, \varphi(n))$ mit Euklidischer Algorithmus, so lange, bis e mit $\text{ggT}(e, \varphi(n)) = 1$ gefunden ist.)

öffentlicher Schlüssel

$$(n, e) = P_A$$

Wähle $d < \varphi(n)$ mit $e \cdot d \equiv 1 \pmod{\varphi(n)}$ (d.h. $\varphi(n) \mid e \cdot d - 1, e \cdot d = 1 + k \cdot \varphi(n)$ für $k \in \mathbb{N}$)

(Wende erweiterten Euklidischen Algorithmus auf $e, \varphi(n)$ an:

Liefert $u, v \in \mathbb{Z}$ mit $u \cdot e + v \cdot \varphi(n) = \text{ggT}(e, \varphi(n)) = 1$

$$d = u \bmod \varphi(n)$$

$$u \cdot e + v \cdot \varphi(n) \bmod \varphi(n) = 1$$

$$\underbrace{(u \bmod \varphi(n)) \cdot e}_{d} \bmod \varphi(n) = 1$$

Geheimerschlüssel

$$G_A = d$$

7.2.2 Verschlüsselung

B Nachricht an A . Codiere Nachricht als Zahl. Zerlege in Blöcke deren Zahlwert $< n$. Sei m so ein Block. ($m < n$)

$$m^e \bmod n = c$$

7.2.3 Entschlüsselung

$$c^d \bmod n = m$$

Gültigkeit basiert auf kleinem Satz von Fermat:

r Primzahl, $\text{ggT}(a, r) = 1$ (d.h. $r \nmid a$)

$$a^{r-1} \equiv 1 \pmod{r}$$

Sei $m < n = p \cdot q$

$$c = m^e \bmod n, c^d \bmod n = m^{e \cdot d} \bmod n$$

$$e \cdot d = 1 + k \cdot \varphi(n) = 1 + k \cdot (p-1) \cdot (q-1)$$

Ist $p \nmid m$, so

$$m^{e \cdot d} = m^{1+k \cdot (p-1) \cdot (q-1)} = m \cdot (m^{p-1})^{k \cdot (q-1)} \stackrel{\text{mod } p}{\Rightarrow} m \cdot 1^{k \cdot (q-1)} (\text{mod } p) = m (\text{mod } p)$$

Ist $p \mid m$:

$$m \equiv 0 \equiv m^{e \cdot d} (\text{mod } p)$$

In jedem Fall:

$$m^{e \cdot d} \equiv m (\text{mod } p)$$

Genauso:

$$m^{e \cdot d} \equiv m (\text{mod } q)$$

$$p \mid m^{e \cdot d} - m, q \mid m^{e \cdot d} - m, p \neq q \Rightarrow n = p \cdot q \mid m^{e \cdot d} - m$$

$$m^{e \cdot d} \equiv m (\text{mod } n), m^{e \cdot d} \text{ mod } n = m$$

Schnelle Berechnung von modularen Produkten

$$m^e \text{ mod } n$$

$$e = \sum_{i=0}^k e_i \cdot 2^i, e_i \in \{0, 1\}, e_k = 1$$

$$m^e = m^{2 \cdot k + e_{k-1} \cdot 2^{k-1} + \dots + e_1 \cdot 2 + e_0}$$

$$(((\dots ((m^2 \cdot m^{e_{k-1}})^2 \cdot m^{e_{k-2}})^2 \dots)^2 \cdot m^{e_1})^2 \cdot m^{e_0})$$

gelöst im worst case mit $2 \cdot k$ Multiplikationen

$$k = \lfloor \log_2(e) \rfloor$$

Nach jedem Rechenschritt $\text{mod } n$ reduzieren!

7.3 Sicherheit vom RSA-Verfahren

Falls p, q bekannt $\Rightarrow \varphi(n), d$ bekannt.

$\varphi(n)$ bekannt $\Rightarrow p, q$ bekannt.

$\varphi(n) = n - p - q + 1$ bekannt $\Rightarrow p + q = s$ bekannt, $p \cdot q = n$ bekannt.

