Datensicherheit

(Dr. Schenzel SS 2005 – verschriftlicht von Georg Kuschk)

1. Einleitung

Kryptologie = Kryptanalyse + Kryptographie

Kryptanalyse :

Wissenschaft von der Wiederherstellung des Klartextes einer Chiffre ohne Zugriff auf den Schlüssel (Auffinden von Schwachstellen.)

Maxime der Kryptanalyse : (Kerckhoffsche Regel)
Die Sicherheit eines Chiffrierverfahrens darf nur von der Geheimhaltung des Schlüssels abhängen auf keinem Fall jedoch vom Algorithmus.

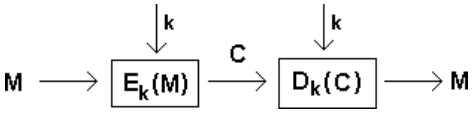
Kryptographie :

Wissenschaft von der Absicherung von Nachrichten.
(Selten ist der genaue/aktuelle Wissensstand der Kryptographie dokumentiert.)

Methoden :
- statistisch , stochastische Verfahren
- algebraisch , kombinatorische Verfahren
- zahlentheoretische Verfahren

Ziele : (Klartext und/oder Schlüssel vor Angreifern schützen.)
- Geheimhaltung
  - organisatorische Maßnahmen (Verschlusssache , Boten)
  - physikalische Maßnahmen (Gehäuse)
  - kryptographische Maßnahmen (Abb. der Nachricht in eine für Außenstehende unverständliche)
    - symmetrische Verfahren (Sender und Empfänger benutzen denselben Schlüssel)
    - asymmetrische Verfahren (Existenz eines geheimen und eines öffentlichen Schlüssels)
    - hybride Verfahren
- Authentikation
  - analog :
    - Aussehen , Stimme , Handschrift
  - elektronisch :
    - Teilnehmerauthentikation (Nachweis durch geheimes Wissen.)
    - Nachrichtenauthentikation (elektronische Unterschrift)
- Anonymität
  - z.B. Geschäftsabwicklung ohne nachweisbare Beteiligung (Bargeld-Einkauf)
  - Bei elektronischer Kommunikation ist es schwierig , Anonymität und Verlässlichkeit in Einklang zu bringen. (z.B. elektronische Bezahlung)

pragmatische Codierungsansätze :
- Steganographie („verborgenes Schreiben“)
  - Verheimlichen einer Nachricht / Vertuschen von Informationen
  - z.B. Einbettung der Nachricht in eine Andere (Audiodateien , Bilder , ...)
  - gewinnt an Bedeutung
- Verschlüsseln einer Nachricht (Klartext->Chiffre)



Anforderung an Verschlüsslung/Codierung :
- $D(E(C)) = M$
- Authentikation (Empfänger soll die Authentizität der Nachricht prüfen können)
- Integrität (Empfänger soll in der Lage sein Manipulationen der Nachricht zu erkennen und gegebenenfalls zu korrigieren.)
- Verbindlichkeit (Ein Sender soll unter Umständen nicht leugnen können , die Nachricht abgeschickt zu haben.)

Beispiel für Anwendungsgebiete :
- Telefonübertragung , Pay-TV , e-banking , Multiuser-Systeme , long distance transmissions

Klassische kryptographische Methoden sind z.B. bei Banknoten : Spezialpapier , Silberdraht , Präzisionsdruck.
Moderne kryptographische Methoden sind besser und effizienter. (Möglichkeit der Risiko-Abschätzung)

Sicherheit bedeutet :
- Aufwand zum Aufbrechen des Algorithmus ist grösser als der Gewinn.
- Zeitraum zum Aufbrechen ist grösser als die notwendige Geheimhaltungszeit.

Kryptographischer Algorithmus :
Mathematische Funktion zum Ver- und Entschlüsseln einer Nachricht.
- Datenkomplexität
- Berechnungskomplexität
- Speicherkomplexität

Berechnungssichere (starke) Algorithmen :
Können mit derzeitigen bzw. absehbaren Resourcen nicht aufgebrochen werden.

One-time-pad :
- zufällige Folge von Schlüsselbits
- bitweise Addition mit Klartext
- Alle Folgen der Länge n werden gleichwahrscheinlich vorkommen.
  (D.h. zufällige Generierung der Schlüsselbits.)
- $\Rightarrow$ Es gibt keine Information in der Chiffre
  (D.h. absolut sicher solange der Angreifer den Schlüssel nicht kennt.)

Nachteile :
- lange Schlüsselfolgen
- Austausch der Schlüssel ist notwendig
- Anwendung des Schlüssels kann nur einmalig erfolgen.

Brute-Force-Angriff : Ausprobieren aller denkbaren Schlüssel.

Cäsar-Chiffre : Zyklische Verschiebung der Buchstaben

Annahme : Gegner haben vollständigen Zugriff auf Kommunikation von Sender und Empfänger.

2. Grundlagen

Definition Teilbarkeit :

$a,b \in N \qquad a \mid n \quad \Leftrightarrow \quad \exists b \in N : a \cdot b = n$

Eigenschaften :
- $\forall a \in N/\{0\} : a \mid 0$
- $\forall a \in N/\{0\} : a \mid a$
- $a \mid b \quad \wedge \quad b \mid c \quad \Rightarrow \quad a \mid c$
- $c \mid a \quad \wedge \quad c \mid b \quad \Rightarrow \quad c \mid (u \cdot a + v \cdot b) \qquad \forall a,b,c \in N \quad \forall u,v \in Z$

Beispiel : $s = (a_1 ... a_n)_{10} \Rightarrow \qquad 9 \mid s \Leftrightarrow 9 \mid \sum a_i$

Definition Rest-Division / Modulo-Division :

$a,b \in Z \quad , b > 0 \quad \Rightarrow$ Es existieren eindeutig bestimte Zahlen $q,r \in Z$ mit den Eigenschaften :
- $a = qb + r$, d.h. $r = a - qb \quad = a \bmod b$
- $0 \le r < b$
- $q = \left\lfloor \frac{a}{b} \right\rfloor$

Bsp : -50 div 8 = -7 , -50 mod 8 = 6

Zahlendarstellung :

- $a = \sum_{i=1}^{k} a_i g^{k-i}$ $\qquad (a_1,...,a_k)_g \in \{0,...,g-1\}$, $a_1 \neq 0$
- $k = \lfloor \log_g a \rfloor + 1$
- $a_i = \left(a - \sum_{j=1}^{i-1} a_j g^{k-j} \right) \text{div } g^{k-i}$

Def. ggT (bzw. gcd) :

$a,b \in Z$ $\qquad c \in Z$ heißt $ggT(a,b)$, falls $c \mid a \wedge c \mid b$ und c maximal bzgl. „$\leq$“

(ggT(0,0) = 0)

Def. ganzzahlige Linearkombinationen :

$aZ + bZ = \{ax + by : x, y \in Z\}$ $\qquad (a,b \in Z)$

D.h. z.B. $1 \in 3Z + 4Z$

Kommunikation erfordert Regelwerk
Nachfrage nach hochwertigen Sicherheitsleistungen , welche komplexe Protokolle erfordern

Protokoll = Regelwerk (z.B. Geldentnahme am Bankautomaten , Schlüsseltausch bei Public-Key-Kryptographie)

Methode von Protokollen :

- Partner kennen Protokoll und Aktionen im Voraus
- Partner müssen sich an vereinbarte Regeln halten
- Protokolle müssen eindeutig sein (d.h. erlaubte Aktionen klar definiert)
- Protokolle müssen vollständig sein (d.h. jeder denkbaren Situation muss eine Aktion zugeordnet sein)

Ziel kryptographischer Protokolle : Verhinderung von Inkorrektheiten und Betrug

Angriffe gegen Protokolle :

- Angriffe gegen Verschlüsselungsalgorithmen
- Ausnutzung von Protokolllücken
- Eingriffe zur Veränderung des Protokolls

Hintergrundinformationen :
RSA Data Security Inc. (Entwicklung , Lizenzierung und Vermarktung des RSA-Patents (Rivest-Shamir-Adleman))
kryptographische Algorithmen sind in den USA patentierbar (z.B. IBM : DES)
Patente können durch die NSA blockiert werden (Invention Secrecy Act , 1940)
Nasa kann ebenfalls Patente einreichen und blockieren
Kryptographie unterliegt in den USA den Ausfuhrbestimmungen und wird als Rüstungsgegenstand behandelt ,
d.h. Verstoss = Waffenschmuggel

Klassische Chiffriertechniken :

- Steganographie (Verstecken von Informationen)
  In Audio- oder Bild-Dateien Änderung des am wenigstens signifikanten Bits
  Nachteil : große Datenmengen

- Transpositionsalgorithmen (Permutation der Stellen der Buchstaben)

Skytale (Stoffumwicklung eines Stockes)

Cäsar-Chiffre (Substitution durch zyklische Permutation modulo 26)

$C = E(p) = p + k \bmod 26$

$p = D(C) = C - k \bmod 26$

Def. : Eine Chiffrierung heißt ***monoalphabetische Substitution*** , falls sie durch Substitution eines Alphabetes hervorgeht , ansonsten polyalphabetisch.

affine Chiffren :

Klartextalphabet , Schlüssel $(a,b) \in (Z_m)^2$

$E : \Sigma \to \Sigma$

$x \mapsto a \cdot x + b \bmod m$ (d.h. Cäsar-Chiffre für $a = 1$)

Beispiel : $a = 3 \quad , \quad b = 0$

$E([x]_{26}) = 3x \bmod 26$

$a \mapsto c \quad , \quad b \mapsto f \quad , \quad c \mapsto i \quad , \quad d \mapsto o \quad ,..., \quad z \mapsto z$

Dechiffrierung durch modulares Invertieren :

$a = 3 \quad \Rightarrow \quad [3]_{26}^{-1} \equiv 9 \bmod 26$

(Da $(3 \bmod 26) \cdot (9 \bmod 26) = (1 \bmod 26)$)

Verallgemeinerung :

affin lineare Blockschriften :

$0 \mapsto [1]_{37} \quad ,..., \quad 9 \mapsto [10]_{37} \quad , \quad a \mapsto [11]_{37} \quad ,..., \quad z \mapsto [36]_{37}$

Beispiel : $m = 37$, Nachricht : „good“

$"good" \hat{=} [17]_{37}[25]_{37}[25]_{37}[14]_{37}$

Aufteilung in Blockcode :

$\begin{pmatrix} [17]_{37} \\ [15]_{37} \end{pmatrix}, \begin{pmatrix} [25]_{37} \\ [14]_{37} \end{pmatrix}$

Multiplikation mit Verschlüsselungsmatrix , wobei diese invertierbar sein muss (d.h. det $\neq 0$)

$$\begin{pmatrix} [3]_{37} & [13]_{37} \\ [22]_{37} & [15]_{37} \end{pmatrix} \cdot \begin{pmatrix} [17]_{37} \\ [25]_{37} \end{pmatrix} = \begin{pmatrix} [376]_{37} \\ [749]_{37} \end{pmatrix} = \begin{pmatrix} [6]_{37} \\ [9]_{37} \end{pmatrix}$$

$$\begin{pmatrix} [3]_{37} & [13]_{37} \\ [22]_{37} & [15]_{37} \end{pmatrix} \cdot \begin{pmatrix} [25]_{37} \\ [14]_{37} \end{pmatrix} = \begin{pmatrix} [257]_{37} \\ [760]_{37} \end{pmatrix} = \begin{pmatrix} [35]_{37} \\ [20]_{37} \end{pmatrix}$$

$\Rightarrow "good" \hat{=} [17]_{37}[25]_{37}[25]_{37}[14]_{37} \quad \mapsto \quad [6]_{37}[9]_{37}[35]_{37}[20]_{37} \hat{=} "58yj"$

Dechiffrierung :

$$\det \begin{pmatrix} [3]_{37} & [13]_{37} \\ [22]_{37} & [15]_{37} \end{pmatrix} = [-241]_{37} = [-19]_{37} = [18]_{37} \quad \Rightarrow [18]_{37}^{-1} \equiv 35 \bmod 37$$

$$\begin{pmatrix} [3]_{37} & [13]_{37} \\ [22]_{37} & [15]_{37} \end{pmatrix}^{-1} = 35 \cdot \underbrace{\begin{pmatrix} [-15]_{37} & [-13]_{37} \\ [-22]_{37} & [-3]_{37} \end{pmatrix}}_{\text{=Matrix der Konjugierten}} = \begin{pmatrix} [7]_{37} & [26]_{37} \\ [7]_{37} & [31]_{37} \end{pmatrix}$$

D.h. die modulare inverse Matrix liefert die Dechiffrierung

$$\begin{pmatrix} [7]_{37} & [26]_{37} \\ [7]_{37} & [31]_{37} \end{pmatrix} \cdot \begin{pmatrix} [6]_{37} \\ [9]_{37} \end{pmatrix} = \begin{pmatrix} [276]_{37} \\ [321]_{37} \end{pmatrix} = \begin{pmatrix} [17]_{37} \\ [25]_{37} \end{pmatrix}$$

$$\begin{pmatrix} [7]_{37} & [26]_{37} \\ [7]_{37} & [31]_{37} \end{pmatrix} \cdot \begin{pmatrix} [35]_{37} \\ [20]_{37} \end{pmatrix} = \begin{pmatrix} [765]_{37} \\ [865]_{37} \end{pmatrix} = \begin{pmatrix} [25]_{37} \\ [14]_{37} \end{pmatrix}$$

Euklidscher Algorithmus (Bestimmung des ggT)

Voraussetzung : $(|a| \geq |b|)$

```
int r;
do
{
  r = a % b;
  a = b;
  b = r;
}
while(b!=0);

ggt = a
```

Variante des Euklidschen Algorithmus

Für ganze Zahlen $a,b \quad (|a| \geq |b|)$ werden iterativ die Tripel $W_i = (t_i, u_i, v_i)$ konstruiert

$W_0 = (a,1,0)$

$W_1 = (b,0,1)$

$W_{i+1} = W_{i-1} - q_{i+1} \cdot W_i \qquad q_{i+1} = t_{i-1} \; div \; t_i$

und zwar bis $t_j = 0$.

Dann gilt $ggT(a,b) = t_{j-1}$ und die BEZOUT-Gleichung der Zahlen $a,b \quad ggT(a,b) = u_{j-1}a + v_{j-1}b$.

Für jeden Schritt also $t_{j-1} = u_{j-1}a + v_{j-1}b$.

Anwendung : ganzzahlige Lösungen für Gleichungen bestimmen

Beispiel : $x \cdot [17]_{29} = [3]_{29}$

$[x \cdot 17]_{29} = [3]_{29}$, d.h.

