

Codierung und Verschlüsselung

Steffen Lindner

January 7, 2016

Inhaltsverzeichnis

1 Kryptologie	4
1.1 Grundbegriffe und einfache Verfahren	4
1.1.1 Definition	4
1.1.2 Anzahl Schlüssel symmetrisch vs. asymmetrisch	5
1.2 Beispiel	5
1.3 Übersprungen	6
1.4 Übersprungen	6
1.5 Prinzip von Kerkhoffs (1883)	6
1.6 Kryptoanalyse	6
1.6.1 Arten von Angriffen	6
2 One-Time-Pad und perfekte Sicherheit	7
2.1 Lauftextverschlüsselung	7
2.2 One-Time-Pad	7
2.3 Was ist Zufallsfolge der Länge n?	7
2.4 One-Time-Pad ist perfekt sicher	8
3 Symmetrische Blockchiffren	9
3.1 Blockchiffre	9
3.2 Lineare Chiffren	9
3.2.1 Beispiel	11
3.3 Known-Plaintext-Angriff auf lineare Chiffren	12
3.4 Bemerkung	13
3.5 Diffusion und Konfusion	13
3.5.1 Diffusion	13
3.5.2 Konfusion	13
3.6 Feistel-Chiffren	13
4 Der Advanced Encryption Standard (AES)	15
4.1 Struktur des AES	15
4.2 SubBytes-Transformation	16
4.3 Shift Rows und MixColumns-Transformation	17
4.4 Rundenschlüsselerzeugung	17
4.5 Entschlüsselung	18
4.6 Schnelligkeit und Sicherheit	18

5	Public-Key-Systeme	19
5.1	Grundidee (Diffie, Hellman, 1976)	19
5.2	Das RSA-Verfahren	19
5.3	Berechnung modularer Potenzen	21
5.4	Sicherheit des RSA-Verfahrens	21
5.5	Bestimmung großer Primzahlenn	22
5.6	Das Diffie-Hellman-Verfahren zur Schlüsselvereinbarung (1976)	23
5.7	Der Man-in-the-Middle-Angriff auf D-H-Verfahren	24
5.8	ElGamal-Verschlüsselungsverfahren (T. ElGamal, 1984)	25
5.8.1	Schlüsselerzeugung:	25
5.8.2	Verschlüsselung	25
5.8.3	Entschlüsselung	25
5.9	Bug-Attacks	25
6	Signaturen, Hashfunktionen, Authentifizierung	27
6.1	Digitale Signaturen	27
6.2	RSA-Signatur	27
6.3	Definition Hashfunktion	28
6.4	RSA-Signatur mit Hashfunktion	28
6.5	Anforderung an H	28
6.6	Definition: Kryptographische Hashfunktion	28
6.7	Satz (Geburtstagsparadox)	29
6.7.1	Beweis	29
6.8	Geburtstagsattacke (gegen starke Koll.res.)	29
6.9	Bemerkung	30
6.10	Authentifizierung	30
6.11	Passwörter	30
6.12	Challenge-Response-Authentifizierungen	31
7	Secret Sharing Schemes	32
7.1	Definition	32
7.2	Definition	32
7.3	Shamirs Konstruktion eines Schwellenwertsystems	33
8	Codierungstheorie	34
8.1	Grundbegriffe und einfache Beispiele	34
8.1.1	Codierung (genauer Kanalcodierung)	34
8.1.2	Beispiele	34
8.1.3	GTIN-Prüfzifferncode (GTIN-13)	35
9	Blockcodes	37
9.1	Satz (10.7)	37
9.1.1	Beweis	37
9.2	Bemerkung	37
9.3	Beispiel	37
9.4	Satz und Definition	38
9.4.1	Beweis	38
9.5	Bemerkung	39
9.6	Beispiel	39

Kryptologie

1.1 Grundbegriffe und einfache Verfahren

Klartext: Unverschlüsselter Text (plain text)

Chiffre: Verschlüsselter Text (cipher text)

Sender Alice: Schlüssel \rightarrow Verschlüsselung

Empfänger Bob: Schlüssel \rightarrow Entschlüsselung

Encryption über Alphabet R, decryption über Alphabet S.

Verschlüsselung erfordert:

- Verschlüsselungsverfahren (Funktion E)
- Schlüssel k_e (encryption key) aus Menge K von möglichen Schlüsseln

$E(m, k_e) = c$, mit m Klartext, k_e Schlüssel und c Chiffre-text.

Entschlüsselung erfordert

- Entschlüsselungsfunktion (Funktion D)
- Schlüssel k_d (decryption key, hängt ab von k_e)

$D(c, k_d) = m$

Für festes k_e soll die Funktion $E(., k_e)$ injektiv sein, d.h.

$$m_1 \neq m_2 \rightarrow E(m_1, k_e) \neq E(m_2, k_e) \quad (1.1)$$

$$D(., k_d) = E(., k_e)^{-1}$$

1.1.1 Definition

Ist $k_e = k_d$ (oder falls k_d einfach aus k_e zu berechnen ist), so spricht man von symmetrischen Verschlüsselungsverfahren.

Ist k_d nicht oder nur mit großem Aufwand aus k_e berechenbar, so spricht man von asymmetrischen Verfahren.

Im zweiten Fall kann man k_e auch veröffentlichen: Public-Key-Verfahren.

Sicherheit eines Verschlüsselungsverfahrens darf nur von der Geheimhaltung von k_d abhängen. Bei symmetrischen Verfahren muss $k_e = k_d$ auf sicherem Wege ausgetauscht werden.

Asymmetrische Verfahren:

Bob erzeugt k_e, k_d .

1.1.2 Anzahl Schlüssel symmetrisch vs. asymmetrisch

Für symmetrische Verfahren gilt:

$$\binom{n}{2} = \frac{n \cdot (n-1)}{2} = O(n^2) \quad (1.2)$$

Für asymmetrische Verfahren gilt:

$$2n = O(n) \quad (1.3)$$

1.2 Beispiel

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

Verfahren: Verschiebeciffre

Menge der Schlüssel: $K = \{0, 1, \dots, 25\}$

Elemente aus R werden einzeln verschlüsselt:

Wähle Schlüssel $i \in K$

Verschlüsseln: $x \in R \mapsto x + i \bmod 26$

Entschlüsseln: $y \mapsto y - i \bmod 26$

$m = x_1 \dots x_r, x_j \in R$

$E(m, i) = ((x_1 + i) \bmod 26) ((x_2 + i) \bmod 26) \dots ((x_r + i) \bmod 26) = c$

$D(c, i) = (y_1 - i) \bmod 26 \dots ((y_r - i) \bmod 26) = m$

Verfahren unsicher, da K klein.

(b) Verallgemeinerung: zeichenweise Substitutionschiffre

$R = S = \{0, \dots, 25\}$

K = Menge aller Permutationen von R

Wähle Schlüssel $\pi \in K, m = x_1 \dots x_r, x_j \in R$

Verschlüsseln:

$E(m, \pi) = \pi(x_1) \dots \pi(x_r) = c$

Entschlüsseln:

$D(c, \pi) = \pi^{-1}(y_1) \dots \pi^{-1}(y_r) = m$

$|K| = 26! \approx 4 \cdot 10^{26}$

Angenommen 10^{12} Schlüssel pro Sekunde testbar.

Angenommen 50% Schlüssel werden getestet.

Man benötigt dazu: $2 \cdot 10^{14}$ Sekunden $\approx 6.000.000$ Jahre.

1.3 Übersprungen

1.4 Übersprungen

1.5 Prinzip von Kerkhoffs (1883)

Sicherheit eines Verschlüsselungsverfahrens darf nur von der Geheimhaltung des Schlüssels k_d abhängen, nicht von der Geheimhaltung des Verschlüsselungsalgorithmus (\Rightarrow sollte offen gelegt werden: Praxistest).

1.6 Kryptoanalyse

Qualitative Unterschiede:

- Schlüssel k_d lässt sich ermitteln
- Ermittlung einer zu $D(., k_d)$ äquivalenten Funktion (evtl. nur für gewisse k_e)
- Finden des Klartextes für einen speziellen Chiffretext

1.6.1 Arten von Angriffen

Ciphertext-only-Angriff

Lediglich der Chiffretext ist dem Angreifer bekannt.

Known-Plaintext-Angriff

Der Angreifer kennt bereits Plaintext - Chiffretext Kombinationen.

Chosen-Plaintext-Angriff

Der Angreifer kann sich mit einem selbstgewählten Plaintext Chiffretexte erzeugen.

Chosen-Ciphertext-Angriff

Der Angreifer kann sich gewünschte Chiffretexte entschlüsseln lassen.

One-Time-Pad und perfekte Sicherheit

2.1 Lauftextverschlüsselung

Früher im militärischen Bereich Lauftextverschlüsselungen.

Klartext: Endliche Folge z.B. über $\{0, \dots, 25\}$.

Addiere (mod 26) anderen Text (z.B. ab gewisser Stelle im Buch).

Problem: Häufige Buchstaben treffen bei Addition häufig auf Buchstaben \Rightarrow kryptoanalytische Möglichkeiten.

