Kapitel 11: Zufall, Strom- und Blockchiffre, Schlüssel

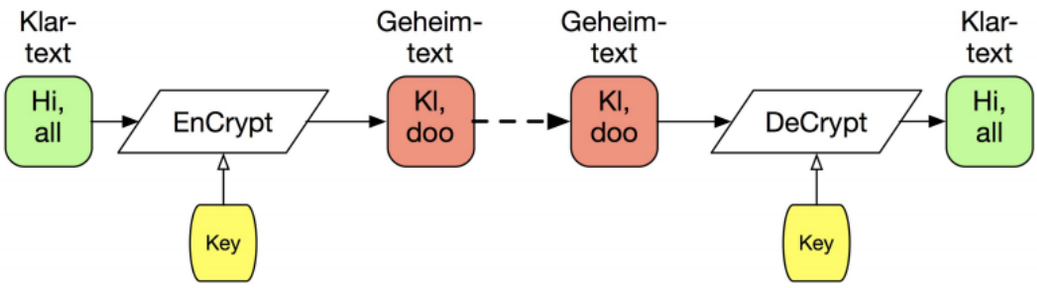
Schutzziele

- **Vertraulichkeit**: Daten sind nur den befugten Personen zugänglich
  → durch Verschlüsselung erreicht
- **Integrität**: Daten sind korrekt und wurden während der Übertragung nicht verändert
  → Änderungen können durch kryptographische Prüfsummen erkannt werden
- **Authentizität**: Daten stammen vom vorgeblichen Erzeuger
  → Identität des Erzeugers kann durch digitale Signaturen überprüft werden
- **Verfügbarkeit**: Daten bzw. Systeme können von befugten Personen gelesen, bearbeitet oder benutzt werden
  → durch redundante Systeme erreicht

Verschlüsselungsverfahren

verschlüsselte Kommunikation

Erwartung: Ohne die Kenntnis des Schlüssels (Key) ist eine Entschlüsselung unmöglich!



Verschlüsselung: Begriffe

- **Klartext**: ursprünglich lesbare Nachricht
- **Geheimtext**: verschlüsselte Nachricht
- **Verschlüsselung**: Verschlüsselungsalgorithmen zur Verschlüsselung der Nachricht
- **Schlüssel**: Parameter, der den Verschlüsselungsalgorithmus steuert
- **Entschlüsselung**: Umkehrung der Verschlüsselung

Verschlüsselungsverfahren: Klassifikation

- **Substitution**: Zeichen werden ersetzt
- **Transposition**: Reihenfolge der Zeichen werden vertauscht
- **Anzahl der Schlüssel:**
  - ⇨ **Symmetrisch**: wenn 1 Schlüssel für das Verschlüsseln und Entschlüsseln zuständig ist (Private Key Verfahren)
  - ⇨ **Asymmetrisch**: wenn Schlüsselpaar verwendet wird (Public Key Verfahren)
- **Verarbeitung des Klartextes:**
  - ⇨ **Blockverschlüsselung**: Klartext wird in Blöcke fester Größe eingeteilt, bevor die Blöcke in „Runden“ verschlüsselt werden
  - ⇨ **Stromverschlüsselung**: Klartext wird als Folge von Zeichen betrachtet und verschlüsselt jedes Zeichen einzeln

Verschlüsselung: Begriffe

- **Klartext**: ursprünglich lesbare Nachricht
- **Geheimtext**: verschlüsselte Nachricht
- **Verschlüsselung**: Verschlüsselungsalgorithmen zur Verschlüsselung der Nachricht
- **Schlüssel**: Parameter, der den Verschlüsselungsalgorithmus steuert
- **Entschlüsselung**: Umkehrung der Verschlüsselung

Private Key Verschlüsselung

- Symmetrische Verschlüsselung
  → One-Time-Pad ist unknackbar

Kerckhoffs'sches Prinzip

- Die Sicherheit eines Verschlüsselungsverfahrens muss auf der Geheimhaltung des Schlüssels beruhen, nicht auf der Geheimhaltung des Verfahrens.
- **Gegenteil ist „Security through/by obscurity"**
- Vorgänge im Verschlüsselungsverfahren sind undurchsichtig und nicht publik

