Zusammenfassung: Kryptographie

Symmetrische Verschlüsselung

Data Encryption Standard (DES)

Schlüssellänge: 56 Bit (plus 8 Paritätsbits)
16 Runden, Rundenschlüssellänge: 48 Bit
Verarbeitet Blöcke der Größe: 64 Bit

Advanced Encryption Standard (AES)

Rijndael Schlüssel- und Blocklängen: 128, 160, 192, 224 oder 256 Bit
AES Schlüssellängen: 128, 192 oder 256 Bit
AES Blocklänge: 128 Bit
AES Rundenzahl: 10, 12 oder 14

```
ByteVector aes(ByteVector plaintext, key)
{
  ByteVector state;
  initState(plaintext, state);
  roundKeys = expandKey(key);
  addRoundKey(state, roundKeys.front());
  for(size_t i = 1; i < roundKeys.size() - 1; ++i) {
    subBytes(state);
    shiftRows(state);
    mixColumns(state);
    addRoundKey(state, roundKeys[i]);
  }
  subBytes(state);
  shiftRows(state);
  addRoundKey(state, roundKeys.back());
  return state;
}
```

Rechnen in Körpern von Polynomen

AES ist Byte-orientiert. Jedes Byte wird als Polynom in $\mathbb{F}_{2^8}$ interpretiert.

- besteht aus Polynomen mit einem maximalen Grad (höchste Potenz) von 7
- Koeffizienten (Zahl vor dem *x*) der Polynome sind aus $\mathbb{Z}_2$ (modulo 2 rechnen)
- Addition: Polynome koeffizientenweise addieren (modulo 2 rechnen)
- Multiplikation: Polynome koeffizientenweise multiplizieren, am Ende durch „Modulo Polynom" teilen → *Rest* ist Ergebnis

S-Box

Jedes Byte in Zustandsmatrix wird invertiert und anschließend mit dem Byte einer anderen Matrix multipliziert.

Invertieren: Byte als Polynom $p(x) \in \mathbb{F}_{2^8}$ darstellen; $x^8+x^4+x^3+x+1$ durch $p(x)$ teilen; falls Rest nicht 1 ist, $p(x)$ durch das Ergebnis der Division teilen bis Rest 1 ist

Expand Key

Orginalschlüssel wird in Rundenschlüssel in Länge der Blockgröße aufgeteilt.

AddRoundKey

Nachrichtenbytes werden mit ihrem entsprechenden Rundenschlüssel geXORt.

SubBytes

Jedes Nachrichtenbyte wird mit seinem Inversen modulo $x^8+x^4+x^3+x+1$ ersetzt. (Vorgehensweise wie bei S-Box)

MixColumns

Jede Spalte der Zustandsmatrix (durch repräsentiert $a(x)$) wird folgendermaßen transformiert: $a(x) \rightarrow (a(x) \cdot c(x)) \bmod (x^4+1)$, wobei $c(x) = 03x^3+01x^2+01x+02$

- Ablesen von $a(x)$ in folgender Reihenfolge: $a(x) = a_0 \cdot x^0 + a_1 \cdot x^1 + a_2 \cdot x^2 + a_2 \cdot x^2$
- statt „normaler" Addition binäres xor rechnen
- für Modulo $a(x) \cdot c(x)$ durch x^4+1 teilen → *Rest* ist Ergebnis

ShiftRows

Zeile 1 bleibt gleich
Shift von Zeile 2 um 1 nach rechts
Shift von Zeile 3 um 2 nach rechts
Shift von Zeile 4 um 3 nach rechts

Operationsmodie

Electronic Codebook Mode (ECB): Blöcke werden unabhängig voneinander verschlüsselt, bei Fehlern *nur* betroffene Blöcke *zufällig*

Cipher-Block Chaining Mode (CBC): xor mit zufälligem Initialisierungsvektor (nicht geheim), bei Fehlern betroffene Blöcke zufällig, Fehler wirken sich auch auf nächsten Block als Bitfehler aus, geht Block komplett verloren ist nächster Block komplett *zufällig*