$x \cdot 17 = y \cdot 29 + 3$

$-29y + 17x = 3$

Algorithmus :

i	0	1	2	3	4	5	6
$a = -29$	-29	17	-12	5	-2	1	0
$\& \; b = 17$	1	0	1	1	3	7	17
	0	1	1	2	5	12	29

Wegen $t_{j-1} = u_{j-1}a + v_{j-1}b$ und dem gewünschten Rest 3 werden nun die Spalten 4 und 5 „addiert“ :

$-t_4 = 2 = -3a - 5b$

$t_5 = \quad 1 = 7a + 12b$

$3 = 4a + 7b$ (d.h. $y = 4 \quad , \quad x = 7$)

$\Rightarrow [3]_{29} = [x \cdot 17]_{29} = [119]_{29} = [3]_{29}$

Algorithmen zur Multiplikation großer Zahlen :

- Karatsuba (bis ca. 1200 Dezimalstellen) $O(n^{1.53})$
- Schönhage-Strassen (>1200 Dez) $O(n \cdot \log_2(n) \cdot \log_2(\log_2(n)))$

Potenzen von Restklassen :

$a^b \pmod m$

Algorithmus :

$a^b = 1$, falls $b = 0$

$a^b = a$, falls $b = 1$

$a^b = \left(a^{b-1}\right)a$, falls $(b\%2 == 1)$

$a^b = \left(a^{b/2}\right)$ sonst

Bestimmung der Anzahl primer Restklassen einer Zahl : **Eulersche Φ-Funktion**

$m \in N$, $p_1,...,p_n$ Primteiler von m

1. $\{p_1,...,p_n\}$ ist eindeutig bestimmt
2. $m = p_1^{a_1} \cdot ... \cdot p_n^{a_n}$ ist bis auf die Reihenfolge ebenfalls eindeutig bestimmt

Def. prime Restklasse :

Falls zu $[a]_m$ ein Inverses (bzgl. Multiplikation) modulo m existiert , so heißt $[a]_m$ prim modulo m .

$Z_m^* = \{[a]_m : ggT(a,m) = 1\}$ = alle primen Restklassen von m

Satz :

$\left|Z_m^*\right| = \Phi(m)$

mit $\Phi(m) = m\left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)..\left(1-\frac{1}{p_n}\right)$

Für Primzahlen gilt $\Phi(m) = m-1$.

Beispiel : $m = 15 = 5 \cdot 3$

$\Phi(m) = 15\left(1-\frac{1}{3}\right)\left(1-\frac{1}{5}\right) = 8$

Stromchiffren

- Erzeugung einer „wilden“ Bitfolge
- XOR-Verknüpfung des Schlüssels mit dem Klartext
- Sicherheit liegt in der Erzeugung der Bitfolge
- Schlüssel kann im Voraus berechnet werden

symmetrische Stromchiffre RC4 (WLAN)

Algorithmus :

- interne Schlüssel
  a) zwei 8-Bit Zahlen i und j
  b) Permutation P der Zahlen $0..255$
- Berechnung des Schlüsselbytes k wiefolgt :
  a) $i = i + 1 \ (\%256)$
  b) $j = j + P_i \ (\%256)$, P_i = i-tes Element in P

c) vertausche P_i und P_j

d) $t = P_i + P_j \ (\%256)$

e) $k = P_t$

$\Rightarrow \quad c_i = k \oplus m_i$

Anfangswerte für i, j, P :

- privater Schlüssel des Benutzers $S_0,...,S_{l-1}$ (l Bytes)
- $i = 0$, $j = 0$, $P_k = k$ für $k = 0,...,255$
- berechne 256 mal :
  1.) $j = j + P_i + S_i \quad (\%256)$
  2.) vertausche P_i und P_j
  3.) $i = i + 1 (\%l)$

Schlüsselraum : $256^2 \cdot 256!$

Ohne Initialisierungsvektor ist RC4 anfällig gegenüber known-plaintext-Angriffe

- Erzeuge einen IV für jede Nachricht neu.
- Hänge den IV an den Benutzerschlüssel an und verändere dadurch die internen Schlüssel.
- Schreibe den IV vor die zu übertragende Nachricht.

Blockchiffren

- Unterteilung des Klartextes in Blöcke gleicher Länge
  Bsp.: Permutationschiffre

ECB Mode (electronic codebook)

- Aufteilung des Klartextes in Blöcke $m_1, m_2, ...$ der Länge n
- Chiffrierung jedes Blockes $\left(c_j = E(m_j)\right)$
- Dechiffrierung durch $m_j = D(c_j)$

Beispiel :

$\Sigma = \{0,1\} \qquad n = 4$

Schlüsselmenge : $4! = 24$

$E_\Pi : \{0,1\}^4 \rightarrow \{0,1\}^4$

$$\Pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix} \qquad \Pi^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 2 & 3 \end{pmatrix}$$

$m = 1011 \quad 0001 \quad 0100 \quad 1010$

$c = 0111 \quad 0010 \quad 1000 \quad 0101$

Nachteile :

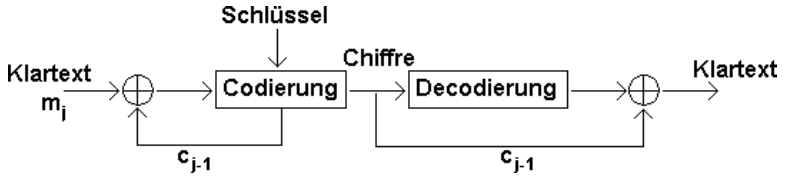
- Gleiche Klartextblöcke werden zu gleichen Chiffreblöcken.
- Angreifer kann Nachricht verändern , indem er einen Chiffretext einfügt , der mit dem gleichen Schlüssel erzeugt wurde.

⇒ **CBC Mode (Cipher Block Chaining Mode)**

Die Verschlüsselung eines Klartextblockes hängt zusätzlich vom vorhergehenden Block ab.
(=kontextabhängige Codierung)
Der erste Block wird mit einem Initialisierungsvektor verschlüsselt.

Vorteile :

- Gleiche Klartextblöcke erzeugen (gewöhnlich) unterschiedliche Chiffreblöcke
- Nachträgliche Veränderung erschwert , da die Decodierung (fast immer) fehlerhaft wird



Codierung :

c_0 = Initialisierungsvektor

$$m_j = c_{j-1} \oplus D(c_j)$$

Korrektheit :

$$c_0 \oplus D(c_1) = c_0 \oplus m_1 \oplus m_0 \quad \Rightarrow m_1$$

Beispiel :

(Voraussetzungen siehe voriges Beispiel)

$$m = 1011 \quad 0001 \quad 0100 \quad 1010$$
$$c_0 = 1010$$
$$c_1 = E(c_0 \oplus m_1) = E(0001) = 0010$$
$$c_2 = E(c_1 \oplus m_2) = E(0011) = 0110$$
$$c_3 = E(c_2 \oplus m_3) = E(0010) = 0100$$
$$c_4 = E(c_3 \oplus m_4) = E(1110) = 1101$$

Decodierung

$$c_0 = 1010$$
$$m_1 = c_0 \oplus D(c_1) = 1010 \oplus 0001 = 1011$$

...usw.

- Bitfehler in c_i bewirken nur falsche Decodierung von c_i und c_{i+1}.
- Bei verschiedenen Initialisierungsvektoren erhält der Empfänger bis auf m_1 die korrekte Nachricht.

Nachteile :

- Effizienzprobleme bei langen Nachrichten
- Empfänger muss warten , bis der Sender alle Chiffretextblöcke erzeugt und versendet hat.

⇒ **CFB Mode (Cipher Feedback Mode)**

Blöcke einer Länge <n werden nicht direkt codiert , sondern durch XOR-entsprechende Schlüsselblöcke verschlüsselt.

- $IV \in \{0,1\}^n$
- $r \in N$ mit $1 \leq r \leq n$
- Aufteilung des Klartextes in Blöcke der Länge r
- Blockfolge $m_1,...,m_n$

Codierung :

$I_1 = IV$, für $1 \le j \le n$ bildet man :

1.) $O_j = E(I_j)$

2.) t_j ist der String der ersten r Bits von O_j

3.) $c_j = m_j \oplus t_j$

4.) $I_{j+1} = \left(2^r I_j + c_j\right) \bmod 2^n$ (Links-Shift um r Bits , anfügen der letzten r Bits von c_j und abschneiden der ersten r Bits)

Decodierung :

$I_1 = IV$

1.) $O_j = E(I_j)$

2.) t_j ist der String der ersten r Bits von O_j

3.) $m_j = c_j \oplus t_j$

4.) $I_{j+1} = \left(2^r I_j + c_j\right) \bmod 2^n$

Bemerkungen zum Ablauf :

- Sender und Empfänger können t_{j+1} bestimmen , sobald sie c_j kennen.
- t_1 kann von Sender und Empfänger gleichzeitig berechnet werden.
- Sender erzeugt und verschickt $c_1 = m_1 \oplus t_1$.
- Sender und Empfänger berechnen simultan $t_2, t_3, \ldots$

Beispiel :

$IV = 1010$, $r = 3$, Schlüssel $\Pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix}$

$m_1 = 101$ $m_2 = 100$ $m_3 = 010$ $m_4 = 100$ $m_5 = 101$

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

Bemerkungen :

- Je kleiner die Blocklänge r , desto häufiger muss $E()$ angewandt werden. (höherer Zeitaufwand)
- Kürzere Chiffretextblöcke werden aber schneller übertragen.
  $\Rightarrow$ Kompromiss nötig
- Fehler in der Übertragung beeinflussen die Decodierung solange , bis der fehlerhafte Chiffreblock aus dem Vektor I herausgeschoben ist.

- CFB-Mode kann nicht bei Public-Key-Verfahren angewandt werden , da beide Seiten den gleichen Schlüssel verwenden.

OFB-Mode (Output Feedback Mode)
Aufteilung des Klartextes in Blöcke der Länge k

Codierung :

$I_1 = IV$, für $1 \le j \le n$ bildet man :

1.) $O_j = E(I_j)$
2.) $t_j =$ String der ersten r Bits von O_j
3.) $c_j = m_j \oplus t_j$
4.) $I_{j+1} = O_j$

Decodierung :

1.) $O_j = E(I_j)$
2.) $t_j =$ String der ersten r Bits von O_j
3.) $m_j = c_j \oplus t_j$
4.) $I_{j+1} = O_j$

Beispiel :

$m_1 = 101 \qquad m_2 = 100 \qquad m_3 = 010 \qquad m_4 = 100 \qquad m_5 = 101$

$\Pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix} \qquad IV = 1010$

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

Affine Blockchiffren

Eine Blockchiffre der Länge n und Klartext- / Chiffreraum Z_m heißt affin-linear , falls die Chiffrier-Fkt. $E : (Z_m)^n \to (Z_m)^n$ die Form $v \mapsto A \cdot v + b \pmod m$ hat.

Sie heißt linear , falls $b = \vec{0}$.

Dabei ist $A \in (Z_m)^{n \times n}$ eine in Z_m invertierbare $n \times n$ - Matrix und $b \in (Z_m)^n$ ein Spaltenvektor.

Desweiteren gilt $ggT(\det A, m) = 1$.

Die Dechiffrier-Fkt. lautet $D : (Z_m)^n \to (Z_m)^n$ mit $v \mapsto A^{-1}(v - b) \pmod m$.

Kryptanalyse affiner Blockchiffren :
Kann mit known-plaintext-Angriff geknackt werden. (siehe Übung 2)

Verwende Klartextworte $w_i \in (Z_m)^n$ und zugehörige Chiffre $c_i = Aw_i + b$.

Betrachte $c_i - c_0 \equiv A(w_i - w_0) \bmod m \qquad i = 0,...,n$ und bilde die Matrizen

$W = (w_1 - w_0 \quad w_2 - w_0 \quad ... \quad w_n - w_0)$ und

$C = (c_1 - c_0 \quad c_2 - c_0 \quad ... \quad c_n - c_0)$

Interpretiere die n Gleichungen mit diesen Bezeichnungen , dann gilt :

$C \equiv A \cdot W (\text{mod } m)$, d.h. $A \equiv W^{-1} \cdot C (\text{mod } m)$

und $b \equiv c_0 - A \cdot w_0 (\text{mod } m)$

Beispiel : Vigenère-Chiffre (polyalphabetische Verschlüsselung) – Blaise de Vigenère (1523-1596)

$A = E_n$, $b \neq (0,...,0)$, $v \mapsto v + b \ (\text{mod } m)$

Codierung :

KLARTEXTDERGEHEIMBLEIBENSOLL
+PASSWORTPASSWORTPASSWORTPAS
XMRGNTHBSDUIFKHGSFDKFUBGULK

Ansätze zur Dechiffrierung :

- Ausschluss einer monoalphabetischen Substitutionschiffre durch Häufigkeitsanalyse.
- Bestimmung der Länge des Schlüsselwortes.
  (Jeder $a \cdot x$-te Buchstabe im Text gehört zum selben monoalphabetischen Code.)

Def. Koinzidenzindex :

Zähle in der Chiffre :

$n_1 = \#A$

$n_2 = \#B$

$n_3 = \#C$

...

Betrachte die Anzahl der Paare , bei denen beide Buchstaben ‚A' sind (nicht notwendigerweise direkt aufeinanderfolgende Buchstaben.)

$= \frac{n_1(n_1 - 1)}{2}$

$\Rightarrow$ Anzahl der Paare gleicher Buchstaben = $\sum_{i=1}^{26} \frac{n_i(n_i - 1)}{2}$

$\Rightarrow$ Chance für die Auswahl eines Paares gleicher Buchstaben = $\frac{\sum_{i=1}^{26} \frac{n_i(n_i - 1)}{2}}{\frac{n(n-1)}{2}} = \frac{\sum_{i=1}^{26} n_i(n_i - 1)}{n(n-1)} = C_I$

Andere Herangehensweise :

$p_i =$ Wahrscheinlichkeit des Vorkommens des i-ten Buchstaben im deutschen Text

$\Rightarrow$ Wahrscheinlichkeit für zufälliges Herausgreifen eines Paares (a,a) $\approx p_i^2$

$\Rightarrow$ Wahrscheinlichkeit für zufälliges Herausgreifen eines Paares gleicher Buchstaben

$C_I = \sum_{i=1}^{26} p_i^2 \approx 0.0762$ (deutscher Text)

≈ 0.0661 (englischer Text)

≈ 0.0385 (zufälliger Text) $p_i = \frac{1}{26} \Rightarrow \sum_{i=1}^{26} \frac{1}{p_i^2} = \frac{1}{26} \approx 0.0385$

Bestimmung der Schlüssellänge einer Vigenère-Chiffre

Friedmann (1925)

$C_I \approx \frac{0.0377n}{l(n-1)} + \frac{0.0385n - 0.0762}{n-1}$

d.h.

$$l \approx \frac{0.0377n}{C_I(n-1) - 0.0385n + 0.0762}$$

Kasiski (1863)

- Falls der Geheimtext einigermaßen zufällig ist , dann erscheint die Wiederholung einer wenigstens 3 Zeichen langen Kette im Geheimtext als sehr unwahrscheinlich.
- Im Klartext hingegen ist eine derartige Wiederholung häufig.
- Falls zwei gleiche Buchstabenfolgen im Klartext einen Abstand haben , der ein Vielfaches der Periodenlänge ist , ergibt sich der gleiche Geheimtext.
- Analyse der Chiffre auf sich wiederholende Wortgruppen mit 3 oder mehr Buchstaben und Berechnung der Abstände.
  z.B. 24,54,18,29,66 (29 Ausrutscher , 6=Teiler)
  d.h. Vermutung : Schlüssellänge 6 (oder 3 , 2)

Autokorrelation (effizienter und genauer)

Angriff gegen Vigenère-Chiffre :

- Bestimmung der Schlüssellänge.
- Decodierung der monoalphabetischen Chiffren durch Häufigkeitsanalyse.

