Ziele der Kryptographie:

- Vertraulichkeit (kein Abhören möglich)
- Integrität der Daten (keine Veränderung von außen)
- Authentizität des Partners

Verfahren:

- Symmetrische Verschlüsselung (z.B. AES): Beide Partner kennen den geheimen Schlüssel
- Asymmetrische Verschlüsselung (z.B. RSA): Der Empfänger kennt den geheimen Schlüssel, der Verschlüsselungsschlüssel ist allgemein bekannt

Chiffren:

- Blockchiffre: Verschlüsselung von Blöcken (feste Länge)
- Stromchiffre: Verschlüsselung von beliebigen Bit-Ketten (z.B. One-Time-Pad)

Attacken:

- Ciphertext-only
- Known-plaintext
- Chosen-plaintext
- Adaptively-chosen-plaintext
- Chosen-, Adaptively-chosen-ciphertext

Challenge Response (User Autentification):

1. B wählt Challenge c
2. A signiert c
3. B überprüft die Signatur

Rechnen im Polynomring

Bitkette $\Leftrightarrow$ Polynom, z.B. 1011 $\rightarrow x^3 + x^1 + 1$

Wenn Koeffizienten **(mod 2)**: $+ = - = \oplus$

Berechnen: Rest $\equiv$ I mod Generatorpolynom)

Bsp: I= 110100, G=$x^3 + x^1 + 1 \widehat{=} 1011$

```
110100:1011=1111001
1011
01100
 1011
01110
  1011
  0101 => 110100 ≡ 101 mod 1011
```

Euklid

$r_0 - q_1 * r_1 = r_2$

$r_1 - q_2 * r_2 = r_3$

...

$r_{k-1} - q_k * r_k = \underline{0} \implies ggt(r_1, r_2) = r_k$

$100 - 5*19 = 5$

$19 - 3*5 = 4$

$5 - 1*4 = 1$

$4 - 4*1 = \underline{0} \implies ggt(100, 19) = 1$

Ausrechnen des Inversen: (wenn ggt=1)

$ggt = r_{k-2} - q_{k-1} * r_{k-1}$

$= r_{k-2} - q_{k-1} * \overbrace{(r_{k-3} - q_{k-2} * r_{k-2})}$

...

$= a * r_0 + b * r_1 \rightarrow r_1^{-1} = b$

$1 = 5 - 1*4$

$= 5 - 1*(19 - 3*5) = -19 + 4*5$

$= -19 + 4*(100 - 5*19) = 4*100 - 21*19$

$19^{-1} = -21$

Fast Modular Exponentiation

Um x^e mod n zu berechnen:

Berechne Binardarstellung des Exponenten e;

Berechne für jede Binarstelle des Exponenten ab der 2.:

$((x^2 \cdot x^{e_{k-2}})^2 \dots)^2 \cdot x^{e_0}$ mod n

Bsp: 5^{10} mod 13

$10 = (1010)_2$

$5^{10} mod 13 = (5^2 \cdot 1)^2 \cdot 5)^2 \cdot 1$

$= (1^2 \cdot 5)^2 = 12$

Phi

Primfaktorzerlegung!

Keine Doppelten Primfaktoren p_i: $\varphi(n) = (p_1 - 1)(p_2 - 1)$... z.B: $\varphi(15) = (3-1)(5-1) = 8$

Doppelte Primfaktoren: p_i: $\varphi(n) = n(1 - \frac{1}{p_1})(1 - \frac{1}{p_2})$ z.B: $\varphi(75) = 75 * (1 - \frac{1}{3})(1 - \frac{1}{5}) = 40$

$a^{\varphi(n)} \equiv 1(mod n)$

Chinesischer Restesatz

Lösung eines Systems $x \equiv a_1$ mod $n_1 \dots x \equiv a_k$ mod n_k (Bedingung n untereinander Teilerfremd):

1. Berechne m = $n_1 \cdot n_2 \dots n_k$
2. Berechne $t_i = m/n_i$ für alle n_i
3. Berechne $d_i = t_i^{-1} mod n_i$ für alle n_i (Euklid)
4. Ergebnis $= a_1 \cdot d_1 \cdot t_1 + \dots + a_k \cdot t_k \cdot d_k (mod m)$

Bsp:

$x \equiv 2(mod 3)$ $x \equiv 3(mod 5)$ $x \equiv 2(mod 7)$

m= $3 \cdot 5 \cdot 7 = 105$

$t_1 = \frac{105}{3} = 35$ $t_2 = 21$ $t_3 = 15$

$d_1 = 2(mod 3)$ $d_2 = 1(mod 5)$ $d_3 = 1(mod 7)$

$2*35*2 + 3*21 + 2*15 = 233 \equiv 23(mod 150)$

Primitive Wurzel

ord(x): kleinstes n mit $x^n = 1$, ord(x) Produkt aus Primfaktoren von Gruppenordnung (p-1)
Anzahl Primitiver Wurzeln: $\varphi(p-1)$

Wähle Zufallszahl g und Teste für jeden Primfaktor q von p-1:
$g^{\frac{p-1}{q}} \neq 1 mod p$ => Primitive Wurzel
Bsp: p = 31, p-1= 2*3*5
Teste g=2: $2^6 = 2 mod 31$, $2^{15} = 1 mod 31$ => keine Primitive Wurzel

Primzahltest (Miller-Rabin)

1. Zerlege n-1 = $2^k + t$ // t ungerade
2. Für Zufallszahl a // Fehlerwahrscheinlichkeit: $1/4^{AnzahlZufallszahlen}$
3. ff: Nach jedem Schritt prüfen ob = -1 (Wenn ja Zusammengesetzt)
4. berechne a^t
5. quadriere nun k-1 mal

DES:

m = 64 Bits; k = 56 Bits
Aus k werden 16 Rundenschlüssel k_i generiert;

Verschlüsseln:

Führe Anfangspermutation durch;
Ablauf einer Runde i:

1. Halbiere die nachricht m in x und y: $m \rightarrow (x,y)$
2. Aus (x,y)berechne $(x \oplus f(k_i, y), y)$
   mit $f(k_i, x) = P(S(E(x) \oplus k_i))$
   S= Substitution (ersetzten des Blocks)
   P= Permutation (Umsortieren der Bits)

Entschlüsseln:

Führe Verschlüsseln in Umgekehrter Reihenfolge durch

AES:

m= 128 Bits; k= 128, 192 oder 256 Bits
Rundenanzahl (N_r): Für k= 128: 10 k= 192: 12 k= 256: 14

Zustandsmatrix: 4x4 Matrix

Ablauf:

1. initialisiere Zustandsmatrix mit m
2. bilde $Zustandsmatrix \oplus k_0$
3. for i = 1 bis i= $N_R - 1$

   1 S-Box: $x = x^{-1} \in F_{2^8}$
   2 Verschiebe die Zeilen der Matrix
   3 Durchmische die Spalten (MixColumns)
   4 bilde $Zustandsmatrix \oplus k_0$

4. S-Box
5. Verschiebe die Zeilen der Matrix
6. Durchmische die Spalten (MixColumns)

Modes of Operation:

Ist |m|>r: Teile m in Blöcke und für jeden Block: (r= Blocklänge)

- ECB: Electronic Codebook Mode
  $c_i = E(m_i)$ (Blöcke einzeln Verschlüsseln)
  Fehler: nur betroffener Block
- CBC: Cipher-Block Chaining Mode
  Wähle zufälliges Kryptogramm c_0 (Startwert, wird mitgeschickt)
  $c_i = E(m_i \oplus c_{i-1})$ (Verschlüssle Nachricht $\oplus$ voriges Kryptogramm
  Fehler: betroffener Block und folgender
- CFB: Cipher Feedback Mode:
  $c_i = m_i \oplus msb_r(E(x_i))$ (nimm die vordersten r Bits aus $E(x)$ und XORe)
  $x_{x_i+1} = lsb_{|x|-r}(x_i) || c_i$ (shifte c_i in x ein)
  Fehler: Fehler in den Folgenden Blocklänge/r c's; zus. bei Bitfehlern => Bitfehler an gleicher Stelle
- OFB: Output Feedback Mode:
  $c_i = m_i \oplus msb_r(E(x_i))$ (siehe CFB)
  $x_{i+1} = E(x_i)$ (x als Pseudozufallsgenerator)
  Fehler: verloren = alle Folgenden, Bitfehler = Bitfehler an gleicher Stelle

RSA

Schlüsselerzeugung
Wähle 2 große Primzahlen p und q; n= $p \cdot q$
Wähle e und berechne d = e^{-1} mod $\varphi(n)$
Schlüssel: öffentlich (n,e), geheim (n,d)

Verschlüsselung
c = $m^e (mod n)$

Entschlüsselung
m = $c^d (mod n)$ (Hier mod p und mod q rechnen, anschließend Chin. Restesatz)