$p \cdot (s - p) = n \Rightarrow p^2 - s \cdot p + n = 0$ quadratische Gleichung für p

Es gilt auch: Bestimmung von d ist „genauso schwierig“ wie die Faktorisierung von n .

Komplexität der besten Faktorisierungsalgorithmen:

$$O(e^{c \cdot (\log n)^{\frac{1}{3}} \cdot ((\log \log n)^{\frac{2}{3}})})$$

Um eine 640 Bit Zahl zu faktorisieren braucht man 30-CPU-Jahre auf einer 2.2 GHz CPU.

Häufig wird $e = 3$ gewählt.

HIER KOMMT NOCH EINE GRAFIK HIN

$$\text{ggT}(n_i, n_j) = 1$$

$$c_1 = m^3 \bmod n_1$$

$$c_2 = m^3 \bmod n_2$$

$$c_3 = m^3 \bmod n_3$$

Eve fängt c_1, c_2, c_3 ab: Chinesisches Restsatz:

$$0 \leq x \leq n_1, n_2, n_3 \text{ mit } x = c_i \bmod n_i$$

$$i = 1, 2, 3$$

x ist eindeutig bestimmbar

$$m^3 \equiv c_i \bmod n_i, m^3 < n_1, n_2, n_3$$

$$\Rightarrow x = m^3 \Rightarrow m = \sqrt[3]{x}$$

Wenn $e = 5$, dann braucht man 5 Nachrichten.

7.3.1 Wie bestimmt man große Primzahlen?

$$p \text{ Primzahl, } a \in \mathbb{Z}, \text{ ggT}(a, p) = 1$$

$$a^{p-1} \equiv 1 \pmod{p} \text{ [kl. Satz von Fermat]}$$

gegeben: $n, \text{ ggT}(a, n) = 1, a^{n-1} \equiv 1 \pmod{n}$?

7.3.2 Fermat-Test

Wenn nicht, so ist n keine Primzahl. Wenn ja, so keine Aussage möglich. Wähle neues a !

Es gibt zusammengesetzte Zahlen n (Carmichael-Zahlen) mit:

$$a^{n-1} \equiv 1 \pmod{n} \forall a \text{ mit } \text{ggT}(a, n) = 1$$

7.3.3 Miller-Rabin-Test

$$\begin{aligned}
 &ggT(a, p) = 1 \\
 &p \text{ Primzahl } p - 1 = 2^s \cdot t, \quad 2 \nmid t \\
 &a^{2^{s \cdot t}} \equiv 1 \pmod{p} \\
 &(a^{2^{s-1} \cdot t})^2 = b \\
 &a^{2^{s-1} \cdot t} = \begin{cases} 1 \pmod{p} \\ -1 \pmod{p} \end{cases} \\
 &b^2 \equiv 1 \pmod{p} \\
 &(b \pmod{p})^2 = 1 \in \mathbb{Z}_p \\
 &x^2 - 1 \in \mathbb{Z}_p[x]
 \end{aligned}$$

Entweder $a^t \equiv 1 \pmod{p}$ oder $a^{2^i \cdot t} \equiv -1 \pmod{p}$ für ein $0 \leq i \leq s$ Teste dies mit n statt p .

Wenn n keine Primzahl ist, dann gibt es mindestens $\frac{3}{4}\varphi(n)$ viele a , so dass der Test fehlschlägt.

→ probabilistischer Primzahltest

p Primzahl $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$ Gruppe bezüglich Multiplikation (zyklisch)

$$\begin{aligned}
 \exists g \in \mathbb{Z}_p^* : \{g^0, g^1, g^2, \dots, g^{p-2}\} &= \mathbb{Z}_p^* \\
 g^{p-1} &\equiv 1 \pmod{p}
 \end{aligned}$$

Primitivwurzel \pmod{p}

$0 \leq a \leq p-2 : a \mapsto g^a \pmod{p}$ Kandidat für Einwegfunktion.

$g^a \pmod{p} \rightarrow a$ (diskreter Logarithmus) ist nach heutigem Stand schwer!

