Vorlesung Kryptologie

FH Bingen **WS 04/05**
Prof. Dr. rer. Nat. Harald Baier

1.2 Mathematische Grundlagen

Definition 1.1 (XOR)
Wir definieren eine Verknüpfung $\oplus$ von zwei Bits wie folgt:

$\oplus$	0	1
0	0	1
1	1	0

Definition 1.2:
Es seien a, b, n ganze Zahlen, $n > 0$.
Wir sagen „a ist kongruent zu b modulo n“, falls n ein Teiler von a-b ist.
Man schreibt: $a \equiv b \bmod n$

Bemerkung: „Kongruent sein“ ist eine Äquivalenzrelation.

Relation:
$R \subset M \times M$
$m_1 \, R \, m_2$

Bsp.: „Gleichheit“
$m_1 = m_2$
$R = \{ (m_1, m_2) \in M \times M \mid m_1 = m_2 \}$

Das heißt:
1.) $a \equiv a \bmod n$ (Reflexivität)
2.) $a \equiv b \bmod n \Rightarrow b \equiv a \bmod n$ (Symetrie)
3.) $a \equiv b \bmod n$ und $b \equiv c \bmod n \Rightarrow a \equiv c \bmod n$ (Transitivität)

Definition 1.5:
Es sei G eine nicht leere Menge.
Es sei: $o: \; G \times G \rightarrow G$ eine Verknüpfung auf G.
Das Paar (G, o) heißt Gruppe, falls folgendes gilt:
1.) Assoziativität: $(g \circ h) \circ r = g \circ (h \circ r)$ für alle $g, h, r \in G$
2.) Existenz des neutralen Elements: Es gibt ein $e \in G$ mit $g \circ e = e \circ g = g$ für alle $g \in G$
3.) Existenz des Inversen: Für jedes $g \in G$ gibt es ein $h \in G : h \circ g = g \circ h = e$

Bsp.: $(\mathbb{Z}, +)$
$(3 \setminus \{o\}, *)$

$\boxed{g^n = h}$

Definition 1.6
$a \equiv b \bmod n \Leftrightarrow n \mid a - b$ (n teilt a-b)
Für jedes $n \in \angle$ ist die Relation ≡ eine Äquivalenzrelation.

Bsp.: Sei n = 3. Es gilt:
$0 \equiv 3 \bmod 3$
$12 \equiv 3 \bmod 3$
$1 \equiv 4 \equiv -2 \equiv -5 \bmod 3$

Es gibt 3 Äquivalenzklassen:
$0 + 3\mathbb{Z} = \{ 3 * x \mid x \in \mathbb{Z} \} =: \overline{0}$
$1 + 3\mathbb{Z} = \{ 3x + 1 \mid x \in \mathbb{Z} \} =: \overline{1}$
$2 + 3\mathbb{Z} = \{ 3x + 2 \mid x \in \mathbb{Z} \} =: \overline{2}$

Satz 1.4:

Es seien a, b, c, d $\in \mathbb{Z}$ und n $\in \angle$.
Es seien $a \equiv c \bmod n$ und $b \equiv d \bmod n$.
Dann gilt:
1.) $-a \equiv -c \bmod n$
2.) $a + b \equiv c + d \bmod n$
3.) $a * b \equiv c * d \bmod n$

Bsp.: n = 5 Die Menge $\{ \overline{0}, \overline{1}, \overline{2}, \overline{3}, \overline{4} \}$ ist bezüglich der Addition eine Gruppe

$\overline{2} + \overline{2} = \overline{4} = \overline{-1}$
$\overline{4} + \overline{3} = \overline{7} = \overline{2}$

+: (a + 5ℤ) + (b + 5ℤ) := (a + b) + 5ℤ
*: (a + 5ℤ) * (b + 5ℤ) := (a * b) + 5ℤ

Definition 1.7: (Potenzen, Vielfache)
Es sei (G, o) eine Gruppe mit neutralem Element e.
o sei multiplikativ
$g^o := e$
$g^n = (g^{n-1}) \text{ o } g$ für $n \geq 1$
$g^n = (g^{-1})^{-n}$ für $n < 0$

Bsp.: $2^0 = 1$
$2^1 = 2$; $2^2 = 2^1 * 2 = 4$; ...
$2^{-2} = (2^{-1})^2 = ¼$

o sei additiv
$0 * g = e$
$n * g = ((n-1) g) \text{ o } g$
$n * g = ((-n) (-g))$

Definition 1.8: (Diskretes-Logarithmus-Problem)
Es sei (G, o) eine Gruppe. Es sei $g \in G$ und $l \in \angle$.
Wir setzen $h := g^l$. Das Diskrete-Logarithmus-Problem (DLP) besteht darin, zu gegebenem g und h den Exponenten l zu berechnen.

Bsp.: $3^l = 81;\ l = 4$

Bei additiver Gruppenoperation schreibt man $h = l * g$ für das DLP.

Bsp.: $G = \{ \overline{1}, \overline{2}, \overline{3}, \overline{4}, \overline{5}, \overline{6} \}$ (Restklassen modulo 7, ohne Restklasse $\overline{0}$)
o = *

k	3^k
1	$\overline{3}$
2	$\overline{2}$ ($= 3^2 = 9$)
3	$\overline{6}$
4	$\overline{4}$
5	$\overline{5}$ ($= 3^2 * 3^3 = \overline{2} * \overline{6} = \overline{5}$)
6	$\overline{1}$

Bsp.: $g = \overline{3}$, $h = \overline{4}$ } $l = 4$

In der Kryptografie verwendet man z.B. die Gruppe $\{ \overline{1}, \overline{2}, ..., \overline{p-1} \}$ mit der Multiplikation als Verknüpfung.
p ist eine Primzahl der Größenordnung 1024 Bit ($\sim 2^{1000}$).
Diese Gruppen werden in DSA verwendet.