Zufall und Zufallszahlengeneratoren

Zufall und Entropie

- Als **zufällig** werden Ereignisse bezeichnet, für die keine **kausale** Erklärung gefunden werden können.
  → Die **kausalen bzw. deterministischen Zusammenhänge sind unbekannt**.
- **Entropie** bezeichnet in der Informationstheorie ein Maß für den **mittleren Informationsgehalt** einer Nachricht.
  - Die Entropie ist am höchsten, wenn die Wahrscheinlichkeit aller zu erwartenden Zahlen gleich ist.
  - ~~Entropie kann als Maß für den Zufall in einem System verstanden werden.~~
  - Idealer Münzwurf liefert Kopf oder Zahl, ~~beide mit der gleichen~~
  - ~~Wahrscheinlichkeit~~ $P = 0.5$ Die Zufallsinformation beträgt 1 bit (Bei eine gezinkten ~~Münze < 1 bit~~).

Zufallszahlengeneratoren

- Verfahren, dass eine Folge von Zufallszahlen erzeugt.
  (Anwendungsbeispiele sind z.B. Spiele, bspw. Steuerung von Spielfiguren)

- **True Random Number Generators (TRNG)**
  - Quelle: zufällige physikalische Prozesse. || Nicht reproduzierbar, **nicht-deterministisch**

- **Pseudo-Random Number Generators (PRNG)**
  - Berechnet, also **deterministisch** || Folge: s_0 = seed, $s_{i+1} = f(s_i)$

- **Cryptographically Secure** Pseudo-Random Number Generators (CSPRNG) sind RNGs **für den Anwendungsbereich der Kryptographie.**

Unterscheidung in

- **deterministische** CS(P)RNG
  - basiert bspw. auf kryptographischen Primitiven wie Verschlüsselungs- oder Hash-Algorithmen
- **nichtdeterministische** CSRNG
  - bspw. hardwarebasierter Zufallsgenerator

Wofür werden Zufallszahlen benötigt?

Kryptografisch sichere Zufallszahlengeneratoren werden für kryptografischen Verfahren benötigt, bspw.

- Schlüsselgenerierung
- Schlüsselstrom einer Stromverschlüsselung
  - Zeichenweise: Chiffretextzeichen = Klartextzeichen XOR (Zeichen aus dem Schlüsselstrom)
- Salt
  - Zufällig Zeichenfolge, die mit einem Passwort verbunden wird. Das Ergebnis dient als Eingabe für eine Hash-Funktion. Der Hashwert wird in einer Passwort- Datenbank gespeichert.
- One-Time-Pad
- Digitale Signaturen
- Schlüsselverteilungsprotokolle

One-Time-Pad

- Polyalphabetisches Substitutionsverfahren
- **Stromverschlüsselung**
- Pro Zeichen wird ein **Einmalschlüssel** verwendet. Ein **Schlüssel darf immer nur** maximal **einmal** verwendet (und muss anschließend vernichtet) werden.
- Der Schlüssel ist genauso lang, wie die Nachricht.
- Der Schlüsselstrom muss aus einem TRNG stammen.
- Mit maximaler Rechenleistung **unmöglich** zu brechen.

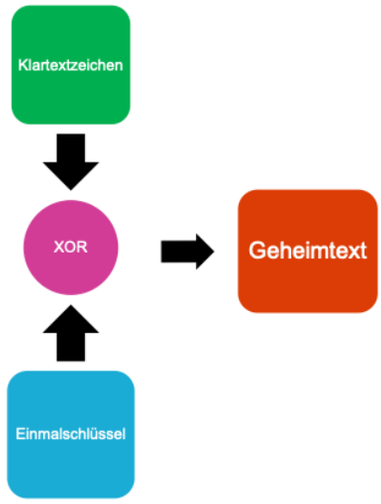
Problem: Schlüssel ist genauso lang, wie die Nachricht. Verlagert somit das Problem auf die Schlüsselverteilung: „Wie kommt Schlüssel S von A nach B?“

Funktionsweise

Zeichenweise durch die Klartextnachricht iterieren

• Pro Zeichen

1. Klartextzeichen aus dem Klartextstrom nehmen
2. Schlüsselstromzeichen (Einmalschlüssel) aus dem Schlüsselstrom nehmen
3. Beide entnommene Zeichen per XOR verknüpfen, um Geheimtextzeichen zu erhalten
4. Geheimtextzeichen in Geheimtextstrom einfügen



Hinführung zu anderen Verfahren

Größter Nachteil: Schlüssel ist genauso lang, wie die Nachricht.