2.2 One-Time-Pad

Klartextalphabet $R = \mathbb{Z}_2 = \{0, 1\}$.

Verschlüsselung: Addiere zu Klartext der Länge n eine binäre Zufallsfolge der Länge n (Addition mod 2 = XOR).

Ergebnis: Chiffretext

$$\underbrace{\text{Klartext}}_m \oplus \underbrace{\text{Zufallsfolge}}_k = c$$

Entschlüsseln: $c \oplus k = m \oplus k \oplus k = m$

One-Time-Pad

k darf nur einmal verwendet werden.

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

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

2.3 Was ist Zufallsfolge der Länge n ?

Frage ist sinnlos! Es kommt auf die Erzeugung an. Folge von Nullen und Einsen, die so erzeugt werden, dass jedes Bit unabhängig von den vorhergehenden mit Wahrscheinlichkeit $\frac{1}{2}$ erzeugt wird.

Jede Folge der Länge n hat die Wahrscheinlichkeit $\frac{1}{2^n}$.

2.4 One-Time-Pad ist perfekt sicher

Gegeben: Chiffretext c .

A-priori W-keit

Dann gilt: $pr(m|c) = \overbrace{pr(m)}$, für alle Klartexte m .

$pr(m|c)$ = Wahrscheinlichkeit, dass m Klartext war, wenn ich c kenne.

Substitutionschiffre ist nicht perfekt sicher:

Chiffretext: OCKTT

$\Rightarrow m = \text{HAUCK}$ kann nicht Klartext gewesen sein. $pr(m|c) = 0$

Symmetrische Blockchiffren

3.1 Blockchiffre

Klartext wird in Blöcke von fester Länge n zerlegt. Jeder Block wird einzeln verschlüsselt. Bei festem Schlüssel wird ein und derselbe Block immer in der gleichen Weise verschlüsselt (einfachstes Szenario).

Häufig: Klartextblocklänge = Chiffretextblocklänge

$R = S = \{1, 1\}$. Es gibt 2^n Blöcke der Länge n .

Jede Blockchiffre ist Permutation dieser 2^n Blöcke. Insgesamt $(2^n)!$ viele Blockchiffren der Länge n .

Schlüssel: Permutation

Speicherung: 2^n Bider von $(0, \dots, 0), \dots, (1, \dots, 1)$. Insgesamt $n \cdot 2^n$ Bit.

Beispiel

$n = 64$. Ein Schlüssel erfordert 2^{70} Bit ≈ 13.5 Millionen Festplatten a 10 TB.

Lösung

Es wird nur eine ausgewählte Menge von Permutationen der Blöcke als Schlüsselmenge verwendet.

3.2 Lineare Chiffren

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

Klartextblöcke: Elemente in \mathbb{Z}_k^n

Lineare Chiffre über \mathbb{Z}_k (Blocklänge n):

$m = (V_1, \dots, V_n)$, $V_i \in \mathbb{Z}_k$, Klartextblock.

Verschlüsseln

$m \xrightarrow[\text{Verschl.}]{} m \cdot A \in \mathbb{Z}_k^n$, A $n \times n$ - Matrix über \mathbb{Z}_k .

$$(V_1, \dots, V_n) \cdot \begin{pmatrix} a_{11} & \cdot & \cdot & \cdot & a_{1n} \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ a_{n1} & \cdot & \cdot & \cdot & a_{nn} \end{pmatrix} = (a_{11}V_1 + \dots + a_{n1}V_n, \dots, a_{1n}V_1 + \dots + a_{nn}V_n)$$

Abb $m \rightarrow m \cdot A$ ist invertierbar (d.h. Permutation), wenn A invertierbar ist, das heißt wenn Matrix A^{-1} existiert mit $A \cdot A^{-1} = A^{-1} \cdot A = E_n$.
 $c = m \cdot A$.

Entschlüsselung:

$$c \cdot A^{-1} = m \cdot (A \cdot A^{-1}) = m \cdot E_n = m.$$

Wann ist A invertierbar und wie berechnet man A^{-1} ?

→ Determinante von A $\det(A) \in \mathbb{Z}_k$.

$$\det(a) = a$$

$$\det \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = a_{11} \cdot a_{22} - a_{12} \cdot a_{21}.$$

Inverse zu A (wenn sie existiert):

$$A^{-1} = \underbrace{(\det(A))^{-1}}_{\text{Inverses in } \mathbb{Z}_k} \cdot B, B = (b_{ij}), b_{ij} = (-1)^{i+j} \det(A_{ji}) \quad (3.1)$$

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

$\det(A)$ ist invertierbar in $\mathbb{Z}_k \Leftrightarrow \text{ggT}(\det(A), k) = 1$

$m \rightarrow m \cdot A$ ist invertierbar $\Leftrightarrow \text{ggT}(\det(A), k) = 1$.

Spezialfall n = 2

$$\text{ggT}(a_{11} \cdot a_{22} - a_{12} \cdot a_{21}, k) = 1$$

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

3.2.1 Beispiel

$\mathbb{Z}_6, k = 6, n = 2$.

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 3 \end{pmatrix} \det(A) = -1 \bmod 6 = 5$$

$$\text{ggT}(5, 6) = 1$$

$$A^{-1} = 5^{-1} \begin{pmatrix} 3 & 5 \\ 5 & 0 \end{pmatrix}, 5^{-1} = 5 \text{ in } \mathbb{Z}_6.$$

$$= 5 \cdot \begin{pmatrix} 3 & 5 \\ 5 & 0 \end{pmatrix}.$$

Verschlüsseln von (1,3) mit A

$$(1,3) \cdot \begin{pmatrix} 0 & 1 \\ 1 & 3 \end{pmatrix} \underbrace{=}_{\text{mod } 6} (3,4).$$

Nachricht (1 3 2 5)

$$(2,5) \begin{pmatrix} 0 & 1 \\ 1 & 3 \end{pmatrix} = (5,5).$$

Chffretext: (3 4 5 5)

Entschlüsselung:

$$(3,4) \cdot A^{-1} = (3,4) \cdot \begin{pmatrix} 3 & 1 \\ 1 & 0 \end{pmatrix} = (1 \ 3)$$

$$(5,5) \cdot A^{-1} = (2,5).$$

Anzahl der Schlüssel = Anzahl der invertierbaren $n \times n$ - Matrizen über \mathbb{Z}_k .

Für $k = 2$: $(2^n - 1)(2^n - 2)(2^n - 2^2) \dots (2^n - 2^{n-1})$

$n = 64$: $\approx 0,29 \cdot 2^{4096}$

3.3 Known-Plaintext-Angriff auf lineare Chiffren

$m \in \mathbb{Z}_k^n$, $m \rightarrow m \cdot A$, A invertierbare $n \times n$ - Matrix über \mathbb{Z}_k .

Angreifer kennt Klartextblöcke m_i und zugehörige Chiffretextblöcke c_i . Er will A ermitteln.

Er benötigt n Klartextblöcke m_1, \dots, m_n mit zugehörigen Chiffretextblöcken c_1, \dots, c_n .

Daraus wird er häufig A ermitteln können:

$$m_i \cdot A = c_i, \quad i = 1, \dots, n.$$

$$M = \begin{pmatrix} m_1 \\ m_2 \\ \vdots \\ m_n \end{pmatrix}, \quad C = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} \quad n \times n \text{ - Matrizen über } \mathbb{Z}_k.$$

$$M \cdot A = C$$

Falls M invertierbar ist, so berechne M^{-1} . Dann: $M^{-1} \cdot C = M^{-1}(MA) = (M^{-1}M)A = E_n A = A$.

Beispiel

$n = 2$, $k = 26$

Angenommen wir wissen: KRYPTO \rightarrow QLIPRL mit linearer Chiffre.

$$\underbrace{\overset{K}{10}}_{10} \underbrace{\overset{R}{17}}_{17} \underbrace{\overset{Y}{24}}_{24} \underbrace{\overset{P}{15}}_{15} \underbrace{\overset{T}{19}}_{19} \underbrace{\overset{O}{14}}_{14} \rightarrow \underbrace{\overset{Q}{16}}_{16} \underbrace{\overset{L}{11}}_{11} \underbrace{\overset{I}{8}}_{8} \underbrace{\overset{P}{15}}_{15} \underbrace{\overset{R}{17}}_{17} \underbrace{\overset{L}{11}}_{11}$$

($n = 2$)

$$M = \begin{pmatrix} 10 & 17 \\ 24 & 15 \end{pmatrix}, \quad \text{ggT}(\det(M), 26) = 1 \text{ ? Nein, da } \det(M) \text{ gerade.}$$

Stattdessen:

$$M = \begin{pmatrix} 10 & 17 \\ 19 & 14 \end{pmatrix}, \quad \det(M) = (10 \cdot 14 - 17 \cdot 19) \bmod 26 = 25.$$

$$(\det(M))^{-1} = -1 \bmod 26 = 25$$

$$M^{-1} = (-1) \begin{pmatrix} 14 & -17 \\ -19 & 10 \end{pmatrix} \bmod 26$$

$$= \begin{pmatrix} 12 & 17 \\ 19 & 16 \end{pmatrix}.$$

Zugehöriges C: $C = \begin{pmatrix} 16 & 11 \\ 17 & 11 \end{pmatrix}$.

$$A = M^{-1} \cdot C = \begin{pmatrix} 12 & 17 \\ 19 & 16 \end{pmatrix} \cdot \begin{pmatrix} 16 & 11 \\ 17 & 11 \end{pmatrix} \underbrace{=}_{\text{mod } 26} \begin{pmatrix} 13 & 7 \\ 4 & 21 \end{pmatrix}$$

$$(10 \ 17) \ A = (16 \ 11)$$

$$(24 \ 15) \ A = (8 \ 15)$$

$$(19 \ 14) \ A = (17 \ 11)$$

→ Test

3.4 Bemerkung

Häufig kann Sicherheit von Blockchiffren erhöht werden durch Hintereinanderausführung mehrerer Blockchiffren. Sinnlos bei linearen Blockchiffren:

2 x lineare Blockchiffre = 1x lineare Blockchiffre

3.5 Diffusion und Konfusion

Shannon, 1949 (Theory of Secrecy Systems)

Kriterien für gute Chiffrierverfahren.

