

# Datenkommunikation und Sicherheit

## Kapitel 6: Internet-Protokolle und Sicherheit

Klaus Wehrle

Communication and Distributed Systems

Chair of Computer Science 4

RWTH Aachen University

<http://www.comsys.rwth-aachen.de>



VI-1

# Themenübersicht

## • Datenkommunikation und Sicherheit

- ▶ Einführung, Begriffe und allgemeine Grundlagen
- ▶ Technische und nachrichtentechnische Grundlagen: Medien, Signale, Bandbreite, Leitungscodes und Modulation, Multiplexing
- ▶ Lokale Netze: Strukturierung des Datenstroms, Fehlererkennung/-behebung, Flusssteuerung, Medienzugriff, Ethernet
- ▶ Internet und Internet-Protokolle:
  - Vermittlungsschicht: IP, Routing
  - Transportschicht: TCP
- ▶ Grundlagen der Sicherheit in/von Kommunikationsnetzen
  - Grundlagen: Verschlüsselung, Authentifizierung, Integrität
  - Sichere Internet-Protokolle
  - Firewalls

- **Grundlagen der Sicherheit**

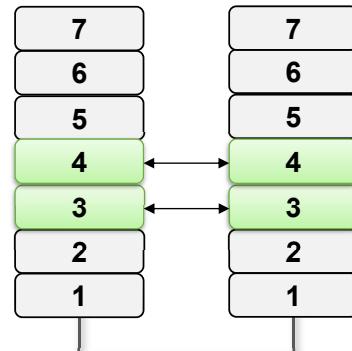
- ▶ Sicherheitsprobleme und -ziele
- ▶ Symmetrische und asymmetrische Verschlüsselung
- ▶ Authentifizierung

- **Sichere Internet-Protokolle**

- ▶ Vermittlungsschicht: IPSec, IKE
- ▶ Transportschicht: TLS

- **Schutz von lokalen Netzen**

- ▶ Firewalls: Packet Filter, Application Gateways



## Sicherheit im Internet?

- **Entwurfsziel bei der Schaffung des Internets:**
  - ▶ Einfache Implementierbarkeit der Protokolle
  - ▶ Offener Zugang zu den angeschlossenen Forschungseinrichtungen
- **Angenommen wurde:**
  - ▶ Überschaubare Nutzerzahl
  - ▶ Wohlmeinendes Verhalten der Nutzer
- **Realität im heutigen Internet:**
  - ▶ Vielzahl an (sicherheitskritischen) Diensten
  - ▶ Nutzer mit niederen Motiven
  - ▶ Die Technologie des Internets zeigt massive Sicherheitslücken:
    - ... sowohl im Design der Protokolle
    - ... als auch in fehlerhaften Implementierungen von Protokollen, die Fehlverhalten einzelner Rechner bewirken können

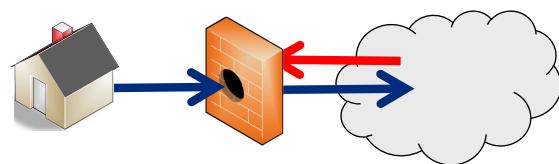
## Sicherheitsprobleme

- **TCP/IP erlaubt die Kommunikation über das Internet**

► Aber: wie schützt man seine eigenen Rechner vor Zugriffen?

- Ausspionieren von vertraulichen Daten
- Einbruch auf Rechner
- Verbreitung von Viren auf Rechnern
- ...

*Firewalls* zum Schutz des  
eigenen Netzes anhand von  
Adressinformationen des  
Absenders und des Ziels



## Sicherheitsprobleme

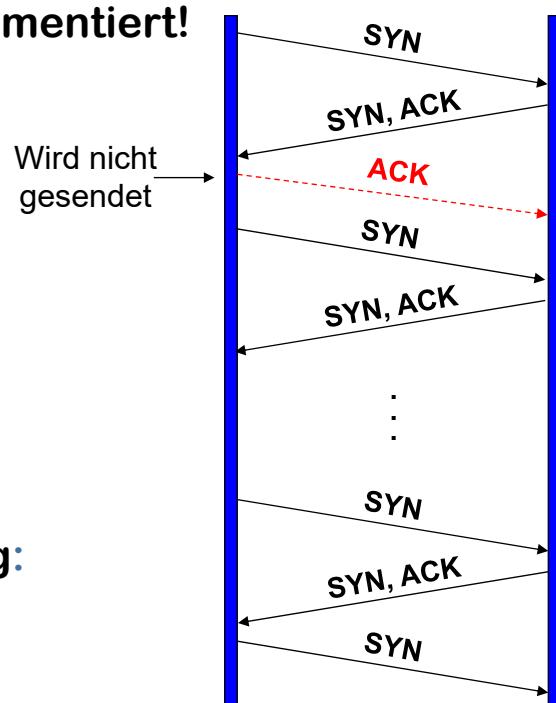
- Internet-Protokolle sind teilweise unzureichend spezifiziert und oft auch implementiert!

- *Denial of Service (DoS)*

- ▶ Durch *SYN-Flooding*
- ▶ Verschärfung: *Distributed DoS*

- System zur Angriffserkennung:

- ▶ *Intrusion Detection*



Einfaches Modell eines DoS-Angriffs: SYN-Flooding - Aufbau mehrerer halb-offener TCP-Verbindungen.

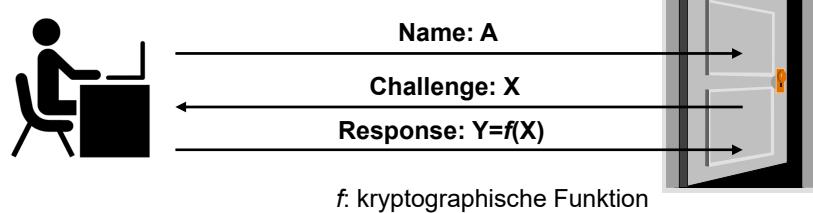
Verzichtet man beim TCP-Verbindungsaufbau auf die dritte Nachricht des Three-Way-Handshakes, bleibt die angesprochene TCP-Implementierung in einem Zustand, bei dem sie auf diese dritte Nachricht wartet; Ressourcen für die Verbindung wurden bereits reserviert, eine Kommunikation wird erwartet. Der Server erkennt halb-offene Verbindungen nach einem Timeout, um zu vermeiden, dass im Fehlerfall eine ausbleibende dritte Nachricht Ressourcen auf ewig blockiert. Werden allerdings mit hoher Anfragerate halb-offene Verbindungen erzeugt, werden nach und nach alle Ressourcen eines Servers aufgebraucht. Der Server kann keine Anfragen mehr entgegennehmen, seinen „Dienst“ nicht mehr erbringen.

Sogenannte Intrusion-Detection-Systeme dienen dazu, Angriffe durch auffällige Kommunikationsmuster zu entdecken und Gegenmaßnahmen zu ergreifen.

## Sicherheitsprobleme

- **Firewalls: Zugriff auf lokale Netze einschränken**
  - ▶ Verwenden Informationen wie z.B. IP-Adressen, Ports, Protokoll
- **Aber: Zugriffskontrolle auf bestimmte Dienste, z.B. Online-Banking**
  - ▶ Zugriff auf bestimmte Personen/Prozesse beschränken
    - Einführung personengebundener Zugriffskontrolle
    - Verwendung von Login und Passwort (bzw. PIN, ...)

→ *Authentizität* der Benutzer durch *Authentifizierung*



## Sicherheitsprobleme

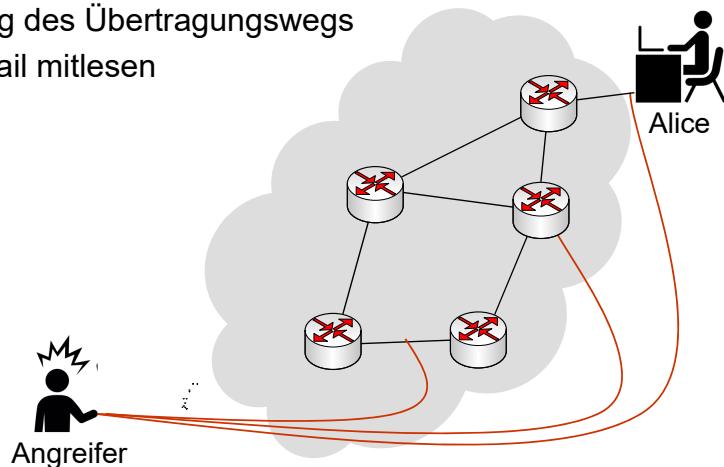
- Internet-Protokolle übertragen Informationen ohne Überprüfung des Absenders
  - ▶ *Spoofing* (= Vortäuschen einer falschen Identität)
    - ARP-Spoofing (= Hinterlegen falscher Adressauflösungen in ARP-Caches)
    - IP-Spoofing (= Senden unter falscher IP-Adresse)
    - ICMP-Spoofing (= z.B. Senden eines Redirect-Befehls unter falscher Adresse)
    - Mail-Spoofing (= Senden unter falscher E-Mail-Adresse)
    - ...
  - ▶ *Hijacking*: Erraten von TCP-Folgenummern und Einfügen eigener Kommandos
- ➡ *Digitale Signaturen zur Sicherstellung von Authentizität/Integrität bzw. Intrusion Detection*

# Sicherheitsprobleme

- Internet-Protokolle sind einfach gehalten

- ▶ Beispiel E-Mails: werden in ASCII-Text übertragen
  - TCP und IP nehmen keine Veränderung des Payloads vor
  - Daten werden im Klartext übertragen
  - Bösartige Nutzer (*Angreifer*), die Zugriff auf einen Router entlang des Übertragungswegs haben, können die E-Mail mitlesen
- ▶ Gleiches gilt für andere Anwendungsprotokolle

→ *Vertraulichkeit* der Daten durch *Verschlüsselung*



# Sicherheitslücken und -lösungen

## • Sicherheitslücken in Infrastruktur und Protokollen:

- ▶ Zugriff auf interne Rechner,  
Angriffe auf geschützte Netze

*Firewalls, Intrusion Detection*

- ▶ Vortäuschen falscher Identität

■ Sicherheitsziel: Authentizität

*Authentifizierung*

- ▶ Zugriff auf persönliche Daten

■ Sicherheitsziel: Zugriffskontrolle

*Authentifizierung*

- ▶ Mitlesen vertraulicher Informationen

■ Sicherheitsziel: Vertraulichkeit

*Verschlüsselung*

- ▶ Modifikation übertragener Daten

■ Sicherheitsziel: Integrität

*Verschlüsselung, Integritätsprüfung*

- ▶ Verantwortungsloser Umgang z.B. mit Passwörtern?

Kurze Zusammenfassung: Die angesprochenen Probleme lassen sich auf ein paar Grundmechanismen reduzieren – auf Protokollseite sind dies Verschlüsselung, Authentifizierung, Integrität. Hierzu werden Erweiterungen für die Internetprotokolle (hier behandelt: IP und TCP) benötigt.

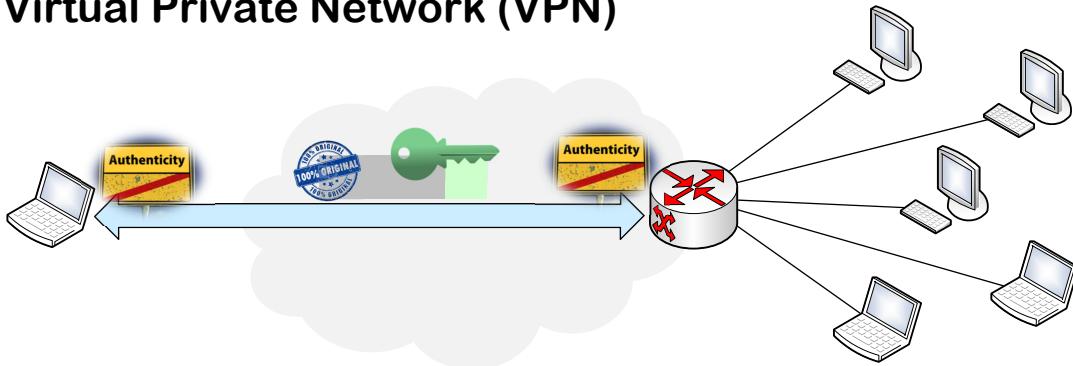
Auf Infrastrukturseite ist eine Ergänzung lokaler Netze durch Firewalls und Intrusion-Detection-Systeme nötig.

