1 Mathematische Grundlagen

$$x^k \bmod p = (x \bmod p)^{k \bmod \varphi(p)} \bmod p$$

$-83 \bmod 12 = 1 \bmod 12$
denn $-83 \div 12 = 6$;R=11 und $12 - 11 = 1$

1.1 φ-Funktion

Die Eulersche φ-Funktion gibt die für eine Zahl n mit Primfaktorzerlegung

$$n = p_1^{k_1} \cdot p_2^{k_2} \cdot \ldots \cdot p_r^{k_r}$$

(p_i sind die Primfaktoren, k deren Anzahl als Potenzschreibweise) an, wie viel ganze Zahlen teilerfremd zu n sind:

$$\varphi(n) = n \prod_{p|n}^{r} \left(1 - \frac{1}{p}\right) \qquad \sum_{d|n} \varphi(d) = n$$

$$\varphi(p) = p - 1 \qquad \varphi(p^k) = p^{k-1}(p-1)$$

Beispiel: $\varphi(72) = \varphi(2^3 \cdot 3^2) = 2^{3-1} \cdot (2-1) \cdot 3^{2-1} \cdot (3-1) = 2^2 \cdot 1 \cdot 3 \cdot 2 = 24$

1.2 XOR-Operation

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

1.3 Diskrete Logarithmus

Der diskrete Logarithmus ist die kleinste Lösung x der Gleichung $a^x \equiv m \mod p$ mit $m,\ a \in \mathbb{N}$, $p \in \mathbb{Z}_p$. Da sich die diskrete Exponentiation leicht berechnen lässt, während für die Umkehrfunktion, den diskreten Logarithmus, meist nur Algorithmen mit polynomialer Laufzeit bekannt sind, wird der Diskrete Logarithmus u. a. im Diffie–Hellman-Key-Exchange, ElGamal-Encryption, Digital Signature Algorithm eingesetzt.

1.4 Schnelles Potenzieren

Seien $a, p, m \in \mathbb{N}$, gesucht ist $e = a^p \mod m$

1. Berechne die Binärdarst. von $p_{10} = b_2$
2. Nun geht man wie folgt vor: Für die erste 1 die Basis a hinschreiben, für weitere $1 \to)^2 \cdot a$, für folgende $0 \to)^2$

Beispiel: $a^p \mod m$ ist a=3, p=19, m=23. Die Binärdarst. der Zahl p ist $(10011)_2$.

$$(((3^2)^2)^2 \cdot 3)^2 \cdot 3 \equiv 6 \mod 23$$

1.5 Chinesischer Restsatz

Sind $m_1, \ldots, m_n \in \mathbb{N}$ paarweise teilerfremd, dann hat das System von Kongruenzen

$$\begin{aligned} x &= a_1 \mod m_1 \\ &\vdots \\ x &= a_n \mod m_n \end{aligned}$$

eine eindeutige Lösung $x \in \mathbb{Z}_m$, wobei $m = m_1 \cdot \cdots \cdot m_n$ das Produkt der einzelnen Module ist. Die Lösung lautet

$$x = \left(\sum_i a_i \cdot M_i \cdot N_i\right) \mod m$$

mit folgenden Voraussetzungen:

1. $m = m_1 \cdot \cdots \cdot m_n$
2. $M_i = \frac{m}{m_i}$
3. $N_i = M_i^{-1} \mod m_i$ hierfür Erweiteter Euklid verwenden.

1.6 Erweit. Euklidischer Algorithmus

- Berechnet den $\mathrm{ggT}(a, b)$, wenn $a, b \in \mathbb{N}$
- Berechnet $a \cdot b + n \cdot k = \mathrm{ggT}(a, b)$
- Berechnet $a^{-1} \mod m$
- Besonders interessant für $\mathrm{ggT}(a, b) = 1$, da dann a und b teilerfremd sind.