Automatische Häufigkeitsanalyse :

g_i = relative Häufigkeit des i-ten Buchstaben im Geheimtext

p_i = relative Häufigkeit des i-ten Buchstaben im Klartext

Minimierung von $\sum_{i=1}^{26} (g_i - p_i)^2$ (D.h. Minimierung der Fläche zwischen den beiden Graphen.)

Def. Kryptosystem :

(P, C, K, E, D) bestehend aus den Mengen :

1.) P - Klartextraum
2.) C - Chiffretextraum
3.) K - Schlüsselraum
4.) $E = \{E_k : k \in K\}$ $E_k : P \to C$ Verschlüsselungsfunktionen
5.) $D = \{D_k : k \in K\}$ $D_k : C \to P$ mit : $Vs \in K\ \exists k \in K\ D_k(E_s(p)) = p \quad (\forall p \in P)$

Kryptanalytische Angriffe :

- known-ciphertext-Angriff
- known-plaintext-Angriff (kleiner Klartext ist bekannt , z.B. Anrede,Gruß)
- chosen-plaintext-Angriff (verschiedene Chiffretexte bekannt und mindestens ein Klartext)

- Jede monoalphabetische Chiffrierung einer natürlichen Sprache kann (aufgrund ihrer Redundanz) leicht geknackt werden.

Statistische Analyse einer Sprache

Ermöglicht Dechiffrierung einfacher Permutationschiffren.

Def. Informationsgehalt :

- Funktion von $p(x_i) = p_i$ $(x_i \in \Sigma)$
- $H_0 : [0,1] \to R$, stetig und streng monoton fallend
- $H_0(p_i\, p_j) = H_0(p_i) + H_0(p_j)$, für $i \neq j$ (Unabhängigkeit der beiden Ereignisse)
- $H_0(0.5) = 1$ (Ausgang eines Experiments mit zwei gleichwahrscheinlichen Ereignissen hat den Informationsgehalt 1.)

$\Rightarrow$ eindeutig bestimmte Funktion

$$H_0(p_i) = -\log_2 p_i$$

Def. Entropie :

Erwartungswert der Zufallsvariablen eines endlichen Alphabetes.

$$H = -\sum p_i H_0(p_i) = -\sum p_i \log_2 p_i$$

Bsp 1 : 26 Buchstaben mit Gleichverteilung : $H = \log_2 26 \approx 4.700$

Bsp 2 : Entropie der deutschen Sprache : $H \approx 4.063$

$\Rightarrow$ Buchstabenhäufigkeiten lassen Rückschlüsse über Sprache zu.

Verbesserung einfacher Substitutionschiffren / Permutationschiffren : Homophone Chiffrierung

Die Chiffrierfunktion ordnet jedem Buchstaben nicht nur ein Zeichen , sondern eine Menge von Zeichen zu. (Bei der Codierung wird dann unter diesen eines zufällig ausgewählt. - Bsp.: a...z $\longrightarrow$ 00...99)

Anforderungen :

- Dechiffrierung muss eindeutig sein , d.h. die Mengen zu verschiedenen Klartextbuchstaben sind disjunkt.
- Die Anzahl der Geheimtextzeichen , die zu einem Klartextzeichen gehören , entspricht der Häufigkeit des Buchstabens.

Angriff :

- Das Buchstabenäquivalent von z.B. ‚c'=28 besitzt nur bestimmte Nachfolger , also z.B. 07,24,23,41,89 (die Geheimtextäquivalenten von ‚h' und ‚k')
- Häufigkeitsanalyse der Paare.

Einschub Pseudozufallszahlen

Def. : Eine Folge $\{s_i\}$ heißt periodisch mit Periode p , falls p die kleinste Zahl $p \in N \setminus \{0\}$ ist mit

$$s_{i+p} = s_i \qquad \forall i \geq 1 .$$

Anmerkung :

$\{s_i\} \in \{0,1\}$ werden als Binärstrings interpretiert mit :

Blöcke der Länge k : $0\underbrace{1....1}_{k \text{ mal}}0$ Lücken der Länge k : $1\underbrace{0....0}_{k \text{ mal}}1$

Def. Autokorrelation :

$k \in N$, $\{s_i\}$ mit Periode p

$$AC(k) = \frac{1}{p}\big(A(k) - D(k)\big)$$

, wobei :

$A(k) = \#\{1 \le i \le p \quad : \quad s_i = s_{i+k}\}$ (Anzahl der Übereinstimmungen)

$D(k) = \#\{1 \le i \le p \quad : \quad s_i \neq s_{i+k}\}$ (Anzahl der Nicht-Übereinstimmungen)

Zufallspostulate nach Golumb :

1.) Anzahl der Nullen und Einsen in einer Periode p sind möglichst gleich.

2.) Die Hälfte der Elemente einer Periode sind von der Länge 1 , ein Viertel von Länge 2 , ... usw. Die Hälfte der Elemente sind Blöcke , die andere Hälfte Lücken.

3.) $\forall k \neq p \quad AC(k) = const$

Folgerung :

Sei $\{s_i\}_{i \ge 1}$ eine Folge , die obige Postulate erfüllt , dann gilt :

$AC(k) = -\frac{1}{p}$, p ungerade

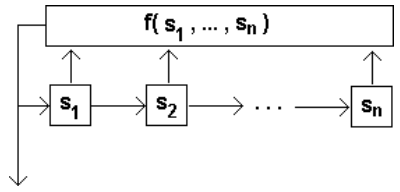
kryptographische Anforderungen :

1.) Periode p von $\{s_i\}_{i \ge 1}$ soll so lang wie möglich sein

2.) $\{s_i\}_{i \ge 1}$ soll leicht generierbar sein

3.) Ein known-plaintext-Angriff sollte nicht die gesamte Folge $\{s_i\} i \ge 1$ liefern.

Lineare Schieberegister



Definition :

Ein LSR ist eine Anordnung von Registern $s_1,...,s_n$ (je 1 Bit) mit folgender Funktionsweise :

- Zeitpunkt $t = 0$: $S(0) = (s_1(0),..., s_n(0))$
- Zeitpunkt t : $S(t) = (s_1(t),..., s_n(t))$
- Zeitpunkt $t+1$: $S(t+1) = (s_1(t+1),..., s_n(t+1))$

mit $s_i(t+1) = s_{i-1}(t)$, $2 \le i \le n$

und $s_1(t+1) = c_1 s_1(t) + ... + c_n s_n(t)$

(D.h. die einzelnen Stellen/Register nach rechts shiften und die vorderste Stelle neu berechnen.)

Beispiel :

	c_1	c_2	c_3	c_4
t	0	1	0	1
0	1	0	0	0
1	**0**	1	0	0
2	**1**	0	1	0
3	**0**	1	0	1
4	**0**	0	1	0
5	**0**	0	0	1
6	**1**	0	0	0

Im Detail :

$$S_1(0) = 1 \qquad \text{(IV)}$$

$$S_1(1) = c_1 S_1(0) + c_2 S_2(0) + c_3 S_3(0) c_4 S_4(0) = 0 + 0 + 0 + 0 = 0$$

$$S_1(2) = c_1 S_1(1) + c_2 S_2(1) + c_3 S_3(1) c_4 S_4(1) = 0 \cdot 0 + 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 0 = 1$$

... usw.

1. Spalte = Ausgabefolge
Periodenlänge = 6

Bemerkung: Ein LSR mit n Registern hat höchstens $2^n - 1$ Einträge
(Periode der Ausgabefolge $\{s_i\}$ ist ebenfalls $\leq 2^n - 1$.)

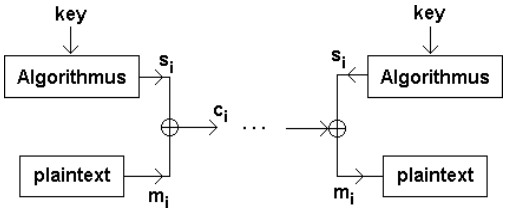
$$S(t+1) = \underbrace{\begin{pmatrix} c_1 & c_2 & \cdots & c_{n-1} & c_n \\ 1 & 0 & & 0 & 0 \\ 0 & 1 & & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & 1 & 0 \end{pmatrix}}_{:=C} \cdot S(t)$$

o.B.d.A. sei $c_n \neq 0$, andernfalls kommt man mit einem LSR mit n-1 Registern aus

Falls $c_n \neq 0$, so ist $\det C \neq 0$ (Entwicklungssatz nach der letzten Spalte)

$\Rightarrow$ Nullvektor kommt nicht als Registerbelegung vor.

<u>Anwendung für Kryptosysteme</u>



- billiges Verfahren zur Chiffrierung
- einfache Implementierung

Analyse :

Voraussetzung : LSR mit Periode $2^n - 1$
Golumb Kriterien :

1.) Ausgabefolge der Länge $2^n - 1$ enthält $2^{n-1} - 1$ Nullen und 2^{n-1} Einsen
(Zustand des LSR zum Zeitpunkt t entspricht einer der Zahlen $[1, 2^n - 1]$ in Binärdarstellung.)

2.) Für $1 \leq t \leq n-2$ enthält die Ausgabefolge der Länge $2^n - 1$ genau 2^{n-t-2} Blöcke der Länge t und

ebenso viele Lücken der Länge t.
(Beim Durchlauf wird ein Block der Länge t $(01...10)$ erreicht , falls ein Registerzustand $01....10x_1...x_{n-t-2}$ existiert. Es gibt 2^{n-t-2} solcher Zustände und jeder kommt vor , da es $2^n - 1$ verschiedene Registerbelegungen gibt.)

3.) $AC(k) = -\frac{1}{2^n - 1}$

$AC(k)$ = # der Übereinstimmungen von $\{s_i\}$ und $\{s_{i+k}\}$

= # der Nullen in $\{s_i + s_{i+k}\}_{i \geq 1}$

= $2^{n-1} - 1$ wegen maximaler Periodenlänge

$AC(k) = \frac{1}{p}\left(A(k) - D(k)\right)$

$= \frac{2A(k) - p}{p}$ $\qquad p = 2^n - 1$

$= -\frac{1}{2^n - 1}$

$\Rightarrow$ Die Ausgabefolgen der LSR mit maximaler Periode erfüllen die Golumbschen Zufallspostulate.

kryptographische Analyse :

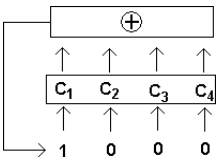
C1 : Periode $2^n - 1$ ist hinreichend lang
C2 : LSR sind sehr leicht zu implementieren
C3 : LSR sind kryptographisch äußerst unsicher

Beispiel :

LSR mit 4 Registern ; known-plaintext Angriff mit 8 Klartext-Bits und der zugehörigen Chiffre 00011110 (die $\oplus$-Addition entspricht der Codierungsvorschrift)

gesucht : Registerbelegung C_1, C_2, C_3, C_4

Vorgehensweise :
1. Initialisierung S(0)=1000
2. Rückkopplung



Takt t	$S_1(t)$	$S_2(t)$	$S_3(t)$	$S_4(t)$
0	1	0	0	0
1	C_1	1	0	0
2	$C_1 \oplus C_2$	C_1	1	0

3	$C_1 \oplus C_3$	$C_1 \oplus C_2$	C_1	1
4	$C_1 \oplus C_2 \oplus C_4 \oplus C_1C_2$	$C_1 \oplus C_3$	$C_1 \oplus C_2$	C_1

Im Detail :

$S_1(0) = 1$ (IV)

$S_1(1) = C_1 \cdot S_1(0) \oplus C_2 \cdot S_2(0) \oplus C_3 \cdot S_3(0) \oplus C_4 \cdot S_4(0) = C_1$

$S_1(2) = C_1C_1 \oplus C_2 \cdot 1 = C_1 \oplus C_2$

$S_1(3) = C_1(C_1 \oplus C_2) \oplus C_2C_1 \oplus C_3 \cdot 1 = C_1 \oplus C_1C_2 \oplus C_1C_2 \oplus C_3 = C_1 \oplus C_3$

$$\begin{aligned} S_1(4) &= C_1(C_1 \oplus C_3) + C_2(C_1 \oplus C_2) + C_3C_1 + C_4 \\ &= C_1 \oplus C_1C_3 \oplus C_1C_2 \oplus C_2 \oplus C_1C_3 \oplus C_4 \\ &= C_1 \oplus C_2 \oplus C_4 \oplus C_1C_2 \end{aligned}$$

Im 4. Takt erhält der Angreifer die restlichen 4 Bits der Folge

$C_1 \oplus C_2 \oplus C_4 \oplus C_1C_2$	= 0
$C_1 \oplus C_3$	= 1
$C_1 \oplus C_2$	= 1
C_1	= 1

Lösung des Gleichungssystems :

$C_1 = 1$, $C_2 = 0$, $C_3 = 0$, $C_4 = 0$

(Lösung : LSR mit Periode 15)

Satz :

Ein LSR der Länge n und maximaler Periode ist entschlüsselbar , wenn $2n$ aufeinanderfolgende Klartext-Bits und die zugehörige Chiffre-Bits bekannt sind.

Beweis :

Gegeben : LSR mit n Registern und Rückkopplungskoeffizienten $c_1,...,c_n$ sowie $2n$ Klartext- und Chiffre-Bits

Hieraus ergeben sich die ersten n Bits nach $\oplus$ Addition als Initialisierungsvektor.
Die zweiten n Bits bilden den Vektor nach dem Durchlauf.
$c_1,...,c_n$ aus den Gleichungen bestimmen

- Verbesserung der LSR möglich durch nichtlineare Rückkopplung.