## Typische Einsatzgebiete

- Sicherer Zugriff auf Webseiten, z.B. Online-Banking



- Virtual Private Network (VPN)



Ein typisches Einsatzszenario, bei dem Vertraulichkeit, Authentizität und Integrität wichtig sind, ist Online-Banking:

- Nutzer und Banking-Server müssen sich gegenseitig authentifizieren, um sicher sein zu können, dass sie jeweils mit dem erwarteten Partner reden.
- Transaktionen werden verschlüsselt übertragen, damit niemand Kontodetails ausspionieren kann.
- Die Übertragung muss gegen Modifikationen gesichert werden, damit niemand z.B. das Zielkonto einer Überweisung abändern kann.

Auch oft verwendet sind Virtual Private Networks – Integration eines Rechners in ein lokales Netz über das öffentliche Internet:

- Rechner und Zugangsrouter des lokalen Netzes müssen sich gegenseitig authentifizieren, damit nur autorisierte Rechner dem lokalen Netz beitreten können und der Rechner sicher sein kann, dem richtigen Netz beizutreten.
- Alle Übertragungen werden verschlüsselt, damit sie genauso wenig belauscht werden können wie im lokalen Netz
- Abhängig vom Einsatzgebiet ist auch Integritätsschutz wichtig.

- **Beschränkung auf Sicherheitserweiterungen für TCP/IP**

- ▶ Erweiterungen auf der Vermittlungsschicht
  - **IPSec** (Internet Protocol Security)
    - Vertraulichkeit, Authentizität, Integrität
  - **IKE** (Internet Key Exchange, v2)
    - Etablierung von Schlüsseln für IPSec
- ▶ Erweiterungen auf der Transportschicht
  - **TLS** (Transport Layer Security)
    - Vertraulichkeit, Authentizität, Integrität
- ▶ Des weiteren: Einsatz von **Firewalls**

Sicherheitsmechanismen sind auf allen Protokollebenen implementiert. Dies fängt auf Schicht 2 an, wo netzspezifische Sicherheitsfunktionen implementiert werden (z.B. Authentifizierung gegenüber einem WLAN-Access-Point, bevor ein Rechner dem Netz beitreten kann), und geht hin bis zur Anwendungsschicht, auf der anwendungsspezifische Sicherheitsfunktionen implementiert werden (z.B. PGP für Verschlüsselung/Authentifizierung bei E-Mails).

Sowohl auf IP- als auch auf TCP-Ebene werden Sicherheitsmechanismen implementiert – auf TCP-Ebene bringt eine Implementierung den Vorteil, dass man der Anwendung näher ist und Ende-zu-Ende-Sicherheit zwischen beliebigen Anwendungen realisieren kann; auf IP-Ebene ist es einfacher, die Sicherung von Übertragungen z.B. zwischen zwei LANs zu realisieren. Auf welcher Ebene man die Sicherheitsmechanismen einsetzt, hängt damit ganz davon ab, welches Ziel man verfolgt.

# Kapitel 6: Internet-Protokolle und Sicherheit

- **Grundlagen der Sicherheit**

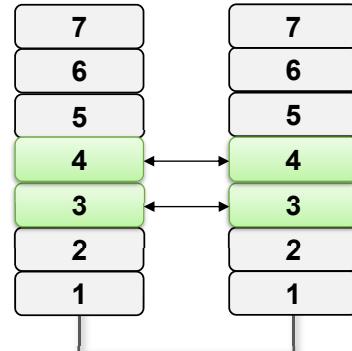
- ▶ Sicherheitsprobleme und -ziele
- ▶ Symmetrische und asymmetrische Verschlüsselung
- ▶ Authentifizierung

- **Sichere Internet-Protokolle**

- ▶ Vermittlungsschicht: IPSec, IKE
- ▶ Transportschicht: TLS

- **Schutz von lokalen Netzen**

- ▶ Firewalls: Packet Filter, Application Gateways



- **Grundlagen der Sicherheitstechniken aller Protokolle**

- **Vertraulichkeit durch Verschlüsselung:**

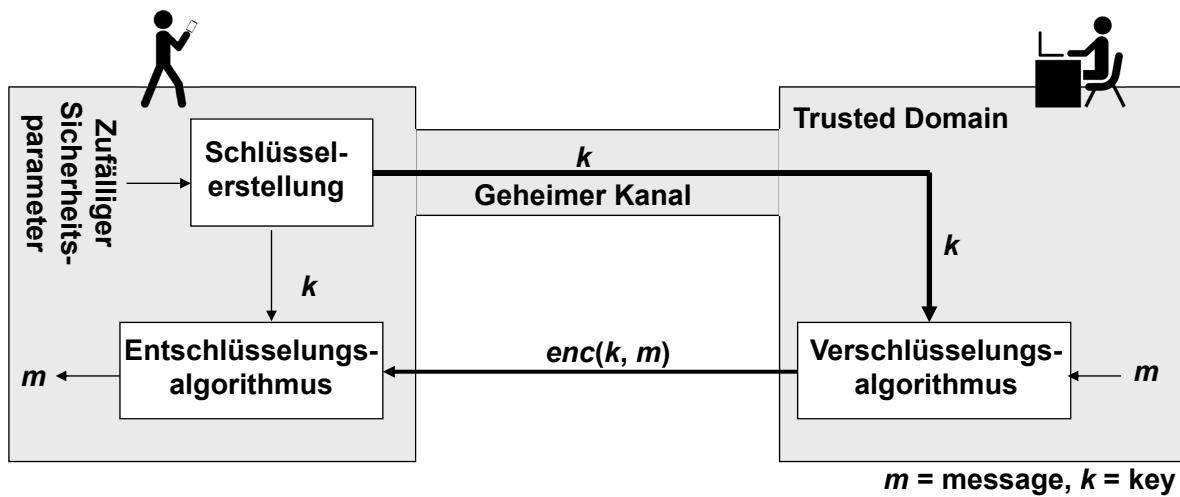
- **encryption\_algorithm:** (Klartext, Schlüssel) → Ciphertext
    - **Symmetrische** Verschlüsselungsverfahren
      - Einigung auf einen gemeinsamen Schlüssel
    - **Asymmetrische** Verschlüsselungsverfahren
      - Unterschiedliche Schlüssel zur Ver- und Entschlüsselung

- **Authentizität und Integrität**

- Durch Verschlüsselungsverfahren und Hash-Funktionen
      - Z.B. digitale Signaturen
      - **encryption\_algorithm:**  
 $(\text{hash}(\text{Klartext}), \text{Schlüssel}) \rightarrow \text{Signatur}$
      - Authentifizierungsstellen

# Symmetrische Verschlüsselung

- Viele Verfahren, z.B. DES, AES, RC4, ...
  - ▶ Zur schnellen Übertragung großer Datenmengen
  - ▶ *Sicherer Kommunikationskanal* ist nötig, um gemeinsamen Schlüssel zu vereinbaren



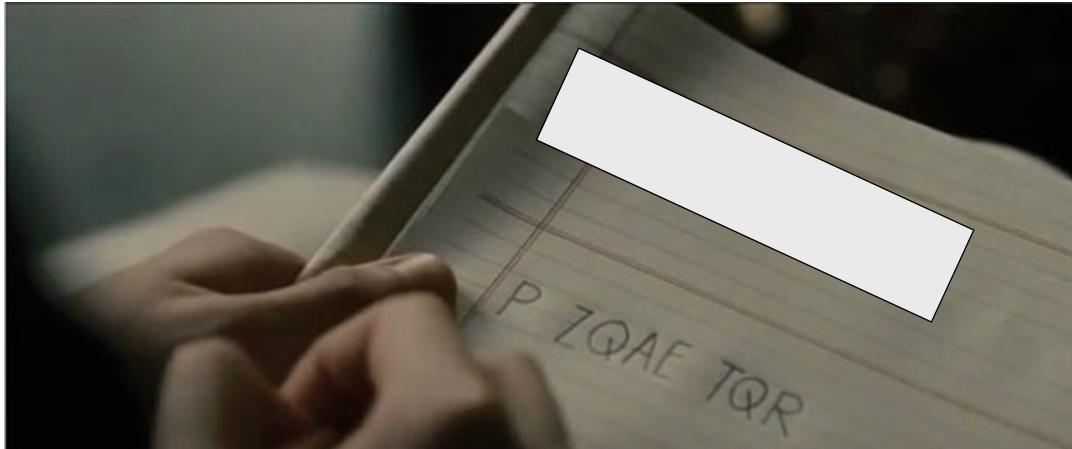
Generelles Merkmal symmetrischer Verschlüsselungsverfahren: die beiden Kommunikationspartner (und nur diese) kennen einen gemeinsamen Schlüssel  $k$ , mit dessen Hilfe die zu übertragende Nachricht durch einen bekannten Algorithmus so verwürfelt wird, dass man sie ohne Kenntnis des Schlüssels nicht dekodieren kann.

Das Problem bei symmetrischen Verfahren ist, dass ein geheimer Kommunikationskanal benötigt wird, um sich auf einen gemeinsamen Schlüssel zu einigen, ohne dass ein Dritter ihn mitbekommt.

## Einfache Verschlüsselungsverfahren

- **Monoalphabetische Substitutionschiffre**

- ▶ Bilde Zeichen  $x_n$  auf andere Zeichen  $y_n$  aus dem Alphabet ab
- ▶ A->K, B->D, C->Z, ...



(aus dem Film: The Imitation Game)

## Einfache Verschlüsselungsverfahren

- **Shift Cipher.**

- ▶ Ersetze ein Zeichen  $x$  durch das Zeichen, welches  $x$  im Alphabet  $k$  Positionen nachfolgt
  - Verschlüsselter Text  $y = x + k \text{ mod } n$
- ▶ Erweiterungen: Verwendung einer Permutationstabelle oder einer Funktion zur Ersetzung

- **Vigenère Cipher.**

- ▶ Verschlüssle ganze Blöcke auf einmal
- ▶ Verwende Schlüssel  $k = (k_1, k_2, \dots, k_m)$
- ▶ Verschlüsselung:  $(x_1, \dots, x_m) = (x_1 + k_1, \dots, x_m + k_m)$
- ▶ Wiederum Varianten definiert (Tabellen, Funktionen)

Eins der einfachsten Verschlüsselungsverfahren ist die Shift Cipher – ersetze ein Zeichen  $x$  eines gegebenen Alphabets durch das Zeichen, welches  $x$  im Alphabet um  $k$  Positionen nachfolgt (modulo  $n$  gerechnet, mit  $n$ : Anzahl der Zeichen im Alphabet).

Bestandteile des Algorithmus‘:

- $k$  ist der Schlüssel
- $y = x + k \text{ mod } n$  ist Verschlüsselungsfunktion mit  $y$  = verschlüsselter Text
- $x = y - k \text{ mod } n$  ist Entschlüsselungsfunktion

Problem dieses einfachen Vorgehens ist, dass ein bestimmtes Zeichen immer auf das gleiche andere Zeichen abgebildet wird und dadurch verschlüsselte Texte sehr einfach analysiert werden können: versuche, Zeichen aufgrund ihrer Häufigkeit zu erraten und den Klartext zu rekonstruieren. (Außerdem wäre die Verschlüsselung einfach zu knacken – es ist kein Problem, alle möglichen Schlüssel zu testen, um den verwendeten zu ermitteln – ein sogenannter Brute-Force-Angriff.)