$$a \cdot b \equiv 1 \mod n$$

oder in einer anderen Schreibw., die uns dann zum Erweit. Euklidischen Algorithmus führt:

$$a \cdot b + k \cdot n = 1$$

1.6.1 Zahlenbeispiel

$e \cdot 85 + k \cdot 352 = 1$
$352 = 4 \cdot 85 + 12 \Leftrightarrow 12 = 352 - 4 \cdot 85$
$85 = 7 \cdot 12 + 1 \Leftrightarrow 1 = 85 - 7 \cdot 12$
$1 = 1 \cdot 85 - 7 \cdot 12$
$1 = 1 \cdot 85 - 7 \cdot (352 - 4 \cdot 85)$
$1 = -7 \cdot 352 + 29 \cdot 85$

1.7 Primitive Wurzel

$\mathbb{Z}_n^*$ hat genau dann Primitivwurzeln, wenn $n = 2, 4, p^k$ oder $2 \cdot p^k$ mit p eine Primzahl ist und $k \geq 1$. Die Anzahl sind dann genau $\varphi(\varphi(n))$.

Primitivwurzeltest Um festzustellen, ob eine Zahl g eine primitive Wurzel von $\mathbb{Z}_p^*$ ist (p ist prim), führe man folgende Schritte aus:

1. Finde die Primfaktorzerlegung von $p - 1$: $p - 1 = p_1 \cdot \ldots \cdot p_n$
2. Wähle $q \in \{p_1, \ldots, p_n\}$
3. Falls nun gilt

   $$\boxed{g^{(p-1)/q} \not\equiv 1 \mod p}$$

   für alle Primfaktoren q von $p-1$, dann ist g eine primitive Wurzel, sonst nicht. Für eine Primzahl p gibt es $\varphi(p-1)$ primitive Wurzel.

Falls g eine Primitivwurzel von $\mathbb{Z}_n^*$ ist, dann ist auch $b = g^i \bmod n$ eine Primitivwurzel von $\mathbb{Z}_n^*$ genau dann wenn i teilerfremd zu $\varphi(n)$ ist $(ggT(i, \varphi(n)) = 1)$. Daraus folgt: hat man schon eine Primitivwurzel gefunden, dann potenziere sie mit jeder Zahl die teilfremd zu $\varphi(n)$ ist:

$$\mathrm{ggT}(i, \varphi(n) = 1) \Rightarrow \langle g^i \rangle = \mathbb{Z}_n^*$$

1.8 Miller-Rabin

Ist n prim? Schreibe $n-1$ als $2^s \cdot d$ mit ungeradem d. Wähle a teilerfremd zu n und kleiner als n.

$$\mathrm{ggT}(a, n) = 1,\ a < n$$

Falls beide folgenden Bedingungen erfüllt sind, dann ist a Zeuge, dass n keine Primzahl ist:

$$\begin{aligned} a^d &\not\equiv \pm 1 \bmod n \\ a^{2^r d} &\not\equiv -1 \bmod n \quad \forall r \in [1, s-1] \end{aligned}$$

Data: Zufallszahlen $a \in [2, p-2]$, auf Primalität zu testende Zahl p
Result: Ob a Zeuge für oder gegen die Primalität von p

```
Zerlege p − 1 als 2^s · d wo d ungerade ist;
Rechne z = a^d mod p;
if z ≡ ±1 mod p then
    return a kein Zeuge und p
      wahrscheinlich prim;
end
while Wiederholungen ≤ s − 1 do
    z = z^2 mod p;
    if z ≡ 1 mod p then
        return p zusammengesetzt und a
          Zeuge hierfür;
    if z ≡ −1 mod p then
        return a kein Zeuge und p
          wahrscheinlich prim;
    end
    Wiederholungen + +;
end
return a ist Zeuge gegen die Primalität
  von p ⇒ p keine Primzahl und
  zusammengesetzt;
```

1.8.1 Irrtumswahrscheinlichkeit

höchstens $\frac{1}{4^x}$ für x Versuche.