3.5.1 Diffusion

Statistische Auffälligkeiten im Klartext sollen im Chiffretex nicht mehr erkennbar sein.

Insbesondere: Jedes Chiffretextzeichen muss von mehreren Klartextzeichen abhängen.

3.5.2 Konfusion

Aus statistischen Eigenschaften des Chiffretextes soll nicht auf den Schlüssel zurückgeschlossen werden.

Insbesondere: Jedes Chiffretextzeichen hängt von mehreren Schlüsselzeichen ab.

Lineare Chiffren: gute Diffusion, halbwegs gute Konfusion.

3.6 Feistel-Chiffren

Horst Feistel, IBM (1915-1990)

1971: Konstruktionsprinzip für symmetrische Blockchiffren. Wichtigste Realisierung: DES (Data Encryption Standard)

Prinzip:

1. Folge von Blocksubstitutionen und Blocktranspositionen (= Permutation der Einträge in einem Block). (Hohe Diffusion und Konfusion).
2. Erzeugung von Rundenschlüsseln aus Ausgangsschlüssel.

$m \in \mathbb{Z}_m^n$, n gerade

Klartextblock m

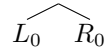


Bild kommt von Maxi

$$m = (L_0, R_0)$$

$$L_i = R_{i+1}$$

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

$$i = 1, \dots, r-1$$

$$L_r = L_{r-1} \oplus f_{k_r}(R_{r-1})$$

$$R_r = R_{r-1}$$

Konkrete Realisierung hängt von f ab + Art der Rundenschlüsselerzeugung.

Die Abbildungen $R_{i-1} \rightarrow f_{k_i}(R_{i-1})$ dürfen nicht affin-linear sein. (affin-linear: $x \mapsto xA + b$, Hintereinanderausführung affin-linearer Chiffren ist wieder affin-linear)

Denn sonst ist die gesamte Feistel-Chiffre affin-linear und angreifbar ähnlich wie lineare Chiffren.

Entschlüsselung einer Feistel-Chiffre = **Verschlüsselung** mit der umgekehrten Reihenfolge der Rundenschlüssel.

Der Advanced Encryption Standard (AES)

1970'er: Entwickelt des DES (auf der Basis einer Feistelchiffre). Blocklänge $n = 64$, effektive Schlüssellänge 56 Bit. Anfällig gegen Brute-Force-Angriffe.

1997: Ausschreibung durch NIST für neuen Verschlüsselungsstandard

2000: Rijndael-Verfahren (J. Daemen, V. Rijmen)

2002: AES

4.1 Struktur des AES

AES ist iterierte Blockchiffre.

Blocklängen: 128, 192, 256 Bit

Schlüssellängen: 128, 192, 256 Bit

Blocklängen unabhängig von Schlüssellängen

Anzahl der Runden (abhängig von gewählten Bitlängen), $r = 10, 12, 14$

Zwischenergebnisse nach jeder Runde:

Zustände (States), S_0, \dots, S_9 .

Jede Runde besteht aus 4 Transformationen:

$S_i \rightarrow \text{Sub Bytes} \rightarrow \text{Shift Rows} \rightarrow \text{Mix Columns} \rightarrow \oplus K_{i+1} \rightarrow S_{i+1}$ (Bild)

Mix Columns fällt in der letzten Runde weg.

Vorbemerkung

128 Bit-Blöcke: 4×4 - Matrizen, jeder Eintrag Byte.
$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix} \equiv$$

Block $(a_{00}a_{10}a_{20}a_{30}a_{01} \dots a_{33})$ (Spaltenweise auslesen).

Bytes werden häufig als Elemente in einem Körper der Ordnung 2^8 aufgefasst (Anzahl der Elemente).

Beispiel für endlichen Körper: $\mathbb{Z}_p = \{0, \dots, p-1\}$, p Primzahl

$$a \oplus b = a + b \mod p \quad (4.1)$$

$$a \odot b = a \cdot b \mod p \quad (4.2)$$

$\mathbb{Z}_2 = \{0, 1\}$, \oplus XOR, \odot \cdot , \wedge

Körper der Ordnung 2^8 : \mathbb{F}_{2^8}

Elemente sind alle Polynome vom Grad $< P$ über \mathbb{Z}_2 :

$$a_7x^7 + \dots + a_1x + a_0 \leftrightarrow (a_7, a_6, \dots, a_0), \quad a_i \in \mathbb{Z}_2 \quad (4.3)$$

Addition in \mathbb{F}_{2^8} : Addition von Polynomen.

Multiplikation in \mathbb{F}_{2^8} : $f, g \in \mathbb{F}_{2^8}$

$$f \odot g := f \cdot g \mod h \quad (4.4)$$

wobei h das irreduzible Polynom $h = x^8 + x^4 + x^3 + x + 1$

Beispiel:

$$(x^7 + x + 1) \odot (x^3 + x) = x^{10} + x^8 + x^4 + x^3 + x^2 + x \mod h = x^6 + x^5 + x^3 + 1$$

In \mathbb{F}_{2^8} existiert zu jedem $0 \neq g \in \mathbb{F}_{2^8}$ ein $g^{-1} \in \mathbb{F}_{2^8}$ mit $g \odot g^{-1} = 1$

Berechnung mit dem erweiterten Euklidischen Algorithmus (wie in \mathbb{Z}).

$$0 \neq f, g \in \mathbb{Z}_2[x] : u, v \in \mathbb{Z}_2[x] \text{ mit } u \cdot f + v \cdot g = \text{ggT}(f, g) \quad (4.5)$$

$0 \neq g \in \mathbb{F}_{2^8}, h$ irred. Polynom. $\text{ggT}(g, h) = 1$, da h irred.

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

$$(v \mod h) \odot g = (v \cdot g) \mod h = 1$$

(Verkürzter) EEA in $\mathbb{Z}_2[x]$:

Input: $f, g \in \mathbb{Z}_2[x], g \neq 0, \text{Grad}(f) \geq \text{Grad}(g)$

Output: $\text{ggT}(f, g), v \in \mathbb{Z}_2[x] : v \cdot g \mod f = \text{ggT}(f, g)$:

1. $s := f, t := g, v_1 := 0, v_2 := 1, v := 1$

2. Solange $s \mod t \neq 0$ wiederhole

$$q := s \text{ div } t, r := s \mod t$$

$$v := v_1 - q \cdot v_2, v_1 := v_2, v_2 := v$$

$$s := t, t := r$$

3. Output: $t = \text{ggT}(f, g), v$

Zur Bestimmung von g^{-1} für $0 \neq g \in \mathbb{F}_{2^8}$ wende EEA an mit $f = h$. $g^{-1} = v \mod h$.

4.2 SubBytes-Transformation

$$\text{Eingabe: Zustand } S_i = \begin{pmatrix} b_{00} & \dots & b_{03} \\ \vdots & & \vdots \\ b_{30} & \dots & b_{33} \end{pmatrix}, b_{ij} \text{ Bytes}$$

Jedes Byte aus S_i wird einzeln verändert. Sei $g = (b_7, b_6, \dots, b_0)$ ein Byte.

1. Schritt: $g \neq 0 : g \mapsto g^{-1}, 0 \mapsto 0$ (fasse g als Element in \mathbb{F}_{2^8})

2. Schritt: Ergebnis (c_7, c_6, \dots, c_0) nach 1. Schritt wird affin-linearer Transformation unterworfen:

$$(c_0, \dots, c_7) \cdot A + b = (d_0, \dots, d_7) \quad (4.6)$$

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

Übrige Zeilen durch zyklischen Shift der 1. Zeile um 1,2,...,7 Position nach rechts.

$$b = (11000110)$$

Ergebnis: (d_7, \dots, d_0)

SubBytes wird in AES durch Table Lookup in einer 16×16 - Matrix beschrieben: Einträge sind 0,...,255 (in gew. Reihenfolge).

