1 Kryptografie

Die Kryptografie wird in drei Hauptkategorien unterteilt:

1. Symmetrische Kryptografie
2. Hashfunktionen
3. Asymmetrische Kryptografie

1.1 Symmetrische Kryptografie

Symmetrische Kryptografie ist eine Menge von kryptografischen Protokollen, bei der derselbe geheime Schlüssel für die Ver- und Entschlüsselung von Daten verwendet wird.

Symmetrische Kryptosysteme

Ein symmetrisches Kryptosystem ist ein 5-Tupel (M, K, C, e, d) bestehend aus:

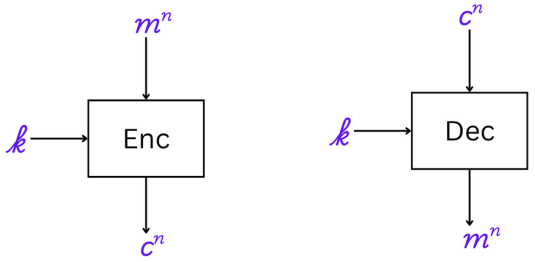
- einer Menge M von Klartexten,
- einer Menge K von Schlüsseln,
- einer Menge C von Chiffretexten,
- einer Verschlüsselungsfunktion $e : M \times K \to C$,
- einer Entschlüsselungsfunktion $d : C \times K \to M$,

so dass für alle Klartexte $m \in M$ und alle Schlüssel $k \in K$ gilt, dass $d(e(m, k), k) = m$.

1.1.1 Blockchiffren

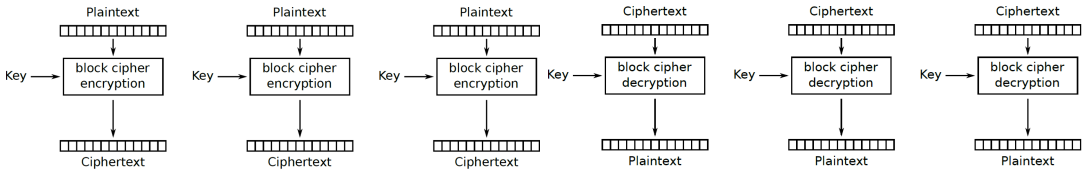
Definition

Blockchiffren sind Kryptosysteme, die nur Blöcke fester Länge verschlüsseln können.



- Ein Blockchiffre arbeitet auf einem Klartextblock der Länge b, um einen Chiffretextblock der Länge b zu erzeugen.
- Der gleiche Schlüssel kann mehrmals auf unterschiedliche Blöcke verwendet werden.
- Beispiele von Blockchiffren: AES, DES, 3DES, Serpent, Twofish, Blowfish, etc.

Electronic Code Book (ECB) Modus



Electronic Codebook (ECB) mode encryption

Electronic Codebook (ECB) mode decryption

Wenn die Blöcke nicht die Länge n haben, können trotzdem beliebige Nachrichten verschlüsselt werden, da eine **Auffüllfunktion** (Padding function) benutzt wird.

Bei vielen Padding-Verfahren wird *immer* ein Padding hinzugefügt, auch wenn die Nachricht bereits ein Vielfaches der Blocklänge n hat. Dies ist notwendig, damit die $unpad()$-Funktion eindeutig feststellen kann, wie viele Bytes entfernt werden müssen. Eine gute Auffüllfunktion sollte umkehrbar sein, d.h. es muss eine $unpad()$-Funktion geben mit $unpad(pad(x)) = x \quad \forall x \in M*$.

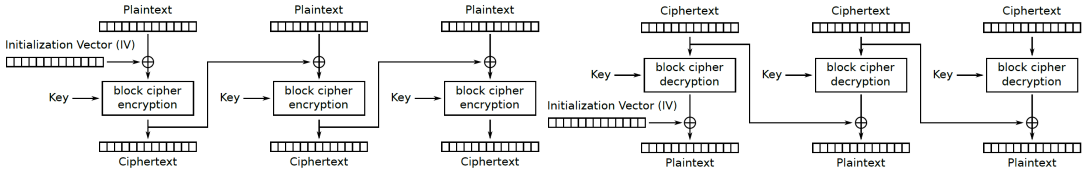
Vorteile:

- Unkomplizierte Bedienung. Jeder Block wird unabhängig bearbeitet.
- Parallelisierbarkeit von Ver- und Entschlüsselungsverfahren.
- Beschädigte Datenblöcke beeinflussen keine anderen Blöcke (Fehlertoleranz).