Cipher Feedback Mode (CFB): Fehler wirken sich im fehlerhaften Block nur als Bitfehler aus, die nächsten „Chiffren-Blocklänge n / Operationsmodie-Blocklänge r" Blöcke sind jedoch *zufällig*, genauso wenn Block komplett verloren geht

Output Feedback Mode (OFB): xor mit zufälligem Initialisierungsvektor (nicht geheim), Fehler wirken sich nur im fehlerhaften Block und nur als Bitfehler aus, geht Block komplett verloren sind jedoch *alle* nachfolgenden Blöcke *zufällig*

Asymmetrische Verschlüsselung

RSA

Konstruktion der Schlüssel

1. zwei (sehr große) Primzahlen p und q bestimmen, z. B. $p = 13$, $q = 17$
2. RSA-Modul N berechnen: $N = p \cdot q = 221$
3. bestimme $\phi(n) = \phi(p \cdot q) = \phi(p) \cdot \phi(q) = (p-1) \cdot (q-1) = 192$
4. wähle e mit $1 < e < \phi(N)$ und $ggT(e, \phi(N)) = 1$, z. B. $e = 31$
5. jetzt bilden (N, e) den öffentlichen Schlüssel
6. bestimme privaten Schlüssel (N, d) mit $e \cdot d \equiv 1 \bmod \phi(N) = 31$ (mit erweiterten Euklidischer Algorithmus)

Ver- und Entschlüsselung, Signieren und Verifizieren

1. gegeben: zu verschlüsselnde Nachricht m, z. B. $m = 42$
2. verschlüsselte Nachricht: $c = m^e \bmod N = 185$
   Signatur: (m, σ) mit $\sigma = m^d \bmod N$
3. entschlüsseln: $m = c^d \bmod N = 42$
   mit Chinesischem Restsatz:
   $a_1 = (c \bmod p)^{d \bmod (p-1)} \bmod p \qquad a_2 = (c \bmod q)^{d \bmod (q-1)} \bmod q$
   $m = \left(a_1 \cdot q \cdot (q^{-1} \bmod p) + a_2 \cdot p \cdot (p^{-1} \bmod q)\right) \bmod N$
   Verifizieren: $m = (m^d \bmod N)^e \bmod N$

Anforderungen an Primzahlen (damit RSA sicher ist)

($p - 1$) hat großen Primfaktor r ($p + 1$) hat großen Primfaktor
(r - 1) hat großen Primfaktor $|p - q|$ muss groß sein (siehe weiter unten)

Angriffsmöglichkeiten

bei kleinem, gemeinsamen Exponenten

z. B. $e = 3$ außerdem bekannt: $N_1 = 319$, $N_2 = 391$, $N_3 = 205$, $c_1 = 43$, $c_2 = 218$, $c_3 = 180$
weitere Voraussetzung: Module N sind teilerfremd

Berechnung von m mit dem Chinesischen Restesatz möglich:
$m^e \equiv 43 \bmod 319$, $m^e \equiv 218 \bmod 291$, $m^e \equiv 180 \bmod 205$ → $m = \sqrt[e=3]{m^e = 1000} = 10$

bei gemeinsamen *N*: „Common Modulus Attack"

neben N bekannt muss bekannt sein: c_1, c_2, e_1, e_2

Berechnung von x und y mit Euklid: $x \cdot e_1 + y \cdot e_2 = 1$ → $m = \begin{cases} ((c_1^{-1})^{|x|} \cdot c_2^y \bmod N & \text{für } x < 0 \\ c_1^x \cdot (c_2^{-1})^{|y|} \bmod N & \text{für } y < 0 \end{cases}$

bei kleinem $|p - q|$

N kann durch Probieren faktorisiert werden, wenn $x > \sqrt{N}$ und es eine ganzzahlige Wurzel für $x^2 - N$ gibt: $y = \sqrt{x^2 - N}$ und $p \cdot q = (x - y) \cdot (x + y)$

weitere

- gleiches e und gleiches p (oder q) bei verschiedenen Schlüsselpaaren:
  $p = \gcd(N_1, N_2)$, $q_1 = \frac{N_1}{p}$, $q_2 = \frac{N_2}{p}$
- wenn $d < N^{1/4}$ existiert Algorithmus mit polynom. Laufzeit zur Berechnung
- „Small Message Space“: Erstellen einer Lookuptable für alle Kryptogramme möglich → kann durch Optimal Asymmetric Encryption Padding (OAEP) verhindert werden, welches RSA probabilistisch macht