Eine Erweiterung wäre z.B. die Vigenère Cipher – verschlüssle einen ganzen Block auf einmal, indem für jedes Zeichen eines Blocks ein eigener Schlüssel verwendet wird. Dadurch wird mehr Zufall in der verschlüsselten Nachricht erzeugt: ersetze jedes Zeichen abhängig von seiner Position im Block durch ein anderes Zeichen.

Dennoch ist auch diese Verschlüsselung zu einfach, Analysen oder Brute-Force-Angriffe sind immer noch einfach machbar.

In der Praxis werden komplexere Verfahren benötigt – Standard heute sind Blockchiffren. Rein prinzipiell ist auch die Vigenère-Cipher eine Blockchiffre, die allerdings für den praktischen Einsatz um Größenordnungen zu einfach ist.

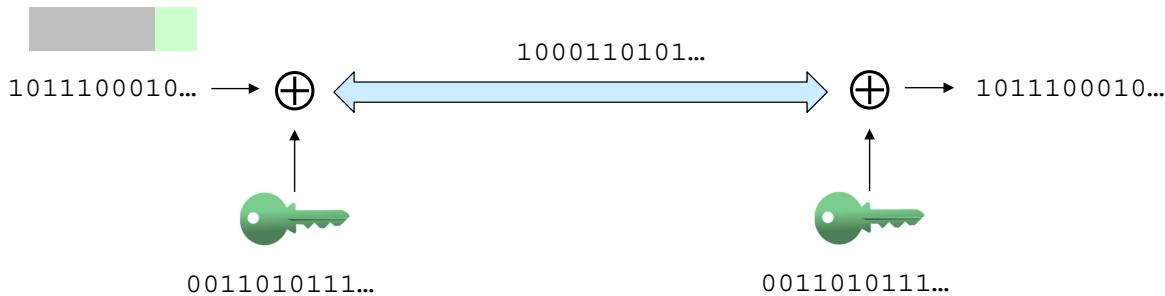
## One-Time Pad

- **Shannon:**

► „*A cipher achieves perfect secrecy if and only if there are as many possible keys as possible plaintexts, and every key is equally likely*“

→ Generiere einen **Schlüsselstrom, der so lang wie der Klartext ist**

- Bitweise XOR-Verknüpfung von Klartext und Schlüssel



Die in den vorherigen Beispielen skizzierten Verfahren haben den Nachteil, dass es sehr einfach ist, die Verschlüsselung zu brechen.

Wie könnte man denn eine perfekte Verschlüsselung erreichen? Dazu kann man ein sogenanntes „One-Time Pad“ verwenden: erzeuge einen zufälligen Strom von Bits, der als Schlüssel verwendet wird. Durch eine einfache XOR-Verknüpfung mit dem Klartext erreicht man eine zufällige Verwürflung, so dass ein Angreifer nicht auf Klartext oder Schlüssel zurückschließen kann.

Nur wer unseren Schlüsselstrom kennt, kann den Ciphertext wieder entschlüsseln.

Das Verfahren ist grundsätzlich sehr effizient zu implementieren, da Ver- und Entschlüsselung jeweils nur eine XOR-Operation sind. Ist der Schlüsselstrom tatsächlich rein zufällig gewählt, hat ein Angreifer keinen Ansatzpunkt – alle möglichen Klartexte sind gleich wahrscheinlich.

Aber in der Praxis stößt man doch auf große Probleme bei der Umsetzung: wie teilt man dem Empfänger den Schlüssel mit? Dies käme der Übertragung des Klartextes selber gleich. Als Lösung kann man einen kürzeren Schlüssel vereinbaren und diesen als Seed für einen Zufallszahlengenerator verwenden, der dann einen zufälligen Bitstrom ausspuckt. Doch sollte jeder Seed nicht mehr als ein Mal verwendet werden, da basierend auf dem gleichen Seed auch immer der gleiche Schlüsselstrom generiert wird. Überträgt man zwei Klartexte mit dem gleichen Schlüsselstrom, kann ein Angreifer durch XOR beider verschlüsselter Nachrichten den XOR der beiden Klartexte bekommen.

## Blockchiffren

- Blockchiffren verschlüsseln *Blöcke der Länge n eines Klartextes gleichzeitig*
  - ▶  $n$  zu groß → komplexe Algorithmen, Verlust der Leistungsfähigkeit
  - ▶  $n$  zu klein → zu schwache Verschlüsselung, unsicheres System
  - ▶ Gewählt wird  $n = 128 - 256$  Bit
- Jeder Block wird mit gleichem Algorithmus verschlüsselt
  - ▶ Variation des Ergebnisses durch *geheimen Schlüssel k innerhalb des Algorithmus* (secret key cryptography, symmetric cryptography)
  - ▶ Verwendete Schlüssellängen: 128 – 256 Bit
    - $k$  zu kurz → systematisches Testen aller Schlüssel (*Brute Force Attack*)

In der Praxis werden meist Blockchiffren eingesetzt, keine Stromchiffren.

Blockchiffren z.B.

- Data Encryption Standard (DES), 3DES
- International Data Encryption Algorithm (IDEA)
- Advanced Encryption Standard (AES)

DES, 3DES, IDEA und andere alte Verfahren sind heutzutage nicht mehr sicher. AES ist Standard für sichere Verschlüsselung.

# Advanced Encryption Standard (AES)

## • Advanced Encryption Standard

- ▶ 1997 vom National Institute of Standards and Technology (NIST) ausgeschrieben
- ▶ Anforderungen an AES
  - Blockchiffre mit 128 Bit Blocklänge
  - Schlüssellängen von 128, 192 und 256 Bit
  - Frei verfügbar, einfaches Design zur effizienten Implementierung
- ▶ Design von AES: *Rundenstruktur* zur mehrfachen Verschlüsselung
  1. Erstellung von Rundenschlüsseln
  2. Variable Anzahl  $N$  an Runden (abhängig von der Schlüssellänge)
    - *Permutation* der Eingangsinformationen und *Polynommultiplikation*
    - Verwendung von S-Boxen zur *Substitution*
    - *XOR-Operation mit dem Rundenschlüssel*
  3. Schlussrunde (ohne Polynommultiplikation)

Im Laufe der Zeit wurden mehrere Blockchiffren standardisiert: Data Encryption Standard (DES), 3DES, IDEA, ... - doch all diese Chiffren gelten mittlerweile als unsicher. Ein wesentlicher Nachteil von DES war eine Schlüssellänge von nur 56 Bit; bereits im Januar 1999 wurde ein DES-Schlüssel durch eine Brute-Force-Attacke innerhalb von 22 Stunden gebrochen. Heutzutage ginge dies noch einmal um Größenordnungen schneller.

3DES als kann zwar als Blockchiffre mit größerer Schlüssellänge eingesetzt werden (112 Bit), aber es war auf Dauer sinnvoll, direkt einen Algorithmus zu spezifizieren, der mit längeren Schlüsseln umgehen kann und daher sowohl sicherer als auch effizienter arbeiten kann als 3DES. Dieser Nachfolger wurde Advanced Encryption Standard (AES) genannt.

AES nutzt im Wesentlichen die Grundmechanismen der einfachen Verschlüsselungsverfahren, die wir oben gesehen haben – verbessert allerdings die Sicherheit durch zwei Faktoren:

- Komplexe Verwürflungsmechanismen durch Kombination von Permutationen, Substitutionen und Polynommultiplikation
- Mehrfache Verschlüsselung mit unterschiedlichen Schlüsseln

Entwurfskriterien von AES:

- Schutz gegen bekannte Attacken: Eliminierung von Symmetrien im Verhalten des Algorithmus durch Verwendung unterschiedlicher Konstanten in einzelnen Runden
- Vorteilhaftes Design des Algorithmus: effiziente Implementierung, einsetzbar auch z.B. auf Smart Cards - Transformationen einer Runde können parallel angewendet werden
- Erweiterbar: als Schlüssellänge ist jedes Vielfache von 32 Bit möglich, es erhöht sich nur die Rundenzahl
- Einschränkung: die Entschlüsselung benötigt mehr Runden und andere S-Boxen als die Verschlüsselung (und ist langsamer)

## AES – Rundenschlüssel

- Anzahl  $N$  der Runden basiert auf Schlüssellänge
  - ▶ 128 Bit-Schlüssel: 10 Runden
  - ▶ 192 Bit-Schlüssel: 12 Runden
  - ▶ 256 Bit-Schlüssel: 14 Runden
- Verwendung eines *eigenen Schlüssels für jede Runde*
  - ▶ Basierend auf einem einzigen geheimen Schlüssel
  - ▶ Schlüsselerstellung siehe Abschnitt 5.2 in  
<https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>
  - ▶ Rundenschlüssel haben gleiche Länge wie Blöcke (128 Bit)
  - ▶ Schlüsseldarstellung als Matrix:  
 $S = s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15}$

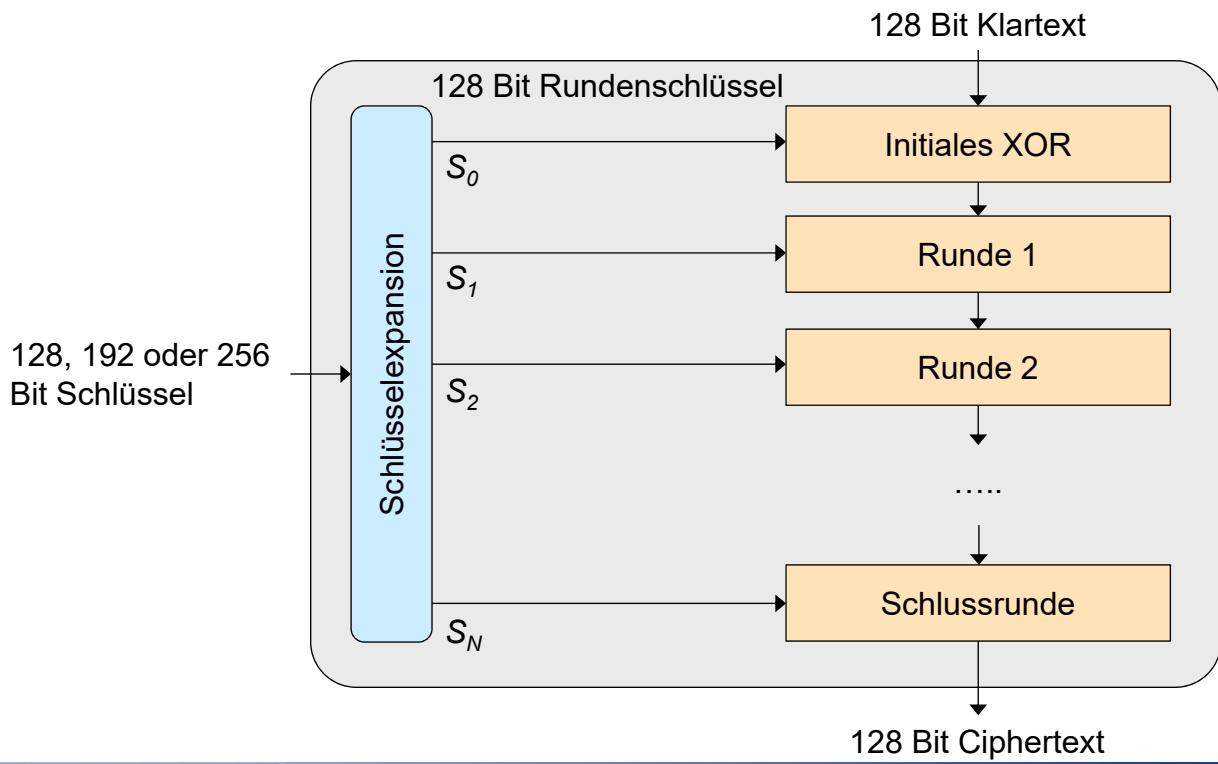
$s_0$	$s_4$	$s_8$	$s_{12}$
$s_1$	$s_5$	$s_9$	$s_{13}$
$s_2$	$s_6$	$s_{10}$	$s_{14}$
$s_3$	$s_7$	$s_{11}$	$s_{15}$

Einer der Faktoren, um eine gute Verschlüsselung zu erreichen, ist die mehrfache Verschlüsselung eines Klartextes.