Nachteile:

- *Deterministisch:* Muster im Klartext sind sichtbar. Identische Klartextblöcke ergeben immer identische Chiffretextblöcke.
- *Keine Diffusion:* Kleine Änderungen im Klartext führen zu lokalisierten Änderungen im Geheimtext.

Cipher Block Chaining (CBC) Modus



Cipher Block Chaining (CBC) mode encryption

Cipher Block Chaining (CBC) mode decryption

Zur Formalisierung von CBC benötigen wir randomisierte Kryptosysteme. Der Zufallswert r wird hier als Initialisierungsvektor (IV) bezeichnet.

Randomisierte symmetrische Kryptosysteme

Ein randomisiertes symmetrisches Kryptosystem ist ein 5-Tupel (M, K, C, e, d) bestehend aus:

- einer Menge M von Klartexten,
- einer Menge K von Schlüsseln,
- einer Menge C von Chiffretexten,
- einer Menge R von Zufallswerten (z.B. IVs),
- einer Verschlüsselungsfunktion $e : M \times K \times R \to C$,
- einer Entschlüsselungsfunktion $d : C \times K \to M$,

(Anmerkung: Die Entschlüsselung d benötigt den IV r, dieser wird aber typischerweise als Teil des Chiffretextes C übermittelt und nicht als separater Zufallseingang für d selbst.)

Sei $r \in R$ der Initialisierungsvektor (IV). **Verschlüsselung**

$$e^*(x_0x_1 \dots x_n, k, r) = y_0y_1 \dots y_n \text{ mit } y_0 = e(x_0 \oplus r, k) \quad \text{und} \quad y_i = e(x_i \oplus y_{i-1}, k) \quad \text{für } i \geq 1$$

Entschlüsselung

$$d^*(y_0y_1 \dots y_n, k, r) = x_0x_1 \dots x_n \text{ mit } x_0 = d(y_0, k) \oplus r \quad \text{und} \quad x_i = d(y_i, k) \oplus y_{i-1} \quad \text{für } i \geq 1$$

- Zur Verschlüsselung muss ein Wert $r \in R$ (der IV) gewählt werden.
- Zufallswerte aus R (IVs) sind nicht geheim, sie können unverschlüsselt mit dem Chiffrat gespeichert und verschickt werden (meist als erster Block).
- Wir wollen $e(x, k, r^1) \neq e(x, k, r^2)$ für $r^1 \neq r^2$.
- Wichtig für die Sicherheit ist, dass der IV (Zufallswert r) **eindeutig** (nie doppelt für denselben Schlüssel) und **unvorhersagbar** ist.
- Muster im Klartext sind im Chiffrat nicht mehr erkennbar.
- Gleiche Klartextblöcke werden unterschiedlich verschlüsselt.
- Ein fehlerhafter Chiffreblock y_i führt nur zur fehlerhaften Entschlüsselung des aktuellen Blocks x_i und des unmittelbar nachfolgenden Blocks x_{i+1}.
- Verschlüsselung ist **nicht** parallelisierbar (sequenziell), Entschlüsselung ist parallelisierbar.

CBC Padding Oracle Attack CBC ist anfällig für Padding-Oracle-Angriffe

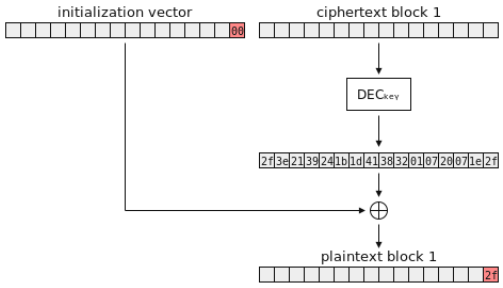
Ein solcher Angriff ermöglicht es, einen Geheimtext Schritt für Schritt zu entschlüsseln, ohne den Verschlüsselungsschlüssel zu kennen. Der Angreifer nutzt aus, wie ein Server auf fehlerhaftes Padding reagiert.