Byte $b = (b_7, b_6, \dots, b_0)$ bestimmt durch $b_7 b_6 b_5 b_4$ die Zeile und $b_3 b_2 b_1 b_0$ Spalte (Zeichen und Spalten sind mit 0,...,15 nummeriert). Eintrag an der entsprechenden Stelle ist Ergebnis von SubBytes angewendet auf b .

4.3 Shift Rows und MixColumns-Transformation

- Shift Rows: Jede der 4 Zeilen nach der 4×4 - Matrix (nach SubBytes) wird zyklisch nach links verschoben: i -te Zeile um $i-1$ Stellen ($i=1,2,3,4$).
- MixColumns: Elemente der Eingangsmatrix werden als Elemente in \mathbb{F}_{2^8}

betrachtet. Diese Matrix wird von links mit $M = \begin{pmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{pmatrix}$

(über \mathbb{F}_{2^8}) über \mathbb{F}_{2^8} multipliziert.

x entspricht $(0, \dots, 0, 1, 0)$

1 entspricht $(0, \dots, 0, 1)$

$$M \begin{pmatrix} a_{00} & \cdot & \cdot \\ a_{10} & \cdot & \cdot \\ a_{20} & \cdot & \cdot \\ a_{30} & \cdot & \cdot \end{pmatrix} = \begin{pmatrix} x \cdot a_{00} + x \cdot a_{10} + a_{10} + a_{20} + a_{30} & \dots \\ a_{00} + x a_{10} + x a_{20} + a_{20} + a_{30} & \dots \\ \dots & \dots \end{pmatrix}$$

4.4 Rundenschlüsselerzeugung

Ausgangsschlüssel hat 128 Bit. Wird geschrieben als 4×4 - Matrix von Bytes, spaltenweise zu lesen.

Spalten $w(0), w(1), w(2), w(3)$.

Definiere 40 weitere Spalten (a 4 Bytes).

$w(0), \dots, w(i-1)$ seine schon definiert. ($i \geq 4$)

$i \not\equiv 0 \pmod{4}$: $w(i) := w(i-4) \oplus w(i-1)$ (Komponentenweise XOR-Verkn.)

$i \equiv 0 \pmod{4}$. $w(i) := w(i-4) \oplus T(w(i-1))$, wobei T folgende Transformation ist:

$$w(i-1) = (a \ b \ c \ d), \text{ a,b,c,d Bytes} \quad (4.7)$$

Wende auf b,c,d,a SubBytes an. Liefert e,f,g,h

Berechne:

$$r(i) = (00000010)^{\frac{i-4}{4}} \quad (4.8)$$

Potenz in \mathbb{F}_{2^8} .

$i = 4, x^0 = 1 \leftrightarrow (00000001)$

$i = 8, x^1 = x \leftrightarrow (00000010)$

$i = 12, x^2 \leftrightarrow (00000100)$

...

$i = 36, x^8$ in F_{2^8} reduziert mod $x^8+x^4+x^3+x+1 = x^4+x^3+x+1 \leftrightarrow (00011011)$

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

Rundenschlüssel K_i besteht aus Spalten $w(4i), \dots, w(4i+3)$

4.5 Entschlüsselung

Alle Transformationen in AES sind invertierbar. mit der umgekehrten Reihenfolge der Rundenschlüssel Transformationen invertieren.

4.6 Schnelligkeit und Sicherheit

- Schnelligkeit:

Software: 200 MBit/sec - 2 GBit/sec

Hardware: 2GBit/sec - 70 GBit/sec

- Sicherheit:

Nach 2 Runden vollständige Diffusion.

Rundenschlüsselerzeugung mit SubBytes verhindert, dass Regelmäßigkeiten im Ausgangsschlüssel fortpflanzen. Verschiedene Ausgangsschlüssel haben nur sehr selten einen Rundenschlüssel gemeinsam.

AES ist resistent gegen klassische kryptoanalytische Angriffe (differenzielle Kryptoanalyse, lineare Kryptoanalyse).

Public-Key-Systeme

5.1 Grundidee (Diffie, Hellman, 1976)

Jeder Teilnehmer A hat Paar von Schlüsseln:

- P_A : öffentlicher Schlüssel
- G_A : geheimer Schlüssel

Zu jedem öffentlichen Schlüssel gehört öffentlich bekannte Verschlüsselungsfunktion E_{P_A} ($= E(., P_A)$).

B \xrightarrow{m} A. $m \rightarrow E_{P_A}(m) = c$ Chiffretext.

Bedingungen:

- 1) $E_{P_A}(m)$ muss schnell berechenbar sein, aber m darf für einen Angreifer aus Kenntnis von $E_{P_A}(m)$ nicht mit vertretbarem Aufwand berechenbar sein. (D.h. Invertierung von E_{P_A} ist schwierig)
 E_{P_A} ist Einwegfunktion.
- 2) A muss m aus $c = E_{P_A}(m)$ mit Hilfe von G_A effizient berechnen können:

$$m = D_{G_A}(c) \quad (5.1)$$

Injektive Einwegfunktion, die mit Zusatzinformation leicht zu berechnen sind, heißen Geheimtürfunktion (trapdoor function).

Aus 1) und 2) folgt:

- 3) G_A darf aus P_A nicht effizient berechenbar sein.

Es ist offenes Problem, ob Einwegfunktionen existieren.

Notwendig: $P \neq NP$.

Es gibt Kandidaten für Einwegfunktionen.

5.2 Das RSA-Verfahren

Rivest, Shamir, Adleman, 1977

Beruht auf der Schwierigkeit der Faktorisierung großer Zahlen.

(a) Schlüsselerzeugung

Wähle zwei große Primzahlen $p \neq q$ (z.B. ca 1000 Bit lang) (wie? später)

Bilde $n = p \cdot q$.

$\varphi(n) = |\{a : 1 \leq a \leq n, \text{ggT}(a, n) = 1\}| = (p-1) \cdot (q-1)$ (Eulersche φ -Funktion)

[Warum? Nicht teilerfremd zu n sind alle Vielfachen von p und alle Vielfachen von q , die $\leq n$ sind.]

$1 \cdot p, 2 \cdot p, 3 \cdot p, \dots, (q-1) \cdot p$

$1 \cdot q, 2 \cdot q, 3 \cdot q, \dots, (p-1) \cdot q, q \cdot p = n$

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

Teilerfremd: $n - q - p + 1 = pq - q - p + 1 = (p-1) \cdot (q-1)$

Wähle $1 < e \leq \varphi(n)$ mit $\text{ggT}(e, \varphi(n)) = 1$ (Zufallszahl + Euklidischer Algorithmus).

Öffentlicher Schlüssel : $P_A = (n, e)$

Wähle $1 \leq d \leq \varphi(n)$ mit $ed \equiv 1 \pmod{\varphi(n)}$

[Wende erweiterten Euklidischen Algorithmus auf e und $\varphi(n)$ an. Liefert $s, t \in \mathbb{Z}$ mit $s \cdot e + t \cdot \varphi(n) = \text{ggT}(e, \varphi(n)) = 1$

$d := s \pmod{\varphi(n)}$

$d \cdot e \pmod{\varphi(n)} = (s \cdot e + \underbrace{t \cdot \varphi(n)}_{\equiv 0 \pmod{\varphi(n)}}) \pmod{\varphi(n)} = 1$

Geheimer Schlüssel: $G_A = d$

(Jetzt kann man p, q und $\varphi(n)$ löschen, und sollte es auch.)

(b) Verschlüsselung

Nachricht

$B \xrightarrow{\quad} A$, B codiert Nachricht als Zahl. zerlege die Zahl in Blöcke, jeder der Blöcke sei als Zahl $< n$.

Sei m solch ein Block, auf jedenfall als Zahl.

Verschlüsseln von m : $m^e \pmod{n} = c$

(c) Entschlüsselung

$c^d = (m^e)^d \pmod{n} = m^{ed} \pmod{n} = m$

Wieso gilt das?

Grund ist der kleine Satz von Fermat:

p Primzahl, $a \in \mathbb{Z}, p \nmid a \Rightarrow a^{p-1} \pmod{p} = 1$

$[a \pmod{p}, a^2 \pmod{p}, \dots, a^{n_1} \pmod{p} = a^{n_2} \pmod{p}, n_1 > n_2$

$a^{n_1-n_2} \pmod{p} = 1$, da $p \nmid a$

Sei $m \in \mathbb{N}$ mit $a^m \pmod{p} = 1$

$U = \{a \pmod{p}, a^2 \pmod{p}, \dots, a^m \pmod{p}\}$

$|U| = m$ U ist Untergruppe von $(\mathbb{Z}_p \setminus \{0\}, \odot)$

Satz von Lagrange: $|U| \mid |\mathbb{Z}_p \setminus \{0\}|, m \mid p-1$

$p-1 = km$

$$a^{p-1} \bmod p = a^{k \cdot m} \bmod p = (a^m)^k \bmod p = 1^k \bmod p = 1.]$$

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

$$m^{ed} \bmod n = m \cdot m^{k \cdot (p-1) \cdot (q-1)} \bmod n$$

$$\text{Ist } p \nmid m, \text{ so } m^{k(p-1)(q-1)} = (m^{p-1})^{k(q-1)} \bmod p = 1$$

$$m^{ed} \bmod p = m \cdot 1 \bmod p = m \bmod p$$

$$\text{Ist } p \mid m, \text{ so } m \bmod p = 0 = m^{ed} \bmod p$$

$$\text{In jedem Fall: } m^{ed} \bmod p = m \bmod p$$

$$\text{Analog: } m^{ed} \bmod q = m \bmod q$$

$$\rightarrow m^{ed} \bmod n = m \bmod n \underbrace{=}_m m$$

$m < n$

5.3 Berechnung modularer Potenzen

$$m^e \bmod n$$

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

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

Square-and-Multiply-Algorithmus $2 \cdot \log_2(e) =$ Maximalzahl der erforderlichen Multiplikationen / Quadrierungen.