Die Kommunikationspartner vereinbaren zu Beginn einen Schlüssel von 128, 192 oder 256 Bit Länge. Je nach Schlüssellänge generiert AES aus diesem Schlüssel 10, 12 oder 14 unterschiedliche Rundenschlüssel.

Jeder Rundenschlüssel besteht aus 128 Bit, d.h. 16 Byte. Diese 16 Bytes werden in einer 4x4-Matrix angeordnet.

## AES – Verschlüsselungsrunden

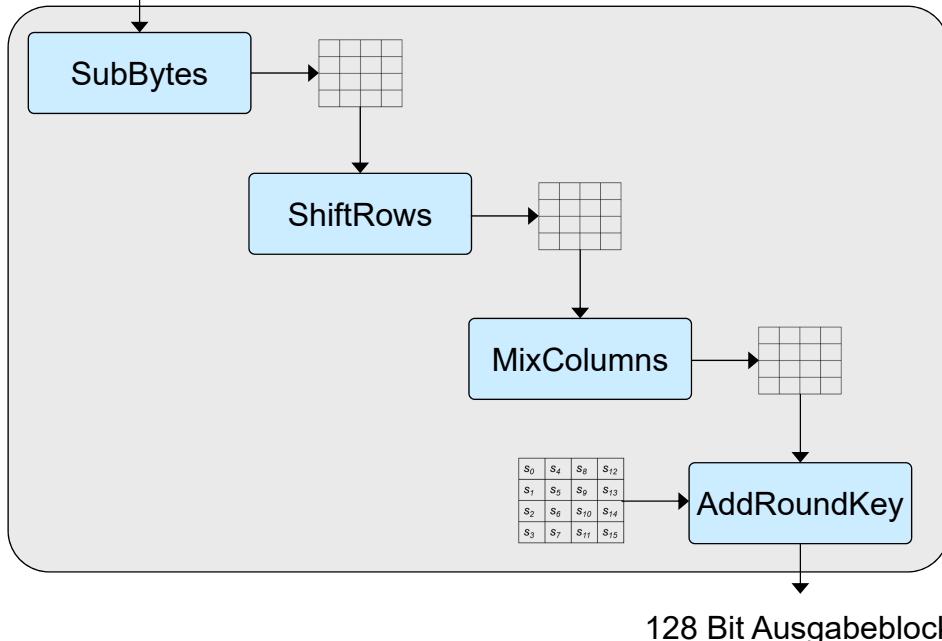


Bevor die erste Verschlüsselungsrunde beginnt, wird der Eingabeblock per XOR mit einem zusätzlichen Schlüssel verknüpft.

## Ablauf einer Verschlüsselungsrounde

$b_0$	$b_4$	$b_8$	$b_{12}$
$b_1$	$b_5$	$b_9$	$b_{13}$
$b_2$	$b_6$	$b_{10}$	$b_{14}$
$b_3$	$b_7$	$b_{11}$	$b_{15}$

128 Bit Eingabeblock B =  $b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 b_{10} b_{11} b_{12} b_{13} b_{14} b_{15}$

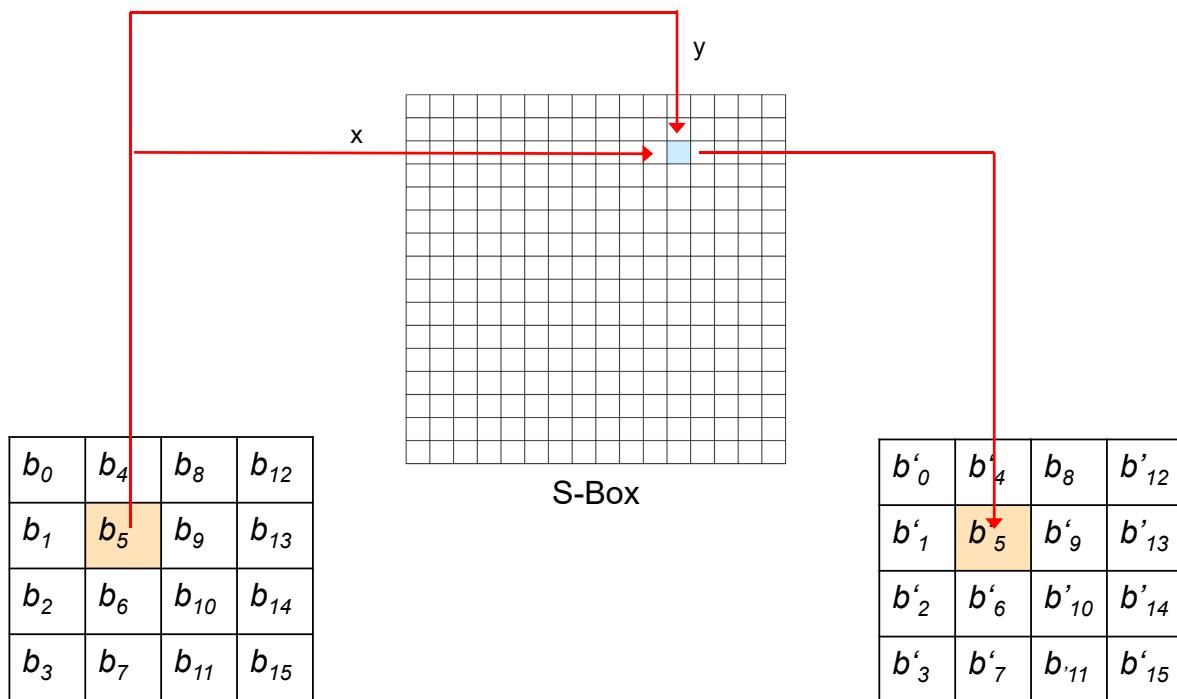


$s_0$	$s_4$	$s_8$	$s_{12}$
$s_1$	$s_5$	$s_9$	$s_{13}$
$s_2$	$s_6$	$s_{10}$	$s_{14}$
$s_3$	$s_7$	$s_{11}$	$s_{15}$

128 Bit Ausgabeblock

$b'_0$	$b'_4$	$b'_8$	$b'_{12}$
$b'_1$	$b'_5$	$b'_9$	$b'_{13}$
$b'_2$	$b'_6$	$b'_{10}$	$b'_{14}$
$b'_3$	$b'_7$	$b'_{11}$	$b'_{15}$

## SubBytes



Zur Substitution wird eine Substitutionsbox (S-Box) verwendet, die eine Permutation der Werte 0 – 255 enthält. Die 8 Bit eines Elements des Eingabeblocks werden als zwei 4-Bit-Werte x und y aufgefasst, die als Indizes verwendet werden, um ein Element der S-Box auszuwählen. Der dort eingetragene Wert wird an gleicher Position in den Ausgabeblock geschrieben.

## ShiftRows

$b_0$	$b_4$	$b_8$	$b_{12}$
$b_1$	$b_5$	$b_9$	$b_{13}$
$b_2$	$b_6$	$b_{10}$	$b_{14}$
$b_3$	$b_7$	$b_{11}$	$b_{15}$

Keine Änderung →

Shift 1 →	$b_0$	$b_4$	$b_8$	$b_{12}$
Shift 2 →	$b_5$	$b_9$	$b_{13}$	$b_1$
Shift 3 →	$b_{10}$	$b_{14}$	$b_2$	$b_6$
	$b_{15}$	$b_3$	$b_7$	$b_{11}$

## MixColumns

- Entspricht Matrixmultiplikation in  $GF(2^8)$  mit Polynom

$$x^8 + x^4 + x^3 + x + 1$$

- Separate Behandlung jeder Spalte

- Jedes Byte wird mit einem Wert ersetzt, der von allen Bytes der Spalte abhängt

2	3	1	1
1	2	3	1
1	1	3	3
3	1	1	2

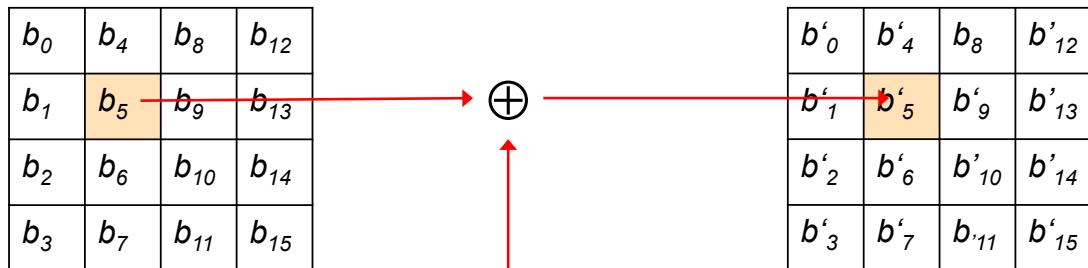
.

$b_0$	$b_4$	$b_8$	$b_{12}$
$b_1$	$b_5$	$b_9$	$b_{13}$
$b_2$	$b_6$	$b_{10}$	$b_{14}$
$b_3$	$b_7$	$b_{11}$	$b_{15}$

=

$b'_0$	$b'_4$	$b_8$	$b'_{12}$
$b'_1$	$b'_5$	$b'_9$	$b'_{13}$
$b'_2$	$b'_6$	$b'_{10}$	$b'_{14}$
$b'_3$	$b'_7$	$b'_{11}$	$b'_{15}$

## AddRoundKey



The diagram shows the selection of s-box values for the addition operation. Below the round key matrix, a 4x4 s-box matrix  $s$  is shown:

$s_0$	$s_4$	$s_8$	$s_{12}$
$s_1$	$s_5$	$s_9$	$s_{13}$
$s_2$	$s_6$	$s_{10}$	$s_{14}$
$s_3$	$s_7$	$s_{11}$	$s_{15}$

A red arrow points from the element  $s_5$  in the s-box matrix to the element  $b'_5$  in the round key matrix, also highlighted in orange.

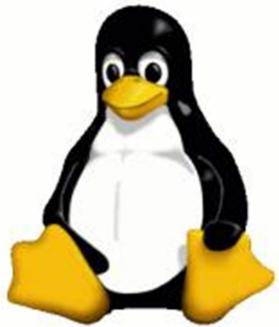
## Verschlüsselung „langer“ Nachrichten

- Anwendungen übertragen meist deutlich mehr Daten als 128 Bit
  - ▶ Blockchiffren kodieren Blöcke fester Länge (AES: 128 Bit)
  - ▶ Nachrichten müssen als mehrere aufeinanderfolgende Blöcke verschlüsselt werden
- Probleme bei der Verschlüsselung
  - ▶ Identische Blöcke werden gleich verschlüsselt
  - ▶ Angreifer kann Informationen über Inhalte erlangen, indem er nach identischen Blöcken sucht
  - ▶ Angreifer kann eine verschlüsselte Nachricht modifizieren, indem er Blöcke permutiert, löscht oder einfügt
- Lösung z.B. *Cipher Block Chaining (CBC)*

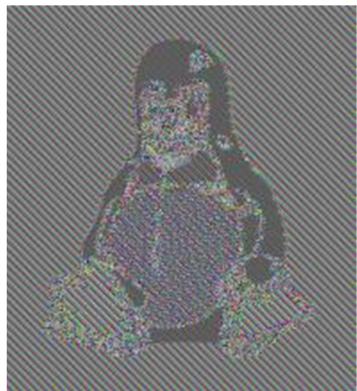
Blockchiffren sind auf eine sehr beschränkt Blocklänge ausgelegt. Daher müssen Datenströme vor der Verschlüsselung mit AES in Blöcke á 128 Bit zerlegt werden. Dies eröffnet ein paar Schwachstellen: ein Angreifer kann durch Abhören einer Übertragung Informationen sammeln, auch wenn er die Inhalte selber gar nicht verstehen kann. Durch längeres Mithören von verschlüsselten Datenströmen kann ein Angreifer versuchen, bestimmte Muster zu erkennen und zu erraten, welche verschlüsselten Blöcke welche Informationen tragen. Dies könnte er nutzen, um Blöcke mitzuschneiden und später einzuspielen, um einen Kommunikationsabtausch zu modifizieren. Es muss daher in der Praxis vermieden werden, dass gleiche Blöcke immer gleich verschlüsselt werden. Zum einen sollte man dazu häufig den Schlüssel wechseln, zum anderen kann man aber auch bereits innerhalb einer einzigen Kommunikationssitzung Zufallszahlen verwenden, um gleiche Eingabeblocks auf unterschiedliche Ausgaben abzubilden.