ElGamal

Basiert auf diskreter Logarithmus-Annahme: wenn $y = g^x \bmod p$ gilt, dann ist $x = \log_g y \bmod p$ nicht berechenbar

Konstruktion der Schlüssel

1. Primzahl p wählen, z. B. $p = 23$
2. primitives Element g in $\mathbb{Z}_p^*$ bestimmen, hier z. B. 5
3. privater Verschlüsselungsexponent x: zufällig gewählte Zahl in aus $\mathbb{Z}_p$, z. B. 10
4. öffentlicher Verschlüsselungsexponent y: $y = g^x \bmod p$, im Bsp. $5^{10} \equiv 9 \bmod 23$
5. (p, g, y) bilden öffentlichen Schlüssel und (p, g, x) bilden privaten Schlüssel

Ver- und Entschlüsselung

1. gegeben: zu verschlüsselnde Nachricht m, z. B. $m = 42$
2. Zufallszahl z aus $[1, p-2]$ wählen, z. B. $z = 7$
   verschlüsselte Nachricht: $c = (g^z \bmod p \ , \ y^z \cdot m \bmod p)$, hier $c = (17, 7)$
3. entschlüsseln: $m = (c_1^{p-1-x} \bmod p) \cdot c_2$, hier $m = (17^{23-1-10} \bmod 23) \cdot 7 = 42$

Signieren

1. z. B. $p = 97$, $g = 5$, $x = 5$, hash(m) = 11
2. wähle zufällige Zahl z aus $[1, p-2]$, die teilerfremd zu $p - 1$ ist, z. B. $z = 13$
3. bestimme $r = g^z \bmod p$, hier $r = 5^{13} \bmod 97 = 29$
4. bestimme $s = (z^{-1} \cdot (\text{hash}(m) - r \cdot x)) \bmod (p-1)$, hier
   $s = 37 \cdot (11 \cdot 29 - 5) \bmod 96 = 34$
5. Signatur: $(\text{hash}(m), r, s)$, hier $(11, 29, 34)$

Verifizieren

1. r und s müssen größer als 0 und kleiner als p sein (siehe Bleichenbacher Angriff)
2. bestimme $v_1 = y^r \cdot r^s \bmod p$, hier $v_1 = 29^{34} \cdot 21^{29} \bmod 97 = 71$
3. bestimme $v_2 = g^{\text{hash}(m)} \bmod p$, hier $v_2 = 5^{11} \bmod 97 = 71$
4. Signatur ist gültig, wenn $v_1 = v_2$

Bleichenbacher Angriff: Signatur fälschen

Wenn man gültige Signatur hat, kann man daraus ohne geheimen Schlüssel noch eine

Signatur für veränderte Nachrichten m' erzeugen. Diese erfüllt allerdings nicht die erste Validierungsbedingung.

1. berechne $u = m' \cdot m^{-1} \bmod (p-1)$ und $s' = s \cdot u \bmod (p-1)$
2. berechne r' mit Chinesischem Restsatz: $\begin{matrix} r' \equiv r \bmod p \\ r' \equiv r \cdot u \bmod (p-1) \end{matrix}$

Angriff wenn zwei mal selbe Zufallszahl beim Signieren verwendet wird

Formel $s = \ldots$ lässt sich umstellen für beide Signaturen: $s_1 \cdot z = m_1 - r \cdot x$ und $s_2 \cdot z = m_2 - r \cdot x$
durch Subtrahieren beider Terme: $z = (s_1 - s_2)^{-1} \cdot (m_1 - m_2) \bmod (p-1)$
daraus lässt sich *privater* Exponent x berechnen: $x = r^{-1} \cdot (m_2 - z \cdot s_2) \bmod (p-1)$