1.9 Geburtstagsparadoxon

Die Wahrscheinlichkeit für eine Kollision ist $1 - p$, und $1 - p \geq \frac{1}{2}$, wenn

$$k \geq \frac{1}{2}\left(\sqrt{1 + 8 \cdot \ln 2 \cdot s} + 1\right) \approx 1,18 \cdot \sqrt{s}$$

wobei z.B. $s = 365$ oder $s = 2^n$ und k die Mindestanzahl der nötigen Personen oder Hashwerte.
Wenn man mehr als $2^{n/2}$ viele Hashwerte bildet, findet die Geburtstagsattacke mit Wahrscheinlichkeit $\geq \frac{1}{2}$ eine Kollision. Um die Geburtstagsattacke zu verhindern, muss man n so groß wählen, dass es unmöglich ist, $2^{n/2}$ Hashwerte zu berechnen und speichern.

1.10 Regel von Laplace

$$\text{Wahrscheinlichkeit} = \frac{\text{Anz. der günstigen F.}}{\text{Anz. aller Fälle}}$$

2 Verschlüsselungsalgorithmen

2.1 Assymetrische Verfahren

2.1.1 RSA

Schlüssel erzeugen

1. Wähle p, q Primzahlen
2. Setze $n = p \cdot q$
3. $\varphi(n) = (p-1)(q-1)$
4. $e \cdot d + k \cdot \varphi(n) = 1 = \mathrm{ggT}(e, \varphi(n)) \wedge 1 < e < \varphi(n)$
5. Berechne d als Inverses von e modulo $\varphi(n)$, also $e \cdot d \equiv 1 \bmod \varphi(n)$
6. (n, e) Public Key
7. (n, d) Private Key

Verschlüsseln $c = m^e \mod n$

Entschlüsseln $m = c^d \mod n$

Signieren $s = m^d \mod n$

Verifizieren $m = s^e \mod n$

Siginieren und Entschlüsseln mit Chinesischem Restsatz

1. $d_p = d \mod (p-1)$
2. $d_q = d \mod (q-1)$
3. $q_{inv} = q^{-1} \mod p$
4.

Signieren	Entschlüsseln
$s_1 = m^{d_p} \mod p$	$m_1 = c^{d_p} \mod p$
$s_2 = m^{d_q} \mod q$	$m_2 = c^{d_q} \mod q$

5. (a) falls $m_1 > m_2$:
$$h = q_{inv} \cdot (m_1 - m_2) \mod p$$
falls $s_1 > s_2$:
$$h = q_{inv} \cdot (s_1 - s_2) \mod p$$
(b) falls $m_1 < m_2$:
$$h = q_{inv} \cdot (m_1 + p - m_2) \mod p$$
falls $s_1 < s_2$:
$$h = q_{inv} \cdot (s_1 + p - s_2) \mod p$$
6.

Signieren	Entschlüsseln
$s = s_2 + (h \cdot q)$	$m = m_2 + (h \cdot q)$

Common-Modulus-Attack Gegeben sei: RSA-Key$_1$: (n, e_1) und RSA-Key$_2$: (n, e_2)

1. Rechne $\alpha = ggT(c_1, n) \wedge \beta = ggT(c_2, n)$
2. Falls $\alpha | \beta \neq 1$ kann man sofort den anderen Pimfaktor von n bestimmen und entschlüssen.
3. Wenn $\alpha | \beta = 1$ bestimme s_1, s_2 mit $e_1 \cdot s_1 + e_1 \cdot s_2 = 1$ (Erweiterter Euklid)
4. Es gilt: $c_1^{s_1} \cdot c_2^{s_2} = (m^{e_1})^{s_1} \cdot (m^{e_2})^{s_2}$
5. Beweise das gilt: $m^{e_1 \cdot s_1} \cdot m^{e_2 \cdot s_2} = m^1$ (mit E.-Euklid: $e_1 \cdot s_1 + e_1 \cdot s_2 = 1$)
6. Somit folgt: $c_1^{s_1} \cdot c_2^{s_2} = m$

Low-Encryption-Exponent-Attack
Folgende Voraussetzungen müssen gelten:

1. (n_1, e), (n_2, e), (n_k, e) als öffentliche Schlüssel mit $k \in \mathbb{N}$
2. e ist klein und für alle gleich
3. n_i sind teilerfremd zueinander
4. Dieselbe Nachricht m wird an alle verschickt
5. alle Kryptogramme $c_i = m^e \mod n_i$ sind bekannt.

$$\begin{aligned} x &= c_1 \mod n_1 \\ x &= c_2 \mod n_2 \\ &\vdots \\ x &= c_k \mod n_k \end{aligned}$$

$$\boxed{\rightarrow m = \sqrt[e]{x}}$$

Angriff durch Primfaktorzerlegung

$$n = \underbrace{(x+y)}_{p} \cdot \underbrace{(x-y)}_{q} = x^2 - y^2$$

besonders schnell, denn wenn $p \approx q$, dann p, $q \approx \sqrt{n} \approx x + y$ mit einem sehr kleinen y.