7.3.4 Diffie-Hellman-Verfahren zur Schlüsselvereinbarung

A, B wollen gemeinsamen Schlüssel K für ein symm. Verfahren vereinbaren; es steht nur unsichere Kommunikationskanal zur Verfügung.

Lösung: p, g (Bitlänge von $p >$ Bitlänge von K) (können öffentlich bekannt sein).

1. A wählt zufällig $a \in \{2, \dots, p-2\}$
 A berechnet $x = g^a \pmod{p}$
 $(a$ geheim halten)
2. B wählt zufällig $b \in \{2, \dots, p-2\}$
 B berechnet $y = g^b \pmod{p}$
 $(b$ geheim halten)

$$\begin{aligned}
3. \quad & A \xrightarrow{x=g^a} B \\
& B \xrightarrow{y=g^b} A \\
& A : y^a \bmod p = g^{b \cdot a} \bmod p = K \\
& B : x^b \bmod p = g^{a \cdot b} \bmod p = K
\end{aligned}$$

7.3.5 Sicherheit

Angreifer: $p, g, g^a \bmod p, g^b \bmod p$

gesucht: $g^{a \cdot b} \bmod p$

Einzig bekannte Möglichkeit ist das Berechnen a aus $g^a : (g^b)^a \bmod p = K$ müsste diskretes Logarithmus-Problem lösen.

7.3.6 Man-in-the-Middle

M fängt g^a und g^b ab und wählt $c \in \{2, \dots, p\}$ und schickt $g^c \bmod p$ an A und B .

$A : g^{c \cdot a} \bmod p, B : g^{c \cdot b} \bmod p$. Beide Schlüssel kennt auch M

7.4 ElGamal-Public Key Verfahren (1984)

7.4.1 Schlüsselerzeugung

A wählt p, g wie bei Diffie-Hellman. Wählt $a \in \{2, \dots, p-2\}, x = g^a \bmod p$.

Öffentlicher Schlüssel: (p, g, x)

Geheimer Schlüssel: a

7.4.2 Verschlüsselung

Klartext $m : 1 \leq m \leq p-1$

$B \xrightarrow{m} A$

B wählt zufällig $b \in \{2, \dots, p-2\}$

$y = g^b \bmod p$

Er berechnet $x^b \bmod p$ und $f = m \cdot x^b \bmod p$, sendet (y, f) an A ,

7.4.3 Entschlüsselung

$y^a \bmod p (= x^b \bmod p)$

Berechnet $(y^a)^{-1} \bmod p [(y^a)^{-1} = \overbrace{(y^{p-1-a})}^{\geq 0}]$

$f \cdot (y^a)^{-1} \bmod p = m$

Nachteil zu RSA

Doppelte Länge wird gebraucht, da Nachricht (Chiffre) und Teilschlüssel versendet werden.

Kapitel 8

Signaturen, Hashfunktionen, Authentifizierung

8.1 Anforderung an digitale Signaturen

Identitätseigenschaft: ID des Unterzeichners des Dokuments wird sichergestellt

Echtheitseigenschaft: des signiertem Dokument

Verifikationseigenschaft: Jeder Empfänger muss digitale Signatur verifizieren können.

8.2 RSA-Signatur (vereinfachte Version)

A will Dokument m signieren.

A besitzt öffentlichen RSA-Schlüssel (n, e) , geheimen Schlüssel d .

Signatur: $m^d \bmod n$ sendet $(m, m^d \bmod n)$ an B.

$$(m^d \bmod n)^e = m^{e \cdot d} \bmod n = m \bmod n$$

$$m < n$$

Wenn $m^{e \cdot d} \bmod n = m$, dann akzeptiert B die Signatur.

$m > n$ $m^d \bmod n \equiv m \bmod n$. Ist $m' \bmod n = m \bmod n$, dann $(m', m^d \bmod n)$ gültige Signatur.

8.2.1 Wie lassen sich lange RSA-Signaturen vermeiden?

Def: Sei R ein endliches Alphabet.

Hashfunktion $H : \mathbb{R}^* \rightarrow R^k (k \in \mathbb{N} \text{ fest})$ soll effizient berechenbar sein.