Der Angreifer:

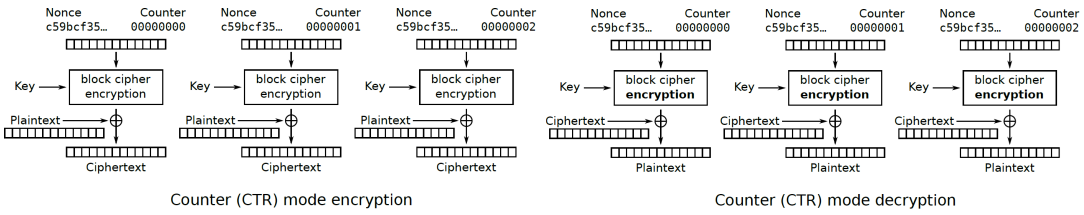
- hat keinen Zugriff auf den geheimen Schlüssel.
- ist in der Lage, gültige Chiffretexte abzufangen.
- ist in der Lage, modifizierte Versionen des Chiffretextes an das Orakel/Server zu senden und dessen Antworten zu beobachten.

Das Orakel (Server):

- muss ein überprüfbares Padding-Schema verwenden.
- muss dem Angreifer verraten, ob ein entschlüsselter Text ein gültiges (oder nicht) Padding hat. Dies kann geschehen durch:
  1. direkte Fehlermeldungen (z.B. HTTP 500) oder
  2. Side-Channel-Messungen (z.B. Unterschiede im Antwortverhalten).



Counter Mode (CTR) Modus



Um diesen Modus zu formalisieren, benötigen wir eine randomisierte Zählfunktion, die einen ”Nonce” (Number used once) r verwendet.

- Ein randomisierter Zähler (Funktion $ctr(r, i)$) bildet einen Zufallswert (Nonce r) und eine natürliche Zahl i (Zähler) auf eine Bitkette fester Länge ab.
- Eine einfache Implementierung benutzt die Binärdarstellung der natürlichen Zahl (LSB- oder MSB-Kodierung) mit 0-Padding, konkateniert mit der Nonce.
- Ein randomisierter Zähler $\text{ctr}(r, \cdot)$ sollte injektiv sein (für ein festes r). Man sollte die Periode (Wiederholung) so lang wie möglich wählen.
- Die Nonce r muss für denselben Schlüssel **nie** wiederverwendet werden.

Verschlüsselung:

$$e^*(x_0x_1 \dots x_n, k, r) = y_0y_1 \dots y_n \text{ mit } y_i = e(\text{ctr}(r, i), k) \oplus x_i$$

Entschlüsselung:

$$d^*(y_0y_1 \dots y_n, k, r) = x_0x_1 \dots x_n \text{ mit } x_i = e(\text{ctr}(r, i), k) \oplus y_i$$

Der CTR Modus unterscheidet sich stark von den vorher betrachteten Betriebsmodi:

- Ver- und Entschlüsselung nutzen beide die Verschlüsselungsfunktion e der Blockchiffre; die Entschlüsselungsfunktion d selbst wird nicht benötigt.
- Ver- und Entschlüsselung sind identisch (XOR mit dem Keystream).
- Die Berechnung des Keystreams $e(\text{ctr}(r, i), k)$ ist unabhängig vom zu verschlüsselnden Text.
- Ver- und Entschlüsselung können vollständig parallelisiert werden.
- CTR ist eine One-Time-Pad-Konstruktion, bei der die Blockchiffre als Pseudozufallsgenerator (Keystream-Generator) dient.