Diffie Hellman Schlüsselaustausch

Initiator: Schlüsselgenerierung mittels ElGamal → öffentlicher Schlüssel wird an Empfänger gesendet

Empfänger: Zufallszahl z aus $[1, p-2]$ wählen, berechne $B = g^z \bmod p$, $K = y^z \bmod p$
→ B wird an Initiator gesendet

Initiator: kann K mittels privatem Schlüssel berechnen: $K = B^x \bmod p$

Digital Signature Algorithm (DSA)

Konstruktion der Schlüssel

1. wähle Primzahlen p und q, wobei q ein Teiler von $(p-1)$ ist, z. B. $p = 23$ und $q = 11$
2. bestimme g mit $g = z^{(p-1)/q} \bmod p$, wobei z Zufallszahl aus $\mathbb{Z}_p$ und $g \neq 1$, hier 2
3. bestimme x und y wie bei ElGamal
4. (p, q, g, y) ist öffentlicher Schlüssel, (p, q, g, x) ist privater Schlüssel

Signieren

1. wähle zufällige Zahl z aus]1; q[
2. berechne $s_1 = (g^s \bmod p) \bmod q$, wobei $s_1 \neq 0$
3. berechne $s_2 = s^{-1} \cdot (\text{hash}(m) + s_1 \cdot x) \bmod q$, wobei $s_2 \neq 0$ (in diesem Fall auch 2. wiederholen)
4. Signatur ist nun (s_1, s_2)

Verifizieren

1. Signatur nur gültig wenn s_1 und s_2 aus [1; q[
2. berechne $w = s_2^{-1} \bmod q$
3. berechne $u_1 = \text{hash}(m) \cdot w \bmod q$ und $u_2 = s_1 \cdot w \bmod q$
4. Signatur gültig, wenn $s_1 = (g^{u_1} \cdot y^{u_2} \bmod p) \bmod q$

Kongruenzgleichungen

Bestimmen des additiven Inversen d der Zahl e in $\mathbb{Z}_m$: $d = m - e$

Bestimmen des multiplikativen Inversen: $x^{-1} (mod\ m) = \frac{l \cdot m + 1}{x}$ (siehe auch Eukl.)

Es existiert dann genau ein multiplikatives Inverses, wenn x teilerfremd zu m ist.

Bsp.: $47^{-1} mod\ 60 = \frac{18 \cdot 60 + 1}{47} = 23$

Beispiel zum **Lösen einer Kongruenzgleichung**:
$347x \equiv 495 (mod\ 60) \Leftrightarrow 47x \equiv 15 (mod\ 60)$ zur Vereinfachung Reste berechnen
$47x \equiv 15 (mod\ 60) \Leftrightarrow x \equiv 345 \equiv 45 (mod\ 60)$ mit multiplikativen Inversen multiplizieren
→ Alle x der Form $x = 45 + n \cdot 60$ sind Lösungen von $347x - 495 = k \cdot 60$ mit $k \in \mathbb{Z}$.

Weiteres Beispiel: $2x \equiv 3 (mod\ 6)$
Da 2 und 6 nicht teilerfremd sind, gibt es keine eindeutige Lösung. Da ggT(2, 6) = 2 kein Teiler von 3 ist gibt es keine Lösung.

Weiteres Beispiel: $2x \equiv 4 (mod\ 6)$
Da 2 und 6 nicht teilerfremd sind, gibt es keine eindeutige Lösung. Da t = ggT(2, 6) ein Teiler von 4 ist gibt es $t = 2$ Lösungen: $x_1 = 2$, $x_2 = 5$ (durch Probieren)

Reduzierte Restsätze: $\mathbb{Z}_m^* := \{k \in \mathbb{Z}_m \mid k \text{ teilerfremd zu } m\}$ Bsp.: $\mathbb{Z}_8^* := \{1, 3, 5, 7\}$