Erraten eines Primfaktors Eve wendet den Euklid auf alle Kryptogramme von Bob um $ggT(c, n)$ zu berechnen. Ist der $ggT \neq 1$ ist ein Primfaktor gefunden dies ist möglich wenn c aus der Menge der vielfachen von p und oder q ist. Die Wahrscheinlichkeit so einen Primfaktor zu finden ist $\frac{p+q}{p \cdot q}$

Small-Massage-Space Attacke Bei einem kleinen Nachrichtenraum, kann ein Angreifer alle möglichen Kryptogramme berechnen und anschließend abgefangene Nachrichten mit den bereits berechneten Kryptogrammen vergleichen; durch den Einsatz von OAEP (Optimal Asymmetric Encryption Padding) kann dies verhindert werden.

2.1.2 ElGamal

Schlüsselerzeugung

1. Wähle eine Primzahl p und eine primitive Wurzel g von $\mathbb{Z}_p$
2. Wähle eine zufällige Zahl $x \in \{1, \ldots, p-2\}$. Diese Zahl ist der geheime Exponent und der eigentlich private Schlüssel.
3. Berechne $y = g^x \bmod p$
4. Die drei Zahlen (p, g, x) sind gemeinsam der private Schlüssel
5. Die drei Zahlen (p, g, y) sind der öffentliche Schlüssel

Verschlüsseln von m

1. Wähle k zufällig aus dem Intervall $[1, p-2]$
2. $c_1 = g^k \mod p$ (g ist öffentlich)
3. $c_2 = y^k \cdot m \mod p$ (y ist öffentlich)
4. das Paar (c_1, c_2) ist das Kryptogramm

Entschlüsseln $m = c_1^{p-1-x} \cdot c_2 \mod p$

Signieren

1. wähle k zufällig aus dem Intervall $[1, p-2]$.
2. $ggT(k, \varphi(p)) = 1$
3. $r = g^k \pmod p$
4. $s = k^{-1}(m - r \cdot x) \pmod{p-1}$
5. (m, r, s) ist die Signatur

Verifizieren Alle Forderungen für (m, r, s) müssen gelten:

- $1 \leq r \leq p-1$ (p ist öffentlich)
- $1 \leq s \leq p-1$
- Es muss $v = w$ gelten mit
  1. $v = g^m$
  2. $w = y^r \cdot r^s$ (y ist öffentlich)
- (alternativ: $g^{h(m)} \equiv y^r \cdot r^s \bmod p$)

Attacke der Signatur falls Alice die gleiche Zufallszahl k benutzt kann man das x (geheime Schlüssel) ausrechnen:

$$\begin{aligned} \sigma &= (s_1 - s_2)^{-1} \bmod (p-1) \\ k &= \sigma \cdot (m_1 - m_2) \bmod (p-1) \\ \rho &= r^{-1} \bmod (p-1) \\ x &= \rho \cdot (m_2 - k \cdot s_2) \bmod (p-1) \end{aligned}$$

Attacke mit *Bleichenbacher* falls eine Signatur (m, r, s) bekannt ist; (p, g, y) ist der öffentliche Schlüssel.