$p(x) = 1 + c_1x + ... + c_nx^n \quad \in Z_2[x] \quad , \quad c_n \neq 0$ (= charakteristisches Polynom des LSR)

Satz :

Das LSR hat genau dann maximale Periode $2^n - 1$, fall $p(x)$ folgende zwei Eigenschaften erfüllt :

1.) $p(x)$ ist irreduzibel (hat keinen echten Teiler)

2.) $p(x)$ ist nicht Teiler von $x^d + 1$ für alle $d < 2^n - 1$

Beispiel :

$$p(x) = 1 + x + x^4 \quad \in Z_2[x]$$

zu 1.) :

$p(x)$ spaltet keinen linearen Faktor ab , denn 0 , 1 sind keine Nullstellen.

$$(x^2 + bx + a)(x^2 + dx + c) = x^4 + (b+c)x^3 + (a+d+bc)x^2 + (ac+bc)x + ac$$
$$= x^4 + x + 1$$

$\Rightarrow a = c = 1$

$\Rightarrow d + b = 1$ und $d + b = 0$ $\Rightarrow$ Widerspruch

zu 2.) :

berechne Potenzen

$$x^4 \equiv x + 1 \mod p(x)$$
$$x^5 \equiv x^2 + x \mod p(x)$$
$$x^6 \equiv x^3 + x^2 \mod p(x)$$
$$x^7 \equiv x^2 + x + 1 \mod p(x)$$
...
$$x^{15} \equiv 1 \mod p(x)$$

Einschub Wahrscheinlichkeitsrechnung :

Sei $S \neq \varnothing$ Ereignismenge z.B. Würfel : $\{1,2,3,4,5,6\}$

Definition :

Ein **Ereignis** über S ist eine Teilmenge von S .

Definition : Wahrscheinlichkeitsverteilung

$p_r : P(S) \to R$ mit den Eigenschaften :

1.) $p_r(A) \geq 0$, $\forall A \in P(S)$

2.) $p_r(S) = 1$

3.) $p_r(A \cup B) = p_r(A) + p_r(B)$, $A, B \in P(S)$ und $A \cap B = \varnothing$

Definition : bedingte Wahrscheinlichkeit

$$p_r(A \mid B) = \frac{p_r(A \cap B)}{p_r(B)}$$

Zwei Ereignisse A, B sind **unabhängig** , falls $p_r(A \cap B) = p_r(A) \cdot p_r(B)$

Satz von Bayes :

$$p_r(A) \cdot p_r(B \mid A) = p_r(B) \cdot p_r(A \mid B)$$

Sicherheit von Kryptosystemen

Definition : perfekte Sicherheit

Ein Kryptosystem heißt perfekt sicher , falls es unmöglich ist , aus einer abgefangenen Nachricht den Inhalt zu ermitteln.

Sei P - Klartextraum
C - Ciphertextraum
K - Schlüsselraum
p_P, p_K - Wahrscheinlichkeitsverteilungen
p_r - Wahrscheinlichkeitsverteilung auf $P \times K$, $p_r(P,K) := p_P(P) \cdot p_K(K)$

Für $c \in C$ betrachte das Ereignis $\left\{ (p,k) \in P \times K : E_K(\underset{\text{plaintext}}{p}) = \underset{\text{Chiffre}}{c} \right\}$

Ein Kryptosystem heißt somit perfekt sicher , falls die Ereignisse , dass ein bestimmter Ciphertext c und ein bestimmter Klartext p vorliegen , unabhängig sind.

D.h. $p_r(p \mid c) = p_r(c) \qquad \forall p \in P, c \in C$

Beispiel :

$P = \{0,1\} \qquad K = \{A, B\} \qquad C = \{a, b\}$

$p_r(0) = 1/4 \quad p_r(1) = 3/4 \quad p_r(A) = 1/4 \quad p_r(B) = 3/4$

Wahrscheinlichkeit , dass das Zeichen 1 auftritt und mit B verschlüsselt wird : $p_r(1) \cdot p_r(B) = \frac{9}{16}$

... usw.

Verschlüsselung :

$E_A(0) = a$, $E_A(1) = b$, $E_B(0) = b$, $E_B(1) = a$

Wahrscheinlichkeit für Ciphertext a : $p_r(a) = p_r(0 \mid A) + p_r(1 \mid B) = \frac{1}{16} + \frac{9}{16} = \frac{5}{8}$

Wahrscheinlichkeit für Ciphertext b : $p_r(b) = p_r(1 \mid A) + p_r(0 \mid B) = \frac{3}{16} + \frac{3}{16} = \frac{3}{8}$

$$p_r(0 \mid a) = \frac{p_r(0 \cap a)}{p_r(a)} = \frac{1/16}{5/8} = \frac{1}{10}$$

$$p_r(0 \mid b) = \frac{3/16}{3/8} = \frac{1}{2}$$

$$p_r(1 \mid a) = \frac{9/16}{5/8} = \frac{9}{10}$$

$$p_r(1 \mid b) = \frac{3/16}{3/8} = \frac{1}{2}$$

$\Rightarrow$ Das Kryptosystem ist nicht perfekt sicher , da bei Betrachtung von a relativ sicher ist , dass 1 im Klartext vorkommt.

Satz von Shannon :

Sei $|P| = |C| = |K|$ und sei $p_r(p) > 0$ für jeden Klartext $p \in P$.

Dann ist das Kryptosystem genau dann sicher , wenn die Wahrscheinlichkeitsverteilung auf dem Schlüsselraum die Gleichverteilung ist und wenn es für jede Chiffre c genau einen Schlüssel $k \in K$ mit $E_k(p) = c$ gibt.

Bsp 1 :

Der Satz von Shannon auf das obige Beispiel angewandt ergibt :
perfekt , falls $P(A) = P(B) = 1/2$

Bsp 2 :

one-time-pad

$P = C = K = \{0,1\}^n$ $\quad E_k : \{0,1\}^n \to \{0,1\}^n$, d.h. $\quad p \to p \oplus k$

Der Algorithmus verschlüsselt den Klartext $p \in \{0,1\}^n$ mit einem zufälligen Schlüssel $k \in \{0,1\}^n$, welcher der Gleichverteilung entspricht.
Die Verschlüsselung ist perfekt sicher , da der Schlüsselraum gleichverteilt ist und $\forall p \in P, c \in C$ existiert genau ein $k \in K$ mit $c = p \oplus k$, nämlich $k = c \oplus p$.

Kapitel 3

Symmetrische Kryptosysteme

bisher : relativ leicht zu knackende Kryptosysteme
nun : Mechanisierung und Elektronisierung der Kryptoverfahren

3.1 Mechanische und Elektromechanische Chiffriersysteme

Chiffrierzylinder : Scheiben , auf deren Rändern permutierte Alphabete eingraviert sind.
Keine ausreichende Sicherheit , da Häufigkeitsanalysen beim Dechiffrieren helfen.
(Friedmann , Kasiski)

elektromechanische Verbesserung : Rotormaschinen (Verwendung mehrerer Scheiben durch Schleifkontakte)
Bewertung :

- feststehender Algorithmus der Scheibenbelegung
- variable Anfangsstellungen
- variable Anordnung der Scheiben

Bei z.B. 3 Scheiben ($26^2 : 26 : 1$) ist die Periode $26^3 = 17576$.
(Nur relative Erschwerung der Vigenére-Attacken.)

Beispiel für Rotormaschine : ENIGMA

Fehler : Einbau einer Umkehrwalze (Reflektor) (nochmaliger Durchlauf der Walzen)
Reflektor : Walze , die nur an einer Seite Kontakte hat , welche untereinander verbunden sind.
$\hat{=}$ zusätzlicher Substitution Π mit $i \neq \Pi(i)$
(Vermeidung von Kurzschlüssen)

Ergebnis :
- Verzicht auf 4. Walze durch Benutzung des Reflektors
- zusätzliches Steckbrett (Schablone) zum Vertauschen der Buchstaben

Schlüssel der ENIGMA :
- Anzahl und Anordnung der Rotoren
- Auswahl von 3 Rotoren aus 5 Möglichen
- Ausgangsstellung
- Beschreibung des Steckbrettes

26^3 Rotorstellungen
Auswahl von 10 verschiedenen Rotorkombinationen (3 aus 5) ,
3! viele Reihenfolgen

$$\Rightarrow \frac{26!}{13!} \cdot 2^{13} \approx 8 \cdot 10^{18} \text{ Schlüssel}$$

Kryptanalyse der ENIGMA :
Polnischer Zoll fängt 1927 ENIGMA ab (Kenntnis des Algorithmus).
Feststellung : Nie wird ein Buchstabe in sich selbst überführt.

$\Rightarrow$ negative Mustersuche
Annahme : Text enthält Klartext
Bsp : „OberkommandoWehrmacht“ = 21 Buchstaben
$1 - \left(1 - 1/26\right)^{21} \approx 0.561$
= Wahrscheinlichkeit für Übereinstimmung von Geheimtext und Klartext an einer Position.

Nun Entlangschieben des Wortes über den Geheimtext ,bei Übereinstimmung nicht bearbeiten (50% der Fälle)

moderne Variante : UNIX-Befehl crypt

3.2 DES (Data Encryption Standard)

Def. Konfusion :
Verschleierung des Zusammenhangs von Klartext und Chiffre.

Def. Diffusion :
Verteilung der im Klartext enthaltenen Information.

Def. Lawinen-Effekt :
Geringe Änderungen im Klartext bewirken große Änderungen in der Chiffre.

DES ist vom Typ der Feistel-Chiffren.

Feistel-Chiffren

Blockchiffren mit Blocklänge t und Alphabet $\Sigma = \{0,1\}^*$

f_k - Verschlüsselungsfunktion zum Schlüssel $k \in K$

Festlegung einer Rundenzahl $r \geq 1$

Aus $k \in K$ wird eine Folge von Rundenschlüsseln erzeugt.

Funktionsweise :

p : Klartext der Länge $2t$

$p = (L_0, R_0)$ (linker und rechter Teil)

Konstruktion nach folgender Vorschrift :

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

$\Rightarrow$ Folge $\{(L_i, R_i)\}_{0 \leq i \leq r}$

Festlegung : $E_k(L_0, R_0) = (R_r, L_r)$

Sicherheit hängt von der Sicherheit der einzelnen Blöcke ab , welche durch iterierte Verwendung gesteigert wird.

Dechiffrierung :

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

Verwendung der Schlüsselfolgen $\{k_r, k_{r-1}, ..., k_1\}$

Aus (R_r, L_r) erhält man (R_0, L_0) und durch Vertauschen den Klartext (L_0, R_0).

Verschlüsselung und Dechiffrierung benutzen denselben Algorithmus.

DES im Detail :

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

$K = \{0,1\}^{64}$

Aufteilung in 8 Bytes mit dem letzten Bit als Parity-Bit (Fehlerkorrektur) , d.h. die ersten 7 Bit legen das 8. Bit fest.

$\Rightarrow 2^{56}$ gültige DES-Schlüssel

1. Schritt :

Anwendung der IP (initiale Permutation – fest gewählt und vom Schlüssel unabhängig) auf Bitvektoren der Länge 64

$p = p_1 \ldots p_{64}$

$IP(p) = p_{58} p_{50} \ldots p_7$ (siehe Zettel)

2. Schritt :

Anwendung einer 16 Runden Feistel-Chiffre

3. Schritt :

$C = IP^{-1}(R_{16}, L_{16})$

interne Blockchiffre (insbesondere S-Boxen) siehe Zettel

$k \in \{0,1\}^{48}$, $f_k : \{0,1\}^{32} \rightarrow \{0,1\}^{32}$

Die rechte Seite R (32 Bit) wird expandiert auf 48 Bit durch Anwenden der

Expansionsfunktion E :

$R = (R_1 ... R_{32})$

$E(R) = R_{32} R_1 ... R_{32} R_1$ (siehe Zettel)

$E(R) \oplus k = B_1 ... B_8$ (8 Blöcke der Länge 6)

S-Boxen $S_i : \{0,1\}^6 \rightarrow \{0,1\}^4$ (siehe Zettel , i=1,...,8)

$C_i = S_i(B_i)_{i=1,...,8}$

$C = C_1 ... C_8$ (32 Bit)

zu den S-Boxen :

$b_1 b_6$ Zeilenindex , $b_2 b_3 b_4 b_5$ Spaltenindex

Der Eintrag $(b_1 b_6 b_2 b_3 b_4 b_5)$ in S_i wird binär dargestellt und mit 0 aufgefüllt , so daß die Länge 4 entsteht.

Beispiel :

$S_1(001011)$, 01 = 1. Zeile , 0101 = 5. Spalte $\Rightarrow$ Eintrag = 2

Danach permutieren mit Permutation P .

Berechnung der Rundenschlüssel k_i , i=1,...,16 der Länge 48

Funktion $v_i = \begin{cases} 1 & für \quad i \in \{1,2,9,6\} \\ 2 & sonst \end{cases}$, i=1,...,16

2 Funktionen $PC1$, $PC2$ zur Schlüsselgenerierung

$PC1 : \{0,1\}^{64} \rightarrow \{0,1\}^{28} \times \{0,1\}^{28}$

$PC2 : \{0,1\}^{28} \times \{0,1\}^{28} \rightarrow \{0,1\}^{48}$

Vorgehensweise :

1. Setze $(C_0, D_0) = PC1(k)$
2. Für i=1,...,17 berechne die k_i wiefolgt :

   $C_i :=$ zirkulärer Linksshift von C_{i-1} um v_i Stellen

   $D_i :=$ zirkulärer Linksshift von D_{i-1} um v_i Stellen

   $k_i := PC2(C_i, D_i)$

Dechiffrierung :

Wende DES mit umgekehrter Schlüsselfolge an , wobei IP und IP^{-1} vertauscht werden

Beispiel : siehe Zettel

DES – Zusammenfassung :

- hardwarefreundlicher Algorithmus (XOR-Additionen , bitweise Verschiebung , Anwendung von Permutationen)
- Diffusion durch Erweiterungspermutation und P-Boxen
  (P-Boxen realisieren Durchlauf durch verschiedene S-Boxen.)
- Die S-Boxen bewirken eine Nichtlinearität der Verschlüsselung $(S(a \oplus b) \neq S(a) \oplus S(b))$
  und erschweren die Kryptanalyse.
- Rotation und Kompression bei Erzeugung des Rundenschlüssels bewirken , dass geringe Änderungen im Klartext grosse Änderungen in der Chiffre erzeugen.
- Ausgangspermutation ist kryptologisch uninteressant.