Attacken
- Faktorisieren von n $\hat{=}$ Berechnen von $\varphi(n)$ $\hat{=}$ Berechnen von d
- für $|p-q|$ klein: teste x>$\sqrt{n}$; wenn $x^2 - n$ ein quadrat ist: x $\pm\sqrt{x^2-n}$ sind die Primfaktoren
- "Common Modulus": wird eine Nachricht m an zwei Empfänger mit gleichem n und unterschiedlichem e gesendet kann ein dritter m ausrechnen;
  zudem kann der eine Empfänger den geheimen Schlüssel des anderen berechnen
- "Low-Encryption-Exponent": wird gleiche Nachricht m an 3 Empfänger mit e=3 gesendet kann m^3 $(mod\ n_1 n_2 n_3)$ berechnet werden (Chin. Restesatz); Da $m^3 < n_1 n_2 n_3$ ist $m = \sqrt[3]{m}$
- Small-Message-Space Attack: gibt es nur wenige, bekannte Nachrichten (z.B. "ja","nein") können diese berechnet werden und durch Vergleich können abgefangene Kryptogramme entschlüsselt werden.
- Chosen-Ciphertext: um c zu entschlüsseln: wähle beliebiges r, lasse $r^e c mod n$ entschlüsseln => m' = rm => m= m'r^{-1}
- Angreifer fängt c ab und berechnet $x = r^e c$ und lässt Empfänger x signieren; (Signatur =x^d) => m = $r^{-1} x^d$
- Bleichenbachers 1-Million-Chosen-Ciphertext Attack

signatur
Signieren von m: $s = m^d$; signed message: (m,s)
Überprüfen von (m,s): prüfe ob $m = s^e$
Existentiell fälschbar: (f^e,f) mit f beliebig wird als signierte Nachricht anerkannt.
(m_1, σ_1) (m_2, σ_2) => z.B. $(m_1 m_2, \sigma_1 \sigma_2)$ oder $(m_1^{-1}, \sigma_1^{-1})$ werden als signierte Nachricht anerkannt.

OAEP
n=k+l: Falltürfunktion f (z.B. RSA): n-Bit $\to$ n-Bit
Pseudozufallsgenerator G: k-Bit $\to$ l-Bit
Hashfunktion h: l-Bit $\to$ k-Bit

Verschlüsseln von m (l-Bit)
1. Wähle r:Zufallszahl mit k-Bit
2. x = (m $\oplus$ G(r)) || (r $\oplus$ h(m $\oplus$ G(r))
3. c = f(x)

Entschlüsseln von c (n-Bits)
1. Zerlege c in a b, a=l-Bits, b=k-Bits
2. r= b$\oplus$h(a)
3. m = a $\oplus$ G(r)

ElGamal

Schlüsselerzeugung
Wähle Primzahl p, so dass p-1 mit großem Primfaktor und suche primitive Wurzel g mod p
wähle zufallszahl $x \in [0, p-1]$, berechne y=$g^x mod p$

privat: (p,g,x) öffentlich: (p,g,y)

Verschlüsselung
wähle $k \in [1, p-2]$ => c= $(c_1, c_2) = (g^k, y^k m(mod p))$

Entschlüsselung
-x= p-1-x
m= $c_1^{-x} c_2$

Signatur
wähle $k \in [1, p-2]$, gcd(k,p-1)=1
berechne r = $g^k, s = k^{-1}(m - rx)$ mod (p-1) => Signatur(m,r,s)
Wichtig: k muss immer neu gewählt werden, k muss gute Zufallszahl sein

Verifizieren
Prüfe $r \in [1, p-1]$; Prüfe $g^m = y^r r^s$

DSA (Digital Signature Algorithm)

Schlüsselerzeugung
Wähle Primzahl p, so dass p-1 mit Primfaktor q(160Bit) und suche g mit ord(g)=q mod p, (Also: $g^{\frac{p-1}{q}} \neq 1$)
wähle zufallszahl $x \in [0, q-1]$, berechne y=$g^x mod p$
privat: (p,q,g,x) öffentlich: (p,q,g,y)

Signatur
Wähle $k \in [1, q-1]$
Berechne r=$g^k mod p mod q$, s=$k^{-1}(m + rx) mod q$ => Signatur (m,r,s)

Verifizieren
Prüfe $r, s \in [1, q-1]$
Berechne t= s^{-1}mod q
Prüfe $((g^m y^r)^t$mod p) mod q = r