Advanced Encryption Standard (AES)

- Blocklänge ist 128 bereits
- AES-Schlüssel können $128, 192,$ oder256 bits lang sein

Sicherheit

- AES ist sicher solange die Implementierung und dazugehörige Systeme richtig konfiguriert sind (s. CBC Padding Attack)
- Schwache Schlüssel und IV-Generierung kann die Sicherheit von AES gefährden
- Side-channel Angriffe wie cache-timing und power analysis können verwendet werden, um den Schlüssel abzuleiten

Gegenmaßnahmen

- Konstantzeit-Implementierung (gegen Timing Angriffe): Ausführungszeit von Code soll unabhängig von den verarbeiteten geheimen Daten sein
-

Stromchiffren Stromchiffren können beliebig lange Bitketten verschlüsseln. Dabei sind Klar- und Chiffretexte beliebiger Länge, nur der Schlüssel hat eine feste Länge. Aus dem Schlüssel wird ein pseudozufälliger Schlüsselstrom erzeugt. Pseudozufallszahlen hängen von ihren Startparametern ab (seed) ab - gleiche Parameter liefern gleiche Zufallszahlen Ver- und Entschlüsselung ist ein bitweise exklusives oder (XOR) mit dem Schlüsselstrom. Ein Kryptosystem heißt Stromchiffre, wenn es eine Funktion $keystream(x, z) = |x|$ gibt, so dass $e(x, y) = d(x, z) = x \oplus keystream(x, y)$. Die Funktion keystream nenn wir Schlüsselstromgenerator und ihren Funktionswert Schlüsselstrom.

- Keystream sollte ein Pseudozufallszahlengenerator sein
- Keystream kann unabhängig vom INhalt der ersten Variable sein, also $keystream(x_1, k) = keystream(x_2, k)$ für beliebige x_1 und x_2 mit $|x_1| = |x_2|$
- Falls der Schlüsselstrom sich wieder, ist die Stromchiffre nicht mehr sicher

ChaCha20 ist eine moderne Stromchiffre, die als Alternative zu AES entwickelt wurde.

1.2 Kryptografische Hashfunktionen

Eine Hashfunktion ist ein Algorithmus, der eine Eingabe beliebiger Größe in einen Hashwert mit einer festen Länge umwandelt. Hashfunktionen sind deterministisch, erlauben eine schnelle Berechnung und bieten Integritätsschutz (Änderung der Eingabe ändert den Hash) Eigenschaften einer Hashfunktion:

1. **Pre-Image Resistance**
   Bei gegebenem Hashwert h ist es rechnerisch unmöglich, die ursprüngliche Nachricht m zu finden, so dass $H(m) = h$.
2. **Second-Image Resistance**
   Bei gegebener Nachricht m_1 ist es rechnerisch unmöglich, eine andere Nachricht m_2 zu finden, die denselben Hashwert erzeugt, so dass $H(m_1) = H(m_2)$
3. **Collision Resistance**
   Es ist rechnerisch unmöglich irgendwelche zwei unterschiedlichen Nachrichten m_1 und m_2 zu finden, die denselben Hashwert erzeugen, so dass $H(m_1) = H(m_2)$

Hashfunktion	Output	Sicherheit	Anwendung
MD5	128 Bits	Unsicher	X
SHA-1	160 Bits	Unsicher seit 2017	X
SHA-256	256 Bits	Sicher	TLS/SSL, hashing, Blockchain
SHA-3/Keccak	224/256/384/512 Bits	Sicher	Ähnlich wie SHA-2 (aber langsamer ohne Hardware Unterstützung)

Vergleich von Hashfunktionen

1.3 Asymmetrische Kryptografie

1.3.1 Grundlagen

Bei einem asymmetrischem Kryptosystem gibt es verschiedene Schlüssel.

1. Öffentliche Schlüsel werden frei für alle interessierten Mitredner veröffentlicht.
2. Eine geheime Nachricht muss erst mit dem öffenltichen Schlüssel verschlüsselt an den Empfänger zugeschickt werden.