Rolle der S-Boxen :
- monatelange Rechenzeit nötig , um die S-Boxen zu entwickeln
- permanentes Misstrauen den S-Boxen gegenüber (vermutete Hintertür)

Sicherheit des DES :

Aussichtsreichste Verfahren ist bislang der brute-force-Angriff , 2^{56} Schlüssel
„deep crack“ 1998 : Knacken einer DES-Verschlüsselung auf einem PC in ca. 4,5 Tagen

These : DES-Verschlüsselung nicht anwenden , wenn der Gegner 6-stellige Eurobeträge einsetzt.

DES-Verbesserungen :
DES ist nicht abgeschlossen bezüglich der Hintereinanderausführung , d.h. man erhält neuen Algorithmus bei Mehrfachverschlüsselung.

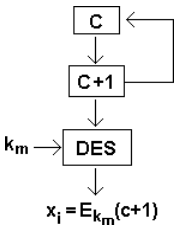
$\Rightarrow$
Triple-DES :
Benutze DES 3 mal mit zwei Schlüsseln k, k' : $DES_k\left(DES_{k'}^{-1}\left(DES_k(m)\right)\right)$

UNIX-Passwortspeicherung mittels DES (/etc/passwd)
„crack“ – Bestandteil der Systemverwaltung
(Leistet brute-force-Angriffe und teilt mit ob ein unsicheres Passwort vorliegt.)
Erweiterung von „crack“ : „satan“ (Aufspüren von Sicherheitslücken im System)

3.3 Kryptographisch generierte Zufallszahlen

DES kann zur Generierung von Zufallszahlen verwendet werden.
(Erzeugen von Bitfolgen , welche die gleichen statistischen Eigenschaften wie Zufallsbits haben und nicht reproduzierbar sind.)

zyklische Verschlüsselung : Zähler C mit Länge $N \quad \left(= 2^{56}\right)$



Eingabe :
Zwei pseudo-zufällige Eingänge steuern den Generator (z.B. 64 Bit Abbild von Datum & Uhrzeit)

DES-Betriebsart OFB (Pseudo-Zufallsgenerator nach ANSI X9 17)

3.4 AES (Advanced Encryption Standard (=Rijndael))

Ziele :
- keine Lizenzgebühren
- effiziente Hardware- und Software-Implementierung
- Resistenz gegen bekannte kryptanalytische Angriffe
- keine Abhängigkeit mehr von der „Mystik“ der S-Boxen

Grobstruktur des Algorithmus

Symmetrische Blockchiffre in mehreren Runden , bestehend aus Substitutionen , Permutationen und Schlüsseladditionen.

Schlüssellänge / Blocklänge des Textes	128	192	256
128	10	12	14
192	12	12	14
256	14	14	14

Anordnung der Bytes durch zwei Varianten :

1.) Bytefolgen der Länge 16 (128 Bit) , 24 (192 Bit) oder 32 (256 Bit).

$s = a^{(0)}...a^{(15)}$ - Bytefolge der Länge 16

2.) Byte-Matrix (wird hier verwendet) mit 4 Zeilen und 4 , 6 oder 8 Spalten

$$s = \begin{pmatrix} a^{(0)} & a^{(4)} & \cdots \\ a^{(1)} & a^{(5)} & \cdots \\ a^{(2)} & a^{(6)} & \cdots \\ a^{(3)} & a^{(7)} & \cdots \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & \cdots \\ a_{10} & a_{11} & \cdots \\ a_{20} & a_{21} & \cdots \\ a_{30} & a_{31} & \cdots \end{pmatrix}$$

Vereinbarungen und Bezeichnungen

M, m_i : Klartextblock M mit Teilblöcken $m_0, m_1, ...$

C, c_i : Chiffreblock C mit Teilblöcken $c_0, c_1, ...$

$\tilde{n}_k$: Schlüssellänge

$\tilde{n}_b$: Blocklänge des Klar- bzw. Geheimtextes $(\tilde{n}_k = \tilde{n}_b = 256)$

n_b : Spaltenzahl der Byte-Matrix $(n_b = \tilde{n}_b / 32)$

n_r : Anzahl der Runden

i : aktuelle Runde

s_i : Eingabeblock der i-ten Runde

s_{i+1} : Ausgabeblock der i-ten Runde

k_i : Rundenschlüssel (wird aus dem AES-Schlüssel k abgeleitet und ist $\tilde{n}_b$ Bits groß)

c : Ergebnis-Chiffre

Grundoperationen :

Byte-Sub : Jedes Byte des Eingabeblocks wird einer 8-Bit Substitution unterzogen.
Row-Shift : Zyklisches Verschieben bestimmter Bytes (Permutation)
Column-Mix : Jeweils 4 Byte werden einer 32 Bit Substitution unterzogen
Key-Add : XOR-Addition eines Blocks mit dem Rundenschlüssel

Hilfsbezeichnungen :

$s_i = s_{i\,a}$: Eingabeblock der i-ten Runde

$s_{i\,b}$: Ergebnis nach Byte-Sub

$s_{i\,c}$: Ergebnis nach Row-Shift

$s_{i\,d}$: Ergebnis nach Column-Shift

$s_{i+1} = s_{i+1\,a}$: Ausgabeblock der i-ten Runde

Hinweise :

Vor den ersten 4 Schritten der ersten Runde wird der Klartextblock m mit k_0 XOR-addiert.

In der letzten Runde entfällt der Column-Mix.

Die Codierung der Bytes für AES $a = a_7 a_6 ... a_0$ erfolgt je nach Kontext in einer der 3 folgenden Möglichkeiten :

1.) Bitstring

$a_i \in \{0,1\}$, z.B. $a = 10010111$

2.) Polynom mit Koeffizienten aus Z_2

$a = \sum_{i=0}^{7} a_i x^i \quad \in Z_2[x]$, z.B. $a = x^7 + x^4 + x^2 + x + 1$

3.) Hexadezimal

platzsparend und für Tabellen geeignet , z.B. $a = 97_{16}$

Darstellung als Polynom : $\in Z / 2Z[x]$

Mathematische Hintergründe des AES :

$F_{256} = GF(256)$ Körper (Galois-Feld) mit 256 Elementen

$Z_2 = Z / 2Z = Z \bmod 2$

$Z_2[x] / m(x)$ mit $m(x) = x^8 + x^4 + x^3 + x + 1$ (irreduzibel über Z_2)

$\left(\hat{=} 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1, x^3, \ldots, x^7 + \ldots + x + 1\right)$

Interpretation eines Bit-Strings $(a_7 \ldots a_0)$ als Polynom $a_7 x^7 + \ldots + a_1 x + a_0 \bmod m(x)$

Rechenoperationen in F_{256} : $a, b \in \{0,1\}^8$

Addition :

$a \oplus b = a \; XOR \; b$

$0101\,0111 \oplus 1000\,0011 = 1101\,0100$

bzw.

$\left(x^6 + x^4 + x^2 + x + 1\right) \oplus \left(x^7 + x + 1\right) = x^7 + x^6 + x^4 + x^2$

Multiplikation :

$$\left(x^6 + x^4 + x^2 + x + 1\right) \otimes \left(x^7 + x + 1\right)$$
$$= \left(x^{13} + x^{11} + x^9 + x^8 + x^7\right) \oplus \left(x^7 + x^5 + x^3 + x^2 + x\right) \oplus \left(x^6 + x^4 + x^2 + x + 1\right)$$
$$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \quad = p(x)$$

nun Reduktion modulo $m(x)$:

$$\left(x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1\right) : \left(x^8 + x^4 + x^3 + x + 1\right) \;=\; x^5 + x^3$$
$$\underline{-\left(x^{13} + x^9 + x^8 + x^6 + x^5\right)}$$
$$x^{11} + x^4 + x^3 + 1$$
$$\underline{-\left(x^{11} + x^7 + x^6 + x^4 + x^3\right)}$$
$$x^7 + x^6 + 1$$

$\Rightarrow p(x) \equiv x^7 + x^6 + 1 \bmod m(x)$ $\left(\hat{=} 1100\,0001\right)$

Erweiterung :

$GF(256)[z]/(z^4+1)$

Elemente hiervon werden als Polynome 3. Grades in Z interpretiert , deren Koeffizienten Elemente aus $GF(256)$ sind , d.h. :

$$c(z) = c_0 + c_1 z + c_2 z^2 + c_3 z^3 \qquad , \; c_i \in GF(256)$$

Bezeichnungen :

$$c(z) = 02 + A1z + 03z^2 + 01z_3$$

bzw. $[02, A1, 03, 01]$, $[00000010, 10100001, 00000011, 00000001]$

Rechenoperationen in $GF(256)[z]/(z^4+1)$ **:** $a, b \in GF(256)[z]/(z^4+1)$

Addition :

erfolgt komponentenweise

Bsp. : $[02, A1, 03, 01] \oplus [FF, 01, 00, 02] = [FD, A0, 03, 03]$

Multiplikation :

$$c(z) = a(z) \otimes b(z) = c_0 + c_1 z + c_2 z^2 + c_3 z^3 + c_4 z^4 + c_5 z^5 + c_6 z^6$$

$$c_0 = a_0 \otimes b_0$$
$$c_1 = (a_1 \otimes b_0) \oplus (a_0 \otimes b_1)$$
$$c_2 = (a_2 \otimes b_0) \oplus (a_1 \otimes b_1) \oplus (a_0 \otimes b_2)$$
$$c_3 = (a_3 \otimes b_0) \oplus (a_2 \otimes b_1) \oplus (a_1 \otimes b_2) \oplus (a_0 \otimes b_3)$$
$$c_4 = (a_3 \otimes b_1) \oplus (a_2 \otimes b_2) \oplus (a_1 \otimes b_3)$$
$$c_5 = (a_3 \otimes b_2) \oplus (a_2 \otimes b_3)$$
$$c_6 = (a_3 \otimes b_3)$$

nun Reduktion modulo $z^4 + 1$:

berücksichtige $z^i \bmod (z^4+1) = z^{i \bmod 4}$:

$$z^4 \equiv 1 \mod z^4 + 1$$
$$z^5 \equiv z \mod z^4 + 1$$
$$z^6 \equiv z^2 \mod z^4 + 1$$
$$z^7 \equiv z^3 \mod z^4 + 1$$
$$z^8 \equiv 1 \mod z^4 + 1$$

$c(z) \equiv d(z) \bmod z^4 + 1$ mit :

$d(z) = d_0 + d_1 z + d_2 z^2 + d_3 z^3$ und

$$d_0 = c_0 + c_4 = (a_0 \otimes b_0) \oplus (a_3 \otimes b_1) \oplus (a_2 \otimes b_2) \oplus (a_1 \otimes b_3)$$
$$d_1 = c_1 + c_5 = (a_1 \otimes b_0) \oplus (a_0 \otimes b_1) \oplus (a_3 \otimes b_2) \oplus (a_2 \otimes b_3)$$
$$d_2 = c_2 + c_6 = (a_2 \otimes b_0) \oplus (a_1 \otimes b_1) \oplus (a_0 \otimes b_2) \oplus (a_3 \otimes b_3)$$
$$d_3 = c_3 = (a_3 \otimes b_0) \oplus (a_2 \otimes b_1) \oplus (a_1 \otimes b_2) \oplus (a_0 \otimes b_3)$$

in Matrizenschreibweise :

$$\begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{pmatrix} = \begin{pmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{pmatrix} \cdot \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

Beispiel :

$$[00,00,00,01] \otimes [47,08,1F,2B]$$

$$\begin{pmatrix} 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \\ 00 & 00 & 00 & 01 \\ 01 & 00 & 00 & 00 \end{pmatrix} \cdot \begin{pmatrix} 47 \\ 08 \\ 1F \\ 2B \end{pmatrix} = \begin{pmatrix} 08 \\ 1F \\ 2B \\ 47 \end{pmatrix}$$

Bemerkungen :

- Nicht jedes Element in $GF(256)[z]/(z^4+1)$ besitzt ein multiplikatives Inverses.
- 32 Bit Substitution S_{32} im AES benutzt $a(z) = 02 + 01z + 01z^2 + 03z^3$ als Multiplikator , welcher das Inverse $a^{-1}(z) = 0E + 09z + 0Dz^2 + 0Bz^3$ besitzt.

Beschreibung der einzelnen Verschlüsselungsschritte des AES :

- **Byte-Sub :**

  Die Einträge $a^{(z,s)}$ der Byte-Matrix werden gemäß der Tabelle mit Substitution S_8 ersetzt.

  $b^{(z,s)} = S_8\left(a^{(z,s)}\right)$, $z = 0,1,2,3$, $s = 0,...,n_b - 1$

  $a_7a_6a_5a_4$ = Zeile

  $a_3a_2a_1a_0$ = Spalte

  Beispiel : $S_8(10110101) = D5 = 11010101$

  Mathematische Hintergründe der Tabelle :

  Die S_8 - Substitution ist eine Zusammensetzung von zwei Transformationen :

  1.) Falls $a^{(z,s)} \neq 0$ ist , so berechne zum Eingabewert $a^{(z,s)}$ das Inverse $\left(a^{(z,s)}\right)^{-1}$ in $GF(256)$.
  Ist $a^{(z,s)} = 0$, dann setzte $\left(a^{(z,s)}\right)^{-1} = 0$.

  2.) Das Zwischenergebnis $\left(a^{(z,s)}\right)^{-1} = \tilde{a}_7\tilde{a}_6...\tilde{a}_0$ wird einer affinen Abbildung über Z_2 unterzogen :

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_7 \end{pmatrix} = \begin{pmatrix} 10001111 \\ 11000111 \\ 11100011 \\ \vdots \\ 00011111 \end{pmatrix} \cdot \begin{pmatrix} \tilde{a}_0 \\ \tilde{a}_1 \\ \tilde{a}_2 \\ \vdots \\ \tilde{a}_7 \end{pmatrix}$$

- **Row-Shift :**

  Ziel : Einträge der Byte-Maxtrix $S_{i,b}$ werden zeilenweise nach links verschoben.

  Die Einträge der 0. Zeile bleibt unverändert
  Die Einträge der 1. Zeile werden um 1 Position verschoben
  Die Einträge der 2. Zeile werden um 2 Position verschoben

...

Mathematischer Hintergrund :

zyklische Verschiebung $\hat{=}$ Multiplikation in $GF(256)[z]/(z^4+1)$ mit dem Multiplikator

$r_1(z) = 01z^3$

$r_2(z) = 01z^2$

$r_3(z) = 01z$

$\left[c^{(1,0)}, c^{(1,1)}, c^{(1,2)}, c^{(1,3)}\right] = \left[b^{(1,0)}, b^{(1,1)}, b^{(1,2)}, b^{(1,3)}\right] \otimes [00,00,00,01]$

usw.