Hash
$\{0,1\}^*$ -> $\{0,1\}^n$ // Beliebige Bitketten -> Bitketten fester Länge Eigenschaften:

1. One-way function: Nicht umkehrbar
2. second pre-image resistance: Nicht möglich auf gegebene Nachricht& Hashwert andere Nachricht zu finden.
3. collision resistance: Nicht möglich zwei Nachrichten mit gleichem Hash zu finden => stärkste Bedingung

Merkle-Damgard
Kollisionsresistente Kompressionsfunktion:$0,1^{n+r} - > 0,1^n$

Padding: hänge 10…0 an, so dass |m|= vielfaches von r;
Hashen:

1. teile m in Blöcke der Länge r
2. füge Block der Länge r in dem die ursprüngliche Länge von m steht
3. v_0= (zufälliger) Startwert
4. v_i= f($v_{i-i}||m_i$)
5. Hashwert ist letztes v

Birthday-Attack
Angriff gegen Kollisionsresitenz: Cachen von Hashwerten und testen auf Kollision

Kompression aus Blockchiffren
$f_1 : (x||y) - > E(y, x)$ // Spalte Wert in Key und Message und Verschlüssle => Gehashter wert

MAC

HMAC
HMAC(k,m) = h((k$\oplus$opad)||h((k$\oplus$ipad)||m))
h: Hash nach Merkle-Damguard mit Kompressionsrate r (in Bytes);

k: Schlüssel mit Padding auf Länge r, ipad: r mal 0x36, opad: r mal 0x5C

CBC-MAC
Aus Blockchiffre:$c_0 = IV$, $c_i = E(k, m_i \oplus c_{i-1}$
=> letztes c ist MAC

Pseudozufallsgenerator PRF
PRF(secret,seed)= HMAC(secret,A(1)||seed) || HMAC(secret,A(2)||seed) ...
A(0)= seed, A(i) = HMAC(secret,A(i-1));

Protokolle

Diffie-Hellman Key Argreement
public Key: p (große Primzahl) g: Primitive Wurzel
Ablauf:

1. A wählt a∈[1,p-2], sendet c=g^a
2. B wählt b∈[1,p-2], sendet d=g^b
3. A berechnet k = d^a, B berechnet k = c^b => Schlüssel wurde ausgetauscht

Nicht alle Bits von k sind sicher => h(k) wird als Schlüssel verwendet;
Zusätzliche Authentification gegen Man-in-the-middle nötig.

Station-to-Station Protocol
public Key: p (große Primzahl) g: Primitive Wurzel; Signaturmechanismus nötig (Zugang zu öffentlichen Signaturschlüsseln notwendig)
Ablauf:

1. A wählt a∈[1,p-2], sendet c=g^a
2. B wählt b∈[1,p-2], berechnet k=g^{ab} und s=Sign($g^a||g^b$)
3. B sendet $(g^b, E_k(s))$
4. A berechnet k=g^{ab}, entschlüsselt E_k und prüft signatur
5. Wenn gültig: A sendet $E_k(sign(g^b||g^a)$, B prüft

Zertifikate
Inhalt: Besitzer, dessen PublicKey, Certification Authority, Seriennummer, Gültigkeitszeitraum,...
Unterschrieben von Certification Authority, eventuell verkettet.

SSL/TLS
SSL Komponenten: Handshake, Change Cipher Spec(Wechsel asymmetrisch->symmetrisch), Alert, Record Layer(Verschlüsselu
Handshake: Server (optional auch Client) wird authentifiziert, Schlüssel für MAC und Verschlüsselung ausgetauscht
Header: Typ,Version,Länge; Trailer: MAC,Padding
Ablauf

1. ClientHello (vorhandene Algorithmen, zufallszahl, session_id wenn Sitzungsfortführung)
2. ServerHello (evtl. neue Session_id, zufallszahl, Certifikate)
3. ClientKeyExchange (premaster_secret vom Client berechnet und mit ServerPKey verschlüsselt => zur Berechnung aller weiteren Schlüssel)
4. ChangeCypherSpec Client->Server, Finished (enthält MAC über alle bisherigen Nachrichten, ist verschlüsselt)
5. ChangeCypherSpec Server->Client, Finished "

Bei Sitzungsfortführung nur austausch von client_random und server_random, keine Authentifizierung;
=> premaster_secret wird wiederverwendet, keys mit den Zufallszahlen neu generiert