Eulersche φ-Funktion

$\phi(m)$: Anzahl der zu m teilerfremden Zahlen in $\mathbb{Z}_m$

- Für Primzahlen p gilt: $\phi(m) = p - 1$
- Für Primfaktorpotenzen $m = p^k$ gilt: $\phi(p^k) = p^k - p^{k-1}$ z. B. $\phi(2 \cdot 5^3 \cdot 17) = 1 \cdot (5^3 - 5^2) \cdot 16$
- Für teilerfremde m und n gilt: $\phi(m \cdot n) = \phi(m) \cdot \phi(n)$

Kleiner Satz von Fermat: Ist a teilerfremd zur Primzahl p, gilt: $a^{p-1} \equiv 1 \mod p$
Verallgemeinerung : $a^{\phi(m)} \equiv 1 \mod m$ → $a^x \equiv a^{x \bmod \phi(m)} \mod m$

Anwendung: hohe Potenzen vereinfachen mit φ-Funktion

Bsp.: $105^{63} \mod 11 \equiv (105 \mod 11)^{63 \bmod \phi(11)} \equiv 6^3 \mod 11$

Ordnung, primitives Element/primitive Wurzel

Gruppenordnung: Anzahl der Elemente (Mächtigkeit) der Gruppe: $ord(G) := |G|$
(in diesem Kontext: jedes Element hoch Gruppenordnung ist immer 1)

Ordnung eines Elements: Wie oft muss man Element *mindestens* mit sich selbst Verknüpfen (in diesem Kontext multiplizieren), damit neutrales Element (in diesem Kontext immer 1) herauskommt? (neutrales Element hat immer Ordnung 1)

primitives Element/primitive Wurzel

Definition: $ord(\text{primitives Element}) := \text{Gruppenordnung}$

Erstes primitives Element in $\mathbb{Z}_p^*$ finden

wenn $g^{\frac{p-1}{q}} \not\equiv 1 \mod p$ für **alle** Primfaktoren q von $(p-1)$ → g primitives Element

Bsp.: $\mathbb{Z}_{11}^*$ → $p = 11$, $q = 2$ und 5
Ist 2 primitive Wurzel? $2^{10/2} \equiv 10 \mod 11$ und $2^{10/5} \equiv 4 \mod 11$ → ja
Ist 3 primitive Wurzel? $3^{10/2} \equiv 1 \mod 11$ und $2^{10/5} \equiv 9 \mod 11$ → nein

Weitere primitive Elemente in $\mathbb{Z}_p^*$ finden

Es existieren $\phi(\phi(p))$ primitive Elemente
x^l mit primitivem Element x und $l \in \mathbb{Z}_p^*$ und $0 < l < \phi(p)$ und $gcd(l, \phi(p)) = 1$

Bsp.: $\mathbb{Z}_{46}^*$, $x = 37$, $\phi(46) = (2-1) \cdot (23-1) = 22 = 2 \cdot 11$
→ primitive Elemente: $37^1 \equiv \mathbf{37}$, $37^3 \equiv \mathbf{7}$, $37^5 \equiv \mathbf{15}$, $37^7 \equiv \mathbf{19}$, $37^9 \equiv \mathbf{21}$,
$37^{13} \equiv \mathbf{11}$, $37^{15} \equiv \mathbf{17}$, $37^{17} \equiv \mathbf{43}$, $37^{19} \equiv \mathbf{33}$, $37^{21} \equiv \mathbf{5}$

Bsp.: $\mathbb{Z}_{11}^*$, $x=2$, $\phi(11)=10=2\cdot 5$

→ primitive Elemente: $2^1\equiv 2,\ 2^3\equiv 8,\ 2^7\equiv 7,\ 2^9\equiv 6$

Gleichungen der Form $x^y = 1 \bmod m$ lösen

Bsp.: $x^7\equiv 1 \bmod 29$ → $\underbrace{2^{\frac{29-1}{7}}}_{\text{2 ist zufällig gewählt}} \equiv 16 \not\equiv 1 \bmod 29$ → $ord(\mathbf{16})=7$

weitere: $16^1\equiv\mathbf{16},\ 16^2\equiv\mathbf{24},\ 16^3\equiv\mathbf{7},\ 16^4\equiv\mathbf{25},\ 16^5\equiv\mathbf{23},\ 16^6\equiv\mathbf{20},\ 16^7\equiv\mathbf{1}$