1. berechne $m^{-1} \bmod (p-1)$
2. wähle zufällig eine Nachricht m' (in der Aufgabe schon gegeben?)
3. $u = m' \cdot m^{-1} \mod (p-1)$
4. $s' = s \cdot u \mod (p-1)$
5. löse mit dem chinesischen Restsatz
$$\begin{cases} r' \equiv r \mod p \\ r' \equiv r \cdot u \mod (p-1) \end{cases}$$
6. Ergebnis: (m', r', s')

2.1.3 DSA

Schlüsselerzeugung

1. Wähle eine Primzahl p der Länge L bit, mit $512 \leq L \leq 1024$, wobei L ein Vielfaches von 64 ist.
2. Wähle eine weitere Primzahl q der Länge 160 bit, die ein Teiler von $p-1$ ist.
3. Wähle h für das gilt:
$$1 < h < p-1 \text{ und } h^{\frac{p-1}{q}} \mod p \neq 1$$
4. Berechne $g = h^{\frac{p-1}{q}} \mod p$
5. Wähle ein zufälliges x für das gilt: $1 < x < q$
6. Berechne $y = g^x \mod p$

Signieren Signiert wird die Nachricht m; SHA-1(m) bezeichnet den *Secure Hash Algorithm (SHA-1)*-Hashwert der Nachricht m.

1. Wähle für jede zu signierende Nachricht ein zufälliges s mit $1 < s < q$
2. Berechne $s_1 = (g^s \mod p) \mod q$
3. Berechne $s_2 = s^{-1} \cdot (\text{SHA-1}(m) + s_1 \cdot x) \mod q$

Die Signatur der Nachricht ist nun (s_1, s_2). s darf nicht übermittelt werden, da sonst der geheime Schlüssel x aus der Signatur berechnet werden kann. Der erweiterte euklidische Algorithmus kann benutzt werden, um das modulare Inverse von $s^{-1} \mod q$ zu berechnen.

Verifizieren Gegeben ist die Signatur (s_1, s_2) sowie die Nachricht m. Der Wert s wird nicht übermittelt.

1. Überprüfe, ob $0 < s_1 < q$ und $0 < s_2 < q$. Ist das nicht der Fall, weise die Signatur als ungültig zurück.
2. Berechne $w = s_2^{-1} \mod q$
3. Berechne $u_1 = \text{SHA-1}(m) \cdot w \mod q$
4. Berechne $u_2 = s_1 \cdot w \mod q$
5. Berechne $v = (g^{u_1} \cdot y^{u_2} \mod p) \mod q$
6. Wenn $v = s_1$, dann ist die Signatur gültig.

2.2 Symmetrische Verfahren

2.2.1 DES

Struktur	Feistelchiffre
Schlüssellänge	56 Bit
Blocklänge	64 Bit
Rundenanzahl	16

Besonderheiten: 56 Bit-Schlüssel mit weiteren 8 Bits als Paritätsbits ergänzt. Innerhalb der Rundenfunktion wird die Blockgröße 32 Bit und ein Schlüssellänge 48 Bits benutzt.

2.2.2 AES

Struktur	Substitutionschiffre
Schlüssellänge	128, 192 oder 256 Bit
Blocklänge	128 Bit
Rundenanzahl	10, 12 oder 14

Algorithmus

1. Schlüsselexpansion
2. AddRoundKey(Rundenschlüssel[0])
3. Verschlüsselungsrunden ($r = 1$ bis $R-1$)
   (a) SubBytes
   (b) ShiftRows
   (c) MixColumns:
$$\overbrace{\underbrace{\begin{pmatrix}02&03&01&01\\01&02&03&01\\01&01&02&03\\03&01&01&02\end{pmatrix}}_{03X^3+01X^2+01X+02}\begin{pmatrix}a_{0,i}\\a_{1,i}\\a_{2,i}\\a_{3,i}\end{pmatrix}}^{a(X)\mapsto\left(c(X)\cdot a(X)\right) \bmod (X^4+1)} = \underbrace{\begin{pmatrix}02\cdot a_{0,i}\\03\cdot a_{1,i}\\01\cdot a_{2,i}\\01\cdot a_{3,i}\end{pmatrix}}_{b}$$
   (d) KeyAddition
4. Schlussrunde
   (a) SubBytes
   (b) ShiftRows
   (c) AddRoundKey(Rundenschlüssel$[R]$)

(Die Schlussrunde zählt auch als Runde, also $R =$ (Anzahl Verschlüsselungsrunden + 1 Schlussrunde))

2.3 Blockverschlüsselung

Ziel ist die Vorbereitung für eine Verschlüsselung durch Einteilung einer Nachricht m in Blöcke der Länge $r \leq n$ mit

- n als Blocklänge, die das Verschlüsselungsverfahren erwartet ($n = 64$ für DES, $n = 128$ für AES),
- r als Länge der Blöcke m_i,
- m_i als Blöcke, in die die Nachricht m zerteilt wird.