- **Column-Mix :**

  Substitution mit S_{32}

  Die Matrix $S_{i\,c} = \begin{pmatrix} c^{(0,0)} & c^{(0,1)} & \cdots & c^{(0,n_b-1)} \\ c^{(1,0)} & c^{(1,1)} & \cdots & c^{(1,n_b-1)} \\ c^{(2,0)} & c^{(2,1)} & \cdots & c^{(2,n_b-1)} \\ c^{(3,0)} & c^{(3,1)} & \cdots & c^{(3,n_b-1)} \end{pmatrix}$ geht über in die Matrix $S_{i\,d}$, für deren Spalten gilt :

  $$\begin{pmatrix} d^{(0,s)} \\ d^{(1,s)} \\ d^{(2,s)} \\ d^{(3,s)} \end{pmatrix} = S_{32} \cdot \begin{pmatrix} c^{(0,s)} \\ c^{(1,s)} \\ c^{(2,s)} \\ c^{(3,s)} \end{pmatrix} \qquad , s = 0,\ldots,n_b - 1$$

  Hilfsfunktion $xtime: \{0,1\}^8 \to \{0,1\}^8$

  $xtime: x_7x_6x_5x_4x_3x_2x_1x_0 \mapsto x_6x_5x_4x_3x_2x_1x_00$, falls $x_7 = 0$

  $xtime: x_7x_6x_5x_4x_3x_2x_1x_0 \mapsto (x_6x_5x_4x_3x_2x_1x_00) \oplus (00011011)$, falls $x_7 \neq 0$

  Mathematischer Hintergrund :

  S_{32} ist eine Multiplikation mit $[02,01,01,03]$ in $GF(256)[z]/(z^4+1)$, d.h. :

  $\left[d^{(0,s)}, d^{(1,s)}, d^{(2,s)}, d^{(3,s)}\right] = \left[c^{(0,s)}, c^{(1,s)}, c^{(2,s)}, c^{(3,s)}\right] \otimes [02,01,01,03]$

$xtime$ $\hat{=}$ Multiplikation mit $0x02$ $\hat{=}$ Multiplikation mit z in $GF(256)[z]/\left(z^4+1\right)$

Key-Add

$s_{i+1,a} = s_{i,d} \oplus k_i$ (XOR-Addition)

Dechiffrierung

Umkehroperationen

1.) Byte-Sub $^{-1}$

$\forall a \in GF(256) \quad Byte-Sub^{-1}(Byte-Sub(a)) = a$ (ebenso für die anderen Umkehroperationen)

Subsitution mit S_8^{-1} :

Gegebenen Wert in der Tabelle suchen und Zeile und Spalte auslesen.

2.) Row-Shift $^{-1}$

Verschiebung der Bytes um jeweils 1,2,3,4 Stellen nach rechts

3.) Column-Mix $^{-1}$

$$\begin{pmatrix} c^{(0,s)} \\ c^{(1,s)} \\ c^{(2,s)} \\ c^{(3,s)} \end{pmatrix} = S_{32}^{-1} \cdot \begin{pmatrix} d^{(0,s)} \\ d^{(1,s)} \\ d^{(2,s)} \\ d^{(3,s)} \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \cdot \begin{pmatrix} d^{(0,s)} \\ d^{(1,s)} \\ d^{(2,s)} \\ d^{(3,s)} \end{pmatrix}$$

S_{32} war als Multiplikation mit $[02,01,01,03]$ in $GF(256)[z]/\left(z^4+1\right)$ erklärt , das Inverse dazu ist $[0E,09,0D,0B]$.

Es gibt zwei Möglichkeiten S_{32}^{-1} effizient zu implementieren :

1.) Multiplikationstabellen für 09 , 0B , 0D , 0E
2.) Logarithmentafel

$$\exp : (R\ ,+) \to (R_+\ ,\cdot)$$
$$r \mapsto e^r$$
$$r+s \mapsto e^r \cdot e^s = e^{r+s}$$

$$\Rightarrow$$
$$\log : (Z/255Z\ ,\oplus) \to (GF(256)\setminus\{0\}\ ,\otimes)$$

Rückführung der Multiplikation in $GF(256)\setminus\{0\}$, $\otimes$ auf die Addition in $Z/255Z$.

4.) Key-Add $^{-1}$

XOR-Addition ist selbstinvers

Zur Auswahl der Rundenschlüssel :

- Schlüssel-Expansions-Algorithmus zum Vergrößern des Schlüssels
  Blöcke aus je 4 Byte : $k^{(0)},k^{(1)},...$
  Länge des Schlüssels ist abhängig von Blocklänge des Klartextes und der Länge des Originalschlüssels
- Auswahl der Rundenschlüssel $k_0,k_1,...,k_{n_r}$ aus dem expandierten Schlüssel

Schlüssellänge \ Klartext-Blocklänge	128	192	256
128	16*11	24*13	32*15
192	16*13	24*13	32*15
256	16*13	24*15	32*15

Zusammenfassung des expandierten Schlüssels zum Rundenschlüssel

$\underbrace{k^{(0)}...k^{(n_b-1)}}_{k_0}$ $\underbrace{k^{(n_b)}...}_{k_1}$ usw.

$n_b/4$ = Schlüsselbytes pro Runde

$(n_b/4)\cdot(n_r+1)$ = Schlüsselbytes insgesamt

Zahlentheorie

Eulerscher Satz :

Seien $a, n \in Z$ mit $a > 1$ und $ggT(a,n) = 1$.

Dann gilt : $a^{\Phi(n)} \equiv 1 \bmod n$.

Beweis:

$Z_n^* = \{[x_1]_n, ..., [x_\Phi]_n\}$ ist eine prime Restklassengruppe (mit $\Phi = \Phi(n)$)

Sei $a \in Z$ mit $ggT(a,n) = 1$.

$a : Z_n^* \to Z_n^*$, $[x]_n \to [a \cdot x]_n$ ist eine Bijektion und es gilt :

$$\begin{aligned} (a \cdot [x_1]) \cdot (a \cdot [x_2]) \cdot ... \cdot (a \cdot [x_\Phi]) &\equiv a^\Phi \cdot [x_1] \cdot ... \cdot [x_\Phi] \bmod n \\ &\equiv [x_1] \cdot ... \cdot [x_\Phi] \qquad |: [x_1] \cdot ... \cdot [x_\Phi] \\ \Rightarrow \quad a^\Phi &\equiv 1 \bmod n \end{aligned}$$

Als Folgerung ergibt sich der kleine Fermatsche Satz.

Fermatscher Satz :

Test auf Primzahleigenschaft

Für $p \in \{\text{Primzahlen}\}$ mit $ggT(a,p) = 1$ gilt : $a^{(p-1)} \equiv 1 \bmod p$

- versagt bei Carmichael-Zahlen
- effizient , da $a^{(p-1)}$ in logarithmisch vielen Schritten berechnet werden kann

Def. Carmichael-Zahl :

Jede Carmichael-Zahl ist das Produkt aus mindestens 3 Primzahlen.

Eine zusammengesetzte natürliche Zahl q heißt Carmichael-Zahl , falls für alle zu q teilerfremden Zahlen a gilt :

$$a^{q-1} \equiv 1 \bmod q$$

Die kleinste Carmichael-Zahl ist $561 = 3 \cdot 11 \cdot 17$

Für alle Zahlen a mit $a \nmid 561$ gilt : $a^{560} \equiv 1 \bmod 561$

Def. : Ordnung von a modulo n

$Ord_n(a) =$ kleinste Zahl $e \in N$, für die $a^e \equiv 1 \bmod n$ gilt.

Beispiel 1 : $Ord_5(4) = 2$

Beispiel 2 : $Ord_{101}(5)$

Nach dem Fermatschen Satz gilt :

$5^{100} \equiv 1 \bmod 101$

$5^{25} \equiv 1 \bmod 101$

$5^5 \equiv -6 \bmod 101$

$\Rightarrow Ord_{101}(5) = 25$

, denn falls $a^f \equiv 1 \bmod n$ und $e = Ord_n(a)$, d.h. $a^e \equiv 1 \bmod n$ mit $e \le f$, dann gilt $e \mid f$ (siehe Satz)

Miller-Rabin-Test

- effizienter Test auf Primzahleigenschaft (allerdings nicht absolut sicher)
- Verschärfung des Fermatschen Satzes

Sei $n \in N$, $n > 2$ und ungerade

Setze $s := \max\{r \in N : 2r \mid (n-1)\}$

, d.h. 2^s ist die größte Zweierpotenz , die $n-1$ teilt

Setze $d := \frac{(n-1)}{2^s}$

Angenommen , n wäre eine Primzahl , und $ggT(a,n) = 1$, dann gilt entweder :

1.) $a^d \equiv 1 \bmod n$ oder

2.) $\exists t \in \{0,1,...,s-1\}$ mit $a^{2^t \cdot d} \equiv -1 \bmod n$

Beweis :

- $Ord_n(a) \mid (n-1)$, $\left(n \text{ prim} \Rightarrow a^{(n-1)} \equiv 1 \bmod n\right)$

  Falls $Ord_n(a) < n-1$, so gilt $\left(a^{Ord_n(a)}\right)^{\frac{n-1}{Ord_n(a)}} \equiv 1 \bmod n$

- $n-1 = d \cdot 2^s \Rightarrow Ord_n(a^d) \mid 2^s$, d.h. $\exists l$ mit $Ord_n(a^d) \mid 2^l$

  , $\left(a^{n-1} \equiv 1 \bmod \Rightarrow a^{2^s \cdot d} \equiv 1 \bmod n \Rightarrow (a^d)\wedge 2^s \equiv 1 \bmod n \Rightarrow Ord_n(a^d) \mid 2^s\right)$

- $l = 0 : Ord_n(a^d) = 1$, $a^d \equiv 1 \bmod n$ **Bedingung 1.)**
- $1 \le l \le s : Ord_n\left(a^{2^{l-1} \cdot d}\right) = Ord_n(x) = 2$

  , d.h. $x^2 \equiv 1 \bmod n \Rightarrow x^2 - 1 \equiv 0 \bmod n \Rightarrow n \mid (x+1)(x-1)$

  $\Rightarrow n \mid (x+1)$

  Angenommen , $n \mid (x-1) : \Rightarrow (x-1) \equiv 0 \bmod n \Rightarrow x \equiv 1 \bmod n$

  $\Rightarrow$ Widerspruch , da $x \equiv 2 \bmod n$ (s.o.)

  $\Rightarrow (x+1) \equiv 0 \bmod n$

  $\Rightarrow x \equiv -1 \bmod n$ **Bedingung 2.)**

Schlussfolgerung :

- Wenigstens eine der zwei Bedingungen ist notwendig , um n als Primzahl zu klassifizieren.
- Die Zahl a heißt Zeuge gegen die Primzahleigenschaft von n .
- Ist $n \ge 3$ und ungerade , dann gibt es in der Menge $\{1,2,...,n-1\}$ höchstens $\frac{n-1}{4}$ Zahlen , die zu n teilerfremd und keine Zeugen gegen die Primzahleigenschaft von n sind.
- Nach Anwendung von k Tests erhält man mit Wahrscheinlichkeit $> \left(1 - \frac{1}{4^t}\right)$ die Gewissheit , dass n eine Primzahl ist.

Beispiel :

Carmichael-Zahl $n = 561$ $\quad a = 2$, $s = 4$, $d = 35$

$2^{35} \equiv 263 \bmod 561$

$2^{2 \cdot 35} \equiv 166 \bmod 561$

$2^{4 \cdot 35} \equiv 67 \bmod 561$

$2^{8 \cdot 35} \equiv 1 \bmod 561$

$\Rightarrow$ keine Primzahl

Primitive Wurzeln modulo einer Primzahl

Sei p eine Primzahl. Eine Zahl $a \in \{2,...,p-1\}$ heisst primitive Wurzel von p , falls $Ord_p(a) = p-1$ ist.

Verwendung in der Kryptographie , da das Potenzieren modulo p eine Falltürfuktion ist ,
d.h. Potenzieren leicht , Logarithmieren schwer

Algorithmus zum Auffinden einer primitiven Wurzel :

1.) Wähle eine Zufallszahl $a \quad (1 < a < p)$

2.) Liste der Primteiler von $p-1$ berechnen $\quad p_1,...,p_r$

3.) Berechne $x \equiv a^{\frac{p-1}{p_i}} \bmod p \quad$ für $i = 1,...,r$

4.) Falls $\forall i \quad x \neq 1 \quad \Rightarrow \quad a$ ist primitive Wurzel

5.) Gehe zu 1.)

Beispiel 1 :

$p = 13 \quad , \quad a = 2$

$2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 3$, $2^5 = 6$, $2^6 = 12$, $2^7 = 11$, $2^8 = 9$,

$2^9 = 5$, $2^{10} = 10$, $2^{11} = 7$, $2^{12} = 1$

$\Rightarrow$ 2 ist primitive Wurzel von 13

Die Zahl 13 hat insgesamt 4 primitive Wurzeln : 2 , 6 , 7 , 11

Beispiel 2 :

$Z_8 = \{0,1,2,3,4,5,6,7\}$ besitzt keine primitiven Wurzeln

$x^2 \equiv 1 \bmod 8$ ist erfüllt für $x = 1,3,5,7$

$x^2 \equiv 4 \bmod 8$ ist erfüllt für $x = 2,6$

$x^2 \equiv 16 \bmod 8$ ist erfüllt für $x = 4$

Satz von Gauß :

Sei p eine Primzahl.

Dann existiert eine primitive Wurzel und es gibt insgesamt $\Phi(p-1)$ primitive Wurzeln.

Bemerkungen zur Bestimmung primitiver Wurzeln :

Sei $p > 2$ eine Primzahl und $a \in \{2,...,p-1\}$.

Dann gilt : a ist primitive Wurzel von p gdw. eine der folgenden äquivalenten Bedingungen erfüllt ist :

1.) $\min\{n \in N \, , \, n \geq 1 : g^n \equiv 1 \bmod p\} = p-1$

2.) Für jeden Primteiler q von $p-1$ gilt : $g^{\frac{p-1}{q}} \not\equiv 1 \bmod p$.

Anwendung :

Sei a primitive Wurzel modulo p .

Dann ist die Abbildung $j : (Z_{p-1}, +) \to (Z_p^*, \cdot)$ mit $a \bmod (p-1) \mapsto g^a \bmod p$ eine bijektive Abbildung mit $j\left((a \bmod (p-1)) + (b \bmod (p-1))\right) = j\left(a \bmod (p-1)\right) \cdot j\left(b \bmod (p-1)\right)$.