Schnelles Potenzieren

Startwert: 1; jede Binärstelle des Exponenten durchgehen und Wert

1. mit Basis multiplizieren, wenn Wert der Binärstelle 1
2. unabhängig von Wert der Binärstelle quadrieren, außer am Ende

Miller-Rabin-Test

Zeugen: per Definition Zahlen, die zur testenden Zahl teilerfremd sind

Wahrscheinlichkeit, dass erkannte Primzahl wirklich eine Primzahl ist: $1-4^{-\text{Anzahl der Zeugen}}$
Mindestanzahl benötigter Nicht-Zeugen für *Irrtums*wahrscheinlichkeit p_i: $\lceil \log_4(1/p_i) \rceil$

Algorithmus am Beispiel der Zahl 25:

$\underbrace{n:=25}_{\text{zu testen}}$, zunächst $\underbrace{m}_{\text{Exponent}}$ und $\underbrace{t}_{\text{Testdurchläufe*}}$ berechnen: $n-1= \underbrace{m\cdot 2^t}_{\text{wie Primfaktorzerlegung}} = \underbrace{24=3\cdot 2^3}_{\text{hier also: } m=3,\ t=3}$

teste potentiellen Zeugen $z=7$: $u:=z^m=7^3=\underbrace{18\neq\pm 1}_{\text{→evtl. Zeuge}} \bmod n=25$

$u:=u^2=18^2=\underbrace{24=-1}_{\text{→doch kein Zeuge}} \bmod 25$

teste potentiellen Zeugen 8: $u:=8^3=\underbrace{12\neq 1}_{\text{→evtl. Zeuge}} \bmod 25$

$u:=12^2=\underbrace{19\neq -1}_{\text{→evtl. Zeuge}} \bmod 25$

$u:=19^2=\underbrace{11\neq -1}_{\text{→Zeuge}} \bmod 25$ → keine Primzahl

*t: maximale Anzahl an „$u :=$ …"-Zeilen (z^m und alle u^2 Zeilen) pro Zeuge

Euklidischer Algorithmus

i	$\mathbf{r_i}$	$\mathbf{q_i}$	$\mathbf{x_i}$	$\mathbf{y_i}$
-1	m		$x_{-1} = 1$	$y_{-1} = 0$
0	n	q_0	$x_0 = 0$	$y_0 = 1$
1	$r_1 = x_1 \cdot m + y_1 \cdot n$	q_1	$x_1 = x_{-1} - q_0 \cdot x_0 = 1$	$y_1 = y_{-1} - q_0 \cdot y_0 = -q_0$
2	$r_2 = x_2 \cdot m + y_2 \cdot n$	q_2	$x_2 = x_0 - q_1 \cdot x_1$	$y_2 = y_0 - q_1 \cdot y_1$
…	…	…	…	…
k	$r_i = x_i \cdot m + y_i \cdot n$	q_k	$x_i = x_{i-2} - q_{i-1} \cdot x_{i-1}$	$y_i = y_{i-2} - q_{i-1} \cdot y_{i-1}$
k + 1	0			

- größter gemeinsamer Teiler: $\gcd(m,n)=r_i$
- multiplikative Inversen: $n^{-1}\equiv y_i \bmod m$ und $m^{-1}\equiv x_i \bmod n$
- Lösungen von: $r_i=m\cdot x_i+n\cdot y_i \quad (-1\leq i\leq k)$

nämlich $x = x_i + k \cdot \frac{n}{r_i}$ und $y = y_i - k \cdot \frac{m}{r_i}$ mit beliebigem $k \in \mathbb{Z}$