8.3 RSA-Signatur mit HASH-Funktion

H öffentlich bekannte Hashfunktion.

A will Nachricht m signieren.

Bildet $H(m)$ und signiert $H(m) : H(m)^d \bmod n$ sendet $(m, H(m)^d \bmod n)$

Verifikation durch B : $m \rightarrow H(m)$

$(H(m)^d \bmod n)^e \bmod n = H(m)$

8.3.1 Angriffsmöglichkeiten

- Angreifer kann $H(m)$ bestimmen wenn es ihm gelingt, $m' \neq m$ zu finden, so $(m', H(m)^d \bmod n)$ gültige Signatur von m durch A .
- Angreife wählt zufällig y und berechnet $y^e \bmod n = z$
Gelingt es ihm, m zu finden mit $H(m) = z$, dann ist (m, y) gültige Signatur von m durch A
 $H(m) \cdot y^e = H(m)$

Def: Eine **kryptographische Hashfunktion** ist eine Hashfunktion, die folgende Bedingungen erfüllt.

1. H ist Einwegfunktion (um Angriffe des zweiten Typs zu vermeiden)
2. H ist **schwach kollisionsresistent**, d.h. zu gegebenem $m \in R^*$, soll es effizient nicht möglich sein ein $m' \neq m$, mit $H(m) = H(m')$, zu finden. (um Angriffe des ersten Typs zu vermeiden)

Verschärfung von 2.

- 2' H ist **stark kollisionsresistent**, wenn es effizient nicht möglich ist $m \neq m'$ zu finden, mit $H(m) = H(m')$.

Da R^* unendlich und $|R^k| = |R|^k$ endlich ist, existiert unendlich viele Paare (m, m') , $m \neq m'$ mit $H(m) = H(m')$.

(Bilde $|R|^k + 1$ viele Hashwerte: Kollision)

Kollisionen lassen sich nicht vermeiden, sie sollten aber nicht schnell herstellbar sein.

8.3.2 Satz: Geburtstagsparadoxon

Ein Merkmal komme in m verschiedenen Ausprägungen vor. Jede Person besitze genau eine dieser Merkmalsausprägungen. Ist $c \geq \frac{1 + \sqrt{1 + 8 \cdot m \cdot \ln 2}}{2} \approx 1.18 \sqrt{m}$, so ist die Wahrscheinlichkeit, dass unter l Personen zwei die gleiche Merkmalsausprägung haben, mindestens $\frac{1}{2}$ (Geburtstage: $m = 366, l = 23$).

Beweis l Personen

Alle Möglichkeiten $(g_1, g_2, \dots, g_l), g_i \in \{1, \dots, m\}$ m^l Möglichkeiten.

Alle Merkmalausprägungen verschieden: $m \cdot (m-1) \cdot (m-2) \cdot \dots \cdot (m-(l-1))$

Wahrscheinlichkeit, dass l Personen lauter verschiedene Geburtstage haben.

$$q = \frac{m \cdot (m-1) \cdot (m-2) \cdot \dots \cdot (m-(l-1))}{m^l} = \prod_{i=0}^{l-1} 1 - \frac{i}{m}$$

Wann ist $q \leq \frac{1}{2}$?

$$e^x \geq 1 + x$$

$$\prod_{i=0}^{l-1} 1 - \frac{i}{m} \leq \prod_{i=0}^{l-1} e^{-\frac{i}{m}} = e^{\sum_{i=0}^{l-1} -\frac{i}{m}} = e^{-\frac{1}{m} \sum_{i=0}^{l-1} i} = e^{-\frac{1}{m} \cdot \frac{l(l-1)}{2}}$$

$$\ln a \leq -\frac{1}{m} \cdot \frac{l \cdot (l-1)}{2} = -\frac{l^2 - l}{2 \cdot m}$$

8.3.3 Hashfunktion

$$H(m) = H(m'), m \neq m'$$

$$H : \mathbb{Z}_2^* \rightarrow \mathbb{Z}_2^n \text{ (} 2^n \text{ Hashwerte)}$$

Bei Erzeugung von circa $2^{\frac{n}{2}}$ Hashwerten ist die Wahrscheinlichkeit, dass zwei gleich sind ungefähr $\frac{1}{2}$.