Blockorientiert	**Stromorientiert**
ECB	OFB
CBC	CFB
$r = n$	

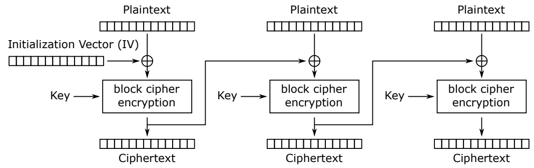
Keine asymmetrische Verschlüsselung bei stromorientierten Operationsmodi möglich

2.3.1 ECB – Electronic Code Book

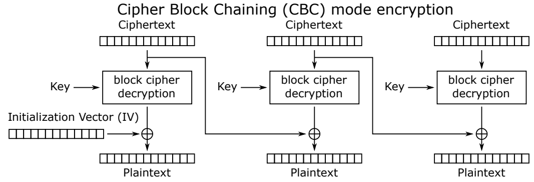
$r = n, \quad c_i = E_k(m_i), \quad m_i = D_K(c_i)$

- Bitfehler in $c_i \Rightarrow m_i$ ist zufällig, alle anderen werden korrekt entschlüsselt.
- Verlust von $c_i \Rightarrow m_i$ ist verloren, alle anderen werden korrekt entschlüsselt.
- Reparatur: nicht möglich
- Angriff: Gleiche verschlüsselte Blöcke enthalten die gleiche Nachricht.
- Einsatz: Anwendungen mit wahlfreiem Zugriff auf verschlüsselte Daten (z. B. Datenbanken), unsicher

2.3.2 CBC – Cipher Block Chaining



Cipher Block Chaining (CBC) mode encryption

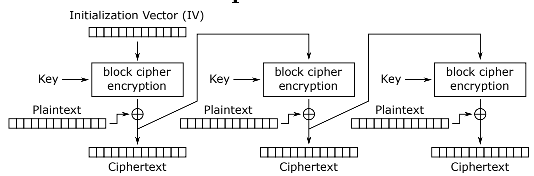


Cipher Block Chaining (CBC) mode decryption

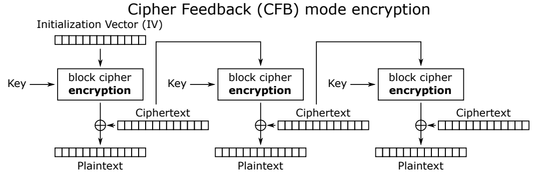
$r = n, \quad c_0 = IV, \quad c_i = E_k(m_i \oplus c_i), \quad m_i = c_{i-1} \oplus E_k^{-1}(c-i)$, jeweils für $i \geq 1$

- Bitfehler in $c_i \Rightarrow m_i$ ist zufällig und m_{i+1} hat den Bitfehler an gleicher Stelle wie c_i, m_{i+2} und folgende werden dagegen wieder korrekt entschlüsselt.
- Verlust von $c_i \Rightarrow m_i$ ist verloren und m_{i+1} ist zufällig (reparaturfähig), m_{i+2} und folgende werden dagegen wieder korrekt entschlüsselt.
- Reparatur: möglich durch $m_{i+1}^{korrigiert} = m_{i+1} \oplus m_i \oplus c_i$, nur durch Sender möglich.
- Angriff: nicht möglich
- Einsatz: Verschlüsselung von Dateien und langen Nachrichten, sehr beliebt

2.3.3 CFB – Cipher Feedback



Cipher Feedback (CFB) mode encryption

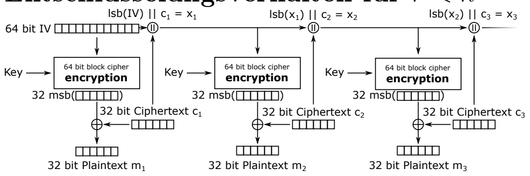


Cipher Feedback (CFB) mode decryption

$1 \leq r \leq n$, $c_0 =$ Initialvektor, $c_i = m_i \oplus msb_r(E_k(x_i))$, $m_i = c_i \oplus msb_r(E_k(x_i))$, $x_{i+1} = lsb_{n-r}(x_i) \mid\mid c_i$