1.2.1 Verschlüsselungsverfahren

Definition 1.10: (Verschlüsselungsverfahren oder Kryptosystem)
Ein Verschlüsselungsverfahren ist ein Fünftupel $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ wobei gilt:

1.) $\mathcal{P}$ ist eine Menge, die Klartextmenge ($\mathcal{P}$laintext)
2.) $\mathcal{C}$ ist eine Menge, die Schlüsseltextmenge ($\mathcal{C}$yphertext)
3.) $\mathcal{K}$ ist eine Menge, die Schlüsselmenge ($\mathcal{K}$ey)
4.) $\mathcal{E}$ ist die Menge aller Verschlüsselungsfunktionen $\mathcal{E}_e : \mathcal{P} \rightarrow \mathcal{C}$, $e \in \mathcal{K}$ ($\mathcal{E}$ncryption)
5.) $\mathcal{D}$ ist die Menge aller Entschlüsselungsfunktionen $\mathcal{D}_e : \mathcal{C} \rightarrow \mathcal{P}$, $d \in \mathcal{K}$ ($\mathcal{D}$ecryption)
6.) Für jedes $e \in \mathcal{K}$ gibt es ein $d \in \mathcal{K}$ mit: $\mathcal{D}_d(\mathcal{E}_e(m)) = m$

Beispiel: Caesar-Chiffre
$\mathcal{P}$ = { A, …, Z } = $\mathcal{C}$ = $\mathcal{K}$
= = =
{ 0, …, 25 }
$e \in \mathcal{K}$: $\mathcal{E}_e(m) = m + e \bmod 26$
$d \in \mathcal{K}$: $\mathcal{D}_d(c) = c - d \bmod 26$
$\mathcal{D}_e(\mathcal{E}_e(m)) = m$ für alle $m \leftarrow \{ 0, …, 25 \}$
Für alle $m \in \mathcal{P}$ und alle $e \in \mathcal{K}$ gibt es ein $d \in \mathcal{K}$ mit: $\mathcal{D}_d(\mathcal{E}_e(m)) = m$

Es gibt zwei Klassen von Verschlüsslungsverfahren:

1.) Symetrische Verfahren: d = e oder „d einfach aus e zu berechnen“

⇒ e darf nicht veröffentlicht werden!

2.) Asymetrische Verfahren: d ist aus e praktisch nicht berechenbar!

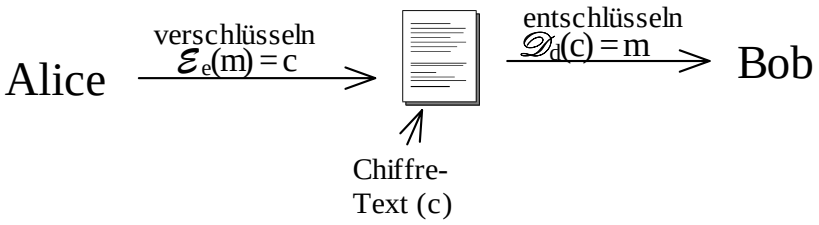
Praktisch nicht berechenbar heißt:

- ein Algorithmus rechnet zu lange
- ein Algorithmus braucht zu viel Speicher

⇒ Der Empfänger (Bob) kann e veröffentlichen

e heißt der öffentliche Schlüssel: Public Key

d heißt der private Schlüssel: Private Key



1.2.2 Hashfunktionen

Besondere Bedeutung in folgenden Kryptografischen Verfahren:

- elektronische Signaturen
- Message Authentication Codes (Symetrische elektr. Signaturen): MAC

Definition 1.11: (Hashfunktion)

Es sei N eine natürliche Zahl.

Eine Funktion

$H: \{ 0, 1 \}^* \longrightarrow \{ 0, 1 \}^N$ heißt Hashfunktion

Bitstring beliebiger Länge — Bitstring der Länge N

Definition 1.12: (Einwegfunktion)

Es sei $H: \{ 0, 1 \}^* \longrightarrow \{ 0, 1 \}^N$ eine surjektive Hashfunktion

H heißt Einwegfunktion, falls es praktisch unmöglich ist, zu einem gegebenen $s \in \{ 0, 1 \}^N$ ein $m \in \{ 0,1 \}^*$ mit H(m) = s zu finden.

Definition 1.13: (Kollision)

Es sei $H: \{ 0, 1 \}^* \longrightarrow \{ 0, 1 \}^N$ eine Hashfunktion.

Ein Tupel $(m, m') \in \{ 0, 1 \}^* \times \{ 0, 1 \}^*$, $m \neq m'$ heißt Kollision, falls H(m) = H(m') gilt.

Es sei $m \in \{ 0, 1 \}^*$. H heißt schwach kollisionsresistent für m, falls es praktisch unmöglich ist, eine Kollision (m, m') zu finden.

H heißt kollisionsresistent, falls es praktisch unmöglich ist, irgendeine Kollision zu finden.

Frage: Gibt es solche Hashfunktionen?

H ist Einwegfunktion ⇒ H ist kollisionsresistent

Antwort: Man weiß es nicht, man glaubt es aber.

Hashfunktionen für kryptografische Zwecke müssen erfüllen:
1) H(m) muss effizient berechenbar sein
2) H ist kollisionsresistent

H sei kryptografisch geeignet. Dann ist der Geburtstagsangriff die beste Attacke (d.h. „finde eine Kollision“)

m_1	$H(m_1)$
m_2	$H(m_2)$
m_k	$H(m_k)$: Gibt es m_j mit $j < k$ und $H(m_j) = H(m_k)$?

Nach $\sim 2^{N/2}$ Hashwerten finden sie mit einer Wahrscheinlichkeit von ≥ ½ eine Kollision.
Bsp.:

N = 128: Speicherbedarf: $2^{64} * 128bit = 2^{64} * 16Byte$
→ ~~MD5~~

N = 160: Speicherbedarf: $2^{80} * 16Byte$
→ SHA-1
RIPEMD-160

1.3 Angreifermodelle und Kryptoanalyse

Kerckhoffsche Prinzip:
Alles bis auf den Entschlüsselungsschlüssel d ist bekannt.
z.B.: DES, AES, RSA, …
Ist heute allgemein akzeptiert.

Konsequenz: Brute-Force- Angriff ist immer möglich. Dabei testet man alle möglichen Entschlüsselungsschlüssel d.