$n = 64 : 2^{32}$ Hashwerte ($4 \cdot 10^9$) unsicher.

Weit verbreitet waren und sind:

MD5 (message digerst / Ron Rivest, 1991, 128 Bit)

SHA-1 (Secure Hash Algorithm, NSA, 1992/1993, 160 Bit)

8.4 Authentifizierung

Nachweise bzw. Überprüfung, dass jemand derjenige ist für den er sich ausgibt.

Möglichkeiten der Authentifizierung durch:

Wissen

Besitz

biometrische Merkmale

gängigste Methode: Passwort

Im Allgemeinen: Passwort w abgespeichert als $f(w)$ f Einwegfunktion.

$w \xrightarrow{\text{sicher}} f^n(w) \rightarrow \text{Id. überprüfer } f \text{ Einweg.}$

1. Auth. $w_1 = f^{n-1}(w) \rightarrow f(f^{n-1}(w)) = w_0$ ersetzt w_0 durch w_1

2. Auth. $w_2 = f^{n-2}(w) \rightarrow \dots$

Passwortsicherheit: <http://www.schneier.com/crypto-gram-0701.html>

8.5 Challenge-Response-Authentifizierung

RSA-Verfahren $A \xrightarrow{\text{auth.}} B$

Öffentlicher Schlüssel: (n, e)

geheimer Schlüssel: d

$A \xleftarrow{\text{Zufallszahl } r} B, r < n \leftarrow \textbf{Challenge}$

$A \xrightarrow{r^d \bmod n} B$ überprüft, ob $r^{de} \bmod n = r \leftarrow \textbf{Response}$

Damit B sich sicher sein kann, dass es wirklich A ist, kann B so oft wie es für nötig hält neue r schicken und dadurch die Chance verringern, dass A nicht A ist.

Kapitel 9

Secret Sharing Scheme

Geheimnis wird auf mehrere Teilnehmer verteilt (Teilgeheimnisse), so dass gewisse Teilmengen der Teilnehmer das Geheimnis mit ihren Teilgeheimnissen rekonstruieren können, die anderen nicht.

$T = \{t_1, \dots, t_n\}$, $k < n$ (T Menge der Teilnehmer)

Jede Teilmenge von T mit mindestens k Teilnehmer sollen Geheimnis rekonstruieren können, Teilmengen von T mit weniger als k Teilnehmer nicht.

9.1 (k, n) - Schwellenwertsysteme

1979 Shamir (How to share a secret)

9.1.1 Konstruktion

Vereinbarung von großer Primzahl p , mindestens $p \geq n + 1$

$$g \in \mathbb{Z}_p = \{0, \dots, p - 1\}$$

9.1.2 Verteilung der Teilgeheimnisse

Dealer wählt zufällig $a_1, \dots, a_{k-1} \in \mathbb{Z}_p$, $a_{k-1} \neq 0$, k = Schwelle

$$f(x) = g + a_1x + \dots + a_{k-1}x^{k-1} \in \mathbb{Z}_p[x]$$

(a_1, \dots, a_{k-1}) hält er geheim, natürlich auch g)

Dealer wählt zufällig $x_1, \dots, x_n \in \mathbb{Z}_p$ (paarweise verschieden).

Teilnehmer t_i erhält als Teilgeheimnis $(x_i, f(x_i))$ (Punkt auf Polynom)

Bei $x = 0$ hast du g .