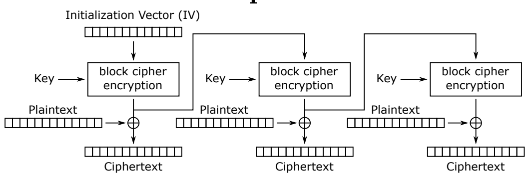
- Bitfehler in $c_i \Rightarrow m_i$ mit Bitfehler an gleichen Stelle wie c_i und alle folgenden $\lceil \frac{n}{r} \rceil$ Blöcke sind zufällig.
- Verlust von $c_i \Rightarrow m_i$ ist verloren, alle folgenden $\lceil \frac{n}{r} \rceil$ Blöcke sind zufällig.
- Reparatur: möglich durch $m_{i+1}^{korrigiert} = m_{i+1} \oplus E_k^{-1}(c_{i-1}) \oplus E_k^{-1}(c_i$
- Angriff: nicht möglich
- Einsatz: Stückweise anfallende, kleinere Datenmenge (Ströme)

Entschlüsselungsverhalten für $r < n$

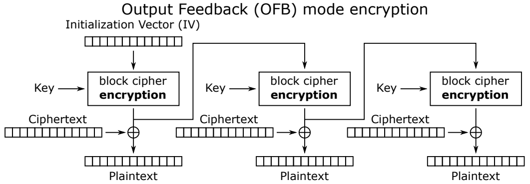


Cipher Feedback (CFB) mode decryption with block length 32 and 64 bit cryptographic algorithm

2.3.4 OFB – Output Feedback



Output Feedback (OFB) mode encryption



Output Feedback (OFB) mode decryption

$1 \leq r \leq n$, $V_0 =$ Initialvektor, $V_i = E_k(V_{i-1})$, $c_0 := V_0$, $c_i = msb_r(V_i) \oplus m_i$, $m_i = msb_r(V_i) \oplus c_i$ für $1 \leq i \leq n$.

- Bitfehler in $c_i \Rightarrow m_i$ mit Bitfehler an der gleichen Stelle wie c_i.
- Verlust von $c_i \Rightarrow m_i$ ist verloren, weitere Blöcke defekt (korrigierbar).
- Reparatur: möglich
- Angriff: bei gleichem Schlüssel und IV möglich.
- Einsatz: Satellitenkommunikation (auf Grund der Fehlertoleranz), Filesysteme/-Datenbanken wegen wahlfreiem Zugriff

3 Kryptographische Hashfunktionen

Eine Hashfunktion ist eine Funktion h, die mindestens folgende Eigenschaften erfüllt:

- **compression** h bildet eine beliebig lange Nachricht m auf einen Hashwert $h(m)$ mit der fixen Länge n ab.
- **ease of computation** für gegebenes h und m muss es leicht sein, $h(m)$ zu berechnen.

3.1 MDC - Manipulation Detection Codes

MDCs (auch bekannt unter *message integraty codes - MICs*) gehören zu der Obergruppe der *unkeyed hash functions* und muss neben den oben genannten folgende Eigenschaften erfüllen:

- **Preimage Resistance:** Praktisch unmöglich, zu gegebenem Hashwert H ein Dokument m vorzuweisen mit $H = H(m)$
- **Collision Resistance:** (auch *stark kollisionsresistent* genannt) Praktisch unmöglich, zwei Dokumente $m \neq m'$ vorzuweisen mit $H(m) = H(m')$
- **Second Preimage Resistance:** (auch *schwach kollisionsresistent* genannt) Praktisch unmöglich, zu gegebenem Dokument m ein zweites Dokument m' vorzuweisen mit $m \neq m'$ und $H(m) = H(m')$

Anmerkungen

– Eine *starke kollisionsresistente* (collision-resistant) Hashfunktion ist auch *schwach kollisionsrestistent* (second-pre-image resistant).