## Beispiel

---



unverschlüsselt



blockweise verschlüsselt

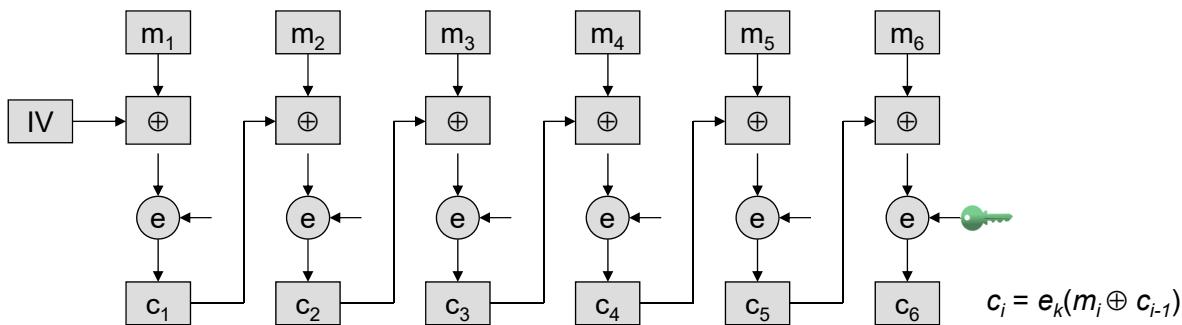


CBC-verschlüsselt

# Cipher Block Chaining (CBC)

- **CBC-Verfahren:**

- ▶ Jeder Block wird vor der Verschlüsselung mit dem Ciphertext des vorherigen Blocks XOR-verknüpft
- ▶ Für den ersten Block muss eine Zufallszahl (*Initialisation Vector, IV*) verwendet werden



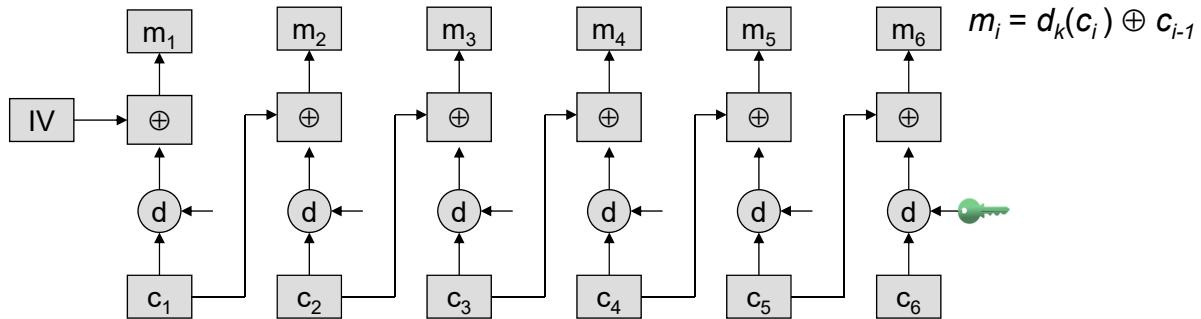
Verfahren wie CBC haben den Zweck, gleiche Klartextblöcke auf unterschiedliche Ciphertextblöcke abzubilden, ohne einen Zusatzaufwand bei Ver- oder Entschlüsselung zu produzieren. Die generelle Idee ist einfach, jeden Block vor der Übertragung mit einer Zufallszahl zu verknüpfen. Wie und was genau verknüpft wird, hängt vom jeweiligen Verfahren ab – bei CBC wird jeweils der verschlüsselte vorherige Block als Zufallszahl für den nächsten Block verwendet. Lediglich beim ersten Block ist es notwendig, für die XOR-Verknüpfung eine Zufallszahl (Initialization Vector, IV) zu verwenden. Diese muss dem Kommunikationspartner (ebenso wie der verwendete Schlüssel) übermittelt werden.

Für jede zu verschlüsselnde Nachricht sollte ein neuer IV verwendet werden, andernfalls:

- Ein Angreifer kann bei ähnlichen Nachrichten feststellen, ab welcher Position sie sich unterscheiden
- Alte Nachrichten können von einem Angreifer wiederholt und als neue Nachrichten gesendet werden
- Durch Sammlung von genügend Informationen können im Laufe der Zeit eventuell IV und Schlüssel ermittelt werden

# Cipher Block Chaining

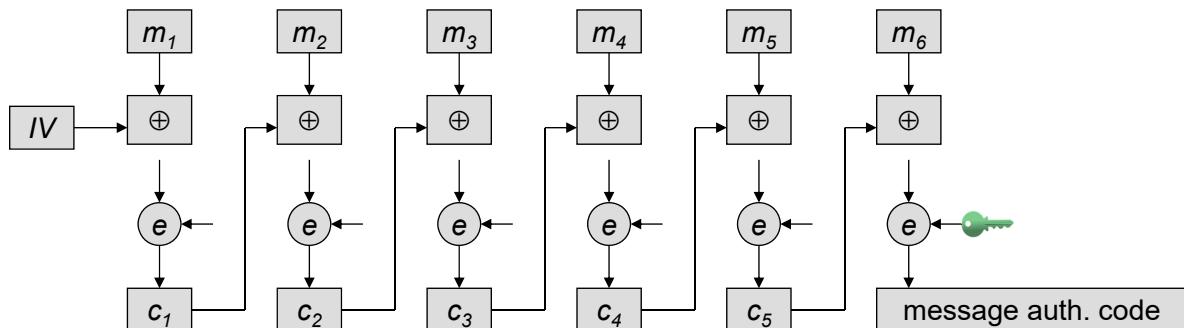
- Entschlüsselung:



# Message Authentication Code (MAC)

- **Integritätsprüfung** mittels AES und CBC:

- ▶ Verwende nur den letzten Block Ciphertext
  - **Message Authentication Code (MAC)**
- ▶ Übertrage die Nachricht zusammen mit dem MAC
- ▶ Der Empfänger verschlüsselt die empfangene Nachricht auf die gleiche Art, um die Korrektheit des MAC zu überprüfen



Das Verfahren kann auch zur Authentizitäts- oder Integritätsprüfung verwendet werden, d.h. um Modifikationen an einer Nachricht während der Übertragung erkennen zu können. Man verschlüsselt die Nachricht mittels eines symmetrischen Verfahrens im CBC-Modus, verwirft allerdings alle bis auf den letzten der Cipherblöcke. Die Nachricht kann nun in Klartext übertragen werden, der letzte Cipherblock dient als Authentizitäts- und Integritätscode und wird mit übertragen.

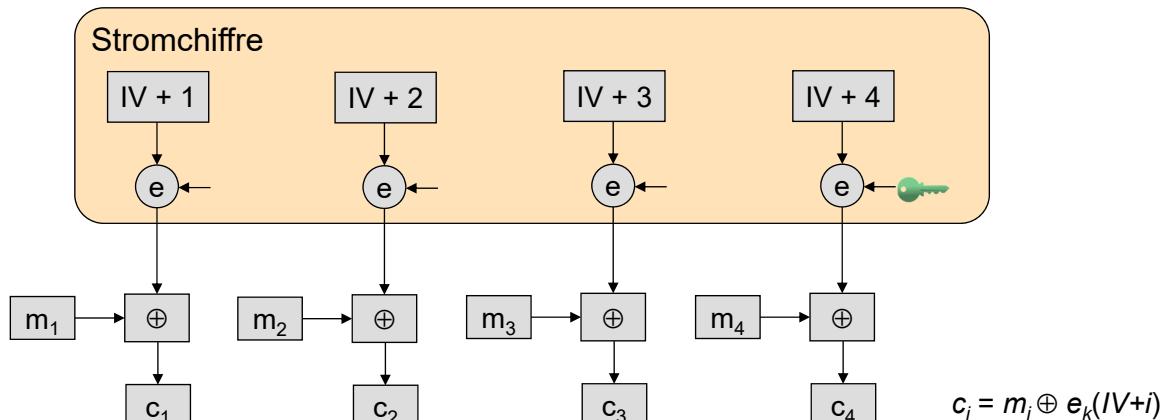
Dieser letzte Block wird bei diesem Vorgehen als Message Authentication Code (älter auch Message Integrity Code, MIC) bezeichnet.

Der Empfänger führt die gleiche Verschlüsselung der empfangenen Nachricht durch und muss auf den gleichen MAC kommen. Errechnet er einen anderen Codeblock, wurde die Nachricht während der Übermittlung modifiziert.

Solche Verfahren können angewendet werden, wenn zwar keine Verschlüsselung der Informationen notwendig ist, aber Modifikationen erkannt werden sollen. Der MAC ist also eine Alternative zur digitalen Signatur.

## Counter Mode (CTR)

- Verwendung einer Blockchiffre als **Stromchiffre**:



- ▶ Einsatz z.B. als **CCM** (Counter with CBC-MAC), RFC 3610
  - CTR zur Verschlüsselung, CBC zur Integritätssicherung
- ▶ Oder als **GCM** (Galois/Counter Mode)
  - CTR-Verschlüsselung, Multiplikation im Galoiskörper zur Authentifizierung

## Diffie-Hellman-Kryptosystem

- Nötig für DES, AES, ....: *Schlüsselaustausch*

- ▶ Datenübertragung ist unsicher
- ▶ Verschlüsselung erst nach einem Schlüsselaustausch anwendbar
- ▶ Wie überträgt man den Schlüssel sicher?
- Algorithmus von *Diffie-Hellman* zum Schlüsselaustausch

Ein Problem bei Einsatz von DES/AES im CBC-Modus ist die Vereinbarung eines geheimen Schlüssels und der Austausch des Initialisierungsvektors.

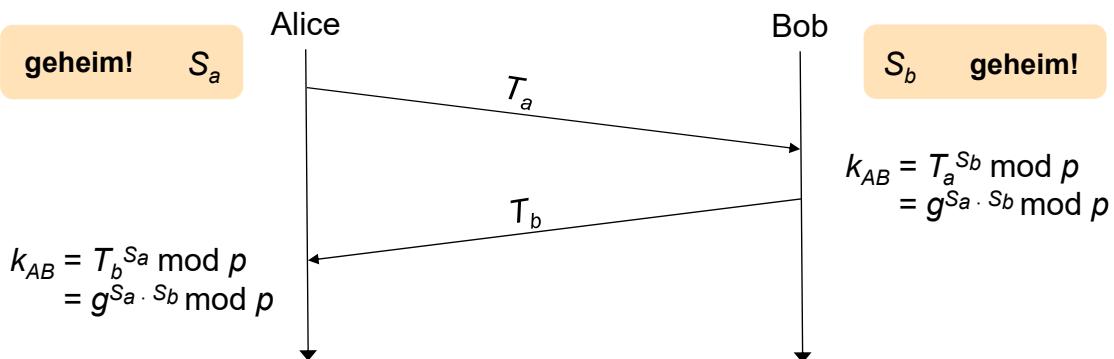
Eine Möglichkeit, einen Schlüssel (oder Initialisierungsvektor) geheim über ein öffentliches Netz auszutauschen, bietet der Diffie-Hellman-Algorithmus.