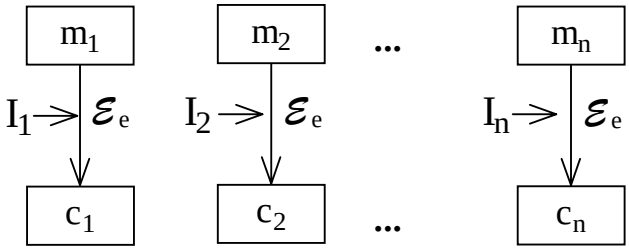
Typische Größenordnungen zum Schutz vor Brute-Force- Angriffen.
128 Bit ⇒ $½ * 2^{128} = 2^{127}$ Schlüssel müssen getestet werden.

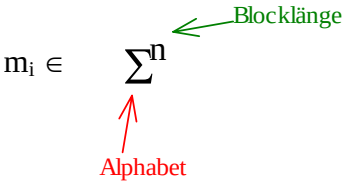
Folgende Angriffe werden unterschieden:
1.: Cyphertext-Only- Angriff: Der Angreifer kennt nur Chiffretexte. Sein Ziel: Berechne den Klartext oder d.
2.: Known-Plaintext- Attacke: Der Angreifer kennt vorgegebene Klartext-Schlüsseltextpaare (m_2, c_2), ... (m_k, c_k).
3.: Chosen-Plaintext- Attacke: Der Angreifer kann Chiffretexte zu selbst gewählten Klartexten erzeugen.
Ist bei Public-Key- Verschlüsselung immer möglich.
4.: Adaptive-Chosen-Plaintext- Attacke: Wie Chosen-Plaintext- Attacke. Die Klartexte können in Abhängigkeit von vorhergehenden Verschlüsselungsergebnissen gewählt werden.
5.: Chosen-Cyphertext- Attacke: Der Angreifer kann selbst gewählte Schlüsseltexte ungleich c_1 entschlüsseln.

Kapitel 2: Symmetrische Kryptoverfahren

2.1 Blockchiffren

Idee :
Klartext m :





Ein Alphabet ist die Grundmenge an zulässigen Zeichen.
Zum Beispiel: $\Sigma_1 = \{ A, B, C, ..., X, Y, Z \}$
$\Sigma_2 = \{ 0, 1 \}$

Definition 2.1:
Eine Blockchiffre ist ein Kryptosystem mit $\mathcal{P} = \Sigma^n = \mathcal{C}$.

Beispiel:

- DES: $\Sigma^n = \Sigma_2^n = \{0,1\}^n$; n = 64
- Caesar: $\Sigma = \Sigma_1$; n = 1

Frage: Wie viele Möglichkeiten hat man, um einen Block aus Σ^n zu verschlüsseln?
$\Rightarrow$ $|\Sigma^n|$

Wie viele unterschiedliche Blockchiffren gibt es?
$|\Sigma|^n (|\Sigma|^n - 1) * ... = |\Sigma|^n!$
$2^{64}!$

Permutationschiffre:

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

$m_1\ m_2\ m_3\ m_4 \rightarrow m_{\pi(1)}\ m_{\pi(2)}\ m_{\pi(3)}\ m_{\pi(4)} = m_4\ m_1\ m_2\ m_3$

1 1 0 1 $\rightarrow$ 1 1 1 0

2.1.2 Mehrfachverschlüsselung

Vorgaben : Blockchiffre der Blocklänge n

$\mathcal{E}(k_1, k_2, k_3) = \mathcal{E}_{k3}(\mathcal{D}_{k2}(\mathcal{E}_{k1}(m)))$

Bei Permutationschiffre kein Sicherheitsgewinn!

Aber: Im Allgemeinen gibt es kein $k \in \mathcal{K}$ mit $\mathcal{E}_k = \mathcal{E}(k_1, k_2, k_3)$

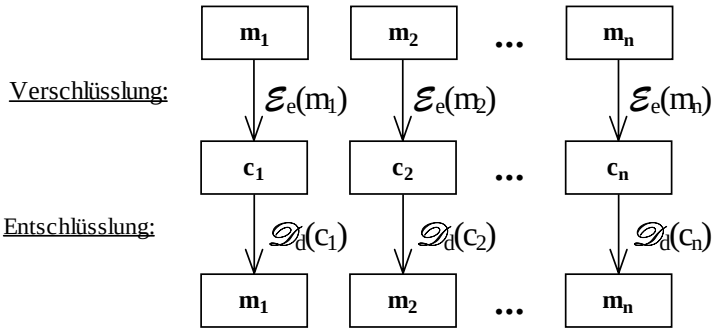
Gilt z.B. $\mathcal{K} = \{ 0, 1 \}^n$ für die Blockchiffre, so hat man bei Mehrfachverschlüsselung mit k_1, k_2, k_3 ca. 2^{3n} Schlüssel.

Beispiel: DES $\rightarrow$ DES – EDE (Tripple-DES)

Oft wird $k_1 = k_3$ gewählt. Dann gibt es ca. 2^{2n} Schlüssel bei Mehrfachverschlüsselung.

2.1.3 Modi von Blockchiffren: Gegeben ist stets eine Blockchiffre $\Sigma^n \rightarrow \Sigma^n$

Electronic Code Book Modus (ECB)



Der Schlüssel e ist für alle Blöcke gleich

d ist der zu e gehörende Entschlüsselungsschlüssel

Nachteile des ECB
Gleiche Blöcke werden bei gleichem Schlüssel e zu gleichen Chiffre-Blöcken.

$\rightarrow$ 1) Statistische Angriffe werden erleichtert (passiver Angreifer)
2) Umsortieren und Einfügen von Textblöcken (aktiver Angreifer)

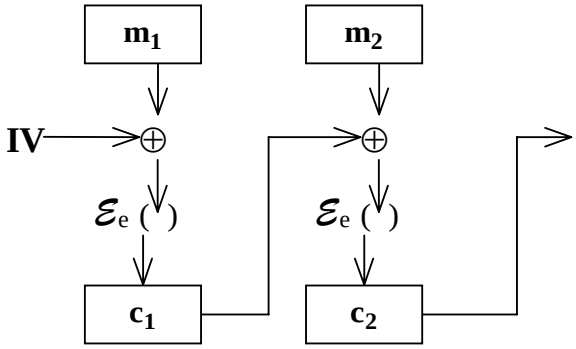
Wie kann man ECB verbessern?