Asymmetrische Kryptografie

Ein asymmetrisches Kryptosystem ist ein 7-Tupel $(M, K_s, K_p, K, C, e, d)$ bestehend aus
- einer Menge M von Klartexten,
- einer Menge K_s von geheimen/privaten Schlüsseln,
- einer Menge K_p von öffentlichen Schlüsseln
- einer Menge $K \subset K_s \times K_p$ von Schlüsselpaaren,
- einer Menge C von Chiffretexten,
- einer Verschlüsselungsfunktion $e : M \times K_p \to C$,
- einer Entschlüsselungsfunktion $d : C \times K_s \to M$,

so dass für alle Klartexte $m \in M$ und alle Schlüsselpaare $(s, p) \in K$ gilt, dass $d(e(m, p), s) = m$.

Prinzip:

- Verschlüsselung mit **öffentlichem Schlüssel** des Empfängers
- Entschlüsselung mit **privatem Schlüssel** des Empfängers
- Kein vorheriger Schlüsselaustausch nötig (im Gegensatz zu symmetrischer Kryptographie)

1.4 RSA-Kryptosystem

Idee

Das RSA-Kryptosystem (nach **Rivest, Shamir, Adleman**, 1977) ist das bekannteste Verfahren der asymmetrischen Kryptographie. Es basiert auf der Schwierigkeit, eine große Zahl $n = p \cdot q$ in ihre Primfaktoren zu zerlegen.

1.4.1 Schlüsselerzeugung

1. **Wähle zwei große Primzahlen** p, q mit $p \neq q$.
2. Berechne das **RSA-Modul**:
$$n = p \cdot q$$
3. Berechne die **Eulersche Totientfunktion**:
$$\varphi(n) = (p-1)(q-1)$$
Diese gibt die Anzahl der zu n teilerfremden Zahlen an.
4. Wähle den **Verschlüsselungsexponenten** e mit
$$1 < e < \varphi(n), \quad \gcd(e, \varphi(n)) = 1$$
(d. h. e und $\varphi(n)$ sind teilerfremd).
5. Berechne den **Entschlüsselungsexponenten** d als **modulares Inverses** von e:
$$d \cdot e \equiv 1 \pmod{\varphi(n)}$$
Dies geschieht mit dem **erweiterten Euklidischen Algorithmus**.

Beispiel: Erweiterter Euklidischer Algorithmus

Gegeben $e = 17$ und $\varphi(n) = 3120$:
Wir suchen d mit $17d \equiv 1 \pmod{3120}$.

$$\begin{aligned} 3120 &= 17 \cdot 183 + 9 \\ 17 &= 9 \cdot 1 + 8 \\ 9 &= 8 \cdot 1 + 1 \\ 8 &= 1 \cdot 8 + 0 \end{aligned}$$

Rückwärtseinsetzen:

$$1 = 9 - 8 = 9 - (17 - 9) = 2 \cdot 9 - 17 = 2(3120 - 17 \cdot 183) - 17 = 2 \cdot 3120 - 367 \cdot 17$$

Daraus folgt:

$$d \equiv -367 \equiv 2753 \pmod{3120}$$

Ergebnis: $d = 2753$.

Damit gilt:

$$\text{Öffentlicher Schlüssel: } (e, n), \quad \text{Privater Schlüssel: } (d, n)$$

1.4.2 Verschlüsselung und Entschlüsselung

- **Verschlüsselung:**
$$c = m^e \bmod n$$
- **Entschlüsselung:**
$$m = c^d \bmod n$$

Sicherheit von RSA:

- Basierend auf **Faktorisierungsproblem**: Schwierigkeit, n in p und q zu zerlegen
- Kenntnis von p, q oder $\varphi(n)$ ermöglicht Berechnung von d
- **Multiplikative Eigenschaft**: $(m_1^e \mod n) \cdot (m_2^e \mod n) \mod n = (m_1 m_2)^e \mod n \rightarrow$ problematisch für Sicherheit
- Durch **Quantencomputer** (Shor-Algorithmus) brüchbar
- p und q sollten groß und ähnlich groß sein (gleiche Bitlänge)

1.5 ElGamal-Kryptosystem

ElGamal Schlüsselerzeugung (Alice)