- Lösungen von: $m \cdot x + n \cdot y = c$ mit $c = f \cdot \gcd(m, n) = f \cdot r_i \Leftrightarrow f = \frac{c}{r_i}$ wobei $f \in \mathbb{Z}$

  nämlich $x = f \cdot x_i + k \cdot \frac{n}{r_i}$ und $y = f \cdot y_i - k \cdot \frac{m}{r_i}$ mit beliebigem $k \in \mathbb{Z}$

Chinesischer Restesatz

geg.: zueinander teilerfremde Module $m_1, m_2, \ldots, m_n$ und $c_1 \in \mathbb{Z}_{m_1}, c_2 \in \mathbb{Z}_{m_2}, \ldots, c_n \in \mathbb{Z}_n$

ges.: x für das gilt: $x \equiv c_1 \bmod m_1$, $x \equiv c_2 \bmod m_2$, ..., $x \equiv c_n \bmod m_n$

zunächst festgelegt: $M := m_1 \cdot m_2 \cdot \ldots \cdot m_n$ und $M_k = \frac{M}{m_k}$ $(1 \leqslant k \leqslant n)$

damit gilt:
$x \equiv (c_1 \cdot M_1 \cdot (M_1^{-1} \bmod m_1) + c_2 \cdot M_2 \cdot (M_2^{-1} \bmod m_2) + \ldots + c_n \cdot M_n \cdot (M_n^{-1} \bmod m_n)) \bmod M$

Bsp.: $x \equiv 3 (\bmod 4)$, $x \equiv 1 (\bmod 5)$, $x \equiv 7 (\bmod 9)$
→ $x \equiv ((3 \cdot 45 \cdot 1) + (1 \cdot 36 \cdot 1) + (7 \cdot 20 \cdot 5)) \bmod 180$ → $x \equiv 151 \bmod 180$

Gleiches Bsp. mit Einsetzmethode:
Erste Kongruenz: $x \equiv 3 (\bmod 4)$ → $x = 3 + 4k$
Zweite Kongruenz: $3 + 4k \equiv 1 \bmod 5$
mit $4^{-1} \bmod 5 = 4$ multiplizieren → $k \equiv 2 \bmod 5$ → $x = 3 + 4 \cdot (2 + l \cdot 5) = 11 + 20l$
Dritte Kongruenz: $11 + 20l \equiv 7 \bmod 9$ → $2 + 2l \equiv 7 \bmod 9$
mit $2^{-1} \bmod 9 = 5$ multiplizieren → $l \equiv 7 \bmod 9$ → $x = 11 + 20 \cdot (7 + 9m) = 151 + 180m$

Hashfunktionen

Anforderungen (jede weitere schließt vorherige mit ein)

Einwegfunktion (Falltür): m lässt sich nicht aus hash(m) berechnen
Zweites Urbild: zu bestimmten m lässt sich kein m' berechnen für das gilt hash(m) = hash(m')
Kollisionsresistenz: zu beliebigen m lässt sich kein m' berechnen für das gilt hash(m) = hash(m')

Geburtstagsattacke

alle Hashwerte und deren Klartext speichern (z. B. bei 128 Bit 2^{128} Werte)
Wahrscheinlichkeit für Kollision: $1.18 \cdot \sqrt{w}$ (w ist die Anzahl der Werte)

Zertifikate

verknüpfen Namen mit öffentlichen Schlüssel und verifizieren deren Authentizität

CAs signieren Zertifikate mit ihrem öffentlichen Schlüssel und garantieren so deren Authentizität

SSL/TLS

- Server benötigt asymmetrisches Schlüsselpaar und Zertifikat
- Sichert *Authentizität der Kommunikationspartner* (i. d. R. Server)
  → MACs (Server), Challenge Response (Client)
- Sichert V*ertraulichkeit und Integrität der Daten* → symmetrische Verschlüsselung

MACs

authentifizieren Ursprung einer Nachricht und garantieren deren Integrität
Standard: HMAC (Hash wird zwei mal angewendet)