1) Man unterteilt den Klartext in Blöcke der Länge $r < n$ und füllt die zu verschlüsselnden Blöcke mit n-r zufälligen Bits auf.
2) CBC – Modus

CBC – Modus

Idee :
Mache die Verschiebung kontext-abhängig.

$m = m_1, m_2, ..., m_T$
$C_i = \mathcal{E}_e (m_i \oplus C_{i-1}) \quad 1 \le i \le C$
$\delta = IV$ (Initialisierungsvektor) $\in \{ 0, 1 \}^n$



Entschlüsselung:

Sei d der zu e gehörende Entschlüsselungsschlüssel.

$\mathcal{D}_d(\mathcal{E}_e(p)) = p \quad ; p \in \mathcal{P}$

$m_i = \underbrace{\mathcal{D}_d(c_i)}_{= m_i \oplus c_{i-1}} \oplus c_{i-1} \quad 1 < i < C$

IV wird entweder mit übertragen oder der Empfänger kann bei Unkenntnis von c_0 (=IV) nur den ersten Chiffretext-Block nicht entschlüsseln.

Vorteile des CBC – Modus

- Bei unterschiedlichen IV ergeben gleiche Klartextblöcke verschiedene Chiffretextblöcke.
- Bei gleichem IV werden gleiche Blöcke an unterschiedlichen Stellen im Klartext unterschiedlich verschlüsselt.

Bsp.:

$m_1 = 1\,0\,1\,1$; $m_2 = 0\,0\,0\,1$;
IV = 1 0 1 0 $\qquad \pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix}$

$C_1 = E_\pi(m_1 \oplus 1\,0\,1\,0) = E_\pi(0\,0\,0\,1) = 0\,0\,1\,0$
$C_2 = E_\pi(m_2 \oplus C_1) = E_\pi(0\,0\,1\,1) = 0\,1\,1\,0$

Übertragungsfehler: $c_0\, c_1\, c_2 \ldots c_t$ $\Rightarrow c_1$ und c_2 kann nicht entschlüsselt werden.

Bitfehler

CFB – Modus (Cypher-Feedback- Modus)

Benötigt bei Blockchiffren mit großer Blocklänge n (~ 10.000).
Empfänger benötigt beim CFB-Mode keinen Block der Länge n zum entschlüsseln, sondern den Klartext. Chiffretextblöcke sind kürzer als n $\rightarrow$ Streamingfähigkeit

- lege „Übertragungsblocklänge“ r fest: $1 \le r \le n$
- Blockchiffre: $\{0,1\}^n \longrightarrow \{0,1\}^n$; Verschlüsselungsschlüssel e
- unterteile den Klartext in Blöcke der Länge r: $m = m_1\ m_2\ \ldots\ m_t$

r Bits

Verschlüsselung:

$I_1 = IV \in \{0,1\}^n$

Für $1 \le i \le t$:
1) $O_i = E_e(I_i) = O_{i_1}\ O_{i_2}\ O_{i_3}\ \ldots\ O_{i_n}$
2) $t_i = O_{i_1}\ O_{i_2}\ \ldots\ O_{i_r}$ (d.h.: String aus ersten r Bits von O_i)
3) $C_i = m_i \oplus t_i \Rightarrow t_i$ ist Einmalschlüssel
4) $I_{i+1} = I_i * 2^r + C_i \bmod 2^n$ (d.h.: erste r Bits von I_i löschen, C_i anhängen)

2.1.4 DES (und Triple-DES)

Ausgangspunkt: Lucifer (128Bit Chiffre)
Entwickler: Horst Feistel (IBM)
1977 als DES standardisiert (64Bit-Variante: $\mathcal{P} = \mathcal{C} = \{ 0, 1 \}^{64}$
$\mathcal{K} \subset \{ 0, 1 \}^{64}$)
Tatsächlich gibt es nur 2^{56} verschiedene Schlüssel

DES als Feistel-Chiffre
Benötigt interne Verschlüsselungsfunktion f

Blocklänge von f sei $t \in \mathbb{N}$
Schlüsselraum von f sei $\mathcal{K}_f$

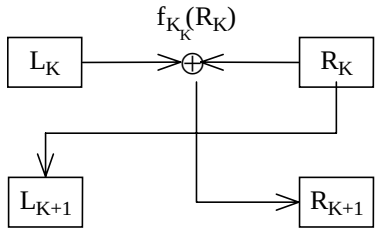
Feistel-Chiffre hat Blocklänge 2t
Feistel-Chiffre hat Schlüsselraum $\mathcal{K}$
Feistel-Chiffre hat Alphabet { 0, 1 }

Festzulegen ist Folgendes:

- Anzahl der Runden r ($r \in \mathbb{N}$)
- Algorithmus, um aus $e \in \mathcal{K}$ insgesamt r Schlüssel für f zu erzeugen:
  Rundenschlüssel $K_1, K_2, \ldots K_r$

Funktionsweise der Feistel-Chiffre
Schreibe Eingabeblock der Länge 2t als $\{ \underbrace{L_0}_{\text{t-Bits}}, \underbrace{R_0}_{\text{t-Bits}} \}$

$(L_1, R_1) = (R_0, L_0 \oplus f_{K_1}(R_0))$
$(L_2, R_2) = (R_1, L_1 \oplus f_{K_2}(R_1))$
$\vdots$
$(L_K, R_K) = (R_{K-1}, L_{K-1} \oplus f_{K_K}(R_1))$



$\mathcal{E}_e(L_0, R_0) = (R_r, L_r)$

Entschlüsselung der Feistel-Chiffre
$(R_K, L_K) = (L_{K+1}, R_{K+1} \oplus f_{K_{K+1}}(L_{K+1}))$

DES: $\mathcal{P} = \mathcal{C} = \{ 0, 1 \}^{64}$ r = 16 $\Rightarrow$ 16 Runden je 4 Operationen:

- Expansion
- XOR mit Schlüssel
- „S“-Boxen
- Permutation

Schlüssel von DES: K $\subset_{\neq}$ $\{ 0, 1 \}^{64}$

k = $\underbrace{k_1}_{8\text{Bit}}$ $\underbrace{k_2}_{8\text{Bit}}$ … $\underbrace{k_8}_{8\text{Bit}}$

$k_1 = k_{1_1} k_{1_2} \ldots k_{1_8}$: $k_{1_1} + k_{1_2} + \ldots + k_{1_8} \equiv 1 \bmod 2$

k_{1_1}: 1. Bit in k_1 $\Rightarrow$ ungerade Anzahl an 1 in k_1

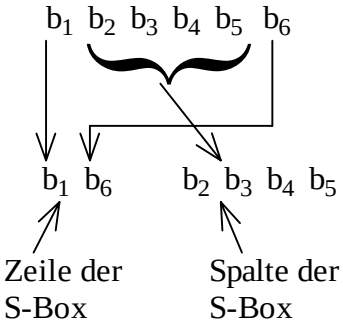
Funktion f in DES: Blockchiffre auf $\{ 0, 1 \}^{32}$ (5 Schritte)

1. Schritt: Expansion E: $\{ 0, 1 \}^{32} \longrightarrow \{ 0, 1 \}^{48}$

2. Schritt: XOR mit Rundenschlüssel der Länge 48Bit

3. Schritt: Bestimme Zeilen & Spalten der S-Boxen aus den 8 Bitstrings der Länge 6Bit

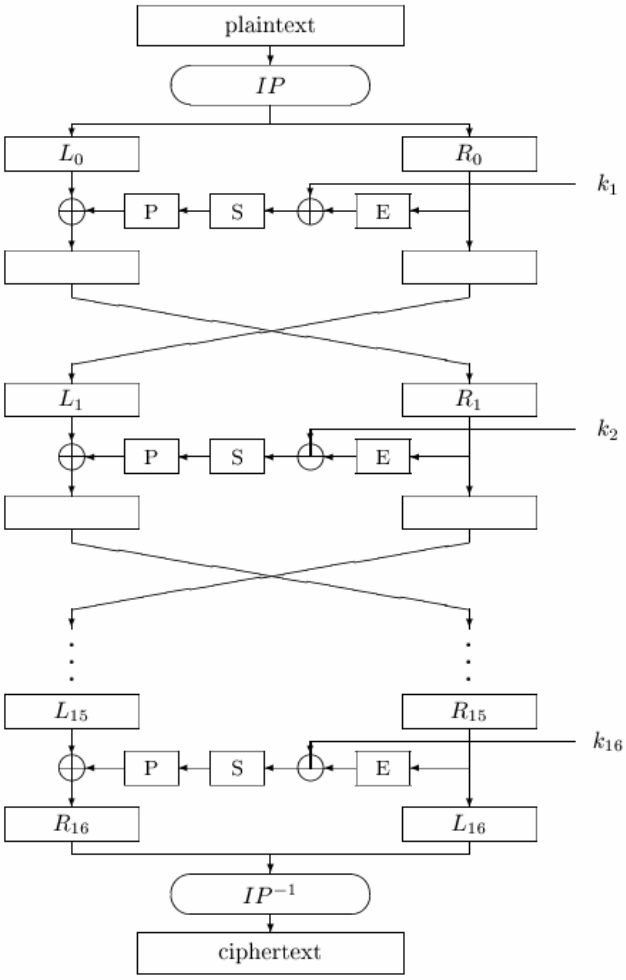
Eingabe:



4. Schritt: Wende die 8 S-Boxen an **Nicht linear**

5. Schritt: Permutation P: $\{ 0, 1 \}^{32} \longrightarrow \{ 0, 1 \}^{32}$

Aus Vorlesung von Nigel Smart:



Sicherheit von DES:

- PC: Abschätzung: 2^{19} Schlüssel pro Sekunde

$\Rightarrow \frac{2^{55}}{2^{19}} s = 2^{36} s \approx 2.000 Jahre$ (bei 2.000 PCs entsprechend 1 Jahr)

- Spezielle Hardware:
  - FPGA (Hardwarearchitektur in der ich etwas programmieren kann)
    Kosten: ~ 10.000 $
    $\Rightarrow \leq 2$ Jahre
  - ASIC
    Kosten: ~ 1 Mio. $
    $\Rightarrow$ ca. 30 min

$\Rightarrow$ Verwende größere Schlüssellängen

Triple-DES: Mehrfachverschlüsselung (EDE) mit Hilfe von DES:

$\mathcal{E} = \mathcal{E}_{k_3}$ o $\mathcal{D}_{k_2}$ o $\mathcal{E}_{k_1}$ (k_1, k_2, k_3 sind DES-Schlüssel)

2 verschiedene Realisierungen

1) k_1, k_2, k_3 alle unterschiedlich $\Rightarrow$ effektive Schlüssellänge: 168Bit (3*56Bit)

2) $k_1 = k_3$ $\Rightarrow$ effektive Schlüssellänge: 112Bit (2*56Bit)

Anforderungen an Kandidaten für AES:

1) Symmetrische Blockchiffre
2) 128Bit Blocklänge
3) 3 mögliche Schlüssellängen: 128Bit, 192Bit, 256Bit
4) Mindestens so sicher wie Triple-DES, aber wesentlich effizienter
5) Mindesthaltbarkeit: 30 Jahre
6) Sensitive Daten sollen 100 Jahre geschützt sein

Literaturhinweis: A. Leustra, E. Verheul: Selecting cryptographic key sizes

Mehrfachverschlüsselung:

$\mathcal{E}_{k_2}(\mathcal{E}_{k_1}(m)) : |\mathcal{K}| = 2^m \Rightarrow |\mathcal{K}_n| = 2^{2m}$

↑ Blockchiffre ↖ Mehrfachverschlüsselung

Meet-in-the-Middle-Attacke: Man berechnet im Schnitt 2^m Chiffretexte.
Man speichert aber 2^m Chiffretext-Schlüssel-Paare: (k, m, c)
Gegeben ist ein Paar (m_1, c_1) mit $\mathcal{E}_{k_2}(\mathcal{E}_{k_1}(m_1)) = c_1$