- Wähle eine **zyklische Gruppe** $G = (\mathbb{G}, \circ, e)$ mit großem Primzahlmodulus (z. B. $\mathbb{Z}_p^\times$) und einem **Erzeuger** g
- Wähle einen privaten Exponenten $a \in \{1, \ldots, \text{ord}(g) - 1\}$ und berechne $A = g^a \bmod p$
- **Privater Schlüssel:** a
- **Öffentlicher Schlüssel:** (G, g, A)

Verschlüsselung (an Alice):

- Wähle zufällig $r \in \{1, \ldots, \text{ord}(g) - 1\}$
- Berechne $R = g^r \bmod p$
- Berechne gemeinsamen Schlüssel $K = A^r = (g^a)^r = g^{ar} \bmod p$
- Berechne $C = (R,\, m \cdot K \bmod p)$ und sende C

Entschlüsselung (Alice):

- Berechne $K = R^a = (g^r)^a = g^{ra} \bmod p$
- Berechne das Inverse $K^{-1} \bmod p$
- Entschlüssele $m = C_2 \cdot K^{-1} \bmod p$

Sicherheit von ElGamal:

- Sicherheit basiert auf dem **Diskreten Logarithmusproblem (DLP)**: gegeben (g, g^a) ist a schwer zu bestimmen
- Angreifbar durch Quantencomputer (Shor-Algorithmus)
- **Probabilistisches Verfahren** durch Zufallswert $r \rightarrow$ semantisch sicher, wenn das **Decisional Diffie-Hellman-Problem (DDH)** schwer ist
- Aus einem gültigen Chiffrat (c_1, c_2) lässt sich leicht $(c_1,\, g \cdot c_2)$ für beliebiges $g \in G$ konstruieren — **nicht deterministisch**, daher keine Wiederverwendung von r

Hinweis: Bei allen Potenzoperationen und Multiplikationen muss stets das **Modulus** p angewendet werden. Der in der Vorlesung gezeigte Fehler (fehlendes mod$p$ bei K) wurde hier korrigiert.

- Kombination von asymmetrischer und symmetrischer Kryptographie

- Nachricht wird mit **symmetrischem Verfahren** verschlüsselt (schnell, effizient)
- Symmetrischer Schlüssel wird mit **asymmetrischem Verfahren** verschlüsselt (sicherer Schlüsselaustausch)
- Vorteile: Effizienz + Sicherheit, einfaches Teilen mit mehreren Empfängern
- Nachteil: Sicherheit von beiden Systemen abhängig

5. Digitale Signaturen

Zweck digitaler Signaturen

- **Authentizität**: Nachweis des Urhebers
- **Integrität**: Nachweis der Unverändertheit
- **Nicht-Abstreitbarkeit (Non-Repudiation)**: Unterzeichner kann Unterschrift nicht abstreiten

Rechtlicher Rahmen (eIDAS/VDG):

- **Einfache elektronische Signatur**: Keine besondere rechtliche Vermutung
- **Fortgeschrittene elektronische Signatur**: Eindeutige Zuordnung, hohes Vertrauen
- **Qualifizierte elektronische Signatur**: Rechtliche Gleichstellung mit handschriftlicher Unterschrift

1.5.1 RSA-Signaturen

RSA-Signaturverfahren

- Schlüsselgenerierung wie bei RSA
- Signieren: $s = (h(m))^d \mod n$
- Verifizieren: Teste ob $h(m) = s^e \mod n$
- **Hashfunktion h notwendig** zur Vermeidung von Angriffen

Digital Signature Algorithm (DSA)

DSA Parametergenerierung

1. Wähle Primzahl q (160/224/256 Bit)
2. Wähle Primzahl p (1024/2048/3072 Bit) mit $q \mid (p-1)$
3. Finde $g \in \mathbb{Z}_p^\times$ mit $\text{ord}(g) = q$
4. Parameter (p, q, g) sind öffentlich