(Analogie zur Exponentialfunktion)

Die Umkehrabbildung ist der „Index“ bzw. der diskrete Logarithmus :

$ind_g : (Z_p^*, \cdot) \to (Z_{p-1}, +)$ mit

$ind_g(x \cdot y) = ind_g(x) + ind_g(y)$

$ind_g(x^{-1}) = -ind_g(x)$

$ind_g(x^k) = k \cdot ind_g(x)$, $k \in N$

Beispiel :

$p = 13$

$ind_2(7 \bmod 13) = 11 \bmod 12$, da $2^{11} \equiv 7 \bmod 13$

$ind_2(5 \bmod 13) = 9 \bmod 12$, da $2^9 \equiv 5 \bmod 13$

$ind_2(35 \bmod 13) = ind_2(9 \bmod 13) = 8 \bmod 12$

Baby-step-giant-step – Algorithmus :

Algorithmus zur Berechnung des diskreten Logarithmus.

Eingabe :

p - Primzahl

g - primitive Wurzel modulo p

y - Element aus Z_p^*

Ausgabe :

$x = \log_g y$, d.h. $g^x \equiv y \bmod p$

Setze $t := \lfloor \sqrt{p} \rfloor$

Für eine Zahl x gilt : $\exists\, q, r$ mit $x = q \cdot t + r$ (für $0 \le r < \sqrt{p}$)

Betrachte $a := g^x \bmod p$

$\equiv g^{qt+r} \bmod p$ $\quad |: q^r$

$\Rightarrow (g^t)^q = a \cdot g^{-r} \bmod p$

Algorithmus :

1.) Berechne Liste „Baby“ der Paare $\left(r \ , \ a \cdot g^{-r} \bmod p\right)$ für $0 \le r < t$

Findet man $(r, 1) \in B$, so ist $x = r$ der diskrete Logarithmus von a .

2.) Anderenfalls berechne Liste „Giant“ der Paare $\left(q \ , \ (g^t)^q \bmod p\right)$ für $0 \le q < t$

3.) Sortiere die zwei Listen nach der zweiten Komponente (effizient : Counting-Sort)

4.) Suche Paare $(i, k) \in$ “Giant“ und $(j, k) \in$ “Baby“

$\Rightarrow$ $x = i \cdot t + j$ ist der diskrete Logarithmus von a .

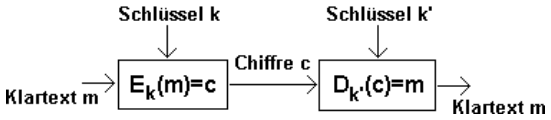
Laufzeit und Speicherbedarf : $O\left(\sqrt{p}\right)$, für $p > 2^{200}$ nicht mehr effektiv.

Probleme :

- Test auf primitive Wurzel (existieren gleiche Paare?)
- Speicherbedarf

Kapitel 4 : Asymmetrische Kryptosysteme

Symmetrische Kryptosysteme sind für mehrere tausend Teilnehmer ungeeignet.
Umgehung des Schlüsseltauschs von Partnern durch öffentliche Kryptosysteme
Empfänger besitzt geheimen Schlüssel k und öffentlichen Schlüssel k'.



Hintergrund : Falltürfunktion (Trapdoor Function)
D.h. $f(m)=c$ lässt sich leicht berechnen , $f^{-1}(c)=m$ hingegen lässt sich aus der Kenntnis von f nur sehr schwer berechnen.

Bsp : klassisches (analoges) Telefonbuch
Nummer zu einem Namen lässt sich sehr leicht finden , andersherum jedoch äusserst schwierig.

Idee des Public-Key-Kryptosystems :

1.) Jeder Teilnehmer A konstruiert eine Trapdoor-Funktion f_A , wobei eine nur A bekannte Zusatzinformation f_A^{-1} liefert.
2.) Teilnehmer A gibt f_A bekannt und hält f_A^{-1} geheim.
3.) Will B eine Nachricht an A senden , so verschlüsselt er sie mittels f_A und sendet $c=f_A(m)$ an A.
4.) Teilnehmer A erhält $c=f_A(m)$, wendet f_A^{-1} an und erhält $f_A^{-1}(c)=f_A^{-1}(f_A(m))=m$

Probleme :

1.) Es muss Sicherheit bestehen , dass aus f_A die Umkehrfunktion f_A^{-1} nicht leicht zu berechnen ist.
2.) Vorspiegeln einer falschen Identität. (elektronische Unterschrift)

4.1 Rucksack-Problem

NP-vollständig

siehe Übung08 (Merkle-Hellmann-Algorithmus)

4.2 RSA-Algorithmus (Rivest,Shamir,Adlemann)

basiert auf zahlentheoretischen Erkenntnissen

zur Erinnerung :

$Z_m = Z/mZ$: Restklassenring modulo m

Z_m^* : prime Restklassengruppe
$= \{[a]_m \in Z_m : ggT(a,m)=1\}$

$|Z_m^*| = \Phi(m)$ Eulersche Φ – Funktion

Satz : $m,n \in N$, $ggT(m,n)=1$
$\Rightarrow \quad \Phi(m\cdot n)=\Phi(m)\cdot\Phi(n)$

Idee von RSA :

1.) Wähle zwei große Primzahlen p, q und setze $n = p \cdot q$

2.) Wähle ein e mit $3 \le e < (p-1)(q-1)$ und $ggT(\, e \,,\, (p-1)(q-1)\,) = 1$

3.) Bestimme ein d mit $1 < d < (p-1)(q-1)$ und $e \cdot d \equiv 1 \bmod (p-1)(q-1)$

4.) Öffentlicher Schlüssel : (n, e)

n : RSA-Modul

e : Verschlüsselungsexponent

5.) Privater Schlüssel : d (Entschlüsselungsexponent)

Beispiel :

$p = 11\,, \qquad q = 23\,, \qquad \Rightarrow n = 253$

$(p-1)(q-1) = 220$

$e = 3 \;\Rightarrow\; d = 147$

Codierung

- Klartextraum : natürliche Zahlen m mit $0 \le m < n$
- Chiffre $c \equiv m^e \bmod n$

Beispiel :

Klartextraum = $(0,...,252)$

$m = 165 \;\Rightarrow\; c = 110$

Decodierung

Empfänger erhält $c \in \{0,...,252\}$ und berechnet $m = c^d \bmod n$

Beispiel :

$110^{147} \bmod 253 \equiv 165$

Satz :

Sei (n, e) der öffentliche- und d der private Schlüssel des RSA-Verfahrens. Dann gilt :

$$\left(m^e\right)^d \equiv m \bmod n \qquad , \; \forall m \in N \text{ und } 0 \le m < n$$

Beweis :

Wegen $e \cdot d \equiv 1 \bmod (p-1)(q-1)$ existiert ein l mit

$e \cdot d = 1 + l \cdot (p-1)(q-1)$

$m^{e \cdot d} = m \cdot m^{l \cdot (p-1)(q-1)}$

$= m \bmod p \qquad$, da $m^{p-1} \equiv 1 \bmod p$, falls $p \nmid m$

Analog für $q \nmid n$

$\Rightarrow m^{e \cdot d} \equiv m \bmod n$

Sicherheit von RSA :

- Kennt ein Angreifer die Primfaktoren p und q , so kann er mittels des Euklidschen Algorithmus d berechnen.
- Umgekehrt kann man aus n, e, d die Faktoren von n berechnen :

- Es gilt $a^f \equiv 1 \bmod m$ gdw. $Ord_m(a) \mid f$

„$\Leftarrow$“ :

$Ord_m(a) \mid f$

$f = b \cdot Ord_m(a)$

$\left(a^{Ord_m(a)}\right)^b \equiv 1 \bmod m$

$\Rightarrow a^f \equiv 1 \bmod m$

„$\Rightarrow$“ :

$a^f \equiv 1 \bmod m$ und $f = k \cdot Ord_m(a) + r$

$0 \le r < Ord_m(a)$

$a^{k \cdot Ord_m(a)+r} \equiv \underbrace{a^{k \cdot Ord_m(a)}}_{\equiv 1} \cdot a^r \equiv 1 \bmod m$

$\Rightarrow r = 0$

- $Ord_m(a) \mid \Phi(m)$

Sei nun $s := \max\left\{t \in N : 2^t \mid (e \cdot d - 1)\right\}$, $k = \dfrac{e \cdot d - 1}{2^s}$

und sei $ggT(a, n) = 1$.

Dann gilt :

$Ord_n\left(a^k\right) \in \left\{2^i : 0 \le i \le s\right\}$

$a^{e \cdot d - 1} \equiv 1 \bmod m$, wegen $a^{e \cdot d} \equiv a \bmod n$, siehe Decodierung

$\left(a^k\right)^{\wedge} 2^s \equiv 1 \bmod n$

Satz :

Sei $ggT(a, n) = 1$

Falls $Ord_p(a^k) \neq Ord_q(a^k)$, so ist $1 < ggT\left(a^{2^t \cdot k} - 1, n\right) < n$ für ein $t \in \{0, ..., s-1\}$.

Beweis :

$Ord_p(a^k), Ord_q(a^k) \in \left\{2^i : 1 \le i \le s\right\}$

Sei o.B.d.A. $Ord_p(a^k) > Ord_q(a^k)$

$Ord_q(a^k) = 2^t \quad < 2^s$, da $t < s$

$a^{2^t \cdot k} \equiv 1 \bmod q$

$a^{2^t \cdot k} \not\equiv 1 \bmod p$, da $t < s$

$\Rightarrow ggT\left(a^{2^t \cdot k} - 1 \, , \, n\right) = q$

Algorithmus :

- Eingabe n

1.) wähle Zufallszahl $a \in \{1, ..., n-1\}$

2.) Berechne $ggT(a, n) = g$

3.) if $g \neq 1$ then fertig else berechne $ggT(a^{2^t \cdot k}, n)$ für $t \in \{0, ..., s-1\}$

4.) if $g \neq 1$ then fertig else goto 1.)

Bemerkungen :

- In jeder Iteration ist die Wahrscheinlichkeit dafür , einen Teiler von n zu finden $> \frac{1}{2}$.
- Die Wahrscheinlichkeit , nach r Schritten einen Teiler gefunden zu haben ist $> \left(1-\left(\frac{1}{2}\right)^r\right)$.
- Falls man nach r Schritten keinen Faktor gefunden hat , so handelt es sich mit der Wahrscheinlichkeit $> \left(1-\left(\frac{1}{2}\right)^r\right)$ um eine Primzahl.

Beispiel :

$n = 253 \quad , \quad e = 3 \quad , \quad d = 147 \quad \Rightarrow \quad e \cdot d - 1 = 440$

$a = 2$:

$ggT(2^{220} - 1 \, , 253) = 253$

$ggT(2^{110} - 1 \, , 253) = 253$

$ggT(2^{55} - 1 \, , 253) = 23$

Realisierung von RSA als eine Art Blockchiffre :

Alphabet Σ besitzt genau N Zeichen codiert als Zahlen $\{0,...,N-1\}$

Konstante $k = \lfloor \log_N n \rfloor$ (=Blocklänge)

Block m der Länge k : $m_1 ... m_k$ mit $m = \sum_{i=1}^{k} m_i \cdot N^{k-i}$

Es gilt $0 \le m \le (N-1) \cdot \sum_{i=1}^{k} N^{k-i} = N^k - 1 < n$

Codierung :

$c \equiv m^e \bmod n$

- c kann ebenfalls zur Basis N dargestellt werden
- $c = \sum_{i=0}^{k} c_i \cdot N^{k-i}$, $c_i \in \Sigma$
- Die N - adische Darstellung von c kann $k+1$ Stellen haben

Beispiel :

$\Sigma = \{0, a, b, c\} \mathrel{\hat{=}} \{0,1,2,3\}$, $|\Sigma| = 4 = N$

$k = \lfloor \log_4 253 \rfloor = 3$

$\Rightarrow$ Chiffreblöcke der Länge 4

Codierung von abb :

$abb \mathrel{\hat{=}} 122$

$m = 4^2 + 2 \cdot 4 + 2 = 26$

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

$c = 1 \cdot 4^3 + 3 \cdot 4^2 + 1 \cdot 4 + 3 \quad \mathrel{\hat{=}} acac$

Problem der großen Primzahlen :
Anzahl der Primzahlen unterhalb einer Schranke M ?
genaues Verfahren : Zählen mit Sieb des Eratosthenes
Schätzung : Pi-Funktion $\Pi(x)$ = Anzahl der Primzahlen $\leq x$

$$\Pi(x) \approx \frac{x}{\ln x}$$

Beispiel :

Die Wahrscheinlichkeit dafür , dass eine zufällige , ungerade , 200-stellige Zahl prim ist , beträgt $\approx 0.4\%$.

effizienter Primzahltest siehe Miller-Rabin-Test

Risiken des RSA :
Beim Stand von 2005 sind für p und q 1024-Bit Zahlen angeraten. (ca. 300-stellig)

- Nicht die gleichen Primzahlen in verschiedenen RSA-Modulen verwenden.
  (ggT der verschiedenen Module liefert Teiler.)

- Angriff mit ausgewähltem Geheimtext :
  Wer fremde Zeichenfolgen mit RSA dechiffriert und bekannt gibt , kann kompromittiert werden.

  - Abfangen einer Nachricht $c \equiv m^e \bmod n$, (n,e) - öffentlicher Schlüssel einer Person A
  - Multiplikation mit r^e , $ggT(r,n)=1$, liefert $y = c \cdot r^e \bmod n$
  - Fordere A auf , die Mitteilung y zu signieren.
    D.h. $y^d \equiv c^d r^{ed} \bmod n$
    Anwendung der e - ten Potenz liefert y zurück.
    $y^{de} = y \bmod n$
  - Kenntnis von $y^d = c^d r^{ed} \bmod n = c^d r \bmod n$
    Bestimme $r^{-1} \bmod n$

    $$\begin{aligned} \Rightarrow \quad y^d r^{-1} &= c^d \bmod n \\ &= \left(m^e\right)^d \bmod n \\ &= m \bmod n \end{aligned}$$

- Angriff mit kleinem Exponenten :
  Sei $e \in N$ „klein“ und $n_1,...,n_l$ paarweise teilerfremd mit $m < n_i$
  Sei $c \in N$ mit $c = m^e \bmod n_i$ und $0 \leq c < \prod_{i=1}^{l} n_i$, dann gilt : $c = m^e$
  Begründung :
  $c' = m^e$ erfüllt $c' = m^e \bmod n$, und wegen $0 \leq c < \prod_{i=1}^{l} n_i$ folgt die eindeutige Lösung $c = c'$.
  Angreifer erhält den Klartext durch $m = \sqrt[e]{c}$.

- Multiplikativität :
  m_1, m_2 werden verschlüsselt , $c_i = (m_i)^e \bmod n$
  , d.h. $c = c_1 \cdot c_2 = m_1^e m_2^e \bmod n = (m_1 m_2)^e \bmod n$
  (Zusätzliche Struktur für gültige Nachrichten.)