# Algorithmus von Diffie-Hellman

- **Parameter bei Diffie-Hellman:**

- ▶ Wähle eine Primzahl  $p$  ( $\geq 2000$  Bit) und einen Generator  $g < p$ 
  - $p$  und  $g$  sind öffentlich!
- ▶ Alice wählt zufällig ein  $S_a$  und berechnet  $T_a = g^{S_a} \text{ mod } p$
- ▶ Bob wählt zufällig ein  $S_b$  und berechnet  $T_b = g^{S_b} \text{ mod } p$

- **Nachrichtenaustausch und Schlüsselgenerierung:**



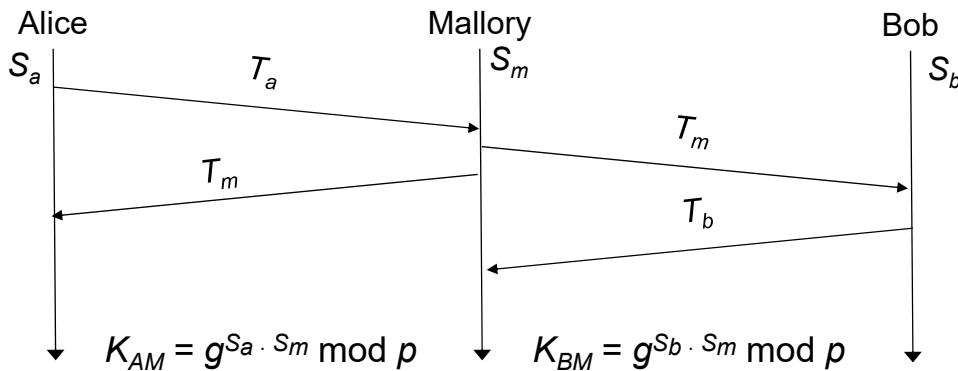
Die Idee ist simpel: zwei Kommunikationspartner A(lice) und B(ob) wählen jeweils einen geheimen Wert und nach einer bestimmten Berechnungsvorschrift ermitteln sie einen zugehörigen öffentlichen Wert. Wichtig ist diese Berechnungsvorschrift: es soll unmöglich sein, außer durch Brute-Force aus dem öffentlichen den geheimen Wert zu ermitteln. (Problem der Berechnung diskreter Logarithmen.)

Es ist unmöglich,  $g^{S_a \cdot S_b}$  effizient zu berechnen, wenn man nicht einen der Werte  $S_a$  und  $S_b$  kennt.

Für einen längerfristigen Einsatz (über 2022 hinaus) wird eine Schlüssellänge (Größe von  $p$ ) von mindestens 3000 Bit empfohlen.

## Man-in-the-Middle-Angriff auf Diffie-Hellman

- Keine Authentifizierung zwischen Alice und Bob
  - ▶ Alice erhält  $T_x$ , kann aber nicht sicher sein, dass es von Bob ist
- **Man-in-the-Middle-Attacke**
  - ▶ Ein Angreifer Mallory fängt die Nachrichten ab und erstellt je einen Schlüssel mit Alice und Bob



- Diffie-Hellman schützt nur gegen passive Angriffe!

Diffie-Hellman schützt zwar gegen passive Angriffe, d.h. die Ermittlung des Schlüssels durch Mitlesen der ausgetauschten Nachrichten, aber nicht gegen aktive Angriffe. Hat ein Angreifer die Möglichkeit, Nachrichten zu modifizieren, braucht man weitere Schutzmechanismen.

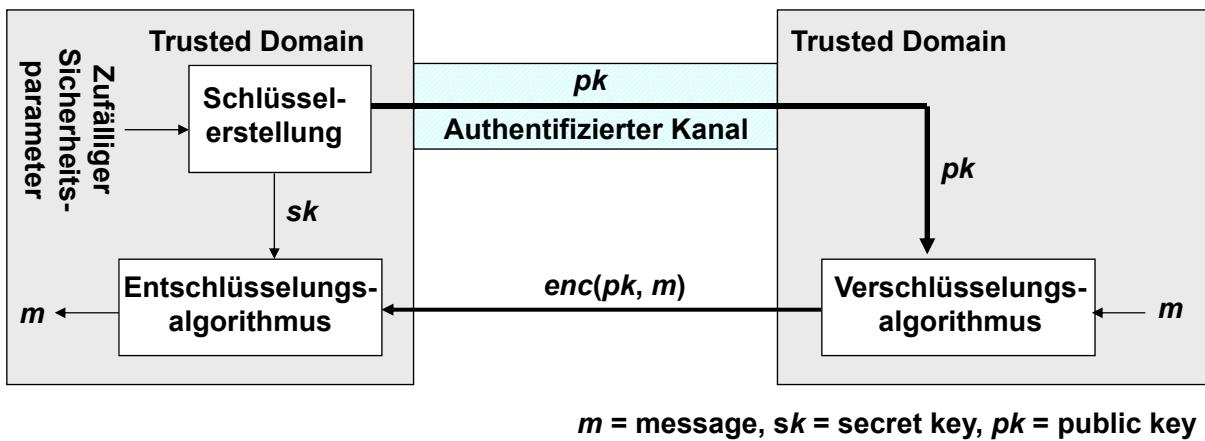
Ein Schutz hätte hier durch Authentifizierung oder Integritätsprüfung stattfinden können. Aber – auch für diese Mechanismen werden Schlüssel benötigt, siehe CBC-MAC.

Alternativ dazu könnte man die Werte  $g^{S_i}$  im Voraus an vertrauenswürdiger Stelle veröffentlichen. Dies hätte allerdings den Nachteil, dass man vor Erstellung eines neuen Schlüssels den Wert neu vertrauenswürdig veröffentlichen müsste. Da Schlüssel nie lange verwendet werden sollten, um es einem Angreifer nicht zu ermöglichen, zu viele Informationen über den Schlüssel zu sammeln, ist dies nicht praktikabel.

# Asymmetrische Verschlüsselung

- Zwei sich ergänzende Schlüssel: geheimer Schlüssel (*private key*) und öffentlicher Schlüssel (*public key*)

- ▶ Wesentlich langsamer als symmetrische Verfahren
- ▶ Z.B. zum Schlüsselaustausch für ein symmetrisches Verfahren
- ▶ Schlüssellängen ab 2048 Bit



Als Alternative zu symmetrischen Verfahren können asymmetrische Verfahren verwendet werden. Hierbei existieren für jeden Kommunikationspartner zwei Schlüssel, einer zur Verschlüsselung und ein inverser zur Entschlüsselung. Den Schlüssel für die Entschlüsselung kennt nur der Ersteller des Schlüsselpaars, der Schlüssel für die Verschlüsselung wird öffentlich bekannt gemacht. Sollen nun Daten verschlüsselt übertragen werden, kann ein Kommunikationspartner den öffentlichen Schlüssel verwenden – entschlüsseln kann dann nur der Ersteller des Schlüsselpaars mit dem geheimen Schlüssel.

Während im Vergleich zu symmetrischen Verfahren der Schlüsselaustausch deutlich vereinfacht wird, sind die Algorithmen um mehrere Größenordnungen langsamer als symmetrische Verfahren. Sie eignen sich daher nicht für den Einsatz zur Verschlüsselung regulären Datenverkehrs, sondern nur für sehr kurze Nachrichten – beispielsweise für den Schlüsselaustausch für ein symmetrisches Verfahren.

Problematisch ist hier – wie bei Diffie-Hellman – die Übergabe des öffentlichen Schlüssels an einen Kommunikationspartner. Da keine Authentifizierung existiert, kann man nicht hundertprozentig sicher sein, dass nicht ein Man-in-the-Middle-Angriff stattgefunden hat und der öffentliche Schlüssel tatsächlich dem geplanten Empfänger gehört.

## Beispiel für asymmetrische Kryptosysteme: RSA

- Entwickelt durch Rivest, Shamir und Adleman mit folgenden Eigenschaften:

- ▶ Variable Schlüssellänge (mindestens 2048 Bit)
- ▶ Variable Klartextlänge (aber kürzer als der Schlüssel)
- ▶ Ciphertext-Blocks haben Länge des Schlüssels
- ▶ Verwendungszweck:
  - Übertragung von Schlüsseln für symmetrische Verfahren
  - Digitale Signaturen
- ▶ Nutzt Eulers und Fermats Theoreme
  - Sicherheit basiert auf Schwierigkeit von Faktorisierung / diskretem Logarithmus

# Grundlagen RSA

- **Sicherheit asymmetrischer Verfahren**
  - ▶ basiert oft auf der Schwierigkeit von Faktorisierung (zerlege eine gegebene Zahl in ihre Primfaktoren)
  - ▶ und/oder diskretem Logarithmus (Finde ein  $x$  mit  $a^x \equiv b \pmod{n}$  zu gegebenen  $a$  und  $b$ ).
  - ▶ Diese Probleme sind für große Zahlen sehr zeitaufwändig zu berechnen.
- **Eulers Theorem**
  - ▶ Die Euler-Funktion  $\varphi(n)$  bezeichnet die Anzahl ganzer Zahlen, die relativ prim (teilerfremd) zu  $n$  sind
  - ▶ Ist  $n$  prim:  $\varphi(n) = n - 1$
  - ▶ Für jedes  $a$  relativ prim zu  $n$  gilt:  $a^{\varphi(n)} \equiv 1 \pmod{n}$
  - ▶ Ist  $n$  Produkt zweier Primzahlen  $p$  und  $q \rightarrow \varphi(n) = (p - 1) \cdot (q - 1)$
  - ▶ Ist  $n$  prim oder Produkt zweier Primzahlen  $\rightarrow x^y \pmod{n} = x^{y \pmod{\varphi(n)}} \pmod{n}$
- **Fermats Theorem**
  - ▶ Ist  $n$  prim und  $0 < a < n \rightarrow a^{n-1} \equiv 1 \pmod{n}$

## Schlüsselerstellung bei RSA

- Erstellt werden ein *privater* und ein zugehöriger *öffentlicher* Schlüssel

1. Wähle zwei große Primzahlen  $p$  und  $q$   
( $p$  und  $q$  müssen geheim sein!)
2. Berechne  $n = p \cdot q$
3. Berechne  $\Phi(n) = (p - 1) \cdot (q - 1)$
4. Wähle eine Zahl  $e$  relativ prim zu  $\Phi(n)$
5. Finde ein  $d$  mit  $d \cdot e = 1 \pmod{\Phi(n)}$   
( $d$  ist das multiplikativ Inverse zu  $e$ )  
⇒  $\langle e, n \rangle$  ist *öffentlicher Schlüssel*  
⇒  $\langle d, n \rangle$  ist *privater Schlüssel*

*Prinzip:*  $x^{d^*e} = x^{1 \pmod{\Phi(n)}} = x \pmod{n}$ , wenn  $n$  Produkt zweier Primzahlen ist

Warum funktionieren diese Schlüssel?

- Wir nutzen modulare Arithmetik ( $\pmod{n}$ ) mit  $p \cdot q = n$
- $d$  und  $e$  werden gewählt als  $d \cdot e = 1 \pmod{\Phi(n)}$
- Weil  $n$  Produkt zweier Primzahlen ist, gilt für alle  $x$ :  $x^{d \cdot e} = x^{1 \pmod{\Phi(n)}} = x \pmod{n}$

RSA ist sicher – es zu brechen bedeutet,  $d$  aus bekanntem  $e$  und  $n$  zu berechnen.

- Ein Angreifer weiß nur:  $d$  ist exponentiell Inverses zu  $e \pmod{\Phi(n)}$
- Einfach: mittels  $p$  und  $q$  kann  $\Phi(n)$  berechnet werden
- Faktorisierung von  $n$  in  $p$  und  $q$  ist nötig, Faktorisierung großer Zahlen ist zeitaufwändig, die besten Algorithmen sind zu langsam, wenn die Schlüssellänge  $n$  ausreichend groß gewählt wird. 1024 Bit lange Schlüssel werden mittlerweile als zu kurz angesehen, 2048 Bit werden noch als sicher angesehen. Trotzdem werden Schlüssellängen von 3072+ (langfristig 15360 Bit) empfohlen.