Nach jedem Rechenschritt mod n reduzieren. (In der Praxis will man Verschlüsseln beschleunigen: Man wählt e von der Form $2^a + 1$)

5.4 Sicherheit des RSA-Verfahrens

(a) Angreifer kennt n , nicht p, q (Faktorisierung ist schwer)

Wenn er $\varphi(n)$ kennt, so kann er d bestimmen (EEA)

$\varphi(n)$ aus n per Definition zu bestimmen ist aussichtslos.

Geht schnell, wenn er p, q kennt.

Tatsächlich: Fakt von n zu bestimmen bzw. $\varphi(n)$ zu bestimmen ist gleich schwierig.

Angenommen er kennt $\varphi(n)$.

$$n = p \cdot q.$$

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

Dann kennt er $p + q = ir$.

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

Quadratische Gleichung für p (und für q). Lösung ist einfach.

Tatsächlich:

- Faktorisierung von n
- Berechnung von $\varphi(n)$
- Bestimmung von d

Gleichschwierige Probleme.

Offen: Ist die Bestimmung von m aus $c = m^e \bmod n$ (bei Kenntnis von e und n) genauso schwierig wie Bestimmung von d ?

- (b) Berechnung von m aus $m^e \bmod n = c$ ist einfach, falls $m^e < n$.

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

e -te Wurzel aus c : m

Einfach (binäre Suche).

Lösung für dieses Problem: Nur Teil von m ist tatsächlich Nachricht.

Rest wird durch zufallsabhängiges (aber systematisches) Padding ergänzt.

Standard: OAEP (Optimal asymmetric encryption padding)

- (c) Brute Force für Faktorisierung.

Teste alle Zahlen $\leq \sqrt{n}$, ob sie Teiler von n sind.

Inputlänge: $\log(n)$

$\sqrt{n} = 2^{\frac{1}{2}\log(n)}$ exponentiell.

Unbekannt: Gibt es polyn. Faktorisierungsalgorithmen, d.h. mit Komplexität $O(\log(n)^k)$, k fest?

Beste Faktorisierungsalgorithmen haben Komplexität $O(e^{c(\log(n))^{\frac{1}{3}}(\log(\log(n)))^{\frac{2}{3}}})$, $c \approx 1,923$

Dezember 2009: 768-Bit-Zahl faktorisiert (Kleijung), Dauer: 2000 CPU-Jahre (2GHz)

1024 Bit: c.a 1000 mal größerer Aufwand.

Gilt nicht mehr als sicher.

2048 bit: 10^9 mal schwieriger als bei 1024-Bit Zahl

Vergleich: Brute Force für AES \approx Faktorisierung von RSA - n mit 3064-Bit n .

5.5 Bestimmung großer Primzahlenn

Geht schnell.

Idee: Wenn p eine Primzahl, $1 \leq a \leq p-1$, so ist $a^{p-1} \bmod p = 1$ (kleiner Satz von Fermat)

Sei $n \in \mathbb{N}$. Wähle $1 \leq a \leq n-1$. Teste, ob $a^{n-1} \bmod n = 1$.

Wenn nein: n ist keine Primzahl!

Wenn ja: Wähle neues a . Fermat-Test

Wenn n Fermat-Test für alle $1 \leq a \leq n-1$ besteht, so ist n Primzahl.

5.6. DAS DIFFIE-HELLMAN-VERFAHREN ZUR SCHLÜSSELVEREINBARUNG (1976)23

Leider gibt es unendlich viele zusammengesetzte Zahlen n , die den Fermat-Test für "fast" alle a bestehen (nämlich für alle a mit $\text{ggT}(a,n) = 1$) (Carmichael-Zahlen)

Verbesserung: Miller-Rabin-test

Sei p eine Primzahl, $p \neq 2$. $p - 1 = 2^e \cdot t$, $2 \nmid t$

$a^{p-1} = 1$ (Fermat)

$(a^{2^{e-1} \cdot t})^2 = 1$

$a^{2^{e-1} \cdot t}$ ist Nullstelle des Polynoms: $x^2 - 1 \in \mathbb{Z}_p[x]$

\mathbb{Z}_p Körper $\Rightarrow x^2 - 1$ hat genau 2 Nullstellen in \mathbb{Z}_p , nämlich 1 und $-1 \pmod{p}$

$a^{2^{e-1} \cdot t} \equiv \begin{cases} 1 \pmod{p} \\ -1 \pmod{p} \end{cases}$

Falls $e - 1 \geq 1$ und falls $a^{2^{e-1} \cdot t} \equiv 1 \pmod{p}$

so $(a^{2^{e-2} \cdot t})^2 \equiv 1 \pmod{p} \Rightarrow a^{2^{e-2} \cdot t} \equiv \begin{cases} 1 \pmod{p} \\ -1 \pmod{p} \end{cases}$

Ist p eine Primzahl, $p-1 = 2^e \cdot t$, $1 \leq a \leq p-1$, so gilt:

entweder $a^t \equiv 1 \pmod{p}$ oder $a^{2^i \cdot t} \equiv -1 \pmod{p}$ für ein $i \in \{0, \dots, e-1\}$

Miller-Rabin: Teste mit n statt p .

Test nicht erfüllt: n ist keine Primzahl.

Test erfüllt: ? Wähle neues a !

Es gilt: Ist n keine Primzahl, so gibt es mindestens $\frac{3}{4}\varphi(n)$ a 's, $1 \leq a \leq n-1$ und $\text{ggT}(a,n) = 1$, die zeigen, dass n keine Primzahl ist.

Besteht das n für mindestens $\frac{1}{4}\varphi(n)$ a 's mit $1 \leq a \leq n-1$ und $\text{ggT}(a,n) = 1$ den Miller-Rabin-Test, so ist n Primzahl.

5.6 Das Diffie-Hellman-Verfahren zur Schlüsselvereinbarung (1976)

D-H ist kein Public-Key-Verfahren.

Schlüsselvereinbarung über unsicheren Kanal.

Es beruht auf folgendem Kandidaten für Einwegfunktion:

(a) p Primzahl, $\mathbb{Z}_p^* := \mathbb{Z}_p \setminus \{0\}$ ist Gruppe bezüglich Multiplikation \odot

Algebra: $\exists g \in \mathbb{Z}_p^*$ mit $\mathbb{Z}_p^* = \{g, g^2, \dots, g^{p-1} = 1\}$

g heißt Primitivwurzel mod p .

$\{0, \dots, p-2\} \rightarrow \{1, \dots, p-1\}$

$a \mapsto g^a \pmod{p}$

Kandidat für Einwegfunktion.

Umkehrfunktion: diskreter Logarithmus

Keine polyn. Algorithmen bekannt.

Beispiel: $p = 7$, $g = 3$

$\{3, 3^2 = 2, 3^3 = 6, 3^4 = 4, 3^5 = 5, 3^6 = 1\}$

(b) Das Verfahren:

A und B einigen sich auf große Primzahl p und Primitivwurzel $g \bmod p$.
Können öffentlich bekannt sein.

1. A wählt zufällig (geheimes) $a \in \{2, \dots, p-2\}$ und berechnet $x := g^a \bmod p$.
A sendet x an B.
2. B wählt zufällig (geheimes) $b \in \{2, \dots, p-2\}$ und berechnet $y := g^b \bmod p$. B sendet y an A.
3. A berechnet $y^a \bmod p = (g^b)^a \bmod p = g^{ab} \bmod p$
4. B berechnet $x^b \bmod p = (g^a)^b \bmod p = g^{ab} \bmod p$

$K := g^{ab} \bmod p$ ist gemeinsamer Schlüssel.

(c) Sicherheit:

Angreifer kennt $p, g, g^a \bmod p, g^b \bmod p$.
Möchte gerne $g^{ab} \bmod p$ wissen!

D-H-Problem:

Gibt es schnellen Algorithmus zur Bestimmung von $g^{ab} \bmod p$ bei Kenntnis von $p, g, g^a \bmod p, g^b \bmod p$? - Unbekannt

Einziger bekannter Weg:

Berechne a aus $x = g^a \bmod p$ (DLP = diskretes Logarithmus Problem)
und dann $K = (y)^a \bmod p$.

p sollte mindestens 2048 Bit haben.