9.1.3 Rekonstruktion(sversuch) des Geheimnisses

k Teilnehmer $(x_{i_1}, f(x_{i_1})), \dots, (x_{i_k}, f(x_{i_k}))$

Durch diese Punkte ist f eindeutig bestimmt, z.B. durch Lagrange-Interpol.:

$$f(x_{i_j}) = g_{i_j}$$

$$f(x) = \sum_{j=1}^k g_{i_j} \cdot \frac{(x - x_{i_1}), \dots, (x - x_{i_{j-1}})(x - x_{i_{j+1}}), \dots, (x - x_{i_k})}{(x_{i_j} - x_{i_1}), \dots, (x_{i_j} - x_{i_{j-1}})(x_{i_j} - x_{i_{j+1}}), \dots, (x_{i_j} - x_{i_k})}$$

$$f(0) = g$$

$$g = \sum_{j=1}^k g_{i_j} \prod_{l=j}^k \frac{x_{i_l}}{(x_{i_l} - x_{i_j})}$$

Bei mehr als k Teilnehmer selbe Ergebnis.

Weniger als k Teilnehmer (k'): Anderes Polynom wegen weniger Punkte, also wahrscheinlich anderer g .

Erzeugen Polynom vom Grad $\leq k' - 1$

Für alle $k \in \mathbb{Z}_p$ existiert gleich viele Polynome vom Grad $\leq k' - 1$ durch die vorgegebene k' Punkte, die bei h durch y-Achse gehen.

Kapitel 10

Codierungstheorie

10.1 Grundbegriffe und einfache Beispiele

10.1.1 Codierung

(Kanalcodierung)

Sicherung von Daten/Nachrichten gegen zufällig auftretenden Fehler bei Speicherung/Übertragung.

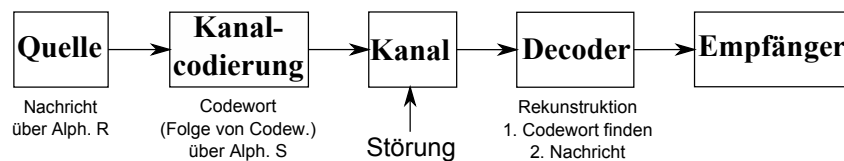


Abbildung 10.1: Schaubild der Codierung

10.1.2 Ziele

- Möglichst viele Fehler erkennen und gegebenenfalls korrigieren.
- Aufwand für Codierung und Decodierung möglichst gering.

10.2 Grundprinzip

Hinzufügen von Redundanz

Es gibt zwei Typen um Redundanz zu erzeugen.

10.2.1 FEC-Verfahren (Forward Error Correction)

Aufgetretene Fehler sollen erkannt und korrigiert werden.

Vorteil: keine Verzögerung der Übertragung aber ggf. große Redundanz notwendig.

10.2.2 ARQ-Verfahren (Automatic Repeat Request)

Aufgetretene Fehler sollen erkannt werden, werden nicht korrigiert. Stattdessen wiederholt die Übertragung beim Sender anfordern.

Vorteil: geringe Redundanz, aber Verzögerung.

Beispiele

1. Parity-Check-Codes

z.B. Nachrichten: 00, 01, 10, 11

Codierung: 00 → 000

01 → 011

10 → 101

11 → 110

(gerade Anzahl von Einsen in den Codewörtern)

1 Fehler wird erkannt, nicht korrigiert.

2 Fehler werden nicht erkannt.

2. Wiederholungscode

Nachrichten wie in 1.

Codierung: 00 → 000000 01 → 010101

10 → 101010

11 → 111111

(3-Fache Wiederholung)

1 Fehler wird erkannt und korrigiert.

010101 → 010101 → 01

3. Nachrichten wie in 1. Codierung: 00 → 00000

01 → 01101

10 → 10110

11 → 11011

Je zwei Codewörter unterscheiden sich an mindestens 3 Positionen.

Angenommen 1 Fehler tritt bei Übertragung auf. Dann gibt es genau ein Codewort, dass sich vom empfangenen Wort an genau einer Stelle unterscheidet; in das wird decodiert.

Muss immer Ungerade unterschiede in Codewörtern sein. Bei 5 diffs sind 2 Fehler korrigierbar.