Idee: Berechne $\mathcal{E}_k(m)$ für alle $k \in \mathcal{K}$ und speichere (k, $\mathcal{E}_k(m)$) : 2^m Einträge
Berechne $D_k(c_1)$ für alle $k \in \mathcal{K}$ und speichere (k, $\mathcal{D}_k(c_1)$) : 2^m Einträge

$\Rightarrow$ Wenigstens Dreifachverschlüsselung
E D E : $\mathcal{E}_k(\mathcal{D}_k(\mathcal{E}_k(m))) = \mathcal{E}_k(m)$ statt E E E aus Gründen der Abwärtskompatibilität

FIPS 197 : Offener Standard, kostenlos verfügbar
↑
AES wird durch dieses Dokument standardisiert

Datentypen in Rijndael:

Bit : { 0, 1 }
Byte : Bit-Array der Länge 8: $b_7\ b_6 \ldots b_1\ b_0$
Word : Bit-Array der Länge 32: $w_0\ w_1 \ldots w_{30}\ w_{31}$
Nb : Blocklänge in Worten: AES: $Nb = \frac{128}{32} = 4$
Rijndael: $4 \le Nb \le 8$
Nk : Länge des Schlüssels in Worten: AES: Nk = 4, 6, 8
Rijndael: $4 \le Nk \le 8$
Nr : Anzahl der Runden: $Nr = \max\{Nb, Nk\}+6$

Für AES: 10 Runden (Nk = 4)
12 Runden (Nk = 6)
14 Runden (Nk = 8)

AES operiert im Wesentlichen auf Bytes

Rechenoperationen auf Bytes in AES/Rijndael:
$\oplus$: XOR $b \oplus c = b_7\ b_6 \ldots b_1\ b_0 \oplus c_7\ c_6 \ldots c_1\ c_0 = b_7 \oplus c_7, b_6 \oplus c_6, \ldots, b_0 \oplus c_0$

Ergänzung: Bytemultiplikation mit Polynomen

$b = b_7\, b_6 \ldots b_1\, b_0 \triangleq b_7X^7 + b_6X^6 + \ldots + b_1X + b_0$

$c = c7\, c_6 \ldots c_1\, c_0 \triangleq c_7X^7 + c_6X^6 + \ldots + c_1X + c_0$

$b \bullet c = (b*c) \bmod m(x)$, $m(x) \triangleq X^8 + X^4 + X^3 + X + 1$

$b*c = q\, m + r$

$b \bullet c := r$

Die Menge der Bytes mit den Verknüpfungen $\oplus$ und $\bullet$ ist ein Körper

$b = \{57\} = 0101\ 0111$

$c = \{83\} = 1000\ 0011$

$b \oplus c = 1101\ 0100$, $b \bullet c = \{C1\} = 1100\ 0001$

$\{03\}^{-1} = \{F6\}$

$\{03\} \bullet \{F6\} = \{01\}$

<u>4 Teilfunktionen in Rijndeal:</u>

1) Add Round Key:

$s \oplus r$

state: Nb Worte — Rundenschlüssel: Nb Worte

Noch zu klären: Wie erhält man den Rundenschlüssel

2) Mix Columns: state $s = s_0\, s_1 \ldots s_{127}$

$s_{0,0} = s_0\ s_1\ \ldots s_7$

$s_{1,0} = s_8\ s_9\ \ldots s_{15}$

$s_{2,0} = s_{16}\, s_{17} \ldots s_{23}$

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

$$\mathbf{A} = \begin{pmatrix} \{02\} & \{03\} & \{01\} & \{01\} \\ \{01\} & \{02\} & \{03\} & \{01\} \\ \{01\} & \{01\} & \{02\} & \{03\} \\ \{03\} & \{01\} & \{01\} & \{02\} \end{pmatrix}$$

$s \mapsto A * s$

3) Shift Rows: Zyklischer Linksschift der Zeilen:

AES:
1.Zeile: 0 Positionen
2.Zeile: 1 Position
3.Zeile: 2 Positionen
4.Zeile: 3 Positionen

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

4) Sub Bytes:

S-Box: $s_{i,j} \mapsto s'_{i,j}$

Permutation auf der Menge der Bytes (mit 256 Elementen)

$s_{i,j} \mapsto$ aff (inv($s_{i,j}$))

$b \mapsto b^{-1}$ (b /= {00})

$0 \mapsto 0$

{00} $\mapsto$ 0110 0011 = {63}

2.2 Message Authentication Codes (MACs)

Auch: symmetrische elektronische Signaturen

!

Sicherheitsziele die mit MACs erreicht werden:

- Vertraulichkeit - nicht durch symmetrische MACs erreicht
- Authentizität - wird durch symmetrische MACs erreicht
- Integrität - wird durch symmetrische MACs erreicht
- Verbindlichkeit - nicht durch symmetrische MACs erreicht

Eingabe: zu signierendes Dokument m
kryptografischer Schlüssel k

Ausgabe: MAC

Möglichkeiten:

- DES_k (H(m))
- H (k || m)
- Verschlüssele m im CBC-Mode mit einer Blockchiffre unter Verwendung von k.
  MAC ist letzter, verschlüsselter Block
  Dieser hängt von allen Blöcken ab.