– Eine *schwach kollisionsrestistente* (second-pre-image resistant) Hashfunktion ist eine Einwegfunktion (one-way function).

– Hashfunktionen sind durch die Voraussetzung der *starken Kollisionsresistenz* (collision Restistance) mächtiger als Einwegfunktionen (one-way hash functions).

3.2 MAC - Message Authentication Codes

Ein MAC-Algorithmus ist eine Funktion h_k, mit einem geheimen Schlüssel als Parameter k, die die folgenden Eigenschaften (neben den oben genannten erfüllt:

- **computation-resistance** Es ist schwer für Null oder mehr gegebene Nachrichten-MAC Paare $(m_i, h_k(m_i))$ ein Nachrichten-MAC Paar $(x, h_k(m))$ zu berechnen für das gilt $m \neq m_i$.

4 Kryptographische Protokolle

4.1 Diffie-Hellman-Schlüsselaustausch

Alice und Bob haben einen gemeinsamen öffentlichen Schlüssel (p, g), p Primzahl, g primitive Wurzel von $\mathbb{Z}_p$ (wenn g keine primitive Wurzel ist, ist das Verfahren möglich aber unsicher).

1. Alice wählt zufällig $a \in [0, p-2]$, rechnet $c = g^a \bmod p$ und schickt c an Bob
2. Bob wählt zufällig $b \in [0, p-2]$, rechnet $d = g^b \bmod p$ und schickt d an Alice
3. Alice rechnet $k = d^a \bmod p$
4. Bob rechnet $k = c^b \bmod p$

Signieren Alice signiert eine Nachricht m mit dem Geheimschlüssel d. Signierte Nachricht ist $(m, m^d) = (m, \sigma)$.

Verifizieren Bob erhält (m, σ) von Alice und nutzt ihr Public Key e. Falls $m = \sigma^e = (m^d)^e$, ok.

Mögliche Angriffe Eve sendet Bob die „Signatur“ (m^e, m), Bob potenziert m mit e und ist reingefallen, da er (m^e, m^e) bekommt. Oder sie schickt ihm eine schon gültige, aber quadrierte Signatur: (m^2, σ^2).

Primzahlen 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997, 1009, 1013, 1019, 1021, 1031, 1033, 1039, 1049, 1051, 1061, 1063, 1069, 1087, 1091, 1093, 1097, 1103, 1109, 1117, 1123, 1129, 1151, 1153, 1163, 1171, 1181, 1187, 1193, 1201, 1213, 1217, 1223, 1229, 1231, 1237, 1249, 1259, 1277, 1279, 1283, 1289, 1291, 1297, 1301, 1303, 1307, 1319, 1321, 1327, 1361, 1367, 1373, 1381, 1399, 1409, 1423, 1427, 1429, 1433, 1439, 1447, 1451, 1453, 1459, 1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511, 1523, 1531, 1543, 1549, 1553, 1559, 1567, 1571, 1579, 1583, 1597, 1601, 1607, 1609, 1613, 1619, 1621, 1627, 1637, 1657, 1663, 1667, 1669, 1693, 1697, 1699, 1709, 1721, 1723, 1733, 1741, 1747, 1753, 1759, 1777, 1783, 1787, 1789, 1801, 1811, 1823, 1831, 1847, 1861, 1867, 1871, 1873, 1877, 1879, 1889, 1901, 1907, 1913, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993, 1997, 1999, 2003, 2011, 2017, 2027, 2029, 2039, 2053, 2063, 2069, 2081, 2083, 2087, 2089, 2099, 2111, 2113, 2129, 2131, 2137, 2141, 2143, 2153, 2161, 2179, 2203, 2207, 2213, 2221, 2237, 2239, 2243, 2251, 2267, 2269, 2273, 2281, 2287, 2293, 2297, 2309, 2311, 2333, 2339, 2341, 2347, 2351, 2357, 2371, 2377, 2381, 2383, 2389, 2393, 2399, 2411, 2417, 2423, 2437, 2441, 2447, 2459, 2467, 2473, 2477