5.7 Der Man-in-the-Middle-Angriff auf D-H-Verfahren

Mallory wählt zufällig $c \in \{2, \dots, p-2\}$.

$$A \leftarrow M \rightarrow B$$

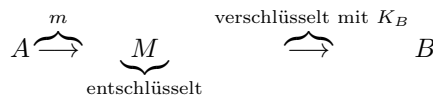
M fängt $x = g^a \bmod p$ von A ab, $y = g^b \bmod p$ von B ab.

M schickt A und B: $g^c \bmod p$.

A berechnet $K_A = g^{ac} \bmod p$ und glaubt, das ist der gemeinsame Schlüssel mit B.

B berechnet $K_B = g^{bc} \bmod p$ und glaubt, das ist der gemeinsame Schlüssel mit A.

M kennt K_A und K_B .



Authentifizierung von A gegenüber B und umgekehrt notwendig.

5.8 ElGamal-Verschlüsselungsverfahren (T. ElGamal, 1984)

5.8.1 Schlüsselerzeugung:

A: p, g wie bei Diffie-Hellman.

a zufällig in $\{2, \dots, p-2\}$, $x = g^a \mod p$

Öffentlicher Schlüssel: (p, g, x)

Geheimer Schlüssel: a

5.8.2 Verschlüsselung

Klartextblock m , $0 \leq m \leq p-1$, $B \rightarrow A$

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

$y = g^b \mod p$ und $f := x^b \cdot m \mod p$

Er sendet (y, f) an A .

5.8.3 Entschlüsselung

A berechnet $y^a \mod p = x^b \mod p (= K)$

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

Wie berechnet sie $(y^a)^{-1} \mod p$?

- Erweiterter Euklidischer Algorithmus (y^a, p)
- $(y^a)^{p-2} \mod p = (y^a)^{-1}$, denn $y^a \cdot (y^a)^{p-2} = (y^a)^{p-1} = 1 \mod p$ (Fermat)

Wichtig:

B muss bei Verschlüsselung von neuem Klartextblock neues b wählen, sonst kennt Angreifer $f_1 = x^b \cdot m_1$, $f_2 = x^b \cdot m_2$.

Angenommen Angreifer kennt m_1 . Kennt $f_1 \cdot m_1^{-1} = x^b \mod p$
 $(x^b)^{-1} \cdot f_2 = m_2 \mod p$

Sicherheit von ElGamal = Sicherheit von Diffie-Hellman

5.9 Bug-Attacks

(Biham, Carmeli, Shamir) 2015 Journal of Cryptology

RSA-Verfahren (deterministisch) (RSA-CRT-Entschlüsselung).

$c = m^e \mod n$, $n = p \cdot q$.

$m_1 = c^d \mod p$

$m_2 = c^d \mod q$

$m_1, m_2 \rightarrow m$ (Chinesischer Restsatz)

a, b 64-bit Zahlen.

$a \cdot b$ wird auf 64-Bit-Multiplier falsch berechnet. (1024-Bit n)

$$p < \sqrt{n} < q$$

Angreifer bildet C in der Nähe von \sqrt{n} .

In der Regel: $p < C < q$

(Chosen-Ciphertext-Angriff)

C entschlüsselt mit RSA-CRT.

$C \bmod p$ enthält in der Regel nicht mehr a und b .

Entschlüsselung mod $p \rightarrow$ korrekt (M_1)

$C \bmod q = C$

$C^d \bmod q$ berechnen $\rightarrow \underline{a \cdot b}$ (BUG) wird auf Chip berechnet.

Entschlüsselung mod q nicht korrekt (M_2).

M_1, M_2 mit Chinesischem Restsatz zu M .

$M \bmod p = M_1$

$M \bmod q = M_2$

Angreifer erhält M : $M^e \bmod n = C' \neq C$

$C' \bmod p = C \bmod p$

$C' \bmod q \neq C \bmod q$

$p | \text{ggT}(C' - C, n)$, $q \nmid \text{ggT}(C' - C, n) \rightarrow \text{ggT}(C' - C, n) = p$. RSA ist geknackt.

Signaturen, Hashfunktionen, Authentifizierung

6.1 Digitale Signaturen

Anforderung: Niemand außer A kann Dokument mit der Signatur von A versehen, selbst wenn er Signatur von A von anderen Dokumenten kennt.

Also auch: A kann nicht abstreiten, Dokument signiert zu haben.

Signatur gewährleistet:

- Identitätseigenschaft des Unterzeichners des Dokuments
- Echtheitseigenschaft des Dokuments

Außerdem: Verifikationseigenschaft

Jeder Empfänger einer von A signierten Nachricht muss Signatur verifizieren können.

Realisierung über RSA-Signatur.

6.2 RSA-Signatur

A will Dokument signieren. Sie besitzt öffentlichen RSA-Schlüssel (n,e) , geheimen RSA-Schlüssel d .

Signatur von Dokument m ($m < n$):

$$s(m) := m^d \mod n$$

A sendet $(m, s_A(m))$ an B.

B verifiziert: $s_A(m)^e \mod n = m^{de} \mod n = m$, Signatur wird akzeptiert.

Falls $s_A(m)^e \mod n \neq m$, Signatur wird nicht akzeptiert.

Funktioniert so gut, da bei RSA Verschlüsselung / Entschlüsselung vertauschbar sind.

Signaturen sind auch mit ElGamal konstruierbar, komplizierter wegen Zufallszahl bei der Verschlüsselung.

Problem: Signatur ist so lang wie Nachricht.

Ausweg: Kryptografische Hashfunktionen.

6.3 Definition Hashfunktion

R endliches Alphabet. Hashfunktion ist Abbildung $R^* \rightarrow R^k$ (k fest)

6.4 RSA-Signatur mit Hashfunktion

A will m signieren (jetzt nicht notwendig $m < n$). Hashfunktion H der Länge k (Hashwerte $< 2^k \leq n$).

H öffentlich bekannt.

Sie sendet $(m, \underbrace{H(m)^d}_{s_A(m)} \bmod n)$

Verifikation: B bildet $H(m)$ aus m mit H . $s_A(m)^e \bmod n = H(m)$?

→ Ja ✓, Nein: nicht akzeptiert

6.5 Anforderung an H

- (a) Angreifer kennt $(m, H(m)^d \bmod n)$.

Er kann $H(m)$ bestimmen. Gelingt es ihm ein $m' \neq m$ zu finden mit $H(m') = H(m)$, so ist $(m', H(m)^d \bmod n)$ gültige Signatur von m' durch A.

- (b) Angreifer wählt zufällig y und berechnet $y^e \bmod n = z$. Gelingt es ihm ein m zu finden mit $H(m) = z$, so ist (m, y) eine gültige Signatur von m durch Alice.

Verifikation:

- $y^e \bmod n = z$ ✓
- $H(m) = z$ ✓

6.6 Definition: Kryptographische Hashfunktion

Eine kryptographische Hashfunktion ist Hashfunktion H , die folgende Bedingungen erfüllen muss:

- (1) H ist Einwegfunktion (um Angriffe vom Typ 6.5.b) zu unterbinden)
- (2) H ist schwach kollisionsresistent, d.h. zu gegebenem n ist es nicht effizient möglich ein $m' \neq m$ zu finden mit $H(m') = H(m)$ (um Angriffe vom Typ 6.5.a) zu unterbinden) (second pre-image resistant)

Verschärfte Anforderung an H als (2):

- (2') H ist stark kollisionsresistent, d.h. es ist nicht effizient möglich zwei $m_1 \neq m_2$ zu finden mit $H(m_1) = H(m_2)$

(Solche Kollisionen gibt es, sogar unendlich viele)

[Kryptographische Hashfunktion = message digest = digital fingerprint]

Wie lang sollte $H(m)$ sein?

- Nicht zu lang (Sinn von Hashfunktion)
- Nicht zu kurz ("Geburtstagsparadox") \rightarrow Geburtstagsattacke

($H(m)$ hat 128 Bit Länge. Angriff auf starke Kollisionsresistenz:
Wenn er $2^{128} + 1$ $H(m)$'s erzeugt hat, hat er sicher Kollision)

6.7 Satz (Geburtstagsparadox)

Ein Merkmal komme in m verschiedenen Ausprägungen vor. Jedes Objekt (einer Grundgesamtheit) besitze genau eine dieser Merkmalsausprägungen (mit gleicher Wahrscheinlichkeit).

Ist dann $l \geq \frac{1+\sqrt{1+8\cdot m\cdot \ln(2)}}{2} (\approx 1.18\sqrt{m})$, so ist die Wahrscheinlichkeit, dass unter l Objekten zwei die gleiche Merkmalsausprägung haben, $\geq \frac{1}{2}$.
(Geburtstagsparadoxon: $m = 366$, $l \geq 23 \rightarrow$ Wahrscheinlichkeit $\geq \frac{1}{2}$, dass 2 Personen am gleichen Tag Geburtstag haben)

6.7.1 Beweis

Gegeben l Objekte.

Alle Ereignisse: $(g_1, \dots, g_l) \in \{1, 2, \dots, m\}^l$

Anzahl: m^l .