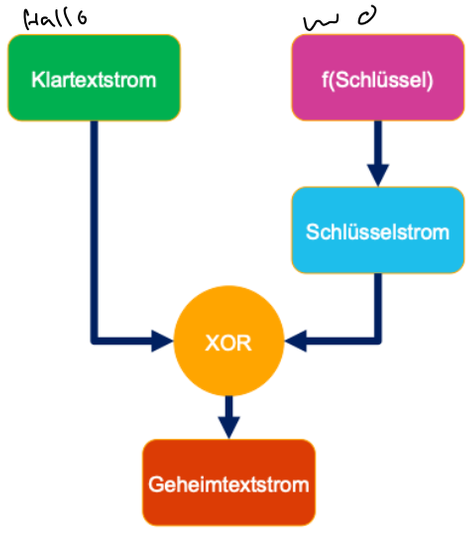
- Verlagert somit das Problem auf die Schlüsselverteilung:
  „Wie kommt Schlüssel S von A nach B?“

→ Moderne symmetrische und asymmetrische Verschlüsselungsverfahren (bereits in GDI vorgestellt) versuchen u.a. genau dieses Problem zu lösen!

Block-/Stromchiffren

Stromchiffre

- Stromchiffre sind der Versuch, ein „**OTP mit einem kurzen Schlüssel**“ zu erstellen;
  OTP birgt bei größeren Datenmengen das Problem der Schlüsselverteilung.
- Eine Stromchiffre erzeugt aus einem Schüssel fester Länge einen beliebig langen Schlüsselstrom, der ähnlich wie ein OTP verwendet werden kann, um Klartextdaten via XOR zu verknüpfen.
- Stromchiffre waren im Vergleich zu Blockchiffren früher viel beliebter - heute ist es genau umgekehrt. **Warum?**
- Früher waren CPUs nicht so leistungsfähig wie heute. Daher wurden Verschlüsselungsverfahren in Hard- statt Software realisiert.
- Stromverschlüsselungsverfahren lassen sich sehr effizient in Hardware umsetzen.



- Schlüsselstrom
  - zufällige (in der Regel pseudozufällig) Zeichenfolge
  - abgeleitet aus dem Schlüssel
    (f sei Schlüsselableitungsfunktion im Diagramm)
- Klartext wir **zeichenweise** mittels **XOR** mit dem Schlüsselstrom verknüpft
- Verwendung in Echtzeitanwendung, bspw. möglichst latenzfreie Sprachübertragung

Stromchiffre am Beispiel von RC4

- RC4: Ron's Code 4
- Von Ronald Rivest in 1987 entwickelt und 1994 geleakt
- Früher sehr weit verbreitet: WEP, WPA-TKIP, TLS, SSH1
  - IETF verbietet mit RFC 7465 den Einsatz von RC4
- Sich selbst modifizierende Umsetzungstabelle, die aus einer Permutation von 0,1, ..., 255 besteht
- Permutation wird mit Hilfe des Schlüssels initialisiert
- Pro Schritt führt RC4 aus:
  1. Zwei Elemente in der Umsetzungstabelle vertauschen
  2. Ein Byte als Zeichen des Schlüsselstroms aus der Umsetzungstabelle auswählen