Wichtig:  $e$  kann klein gewählt werden, ohne dass das Verfahren unsicherer wird;  $d$  sollte einen großen Wert haben, damit es nicht einfach ermittelt werden kann.

## Nutzung von RSA

- **Verschlüsselung und Entschlüsselung**

- ▶ Nachrichten  $m$  können mit dem öffentlichen Schlüssel  $e$  von jedem verschlüsselt werden:

$$c = m^e \text{ mod } n$$

- ▶ Entschlüsselung des Ciphertextes  $c$  ist nur mit dem (geheimen) privaten Schlüssel  $d$  möglich:

$$m = c^d \text{ mod } n$$

- **Elektronische Unterschriften (*Digitale Signatur*)**

- ▶ Der Ersteller verschlüsselt die Nachricht  $m$  mit seinem privaten Schlüssel:

$$s = m^d \text{ mod } n$$

- ▶ Jeder, der den öffentlichen Schlüssel hat, kann die Nachricht dekodieren:

$$m = s^e \text{ mod } n$$

Asymmetrische Verfahren sind beim Einsatz zur Verschlüsselung ineffizient – aber es gibt einen weiteren Anwendungszweck: digitale Signaturen. Verschlüsselt ein Kommunikationspartner Daten mit seinem privaten Schlüssel, kann jeder sie entschlüsseln. Da nur der Ersteller des Schlüsselpaares den privaten Schlüssel kennt, dient die Verschlüsselung als seine Signatur. Lässt sich eine Nachricht mit dem öffentlichen Schlüssel des angeblichen Absenders entschlüsseln, kann man sicher sein, dass die Nachricht korrekt ist; man kann Integrität und Authentizität überprüfen.

## RSA: Beispiel

- Sei  $p = 5, q = 11$

- ▶  $n = p \cdot q = 55$
- ▶  $\varPhi(n) = (p - 1) \cdot (q - 1) = 4 \cdot 10 = 40$

- Schlüssel:

- ▶ Wähle  $e = 7$
- ▶ Wähle  $d$  mit  $d \cdot e = 1 \text{ mod } 40$ ,  $d = 23$

$e$  muss relativ prim sein  
zu  $\varPhi(n)$ , d.h. zu 40

- Verschlüsselung:

- ▶ Verschlüssle mit  $c = m^e \text{ mod } n$
- ▶ Sei  $m = 13$ 
  - $c = m^e = 13^7 = 7 \text{ (mod } 55)$

- Entschlüsselung:

- ▶  $m = c^d = 7^{23} \text{ mod } 55 = 13$

## RSA – praktischer Einsatz

- RSA und *Padding* (RSA-OAEP, Optimal Asymmetric Encryption Padding)
  - ▶ Viele effektive Angriffe gegen RSA, hier nicht behandelt
  - ▶ Daher sollte in der Praxis Randomisierung verwendet werden
    - Zufälliges Padding von  $m$  vor der Verschlüsselung
- Generelle Empfehlung
  - ▶ Kryptoverfahren niemals selbst implementieren! Nutzt Libraries!
    - Zu viele eventuell unbekannte Probleme und unbewusste Lücken in der Implementierung können zu einer unsicheren Implementierung führen

# Kryptographische Hash-Funktion

- **Problem bei digitalen Signaturen**
  - ▶ Es dauert lange, eine ganze Nachricht zu signieren
- **Idee: signiere nur einen Hash-Wert (Message Digest)**
  - ▶ Verwendung einer *kryptographischen Hash-Funktion h*:
    - Bilde Nachricht  $m$  beliebiger Länge auf Hashwert  $h(m)$  fester Länge ab:  
 $h: \{0, 1\}^* \rightarrow \{0, 1\}^k$
  - ▶ Notwendige Eigenschaften von  $h$ :
    - **Preimage resistance**: es ist schwer, aus gegebenem Hashwert  $H$  eine Nachricht  $m$  mit  $h(m) = H$  zu finden
    - **Second preimage resistance**: für eine gegebene Nachricht  $m$  ist es schwer, ein  $m' \neq m$  mit  $h(m) = h(m')$  zu finden
    - **Collision resistance**: es ist schwer, zwei Nachrichten  $m$  und  $m'$  mit  $h(m) = h(m')$  zu finden
  - ▶ Standardisiert sind mehrere Hash-Funktionen:  
~~MD5, SHS, SHA-1, SHA-2, SHA-3, ...~~

Nachteil von Algorithmen wie RSA: asymmetrische Verfahren sind langsam, die Erstellung und Überprüfung digitaler Signaturen dauern lange. Abhilfe schafft die Verwendung eines Message Digest – dies ist im Prinzip ein Hashwert über die Nachricht. Anstatt die gesamte Nachricht zu signieren, wird nur der Hashwert signiert und als Signatur zusammen mit der Nachricht verschickt.

Allerdings kann keine normale Hashfunktion zur Erstellung eines Message Digest verwendet werden: es darf nicht einfach möglich sein, dass ein Angreifer eine zweite Nachricht findet, die auf den gleichen Hashwert abgebildet wird, damit es nicht möglich ist, Nachrichten zu ersetzen, die trotzdem eine gültige Signatur haben. Kryptographische Hashfunktionen verwürfeln daher (ähnlich wie symmetrische Verschlüsselungsverfahren) die Eingabeinformation stark.

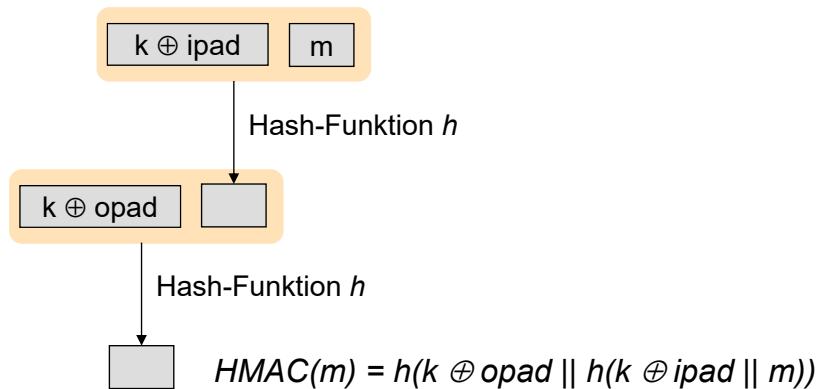
Nach und nach sind für mehrere kryptographische Hash-Funktionen Kollisionen gefunden worden: zwei Eingaben, die auf den gleichen Wert hashen. Ist dies der Fall, wird die Funktion als unsicher klassifiziert und sollte nicht mehr verwendet werden. Dies ist zumindest für die Funktionen bis SHA-1 der Fall.

## Alternativ: HMAC

- Alternativ: berechne Hash über Nachricht und Schlüssel

- *Hash Message Authentication Code (HMAC)*

- Verwendung einer kryptographischen Hash-Funktion  $h$
    - Padding des Schlüssels durch zwei zufällige Werte



Alternativ zur Verwendung von RSA (oder anderer asymmetrischer Verfahren) kann ein symmetrischer Schlüssel direkt als Authentifizierungswert mit in den Input der Hash-Funktion eingehen, um Signaturen zu berechnen.

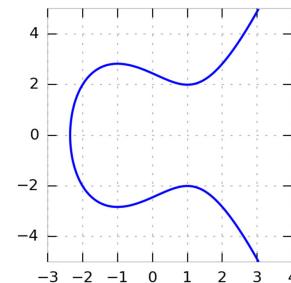
Da es erwiesen ist, dass die alleinige Konkatenation von Schlüssel und Nachricht zu unsicheren MACs führen kann, verwendet man auch hier Padding.

- ***Public Key Cryptography Standard (PKCS)***
  - ▶ Menge an Standards: PKCS#1 - PKCS#9
  - ▶ Festlegung der Implementierung von RSA
    - Wie werden RSA-Nachrichten übertragen?
    - Wie teilt man dem Empfänger mit, dass Padding verwendet wurde?
    - ...
- **Alternative zu RSA:**
  - ▶ *Elliptic Curve Digital Signature Algorithm (ECDSA)*
    - Basiert auf elliptischen Kurven

# Elliptic Curve Cryptography (ECC)

## • Moderner Bereich der Kryptographie

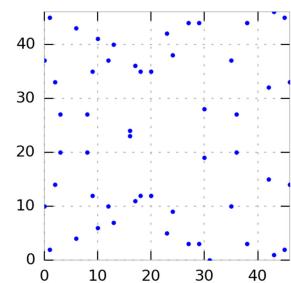
- ▶ Sicherheit basiert auf dem speziellen *Problem des diskreten Logarithmus* für elliptische Kurven
- ▶ Basis z.B. für Bitcoin-Transaktionen



Elliptische Kurve  $E: y^2 = x^3 - 3x + 6$  über  $\mathbb{R}$

## • Grundlagen der ECC

- ▶ Eine *elliptische Kurve* ist eine Gleichung der Form
$$E: y^2 = x^3 + ax + b$$
mit  $a, b$  Element eines Körpers (meist  $\mathbb{Z}_p$ )
- ▶ Ein *Punkt* ist eine Lösung  $P = (x, y)$  der Gleichung



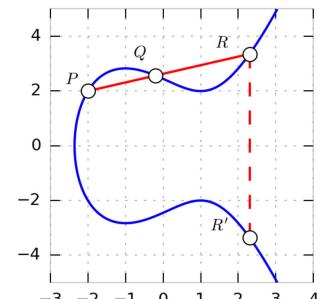
Elliptische Kurve  $E: y^2 = x^3 - 3x + 6$  über  $\mathbb{Z}_{47}$

# Punkt-Operationen und diskreter Logarithmus

## • Wichtige Operationen

### ► Punktaddition

- Geometrische Interpretation des Graphen
- Eine Linie durch die Punkte  $P$ ,  $Q$  schneidet die Kurve in genau einem weiteren Punkt  $R = (x_R, y_R)$
- Definiere  $P + Q := R' = (x_R, -y_R)$
- Funktioniert genauso in  $Z_p$



Beispiel: Punktaddition

### ► Punktmultiplikation

- Multipliziere einen Punkt  $P$  mit einer Konstante  $k$ :

$$k \cdot P := P + P + \dots + P \quad (k \text{ mal})$$

# Problem des diskreten Logarithmus bei elliptischen Kurven

- **Elliptische Kurven formen eine mathematische Gruppe**
  - ▶ Menge aller Punkte (mit „unendlich“ als neutralem Element) zusammen mit Punktaddition bildet eine zyklische Gruppe
    - Oder Vereinigung zweier zyklischer Gruppen, abhängig von Parametern
- **Ordnung der Kurve: Zahl der Punkte auf der Kurve**
  - ▶ Entspricht die Ordnung der Kurve einer Primzahl, ist jeder Punkt ein Generator  $G$  der Gruppe
    - Jeder Punkt  $P$  kann als  $P = kG$  dargestellt werden (für bestimmtes  $k$ )
- **Problem des diskreten Logarithmus:**
  - ▶ Für ein gegebenes  $P$  und den Generator  $G$  einer Kurve finde ein  $k$  mit  $P = kG$
  - ▶ Wird als schwereres Problem angesehen als das klassische Problem des diskreten Logarithmus

# Elliptic-Curve-Verfahren für Sicherheitsziele

- **Elliptic Curve Digital Signature Algorithm (ECDSA) für digitale Signaturen**
  - ▶ Algorithmus zur Signaturerstellung für Nachricht  $m$ 
    - Gegeben: EC der Ordnung  $p$  mit Generator  $G$ , privater Schlüssel  $d$
    - Berechne  $e = h(m)$  mit kryptographischer Hash-Funktion  $h(x)$
    - Wähle zufällig ein Integer  $k$ ,  $0 < k < p$
    - Berechne den Punkt  $(x, y) = kG$  und setze  $r = x$
    - Berechne  $s = k^{-1} \cdot (e + rd)$
    - $(r, s)$  ist Signatur
  - ▶ Algorithmus für Verifikation:
    - Gegeben: Signatur  $(r, s)$ , öffentlicher Schlüssel  $D$ ;  $p, G, m$  wie oben
    - Berechne  $e = h(m)$  mit kryptographischer Hash-Funktion  $h(x)$
    - Berechne  $u = rs^{-1} \bmod p$  und  $v = es^{-1} \bmod p$
    - Berechne Punkt  $(x, y) = uD + vG$
    - Signatur verifiziert, falls  $x = r$
- **Elliptic Curve Integrated Encryption Scheme (ECIES/IES) für Verschlüsselung**
- **Elliptic Curve Diffie-Hellman (ECDH) für Schlüsselgenerierung**

R und s müssen ungleich Null sein, andernfalls muss ein neues k gewählt werden.