Alle g_i paarweise verschieden: $\prod_{i=0}^{l-1} (m-i)$

Wahrscheinlichkeit, dass keine 2 Objekte gleiche Merkmalsausprägung haben:

$$q := \frac{\prod_{i=0}^{l-1} (m-i)}{m^l} = \prod_{i=0}^{l-1} \left(1 - \frac{i}{m}\right)$$

Benutze: $e^x \geq 1 + x$.

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

Wann ist $q \leq \frac{1}{2}$?. Sicher, wenn

$$e^{-\frac{l(l-1)}{2m}} \leq \frac{1}{2} \Leftrightarrow \frac{l(l-1)}{2m} \geq \ln(2) \Leftrightarrow l^2 - l - 2m \cdot \ln(2) \geq 0 \Leftrightarrow l \geq \frac{1+\sqrt{1+8m\cdot \ln(2)}}{2}$$

6.8 Geburtstagsattacke (gegen starke Koll.res.)

Gegeben: $H: \{0, 1\}^* \rightarrow \{0, 1\}^k$

Angriffe erzeugt möglichst viele Hashwerte., diese werden auf Kollisionen untersucht.

Wende 6.7 an:

- Menge der Objekte = $\{0, 1\}^*$
- Merkmalsausprägungen = Hashwerte
- Anzahl aller möglichen Hashwerte $m = 2^k$

Wahrscheinlichkeit $\frac{1}{2}$, dass Kollision vorliegt, bei $\sqrt{2^k} = 2^{\frac{k}{2}}$ vielen erzeugten Hashwerten.

$k = 64$ keine ausreichende Sicherheit. Wahrscheinlichkeit $\frac{1}{2}$ Kollision bei $2^{32} \approx 4 \cdot 10^9$ erzeugten Hashwerten

Forderung: $k \geq 128$, besser $k \geq 160$

6.9 Bemerkung

Weit verbreitete Hashfunktionen:

- MD5 (R. Rivest 1992) $n = 128$
- SHA-1 (NSA, NIST 1992/93/95) $n = 160$
- SHA-2-Familie (SHA-224, 256, 384, 512 NIST 2002/04)

Kollision bei SHA-1 mit ca. $2^{51} - 2^{57}$ Hashoperationen (vgl. mit 2^{80})

2007: NIST - Ausschreibung für neuen Standard bei Hashfunktionen

2012: Gewinner KECCAK (Bertoni, Daemen, Peeters, van Assche). Länge des Hashwertes: 224 - 512 Bit

Konstruktionsprinzipien: Paar, Pelzl, Kap. 11

6.10 Authentifizierung

Nachweis / Überprüfung der Identität.

Forderung: Niemand anderes als A darf sich als A ausgeben können (auch nicht der Verifizierer).

Authentifizierung durch:

- Wissen
- Besitz
- biometrische Merkmale

6.11 Passwörter

A wählt Passwort w . Bei B, gegenüber dem sich A authentifizieren will, ist $f(w)$ gespeichert, f Einwegfunktion (z.B. Hashfunktion).

→ Unsicher !

Besser: Einmal-Passwörter

Lamport, 1981:

f Einwegfunktion (öffentlich bekannt)

A: $w_A, f(w_A), f^2(w_A) = f(f(w_A)), \dots, f^n(w_A)$

Am Anfang sendet A an B: $w_0 = f^n(w_A)$ (auf sicherem Weg)

Authentifizierung:

- A sendet $w_1 = f^{n-1}(w_A)$ an B
- B testet, ob $f(w_1) = w_0$.

Nach der ersten Authentifizierung ersetzt B w_0 durch w_1 . Nächste Authentifizierung $A \rightarrow B$ $f^{n-2}(w_A) = w_2$, B: $f(w_2) = w_1$. Etc.

Vorteil: Aus $f^i(w_A)$ lässt sich $f^{i-1}(w_A)$ nicht rekonstruieren.

6.12 Challenge-Response-Authentifizierungen

Signatur von Zufallsstring, z.B. mit RSA-Verfahren.

(n, e) öffentlicher Schlüssel von A, d geheimer Schlüssel von A)

$A \xrightarrow{\text{auth.}} B$

B wählt Zufallszahl $r < n$. A berechnet $r^d \bmod n = c \rightarrow B$.

B verifiziert, ob $c^e \bmod n = r$.

Wichtig: Authentifizierungsschlüssel und Verschlüsselungsschlüssel müssen verschieden sein !

Secret Sharing Schemes

Verteilung eines Geheimnisses auf mehrere Personen (Teilgeheimnisse), so dass gewisse Teilmengen dieser Personen mit ihren Teilgeheimnissen das Geheimnis rekonstruieren können, die anderen Teilmengen nicht.

7.1 Definition

- (a) Sei T eine Menge von Teilnehmern, $\mathcal{Z} \subseteq \mathcal{P}(T)$ Menge der zulässigen Konstellationen.
(Überlicherweise fordert man Monotonie von \mathcal{Z} :

$$A \subseteq C, A \in \mathcal{Z}, A \subseteq B \subseteq T \rightarrow B \in \mathcal{Z}$$

)

$$[T = \{1, 2, 3, 4\}, \mathcal{Z} = \{\{1, 2\}, \{1, 3, 4\}, \{1, 2, 3\}, \{1, 2, 4\}, \{1, 2, 3, 4\}\}]$$

- (b) Sei g ein Geheimnis (i.d.R. aus einer großen Menge möglicher Geheimnisse).

In einem Secret Sharing Scheme erhalten alle Teilnehmer durch eine vertrauenswürdige Instanz (Dealer) Elemente einer Menge \mathcal{S} (Teilgeheimnisse, shadows), so dass gilt:

- (1) Jede zulässige Konstellation kann mit ihren Teilgeheimnissen g rekonstruieren
- (2) Unzulässige Konstellationen können das nicht

- (c) SSS heißt perfekt, wenn jede unzulässige Konstellation mit ihren Teilgeheimnis genauso viel über g weiß wie ohne ihre Teilgeheimnisse.

7.2 Definition

$$T = \{1, \dots, n\}, \text{ Sei } k \leq n. \mathcal{Z} = \{U \subseteq T : |U| \geq k\}$$

Ein SSS zu einem solchen \mathcal{Z} heißt (k,n) - Schwellenwertsystem (threshold system)

7.3 Shamirs Konstruktion eines Schwellenwertsystems

- (a) Vorgegeben sei große Primzahl p (in jedem Fall $p \geq n + 1$).

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

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

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

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

Dealer wählt paarweise verschiedene $x_1, \dots, x_n \in \mathbb{Z}_p \setminus \{0\}$ (öffentlich bekannt).

Teilnehmer i erhält als Teilgeheimnis:

$$g_i = f(x_i) \text{ (und } x_i)$$

- (b) Zur Rekonstruktion von g müssen k Teilnehmer kooperieren, etwa i_1, \dots, i_k .
Durch $(x_{i_1}, g_{i_1}), \dots, (x_{i_k}, g_{i_k})$ ist das Polynom f eindeutig bestimmt.
Zum Beispiel mit LGS:

$$g + a_1x_{i_1} + \dots + a_{k-1}x_{i_1}^{k-1} = g_{i_1}$$

...

$$g + a_1x_{i_k} + \dots + a_{k-1}x_{i_k}^{k-1} = g_{i_k}$$

k Gleichungen mit k Unbekannten. Hat Lösung, und diese ist eindeutig (f, f' vom Grad $\leq k-1$, die an k Stellen den gleichen Wert haben, so hat $f - f'$ Grad $\leq k-1$ und mindestens k Nullstellen $\rightarrow f - f' = 0, f = f'$)

Oder sie bestimmen f durch sogenannte Interpolation (z.B. Lagrange-Interpolation).

$$f(x) = \sum_{j=1}^k \frac{(x - x_{i_1}) \dots (x - x_{i_{j-1}})(x - x_{i_{j+1}}) \dots (x - x_{i_k})}{(x_{i_j} - x_{i_1}) \dots (x_{i_j} - x_{i_{j-1}})(x_{i_j} - x_{i_{j+1}}) \dots (x_{i_j} - x_{i_k})} \cdot g_{i_j} \quad (7.1)$$

Direkte Bestimmung von g durch Einsetzen von $x = 0$:

$$g = \sum_{j=1}^k g_{i_j} \cdot \prod_{l \neq j} \frac{x_{i_l}}{x_{i_l} - x_{i_j}} \quad (7.2)$$

Mehr als k Teilnehmer erzeugen dasselbe Polynom.

- (c) Angenommen $k' < k$ Teilnehmer wollen g rekonstruieren. Für jedes $h \in \mathbb{Z}_p$ existieren gleich viele Polynome von Grad $\leq k-1$, die durch $(x_{i_1}, g_{i_1}), \dots, (x_{i_{k'}}, g_{i_{k'}})$ und durch $(0, h)$ gehen.

\Rightarrow Shamir-Verfahren ist perfekt.

Codierungstheorie

8.1 Grundbegriffe und einfache Beispiele

8.1.1 Codierung (genauer Kanalcodierung)

Sicherung von Daten gegen Störungen bei der Übertragung / Speicherung.