Algorithmus: S-Box berechnen

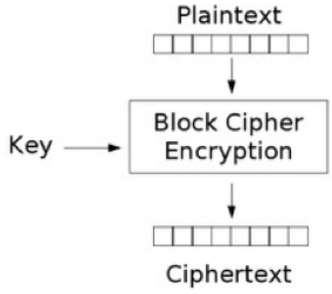
```
k[]: gegebene Schlüssel-Zeichenfolge, meist 40 bis 256 Byte
L := Länge des Schlüssels in Byte
sbox[]: Byte-Vektor der Länge 256
For i = 0 to 255
        sbox[i] := i
j := 0
For i = 0 to 255
        j = (j + s[i] + k[i mod L]) mod 256
        vertausche sbox[i] mit sbox[j]
```

Algorithmus: Ver- und Entschlüsselung

```
klar[]: gegebene Klartext-Zeichenfolge der Länge X
chif[]: Vektor zum Abspeichern des Schlüsseltextes
i := 0
j := 0
For n = 0 to X-1
        i := (i + 1) mod 256
        j := (j + s[i]) mod 256
        vertausche sbox[i] mit sbox[j]
        schlüsselstromzeichen:= sbox[(sbox[i] + sbox[j]) mod 256]
        chif[n] := schlüsselstromzeichen XOR klar[n]
```

Blockchiffre

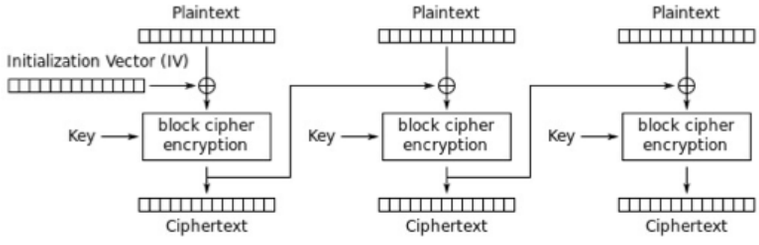
- Im Unterschied zur Stromchiffre, wird **nicht zeichenweise, sondern blockweise** verfahren.
  - Um längere Daten zu verschlüsseln, wird ein **Betriebsmodus** verwendet.
- Typischerweise rundenbasiert: Aus einem Schlüssel werden mehrere Rundenschlüssel abgeleitet.
- Klartextblock und Schlüssel dienen als Eingabe für Verschlüsselungsfunktion: $Sei\ C\ Chiffretextblock, K\ Klartextblock\ und\ S\ Schlüssel$ dann gilt C = f(K, S)



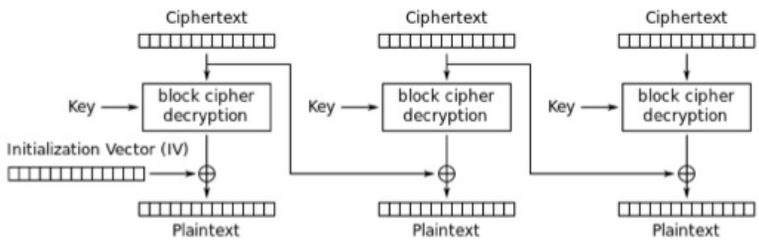
Grundlegende Frage:
Wie werden Nachrichten mit einer Blockchiffre verschlüsselt, wenn die Nachricht länger als die Blockgröße ist?

Cipher Block Chaining Mode (CBC)

- Behebt Schwächen aus ECB
- Sender*in muss Empfänger*in den verwendeten Initialisierungsvektor bereitstellen



Cipher Block Chaining (CBC) mode encryption



Initialisierungsvektoren (IV)

Zufallsdaten fester Länge (echter Zufall oder Pseudozufall)

- Ziel: **Nichtdeterminismus**

Anwendungsbeispiel: Cipher Block Chaining (CBC)

- IV sorgt dafür, dass gleiche Klartextblöcke bei gleichem Schlüssel zu unterschiedlichen Geheimtextblöcken führen

Bedingungen

- **Einzigartigkeit**: kein IV darf bei Verwendung des gleichen Schlüssels wiederverwendet werden
- **Nicht im Voraus zu berechnen**

Schlüssel und Schlüsselgenerierung

Wie sollten Schlüssel generiert werden?