H( ) : Hashfunktion
"||" : Konkatination = aneinander hängen

Kapitel 3: Public Key Verfahren

3.1 Das RSA-Verfahren

RSA-Verschlüsselung:

1) Setup (Erzeugung des Schlüsselpaars beim Empfänger Bob)
2) Verschlüsselung mit Hilfe von Bobs öffentlichem Schlüssel
3) Entschlüsselung des Chiffretextes mit Hilfe von Bobs geheimen Schlüssel

RSA-Setup

1) Finde Primzahlen p und q mit:
   a) $p \neq q$
   b) $p > 2, \quad q > 2$
   c) $p*q \approx 2^k$ Je nach Sicherheitsniveau gilt: k = 512, 1024, 1536, 2048
   d) $\varepsilon_1 < |\log_2 p - \log_2 q| < \varepsilon_2$ Abstand der Schlüssel in Bits
      $\varepsilon_1 = 0{,}1$ Minimalabstand *BSI-Empfehlung*
      $\varepsilon_2 = 30$ Maximalabstand

2) Berechne $n = p * q$ Dann heißt n eine RSA-Zahl

3) Finde e mit: $1 < e < (p-1)(q-1)$
   $e \in \mathbb{N}$ $ggt(e, (p-1)(q-1)) = 1$ Heute: $e = 2^{16}+1$

4) Berechne ein d, $1 < d < (p-1)(q-1)$ mit

   $e * d \equiv 1 \bmod (p-1)(q-1)$

   Verschlüsseln: $C \equiv m^e \bmod n$ (effizient, da e klein)
   Entschlüsseln: $m \equiv c^d \bmod n$

5) Veröffentliche (n,e): Das ist Bobs öffentlicher Schlüssel (public Key)

6) Halte d geheim: Das ist Bobs privater Schlüssel (private Key)

   Beispiel:
   $n = 5 * 7 = 35$; $e = 5$; $m = 8$
   $8^5 = 8^{2^2+1} = 8^{2^2} * 8$
   $8^2 = 64 \equiv -6 \bmod 35$
   $8^{2^2} \equiv (-6)^2 = 36 \equiv 1 \bmod 35$
   $\Rightarrow c \equiv 1 * 8 = 8 \bmod 35$

7) Es gilt $p = \{ 0, 1, \ldots, n-1 \} = c$

Die RSA-Verschlüsselung hat die Eigenschaft
$(m^e)^d \equiv m \bmod n$ für alle $0 \leq m \leq n-1$

RSA-Signatur

1) Setup: Alice erzeugt (m, e_A) sowie d_A
2) Signaturerzeugung: $S \equiv H(m)^d \bmod n_A$
3) Signaturprüfung: $H(m) \equiv S^{e_A} \bmod n_A$

Alice signiert mit ihrem privaten Schlüssel das Dokument m.
Sie verschickt m und S.
Bob prüft diese Signatur mit Hilfe von (n_A, e_A), S und H.

Algorithmus zum Brechen von RSA:

1) Faktorisiere n (Finde p & q)
2) Berechne (p-1)*(q-1)
3) Berechne d mit $d * e \equiv 1 \bmod (p-1)(q-1)$
4) Berechne $c^d \bmod n$

Wie berechnet man d?
Mit Hilfe des Euklidischen Algorithmus.

Beispiel:

$$\left.\begin{array}{r}(p-1)(q-1) = 10.600 \\ e = 17\end{array}\right\} \text{ ggT (} (p-1)(q-1), e) = 1$$

$10.600 = 623 * 17 + 9$
$17 = 1 * 9 + 8$
$9 = 1 * 8 + 1$
$8 = 8 * 1 + 0$

zum berechnen: das Ganze rückwärts

$1 = 9 - 1 * 8$
$= 9 - 1 (17 - 1 * 9)$
$= 2 * 9 - 1 * 17$
$= 2 (10.600 - 623 * 17) - 1 * 17$
$= 2 * 10.600 - 1247 * 17$
$\Rightarrow d \equiv -1247 \bmod 10.600$

zur Verdeutlichung: $-1247 * 17 - 1 = -2 * 10.600$
entspricht $d * e \equiv 1 \bmod (p-1)(q-1)$ (siehe 3)

Wie sicher ist RSA?
RSA-Problem: Berechne e-te Wurzel mod n

Heute wird das RSA-Problem wie oben geschildert gelöst.
$\Rightarrow$ das RSA-Problem ist höchstens so schwer wie das Faktorisierungsproblem.

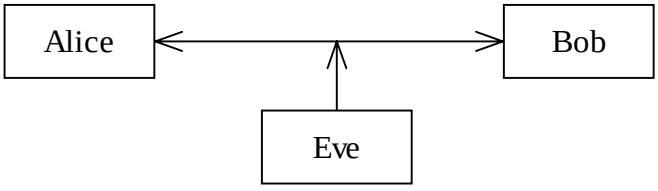
Wie schwer ist das Faktorisierungsproblem?
Bestes Verfahren: Zahlkörpersieb (Number Field Sieve)
RSA-Faktorisierungsrekord: $n \approx 2^{576}$
Benchmark: RSA-Challenge

Und in Zukunft?
- Bessere Faktorisierungsalgorithmen?
- Schnellere konventionelle Hardware
- Quantencomputer

3.2 Weitere Public-Key- Verfahren

3.2.1 Der Diffie-Hellman- Schlüsselaustausch

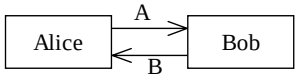
1976 Veröffentlicht
Idee: Tausche symmetrischen Schlüssel über eine unsichere Leitung aus.



passiver oder
aktiver Angreifer

Alice und Bob einigen sich auf eine Primzahl p und eine Zahl g
mit $\{ g^x \bmod n : x \in \mathbb{N} \} = \{ 1, 2, \ldots, p-1 \}$

Alice wählt zufällig a mit $1 \le a \le p-1$. Sie berechnet $A \equiv g^a \bmod p$.
Bob wählt zufällig b mit $1 \le b \le p-1$. Er berechnet $A \equiv g^b \bmod p$.



Alice berechnet $B^a \equiv (g^b)^a \equiv g^{a*b} \bmod p$
Bob berechnet $A^b \equiv (g^a)^b \equiv g^{a*b} \bmod p$

Die Zahl $g^{a*b} \bmod p$ ist das gemeinsame Geheimnis.
Daraus wird der symmetrische Schlüssel abgeleitet.

Eve kennt p, g, A, B.
Sie kann $A * B \equiv g^a * g^b \equiv g^{a+b} \bmod p$ berechnen.

Diffie-Hellman-Problem: Bestimme $g^{a*b} \bmod p$ aus g, p, A und B

Heute: Effizienteste Methode zur Lösung des Diffie-Hellman-Problem zu lösen, ist die Lösung des DLP (diskretes Logarithmus Problem):
Berechne a aus Kenntnis von p, g und A.