DSA Schlüsselgenerierung und Signatur

- Privater Schlüssel: x mit $1 < x < q$
- Öffentlicher Schlüssel: $y = g^x \mod p$
- Signieren: Wähle k, berechne $r = (g^k \mod p) \mod q$, $s = k^{-1} \cdot (H(m) + r \cdot x) \mod q$
- Verifizieren: Berechne $w = s^{-1} \mod q$, $u_1 = H(m) \cdot w \mod q$, $u_2 = r \cdot w \mod q$, $v = (g^{u_1} \cdot y^{u_2} \mod p) \mod q$, akzeptiere wenn $v = r$

1.5.2 Sicherheitsbegriffe für Signaturen

Angreiferwissen

- **Key-Only Attack**: Nur öffentlicher Schlüssel bekannt
- **Known Signature Attack**: Nachricht-Signatur-Paare bekannt
- **Chosen Message Attack**: Signaturen für selbstgewählte Nachrichten erhältlich
- **Adaptive Chosen Message Attack**: Signaturen auch während Angriff erhältlich

Angreiferziele

- **Existential Forgery**: Neues gültiges Nachricht-Signatur-Paar
- **Selective Forgery**: Signatur für bestimmte neue Nachricht
- **Universal Forgery**: Signatur für beliebige Nachricht
- **Total Break**: Bestimmung des privaten Schlüssels

1.6 Schlüsselverteilung und Schlüsselaustausch

1.6.1 Schlüsselarten

- **Langzeitschlüssel**: Lange Gültigkeit (Monate/Jahre), häufig für Authentifizierung
- **Sitzungsschlüssel (Session Keys)**: Kurze Gültigkeit (eine Sitzung), reduziert Risiko bei Kompromittierung

Public Key Infrastructure (PKI)

Zertifikate

- Bestätigung der Zuordnung von öffentlichen Schlüsseln zu Identitäten durch vertrauenswürdige dritte Partei
- Enthalten: Öffentlicher Schlüssel, Name, Gültigkeitszeitraum, Aussteller, Signatur
- **X.509**: Hierarchisches, zentralisiertes System mit Root-Zertifikaten
- **Web of Trust**: Dezentrales System (PGP), gegenseitige Zertifizierung

1.6.2 Schlüsselaustauschprotokolle

Dolev-Yao-Angreifermodell

- Angreifer hat volle Kontrolle über Kommunikationskanal
- Kann: Abfangen, Verzögern, Unterdrücken, Ersetzen, Unter falscher Identität senden
- Kann **nicht**: Kryptographische Primitive brechen (perfekte Kryptographie angenommen)

Needham–Schroeder

- Symmetrische Version anfällig für Replay-Angriffe (veraltete Schlüssel)
- Asymmetrische Version sicherer, aber anfällig gegen aktive Angreifer ohne Authentifizierung

1.6.3 Diffie-Hellman-Schlüsselaustausch

Diffie-Hellman Protokoll

1. Einigung auf Primzahl p und Generator g von $\mathbb{Z}_p^\times$
2. A wählt a, sendet $g^a \mod p$ an B
3. B wählt b, sendet $g^b \mod p$ an A
4. A berechnet $(g^b)^a = g^{ab} \mod p$
5. B berechnet $(g^a)^b = g^{ab} \mod p$

Gemeinsamer Schlüssel: $K = g^{ab} \mod p$

Sicherheit:

- Basierend auf **Computational Diffie-Hellman Problem (CDH)**: Berechnung von g^{ab} aus g, g^a, g^b
- **Man-in-the-Middle-Angriff** möglich: Angreifer führt zwei separate DH-Protokolle
- Lösung: **Authenticated Diffie-Hellman** oder **Station-to-Station (STS)** Protokoll mit Signaturen

Station-to-Station (STS) Protokoll

- $A \to B$: g^a
- $B \to A$: g^b, $\{\text{sig}(sk_B, (g^a, g^b))\}_K$ mit $K = g^{ab}$
- $A \to B$: $\{\text{sig}(sk_A, (g^a, g^b))\}_K$

Signatur gewährleistet Authentizität und Integrität.

Logjam-Angriff (2015):

- Vorberechnung des diskreten Logarithmus für häufig verwendete Primzahlen
- Betraf 512/768 Bit, Abschätzung für 1024 Bit möglich
- Lösung: Verwendung größerer, individueller Primzahlen