4. (ehmaliger) ISBN-Code International Standard Book Number

10-Stelliger Code

Erste 9 Ziffern haben inhaltliche Bedingung ($\hat{=}$ Nachricht)

10. Ziffer: Prüfziffer

Beispiel: 3-540-26121-? (Land - Verlag - Buchnummer - Prüfziffer)

Uncodierte Wörter sind gebildet über $R = \{0, \dots, 9\}$

Codierte Wörter sind gebildet über $S = \{0, \dots, 9, X\}$

ISBN-Wort $C_{10}C_9 \dots C_2C_1$

$C_{10} \dots C_2$ inhaltliche Bedingung, C_1 wird so gewählt, dass

$$\sum_{k=1}^{10} k \cdot C_k \equiv 0 \pmod{11}$$

$10 \cdot C_{10} + \dots + 2 \cdot C_2 + C_1 \equiv 0 \pmod{11}$ falls $C_1 = 10$ so setze $C_1 = X$
 C_1 vom Beispiel ausrechnen.

$$10 \cdot 3 + 9 \cdot 5 + 8 \cdot 4 + 7 \cdot 0 + 6 \cdot 2 + 5 \cdot 6 + 4 \cdot 1 + 3 \cdot 2 + 2 \cdot 1 + C_1 = 0 \pmod{11}$$

$$161 + C_1 = 0 \pmod{11} \Rightarrow C_1 = 4$$

Ändern einer Ziffer wird erkannt:

$C_{10}C_9 \dots C_2C_1 \rightarrow C_i$ wird $X_i \neq C_i$ ersetzt

$C_{10} \dots C_{i+1}X_iC_{i-1} \dots C_1$

$$\sum_{k=1, k \neq i}^{10} k \cdot C_k + i \cdot x_i = \underbrace{\sum_{k=1, k \neq i}^{10} k \cdot C_k}_{\equiv 0 \pmod{11}} \cdot \underbrace{i}_{\neq 0 \pmod{11}} \cdot \underbrace{(x_i - C_i)}_{\neq 0 \pmod{11}} \not\equiv 0 \pmod{11}$$

Fehler wird erkannt, Korrektur nicht möglich.

$$3 - 540 - 26121 - 4 \equiv 0 \pmod{11}$$

$$\left. \begin{array}{l} 3 - 540 - 26121 - \mathbf{6} \\ 3 - 540 - 26122 - 4 \end{array} \right\} \text{Prüfsumme 2.}$$

Vertauschung von Zwei Ziffern wird erkannt.

C_i und C_j vertauscht.

O.B.d.A $C_i \neq C_j$

$C_{10} \dots C_j \dots C_i \dots C_1$

$$\begin{aligned} & \begin{array}{c} \uparrow \quad \uparrow \\ i \quad j \end{array} \\ & \sum_{k=1, k \neq i, j}^{10} k \cdot C_k + i \cdot C_j + j \cdot C_i = \sum_{k=1}^{10} k \cdot C_k + i(C_j - C_i) + j(C_i - C_j) \\ & = \underbrace{\sum_{k=1}^{10} k \cdot C_k}_{\equiv 0 \pmod{11}} + \underbrace{(C_j - C_i)}_{\not\equiv 0 \pmod{11}} \underbrace{(i - j)}_{\not\equiv 0 \pmod{11}} \not\equiv 0 \pmod{11} \end{aligned}$$

Vertauschung wird durch gewichtete Quersummen erkannt.

5. EAN-13-Code

European Article Number

13-Stelliger Code, erste 12 Ziffer sind inhaltlich festgelegt.

13. Ziffer ist Prüfziffer.

$R = S = \{0, \dots, 9\}$

$C_1 \dots C_{12} C_{13}$

$C_1 \dots C_{12}$ inhaltliche Angabe (in der Regel):

$C_1 C_2$ Herstellerland (40-43 Deutschland)

$C_6 \dots C_7$ Hersteller $C_8 \dots C_{12}$ interne Produktions Nummer

C_{13} so gewählt, dass

$$C_1 + 3 \cdot C_2 + C_3 + 3 \cdot C_4 + \dots + 3 \cdot C_{12} + C_{13} \equiv 0 \pmod{10}$$

$x \rightarrow 3x$ Permutation auf $\mathbb{Z}_{10} \pmod{10}$, da $\text{ggT}(3, 10) = 1$

1 Fehler wird erkannt. Vertauschung in der Regel nicht erkannt.