- Für asymmetrische Verfahren (wie bspw. RSA) gibt der Algorithmus die Schlüsselgenerierung vor (s. RSA in GDI)
- Für symmetrische Verfahren
  - Zufallszahlengenerator
  - Diffie-Hellmann-Schlüsselaustausch
  - Schlüsselableitungsfunktion, die aus einem Passwort Schlüsselmaterial erstellt

Schlüsselgenerierung

Eine Schlüsselableitungsfunktion (engl. key derivation function) erzeugt Schlüsselmaterial aus einem Eingabeschlüssel:

$$Schlüssel = f(Eingabeschlüssel)$$

Eine key derivation function

- sorgt dafür, dass ein Angreifer*in keine Rückschlüsse auf den eigentlichen Schlüssel erhält
- kann die Verwendung von „schwachen" Schlüsseln verhindern bzw. die Entropie des Schlüsselmaterials (mit Hilfe bspw. eines Salts) steigern
- wird meist als rechenintensiver Algorithmus implementiert, um Brute-Force- Angriffe auszubremsen

Password-Based Key Derivation Function 2 (PBKDF2)

PRF:	Pseudozufallsfunktion, bspw. keyed-HMAC
hLen:	Länge der Ausgabe von PRF
Password:	Passwort
Salt:	kryptografischer Salt
c:	Anzahl der Iterationen
dkLen:	Länge des abgeleiteten Schlüssels
i:	Index

Dann ist

U_1 = PRF(Password, Salt | INT_#@_BE(i)), U_2+PRF(Passwort,U_1),…,

U_c = PRF(Password,U_{c-1})

F(Password, Salt, c, i) = U1 ⊕ U2 ⊕ ... ⊕ UC

T_i = F(Password, Salt, c, i)

DK = T_1 | T_2 | ... | $T_{dkLen/hLen}$ ist der abgeleitete Schlüssel (| ist Konkatenationsoperator)

DK = PBKDF2(PRF, Password, Salt, c, dkLen)

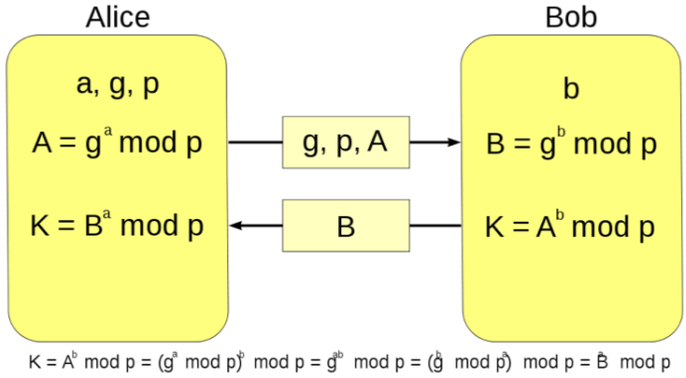
Beispiel: WPA2 verwendet DK = PBKDF2(HMAC−SHA1, passphrase, ssid, 4096, 256)

Diffie-Hellmann-Schlüsselaustausch

Diffie-Hellman-Schlüsselaustausch ermöglicht es zwei Parteien, einen Schlüssel über eine öffentliche, abhörbare Leitung (bspw. Internet) zu vereinbaren.

Voraussetzungen:

- Die beiden Parteien seien Alice und Bob.
- Alice und Bob einigen sich auf eine große Primzahl p
- Alice und Bob einigen sich auf eine natürlich Zahl g < p
- Alice erzeugt Zufallszahl a
- Bob erzeugt Zufallszahl b
- p und g sind öffentlich
- a und b sind geheim



Zusammenfassung

- Die Sicherheit eines Verschlüsselungsverfahrens muss auf der Geheimhaltung des Schlüssels beruhen, nicht auf der Geheimhaltung des Verfahrens.
- Zufallszahlengenerator ist ein Verfahren, das eine Folge von Zufallszahlen erzeugt.
  Zufall: die kausalen Zusammenhänge sind unbekannt.
- Stromchiffre verschlüsselt zeichenweise, Blockchiffre blockweise
  - Blockchiffre benötigt meist ein Betriebsmodus
- Schlüssel sollten generiert werden, aus einer Schlüsselableitungsfunktion (key derivation function) stammen und ausreichend Entropie aufweisen.