- gemeinsamer Modul :
  RSA mit gleichem n und verschiedenen e ist unsicher.

  Sei $c_i = m^{e_i} \bmod n \quad , \; i = 1,2,...$

  $\Rightarrow$ Kenntnis von n, e_1, e_2, c_1, c_2

  Hieraus lässt sich m berechnen :

  Sei o.B.d.A. $ggT(e_1, e_2) = 1$ vorausgesetzt (hohe Wahrscheinlichkeit hierfür)

  $\Rightarrow$ Bezout-Gleichung $re_1 + se_2 = 1$

  Diese sei gelöst

  Betrachte nun $c_1^{\,r} \equiv m^{e_1 r} \bmod n$ und $c_2^{\,s} \equiv m^{e_2 s} \bmod n$.

  Durch Multiplikation erhält man $c_1^{\,r} c_2^{\,s} \equiv m^{e_1 r + e_2 s} \bmod n \quad \equiv \quad m \bmod n$

- Angriff mit kleinem Modul :
  Modul n lässt sich leichter faktorisieren.

Bemerkungen zu RSA :

1.) Auffinden des geheimen Schlüssels ist genauso schwierig wie das Faktorisierungsproblem.
2.) Es ist unbekannt , ob man RSA ohne den Geheimschlüssel (d.h. ohne Faktorisieren) knacken kann.
3.) Es ist unbekannt , wie schwierig/hart Faktorisieren ist.

- 1983 Patentierung in den USA (Ablauf 200) , ausserhalb der USA frei
- Nicht besonders schnell im Vergleich zu symmetrischen Kryptoverfahren

$\Rightarrow$ hybride Kryptosysteme

z.B. RSA für Schlüsseltausch und dann zum Rechnen ein symmetrisches Kryptosystem

- Verallgemeinerung :

G - Gruppe mit $|G| = O$

Wähle einen Verschlüsselungsexponenten e $\Rightarrow$ öffentlicher Schlüssel (G, e)

Verschlüsselung : $m \in G \quad , \quad c = m^e$

geheimer Schlüssel $d \in \{2,...,O-1\}$ mit $e \cdot d \equiv 1 \bmod O$

Entschlüsselung : $c^d = m^{ed} = m^{1+k \cdot O} = m$

Beim RSA gilt : $G = Z_n^*$ und $|G| = (p-1)(q-1)$

Moderne Kryptosysteme : G = Punkte einer elliptischen Kurve

4.3 Diffie-Hellmann-Schlüsseltausch und ElGamal-Kryptosystem

Diffie-Hellmann-Schlüsseltausch

Ziel : A und B wollen mit einem symmetrischen Kryptosystem kommunizieren und hierfür einen Schlüssel über einen unsicheren Kanal austauschen.

Vorgaben : p - Primzahl , $2 \le g \le p-2$ primitive Wurzel

Vorgehensweise :

1.) A wählt eine zufällige Zahl $a \in \{1,..., p-2\}$ und berechnet $X = g^a \bmod p$
2.) A schickt das Ergebnis X an B
3.) B wählt parallel eine zufällige Zahl $b \in \{1,..., p-2\}$ und berechnet $Y = g^b \bmod p$
4.) B schickt das Ergebnis Y an A
5.) A berechnet nun $Y^a \bmod p = g^{ab} \bmod p$
6.) B berechnet $X^b \bmod p = g^{ab} \bmod p$

Das auf beiden Seiten gleiche Ergebnis $k = g^{ab} \bmod p$ ist der gemeinsame Schlüssel

Zur Sicherheit :

Zum Knacken ist die Bestimmung des diskretetn Logarithmus $x = g^a \bmod p$ notwendig.
Die Berechnung dieses ist uneffizient/schwierig. (siehe baby-step-giant-step)

Beispiel :

$p = 17$, $g = 3$, denn $g^8 \equiv 16 \bmod 17$

A wählt $a = 7$ und erhält $3^7 \bmod 17 = 11$

B wählt $b = 4$ und erhält $3^4 \bmod 17 = 13$

A berechnet $13^7 \bmod 17 = 4$

B berechnet $11^4 \bmod 17 = 4$

Risiken :

„Man in the middle"-Angriff

A und B können nicht sicher sein , dass Nachrichten vom jeweiligen Partner sind (C gibt sich gegenüber A als B und gegenüber B als A aus.)

El-Gamal-Verschlüsselung :

gegeben : Primzahl p , primitive Wurzel $g \bmod p$

Schlüsselerzeugung :

A wählt eine Zufallszahl $a \in \{1,2,..., p-2\}$ und berechnet $X = g^a \bmod p$.

$\Rightarrow$ öffentlicher Schlüssel von A : (p, g, X) , privater Schlüssel von A : a

Verschlüsselung :

Klartextraum $\Sigma = \{0,1,..., p-1\}$

B will eine Nachricht m an A schicken.

B wählt eine Zufallszahl $b \in \{1,2,..., p-2\}$, berechnet $Y = g^b \bmod p$ und bildet die Verschlüsselung $c = X^b \cdot m \bmod p$.

$\Rightarrow$ Chiffrat (Y, c)

Entschlüsselung :

A erhält (Y, c) und berechnet $x = p-1-a$ und weiter $Y^x \cdot c \bmod p$

zur Korrektheit :

$$\begin{aligned} Y^x \cdot c &= (g^b)^{p-1-a} \cdot X^b \cdot m \bmod p \\ &= (g^{p-1})^b (g^a)^{-b} \cdot X^b \cdot m \bmod p \\ &= (g^{p-1})^b (g^a)^{-b} (g^a)^b \cdot m \bmod p \\ &= (g^{p-1})^b \cdot m \bmod p \\ &= m \bmod p \end{aligned}$$

Beispiel :

$p = 23 \quad , \quad g = 7$

A wählt $a = 6$

$\Rightarrow X = g^a \bmod p = 4$

$\Rightarrow$ öffentlicher Schlüssel : $(23,7,4)$

B will $m = 7$ verschlüsseln und wählt $b = 3$

$\Rightarrow Y = g^b \bmod p = 21$

$\Rightarrow c = X^b \cdot m \bmod p = 11$

$\Rightarrow (Y, c) = (21,11)$

A entschlüsselt $Y^{p-1-b} \cdot c \bmod p = 7$

Beurteilung des Verfahrens :

- Parameter :
  p als 1024 Bit-Zahl (Stand 2005)

- Effizienz :
  Chiffrierung und Dechiffrierung erfordern Berechnung eines modularen Exponenten.

  $X = g^a \bmod p$ und $Y = g^b \bmod p$ können vorher berechnet werden und sind von der zu verschlüsselnden Nachricht unabhängig.
  (Speicherung z.B. auf Chipkarte)

  Chiffrierung erfordert eine modulare Multiplikation (effizienter als RSA).

- Nachteil :
  Chiffrat benötigt Verdopplung des Textes gegenüber Klartext.

- Vorteil :
  Randomisiertes Verfahren

- Angriffe :
  - El-Gamal und Diffie-Hellmann sind beide gleich schwer zu knacken.
  - Diskreter Logarithmus : Bestimme aus X den geheimen Exponenten (Geheimschlüssel) , und bestimme $m = Y^{p-1-a} \cdot c \bmod p$.
  - Benutze nicht zweimal den gleichen Exponenten b zum Verschlüsseln :

    $c = X^b m \bmod p$

    $c' = X^b m' \bmod p$

    $\Rightarrow c' c^{-1} = m' m^{-1} \bmod p$

    $\Rightarrow$ Ein Angreifer , der m kennt kann m' ermitteln :

    $m' = c' c^{-1} m \bmod p$

Kapitel 5 : Signaturen (einfache Protokolle)

Ziel : elektronisches Äquivalent zur Unterschrift
Eigenschaften : Echtheit , Identität , Abschluss , Komplexität , Verifizierbarkeit

Kryptographische Algorithmen realisieren all diese Eigenschaften bis auf die Komplexität.

Signaturschema :

Ein Teilnehmer T eines Systems erhält

s_T - geheime Signaturfunktion

v_T - öffentliche Verifikationsfunktion

Forderung : s_T soll aus v_T nicht berechenbar sein.

Vorgehensweise :

1.) T unterschreibt die Nachricht m durch $sig = s_T(m)$.

2.) (m, sig) wird an den Empfänger übertragen.

3.) Empfänger verifiziert $v_T(sig) = m$

Analogie zu asymmetrischen Chiffriersystemen.

Hinweis :
Hashfunktionen (z.B. Quersumme eines Textes) sollten nicht verwendet werden und Signaturen sind sehr lang.

$\Rightarrow$
Kombination von Hashfunktion und Signatur.

Dokument $\Rightarrow$ Hashwert berechnen $\Rightarrow$ Hashwert signieren
Empfänger erhält $\left(m\,,\, sig(h(m))\right)$

Überprüfung der Signatur :

Empfänger berechnet $h(m)$ mit öffentlichem Schlüssel und überprüft , ob $v_T(sig(h(m))) = h(m)$ gilt.

RSA-Signatur

Idee : A signiert ein Dokument m , indem $s = m^d \bmod n$ (d = geheimer RSA-Schlüssel) berechnet wird.
B verifiziert die Signatur : $n = s^e \bmod n$ ((n, e) = öffentlicher RSA-Schlüssel)

Ergebnis : RSA bietet die Möglichkeit des Signierens einer Nachricht.

El-Gamal-Signatur

komplexers Signatur-Verfahren
p - Primzahl , $g \bmod p$ - primitive Wurzel
Erzeugung der Signatur durch eine Hashfunktion $h : \{0,1\}^* \to \{1,2,..., p-2\}$

- A wählt eine Zufallszahl $a \in \{1,2,..., p-2\}$ und berechnet $X = g^a \bmod p$
  öffentlicher Schlüssel : (p, q, X) , geheimer Schlüssel : a
- A wählt desweiteren noch eine Zufallszahl $k \in \{1,..., p-2\}$, welche zu $p-1$ teilerfremd ist und berechnet $r = g^k \bmod p$ und $s = k^{-1} \cdot (h(x) - a \cdot r) \bmod (p-1)$ **(*)**
  Signatur : $(r\,,\, s)$
- Verifikation : B kennt den öffentlichen Schlüssel und verifiziert :
  1.) $1 \le r \le p-1$, wenn nicht , dann Rückweisung
  2.) $x^r \cdot r^s = g^{h(x)} \bmod p$, wenn nicht , dann Rückweisung

Begründung :

$$\begin{aligned} x^r \cdot r^s &= (g^a)^r (g^k)^s \bmod p \\ &= g^{ar} g^{ks} \bmod p \\ &= g^{ar} g^{h(x)-ar} \bmod p \\ &= g^{h(x) \bmod p} \end{aligned}$$

(Multipliziere obiges (*) mit $k \Rightarrow ks = h(x) - ar \bmod (p-1)$ und mit dem kleinen Fermatschen Satz folgt $g^{k(p-1)} \equiv 1 \bmod p$)

Ist umgekehrt 2.) für (r,s) erfüllt , und $k = ind_g r$, d.h. $g^k = r \bmod p$, so gilt :

$$\begin{aligned} g^{ar+ks} &= g^{ar+h(x)-ar} \bmod p \\ &= g^{h(x)} \bmod p \end{aligned}$$

Weil g primitive Wurzel modulo p ist , folgt $ar + ks \equiv h(x) \bmod (p-1)$.

Aus $ggT(k, p-1) = 1$ folgt $s = k^{-1} \cdot (h(x) - ar) \bmod (p-1)$.

Es gibt keine andere Möglichkeit die Signatur s von x zu fälschen !

Digital-Signature-Algorithm (DSA)

- Vom NIST 1991 vorgeschlagen und anerkannt.
- beruht auf El-Gamal-Signatur

1.) Schlüsselerzeugung :

- $2^{159} < q < 2^{160}$
- Primzahl p mit :
  1.) $2^{511+64t} < p < 2^{512+64t} \qquad , t \in \{0,1,...,8\}$
  2.) $q \mid p-1$

Länge von p : 512 bis 1024 Bit , desweiteren ein Vielfaches von 64

$q \mid p-1 \Rightarrow Z_p^*$ besitzt eine Untergruppe der Ordnung q

- A wählt primitive Wurzel $x \bmod p$ und berechnet $g = x^{\frac{p-1}{q}} \bmod p$
- $g \bmod p$ erzeugt in Z_p^* eine Untergruppe der Ordnung q
- $a \in \{1,2,...,q-1\}$ Zufallszahl $\Rightarrow$ öffentlicher Schlüssel (p,q,g,x) , privater Schlüssel : a

2.) Erzeugung der Signatur :

- A will Nachricht m signieren
- öffentliche Hashfunktion $h : \{0,1\}^* \rightarrow \{1,2,...,q-1\}$
- $k \in \{1,2,...,q-1\}$ Zufallszahl
- $r = (g^k \bmod p) \bmod q$
- $s = k^{-1} \cdot (h(m) - a \cdot r) \bmod q$
- $\Rightarrow$ Signatur (r,s)

3.) Korrektheit :

wie bei El-Gamal (siehe weiter vorne)

Bemerkung :
Bisher ist nur das Knacken des DSA mittels des diskreten Logarithmus bekannt.

Shamirs No-Key-Protokoll

Problem : Können zwei Teilnehmer eine vertrauliche Nachricht austauschen , ohne dass einer einen Schlüssel des anderen kennt ?

Lösung :

A will B eine geheime Nachricht m schicken.
A verschlüsselt die Nachricht mit dem geheimen Schlüssel a und schickt sie an B.
B verschlüsselt die Nachricht mit dem geheimen Schlüssel b und schickt sie an A zurück.
A „entschlüsselt“ die Nachricht mit dem geheimen Schlüssel a und schickt sie wieder an B.
B wendet den geheimen Schlüssel b an und erhält die Klartextnachricht m.

mathematische Realisierung :
Primzahl p

A erzeugt ein Paar von Zahlen $(a\,,\,a')$ mit $a \cdot a' \equiv 1 \bmod (p-1)$
B erzeugt ein Paar von Zahlen $(b\,,\,b')$ mit $b \cdot b' \equiv 1 \bmod (p-1)$

a,b - Schloß , a',b' - Schlüssel

Vorgehensweise :

1.) A verschlüsselt Nachricht m durch $x := m^a \bmod p$
2.) B erhält x und verschlüsselt $y := x^b \bmod p$
3.) A erhält y und berechnet $z = y^{a'} \bmod p$
4.) B erhält z und berechnet $z^{b'} \bmod p$

$$\begin{aligned} &= (((m^a)^b)^{a'})^{b'} \bmod p \\ &= (m^{a \cdot a'})^{b \cdot b'} \bmod p \\ &= m \bmod p \end{aligned}$$