Richtig-Falsch-Fragen

Große Primzahlen der Form 2^n-1 sind für RSA geeignet.	f
Der Startwert darf bei OFB wiederverwendet werden.	f
MACs dienen meist nur der Authentifizierung des Absenders.	f
$m \to g^m \bmod p$ (p große Primzahl, g primitive Wurzel mod p) ist eine geeignete Verschlüsselungsfunktion für Klartexte $m \in \{0,...p-2\}$	
In SSL wird der Client mittels digitaler Signaturen authentifiziert.	r
Gilt die Diffie-Hellmann-Annahme, so auch die Diskrete Logarithmus-Annahme	r
Der Startwert kann bei CBC problemlos wieder verwertet werden.	f
Beim Bilden eines HMAC wird die Hashfunktion 2 mal angewendet.	r
In SSL wird der Server immer mittels digitaler Signaturen authentifiziert.	f
$a^x \bmod n$ kann mit maximal 2048 modularen Multiplikationen berechnet werden, falls $\lvert n\rvert = 1024$.	r
Bei MD5 kann man heute Urbilder eines Hashwertes effizient berechnen.	f
$m \to g^m \bmod p$ (p eine große Primzahl) kann keine Einwegfunktion sein falls ord(g)nur ca. 2160 ist.	f
Die RSA-Verschlüsselung kann mit Hilfe des Chinesischen Restesatzes effizienter gemacht werden.	f
Die Geburtstagsattacke erfordert bei Hashlänge 256 in der Regel ca. 2^{85} Schritte.	f
In SSL werden immer Server und Client authentifiziert.	(f)
Der Bleichenbacher-Angriff auf ElGamal-Signaturen erfordert den Chinesischen Restesatz.	f
Der Startwert muss bei CFB verschlüsselt übertragen werden.	f
MAC's sind zur Authentifizierung des Absenders bei großen Datenmengen ungeeignet.	
Für einen RSA-Modulus n ist die Berechnung von $\phi(n)$ genauso schwer wie die Faktorisierung von n.	(r)
Basis RSA ist für die Verschlüsselung von EC-Karten-Pins geeignet.	(f)
Für sichere HMACs ist die Kollisionsresistenz der Hashfunktion notwendig.	(f)
In SSL werden kryptografische Hashfuntionen als Pseudozufallsgeneratoren benutzt.	(r)
Der Diffie-Hellman-Schlüsselaustausch ist ohne authentifizierte Kommunikationspartner unsicher.	(r)
Im Zertifikat findet sich der geheime Schlüssel des Eigentümers	(f)
Jeder Rundenschlüssel von DES ist 64 Bit lang.	(f)
RSA mit OAEP ist zur Verschlüsselung von Matrikelnummern geeignet.	(r)
Im SSL Protokoll kommen MACs zum Einsatz.	(r)
$m \to m^3 \bmod (p \cdot q)$ ist für alle großen Primzahlen p und q invertierbar.	(f)
OFB mit AES erfordert ein Padding der Klartexte	(f)
Der diskrete Logarithmus ist notwendig für die Sicherheit von ElGamal.	(r)
Das ElGamal-Verfahren ist zur Verschlüsselung von Matrikelnummern geeignet.	(r)
Die RSA-Verschlüsselung ist zur Verschlüsselung von Matrikelnummern geeignet.	(f)
Einweg-Hashfunktionen sind kollisionsresistent.	(f)
$m \to m^3 + m \bmod 2^n$ ist für große n kollisionsresistent.	(f)
Message-Authentification-Codes werden mit dem öffentlichen Schlüssel berechnet.	(f)
AES ist ein Feistel-Chiffre. (Bemerkung: wäre für DES richtig)	(f)
OAEP verhindert die Attacke gegen den kleinen Nachrichtenraum.	(r)
Rundenschlüssel beim DES sind 56 Bit lang.	(f)
Für einen primitive Wurzel g mod p gilt $g^{(p-1)/2} = \lvert -1\rvert$	(r)
DSA ist ein Verfahren für Unterschriften und zum Verschlüsseln.	(f)
AES erlaubt Schlüssel und Blocklängen von 256 Bit.	(f)
Der CFB-Modus synchronisiert sich nicht nach verlorenen Blöcken.	(f)