Quelle (Nachricht über Alphabet R) \rightarrow Kanalcodierung (codiert Nachricht in Codewort oder Folge von Codewörtern über Alphabet S) \rightarrow Kanal (Störungen!) \rightarrow Decodierer (Rekonstruktion der Nachricht 1. Codewort zurückgewinnen 2. Nachricht zurückgewinnen) \rightarrow Empfänger

Ziel:

1. Möglichst viele Fehler erkennen und möglichst korrigieren
2. Aufwand für Codierung und Decodierung soll gering sein

Grundprinzip: Hinzufügen von Redundanz in systematischer Weise.

Fehlererkennung reicht, wenn Nachricht nochmal gesendet werden kann.

8.1.2 Beispiele

(a) Parity-Check-Code

$R = \{0, 1\}$, $k \in \mathbb{N}$.

Nachricht wird in Blöcke der Länge k zerlegt.

Codierung: Block der Länge $k \rightarrow$ Block der Länge $k+1$

Anhängen eines Bits, so dass Anzahl der Einsen im Block der Länge $k+1$ gerade ist.

$k=2$

$00 \rightarrow 000$

$01 \rightarrow 011$

$10 \rightarrow 101$

$11 \rightarrow 110$

\rightarrow 1 Fehler wird erkannt (kann nicht korrigiert werden), 2 Fehler werden nicht erkannt

(b) Wiederholungscode

Nachricht in Blöcke der Länge k zerlegen. Jeder Block wird m -mal wiederholt. (m -facher Wiederholungscode)

$k=2, m=3$

$00 \rightarrow 000000$

$01 \rightarrow 010101$

$10 \rightarrow 101010$

$11 \rightarrow 111111$

Wenn genau ein Fehler aufgetreten ist, kann er korrigiert werden.

Zum Beispiel:

$001000 \rightarrow 000000$

Angenommen zwei Fehler sind aufgetreten. Es wird erkannt, dass Fehler aufgetreten ist.

Angenommen gesendet wurde 000000. Mögliche 2 Fehler:

- $000011 \rightarrow 000000 \checkmark$
- $000101 \rightarrow 010101$ falsch
- $001001 \rightarrow ?$

(c) Codiere Blöcke der Länge 2 über $\{0, 1\}$ folgendermaßen:

$00 \rightarrow 00000$

$01 \rightarrow 01101$

$10 \rightarrow 10110$

$11 \rightarrow 11011$

Je zwei Codewörter unterscheiden sich an mindestens 3 Stellen.

Wenn genau ein Fehler aufgetreten ist und der Decodierer das “nächstgelegene” Codewort wählt, so Decoder korrekt.

Wenn 2 Fehler auftreten, wird erkannt, dass kein Codewort empfangen wird.

\rightarrow 1 Fehler korrigiert, 2 Fehler erkannt

8.1.3 GTIN-Prüfzifferncode (GTIN-13)

(a) GTIN: Global Trade Item Number (Artikelnr.)

13-stelliger Code: die ersten 12 Ziffern entsprechen Nachricht / Information, 13. Ziffer ist Prüfziffer

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

$c_1 \dots c_{13}$

$c_1 \dots c_{12}$

Herstellungsland (i.d.R. die ersten drei, D: 400-440)

Hersteller (i.d.R. c_4, \dots, c_8)

Produkt (i.d.R. c_9, \dots, c_{12})

c_{13} so, dass $c_1 + 3c_2 + c_3 + 3c_4 + \dots + 3c_{12} + c_{13} \equiv 0 \pmod{10}$

Blockcodes

9.1 Satz (10.7)

G Erzeugermatrix eines $[n,k]$ -Codes \mathcal{C} über K , so

$$C = \left\{ \underbrace{u}_{1 \times k} \cdot \underbrace{G}_{k \times n} : u \in K^k \right\}$$

9.1.1 Beweis

$$u \cdot G = (u_1, \dots, u_k) \cdot \begin{pmatrix} g_1 \\ \dots \\ g_k \end{pmatrix} = u_1 g_1 + \dots + u_k g_k$$

Auf diese Weise entstehen alle Codewörter.

9.2 Bemerkung

- (a) Die Abbildung $\begin{cases} K^k \rightarrow \mathcal{C} \subseteq K^n \\ u \mapsto u \cdot G \end{cases}$ ist injektiv.

Damit: Codierungsmöglichkeit von Informationswörtern der Länge k in Codewörter der Länge n .

- (b) Elementare Zeilenumformung an Erzeugermatrix von \mathcal{C} liefern wieder Erzeugermatrix von \mathcal{C}

9.3 Beispiel

Hamming-Code $[7,4]$ -Code über \mathbb{Z}_2 (9.12.d).

$$\mathcal{C} = \{(c_1, \dots, c_7) \in \mathbb{Z}_2^7 : c_1 + c_4 + c_6 + c_7 = 0, c_2 + c_4 + c_5 + c_7 = 0, c_3 + c_5 + c_6 + c_7 = 0\}$$

c_4, \dots, c_7 frei wählbar, dann c_1, c_2, c_3 bestimmt.

Erzeugermatrix:

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Durch elementare Zeilenumformungen andere Erzeugermatrix:

$$\tilde{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Codierung von $u = (u_1, u_2, u_3, u_4)$ mit $\tilde{G} : u \cdot \tilde{G} = \underbrace{(u_1, u_2, u_3, u_4, *, *, *)}_{\text{Information}}$

(Eine Erzeugermatrix von der Form $(E_n | *)$ heißt in Standardform. Nicht jeder Code besitzt Erzeugermatrix in Standardform.)

9.4 Satz und Definition

Sei \mathcal{C} ein $[n, k]$ -Code über K . Dann existiert $(n - k) \times n$ -Matrix H , sodass gilt:

$$\text{Ist } y \in K^n, \text{ so: } y \in \mathcal{C} \Leftrightarrow H \cdot y^t = 0 \quad (\Leftrightarrow y \cdot H^t = 0)$$

H heißt Kontrollmatrix von \mathcal{C} . Es ist $rg(H) = n - k$.

(Dann gilt auch: $H \cdot G^t = 0$ - Nullmatrix)

9.4.1 Beweis

Sei g_1, \dots, g_k Basis von \mathcal{C} . $G = (g_1, \dots, g_k)$, $g_i = (g_{i1}, \dots, g_{in})$

Betrachte homogenes LGS: $G \cdot \begin{pmatrix} x_1 \\ \dots \\ x_n \end{pmatrix} = 0$, bzw. $(x_1, \dots, x_n) \cdot G^t = 0$ (Zeilen-

vektor der Länge k)

Zeilen von G sind lin. unabhängig, d.h. $rg(G) = k$.

Daher: Dimension des Lösungsraums von $G \cdot \begin{pmatrix} x_1 \\ \dots \\ x_n \end{pmatrix} = 0$ ist $n - k$.

$$h_1 = \begin{pmatrix} h_{11} \\ \dots \\ h_{1n} \end{pmatrix}, \dots, h_{n-k} = \begin{pmatrix} h_{n-k,1} \\ \dots \\ h_{n-k,n} \end{pmatrix} \text{ Basis des Unterraums.}$$

$$H = \begin{pmatrix} h_1^t \\ \dots \\ h_{n-k}^t \end{pmatrix}, G \cdot h_1 = 0, \dots, G \cdot h_{n-k} = 0.$$

Dann gilt:

- $g_j \cdot h_i = 0, \forall i, j$
- $g_j \cdot H^t = 0, \forall j$
- $H \cdot y^t = 0, \forall y \in \mathcal{C}$

(*) $\mathcal{C} \subseteq$ Lösungsraum von $H \cdot y^t = 0$

Dimension Lösungsraum von $H \cdot y^t = 0$ ist $n - rg(H) = n - (n - k) = k$.

Daher gilt (*).

9.5 Bemerkung

- (a) Kontrollmatrix kann zur Fehlererkennung verwendet werden.
- (b) Beweis von 10.10 \rightarrow Verfahren zur Bestimmung von H aus G.
- (c) Geg. H. Bestimme Basis des Lösungsraums von $H \cdot y^t = 0 \rightarrow G$

9.6 Beispiel

- (a) Parity-Check-Code über \mathbb{Z}_2 , $\mathcal{C} = \{(u_1, \dots, u_n) : \sum u_i = 0 \text{ in } \mathbb{Z}_2\}$

Kontrollmatrix: $(1, \dots, 1) \cdot \begin{pmatrix} u_1 \\ \dots \\ u_n \end{pmatrix} = u_1 + \dots + u_n = 0$

- (b) Hamming [7,4]-Cde aus 10.9:

$$H = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

Kontrollmatrix von \mathcal{C} .

- (c) \mathcal{C} Code mit Erzeugermatrix $\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$ [4,2]-Code über \mathbb{Z}_2 .

Gesucht: Kontrollmatrix

Es muss gelten:

$$\begin{aligned} x_1 + x_4 &= 0 \\ x_2 + x_3 + x_4 &= 0 \end{aligned}$$

x_3, x_4 frei wählbar. Damit folgt:

$$H = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix} \text{ Kontrollmatrix}$$

$$\mathcal{C} = \{(y_1, \dots, y_4) : y_2 + y_3 = 0, y_1 + y_2 + y_4 = 0\}$$