Wie schwer ist das DLP?
Für $p \approx 2^k$ ist dieses DLP so schwer wie das Faktorisierungsproblem für $n \approx 2^k$.
Das beste Verfahren ist das Zahlkörpersieb!
$\Rightarrow$ gleiche Größenordnung wie bei RSA

Aktiver Angreifer:
Man-in-the-Middle (heute auch oft: Monkey-in-the-Middle)

Lösung durch Authentifikation

Beispiel zu DH-Schlüsselaustausch:

Alice $\xleftrightarrow{P,g}$ Bob

Alice und Bob suchen kleinste Primzahl der Länge 9 Bit $p = 2^8 = 256$

p = 257 : Wir testen, ob p durch Primzahlen $2 \le q \le \sqrt{p}$ teilbar ist.

2 X 257, 3 X 257, 5 X 257, 7 X 257, 11 X 257, 13 X 257

$\Rightarrow$ p = 257 ist die gesuchte Primzahl
Alice und Bob suchen die kleinste Zahl g
mit $2 \le g \le p-1$ und $\{ g^k \bmod p;\ 1 \le k \le p-1 \} = \{ 1, 2, 3, \ldots, p-1 \}$

Versuch: g = 2
$p = 2^8 + 1 = g^8 - (-1)$ $\Rightarrow g^8 \bmod p = p-1$ $(g^8 \equiv -1 \bmod 257)$
$\Rightarrow g^{16} \equiv 1 \bmod 257$
$\Rightarrow g^{17} \equiv g \equiv 2 \bmod 257$

g = 3 ✓
$\Rightarrow$ Parameter: p = 257, g = 3

Alice wählt a = 12
Sie berechnet $A \equiv g^a \bmod p$
aber: $3^{12} \equiv 3^{2^3+2^2} \equiv 3^{2^3} * 3^{2^2} \bmod 257$
also: $3^2 \equiv 9 \bmod 257$
$3^{2^2} \equiv 9^2 \equiv 81 \bmod 257$
$3^{2^3} \equiv 81^2 \equiv 136 \bmod 257$
$\Rightarrow A \equiv 136 * 81 \equiv 222 \bmod 257$

Bob wählt b = 120
$3^{120} \equiv 3^{2^6} * 3^{2^5} * 3^{2^4} * 3^{2^3} \bmod 257$
$\Rightarrow B \equiv 17 \bmod 257$

Alice schickt A an Bob!
Alice berechnet:
$B^a \equiv 17^{12} \equiv 17^{2^3} * 17^{2^2} \equiv 193 \bmod 257$

Bob schickt B an Alice!
Bob berechnet:
$A^b \equiv 222^{120} \equiv \ldots \equiv 193 \bmod 257$

$\Rightarrow$ K = 193 ist das gemeinsame Geheimnis

Angreifer Eve kennt p, g, A, B. Er kann mit dieser Kenntnis K nicht berechnen.
Eve kann nur $A*B = g^{a+b} \bmod p$ berechnen.

3.2.2 DSA – Digital Signiture Algorithm

- Nach RSA das zweitwichtigste Signaturverfahren.
- Standardisiert im FIPS 186-2 (2000, Revision von FIPS 186 aus 1994)
- Sicherheit beruht auf DLP in $\mathbb{Z}_p$ (1 mod p, 2 mod p, …, p-1 mod p)
- Unterschied zu RSA – Signaturen: Signatur wird von Zufallszahl mitbestimmt.
- FIPS 186 legt SHA-1 als Hashfunktion fest

Setup (Erzeugen des Schlüsselpaars):

1) Wähle zufällig eine Primzahl q mit $2^{159} < q < 2^{160}$ („kleine“ Primzahl)
2) Wähle Größenordnung der „großen“ Primzahl p: $0 \le t \le 8$, $2^{511+64t} < p < 2^{512+64t}$
   $\Rightarrow$ größte Möglichkeit für p: 1024 Bit, Suche p in dieser Größenordnung mit q / p-1
3) Suche g mit $\{ g^k \bmod p: 1 \le k \le p-1 \} = \{ 1, 2, \ldots, p-1 \}$
4) Berechne $\alpha \equiv g^{\frac{p-1}{q}} \bmod p$
5) Wähle zufällig a mit $1 \le a \le q-1$
6) Berechne $A \equiv \alpha^a \bmod p$
7) Veröffentliche p, q, g, α, A. Halte a geheim. $\Rightarrow$ Ein Angreifer, der a aus den öffentlichen Informationen berechnen kann, muss das DLP $A \equiv \alpha^a \bmod p$ lösen.

Signaturerzeugung: Beachte: Hashfunktion H ist SHA-1

1) Wähle eine Zufallszahl k mit $1 \le k \le q-1$
2) Berechne $r \equiv (\alpha^k \bmod p) \bmod q$. Falls r = 0, gehe zu 1) — unabhängig von m und a
3) Berechne $s \equiv \frac{H(m) + r * a}{k} \bmod q$ Gilt s = 0, gehe zu 1) — m: Dokument, a: privater Schlüssel
4) Die Signatur zu m ist (r, s). Die Signatur hat Bitlänge 2 * 160 = 320 Bit.

Beobachtung:
Signiert Alice das gleiche Dokument m zwei Mal mit DSA, dann sind die Signaturen verschieden.

Verifikation:

$$k \equiv \frac{H(m) + r * a}{s} \quad : \quad \alpha^k = \alpha^{\frac{H(m)}{s} \bmod q} * A^{\frac{r}{s} \bmod q}$$

Privater Schlüssel muss (bzw. sollte) in nicht- auslesbarem Medium gespeichert werden:
- Chipkarte
- Hardware Security Module

Wie wird eine Signatur in der Praxis erstellt?

Szenario: Online-Banking mit Chipkarte $\Rightarrow$ Sie benötigen Kartenleser

1) Bank fordert Signatur an über ein Dokument m.
2) PC berechnet H(m) und schickt diesen an die Karte (über den Kartenleser).
3) Karte berechnet Signatur über H(m).
4) Karte schickt Signatur an PC
5) PC übermittelt Signatur an Banking-Server