Neben dem schweren Problem des diskreten Logarithmus hat ECDSA gegenüber RSA den Vorteil, dass Schlüssel und Signaturen kleiner sind als bei RSA. Dadurch sind effizientere Berechnungen möglich.

# Zusammenfassung Kryptographie

- **Symmetrische Kryptographie**
  - ▶ Öffentlicher, komplexer Algorithmus und geheimer Schlüssel
    - Standardverfahren: AES
    - Nötig: Verfahren zum Schlüsselaustausch, z.B. Diffie-Hellman
- **Asymmetrische Kryptographie**
  - ▶ Nutzung von Schlüsselpaaren
    - Standardverfahren: RSA, DSA (ElGamal), Elliptic Curves
    - Langsamer als symmetrische Verfahren
  - ▶ Nützlich zum Schlüsselaustausch und für digitale Signaturen
- **Message Digest**
  - ▶ Berechnung von Hash-Werten zur Beschleunigung digitaler Signaturen
  - ▶ Z.B. SHA-3
- **Authentifizierung?**

Allerdings besteht immer noch ein großes Problem: Authentifizierung. Bei einem Schlüsselaustausch für symmetrische Verfahren sowie bei der Weitergabe von öffentlichen Schlüsseln für asymmetrische Verfahren muss sichergestellt werden, dass keine Angriffe wie Man-in-the-Middle erfolgt sind.

- **Grundlagen der Sicherheit**

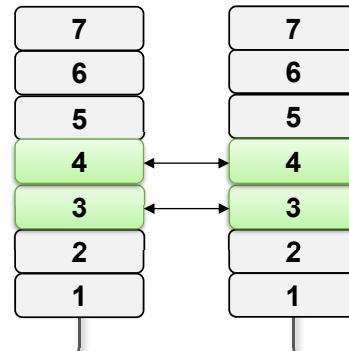
- ▶ Sicherheitsprobleme und -ziele
- ▶ Symmetrische und asymmetrische Verschlüsselung
- ▶ Authentifizierung

- **Sichere Internet-Protokolle**

- ▶ Vermittlungsschicht: IPSec, IKE
- ▶ Transportschicht: TLS

- **Schutz von lokalen Netzen**

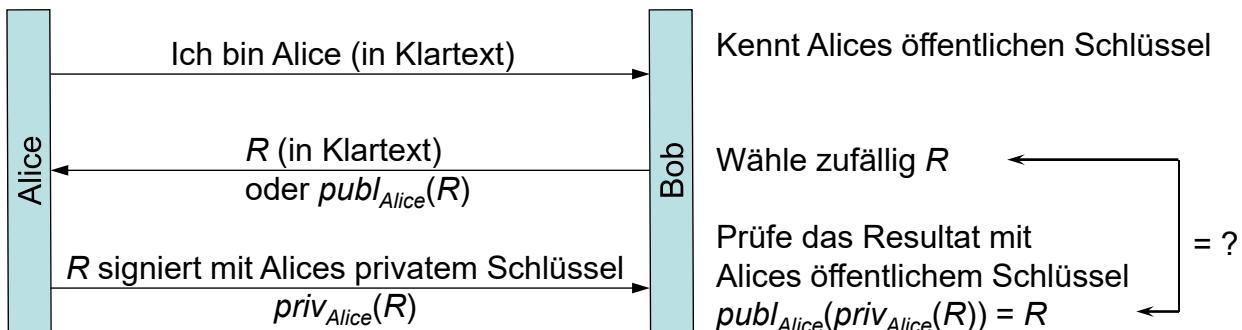
- ▶ Firewalls: Packet Filter, Application Gateways



## Authentifizierung mit RSA

- Weit verbreitet: *Challenge/Response*

- ▶ Beispiel: Alice authentifiziert sich gegenüber Bob
- ▶ Alice nutzt privaten Schlüssel  $priv_{Alice}$ , um eine kryptographische Operation auf einem Wert  $R$  (Challenge) durchzuführen, den Bob liefert hat:

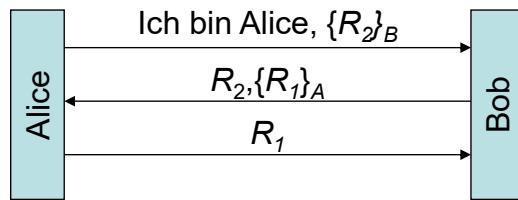


Wichtig: nutze das Challenge  $R$  nur einmal – die Nicht-Vorhersagbarkeit von  $R$  ist wichtig!

Ein wichtiger Begriff in diesem Zusammenhang ist *Nonce*: Number used ONCE. Dieser Begriff bezeichnet einen zufälligen, nur einmal verwendeten Wert.

## Mutual Authentication

- Gegenseitige Authentifizierung (*mutual authentication*)



- ▶ Kennen Bob und Alice gegenseitig ihre öffentlichen Schlüssel, können sie das Challenge mit ihren privaten Schlüsseln ver-/entschlüsseln
- ▶ Während der Authentifizierung wird zusätzlich oft ein Einmal-Schlüssel (*Sitzungsschlüssel*) zur Verschlüsselung der folgenden Kommunikation erstellt, z.B. mittels Diffie-Hellman

Aber: wie kann Bob den öffentlichen Schlüssel von Alice erlangen?

Erhält er ihn von Alice, müsste sich Alice vorher schon authentifizieren, um sicherzustellen, dass wirklich Alice den Schlüssel überträgt...

## Trusted Intermediaries

- **Probleme bei Authentifizierung:**

- ▶ Verwendung symmetrischer Kryptographie: gemeinsamer Schlüssel benötigt – Schlüsselaustausch aber erst nach Authentifizierung möglich
- ▶ Verwendung asymmetrischer Kryptographie: Authentizität des öffentlichen Schlüssels?

- **Lösung:**

- ▶ Führe vertrauenswürdige Stellen ein (*Trusted Intermediaries*)
- ▶ Etabliere geheime Schlüssel mit der vertrauenswürdigen Stelle
- ▶ Die vertrauenswürdige Stelle generiert bei Bedarf einen geheimen Schlüssel bzw. teilt einen öffentlichen Schlüssel mit

- **Konzepte:**

- ▶ Key Distribution Center (KDC) ← für geheime Schlüssel
- ▶ *Certification Authorities (CAs)* ← zur Hinterlegung öffentlicher Schlüssel (*Zertifikate*)

## Certification Authorities (CAs)

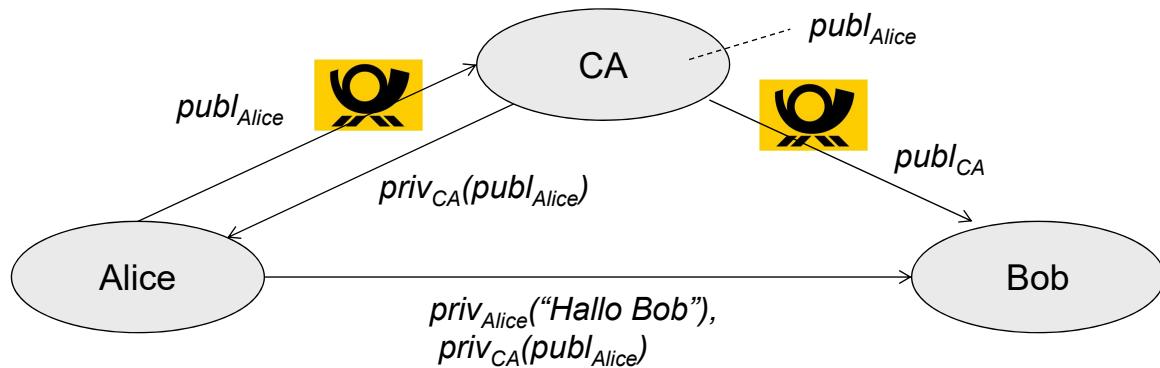
- Asymmetrische Kryptographie: *Certification Authority* (CA) als vertrauenswürdige Stelle
  - ▶ CA generiert *Zertifikate*
    - Zertifikat: öffentlicher Schlüssel und Name des Besitzers, signiert vom CA:  
 $[Alice, \text{priv}_{CA}(\text{publ}_{Alice})]$
  - ▶ Problem: zurückgezogene Zertifikate
    - Hat Bob das Zertifikat erhalten, kann er nicht wissen, ob Alice etwas später immer noch den passenden privaten Schlüssel verwendet
    - *Certificate Revocation List* (CRL) – Veröffentlichung zurückgezogener Zertifikate in bekannten Listen
- Vorsicht
  - ▶ Jeder kann eine CA aufsetzen!
  - ▶ *Trust Stores*: in Betriebssystemen oft Menge an vertrauenswürdigen Zertifikaten vorinstalliert

## Certification Authorities (CAs)

- **Certification Authorities** zur Hinterlegung öffentlicher Schlüssel

► **Zertifikat**: öffentlicher Schlüssel und Name des Besitzers, signiert vom CA

■  $[Alice, \text{priv}_{CA}(\text{publ}_{Alice})]$



Neues Problem: wie authentifiziert man sich gegenüber der CA und wie die CA gegenüber dem, der ein Zertifikat erstellen möchte?

Hier bleibt letztlich nur ein Austausch von Schlüsseln auf anderem Weg – z.B. persönlich oder per Post.

## Organisation von CAs

- **Nicht nur eine einzelne, weltweite CA notwendig**
  - ▶ Vertrauen aller in diese CA notwendig
  - ▶ Skalierbarkeit
- **Stattdessen: Verbund von CAs**
  - ▶ Oft: *CA-Hierarchie*
    - Root-CA vergibt Zertifikate für 2nd-Level-CAs
      - 2nd-Level-CAs vergeben Zertifikate für 3rd-Level-CAs
      - ...
    - Resultat: *Zertifikatsketten*

# Kapitel 6: Internet-Protokolle und Sicherheit

- **Grundlagen der Sicherheit**

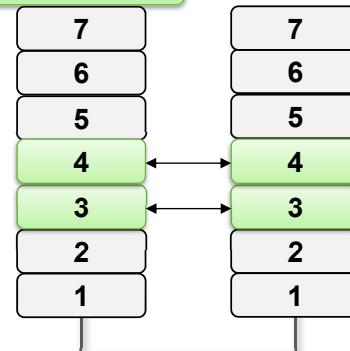
- ▶ Sicherheitsprobleme und -ziele
- ▶ Symmetrische und asymmetrische Verschlüsselung
- ▶ Authentifizierung

- **Sichere Internet-Protokolle**

- ▶ Vermittlungsschicht: IPSec, IKE
- ▶ Transportschicht: TLS

- **Schutz von lokalen Netzen**

- ▶ Firewalls: Packet Filter, Application Gateways



## IP Security Protocol (IPSec)

- **Sicherheitsmechanismen für IPv4:**
  - ▶ *IP Security Protocol (IPSec)*: Umsetzung der Sicherheitsmechanismen von IPv6 für IPv4
  - ▶ Verschlüsselung, Authentifizierung, Integritätsprüfung
- **Einsatzgebiete**
  - ▶ Verbindung von Rechnern/Netzen auf der Vermittlungsschicht
    - Gesamter Verkehr zwischen den Rechnern/Netzen wird gesichert
  - ▶ Haupteinsatzzweck: Virtual Private Network (VPN)

IPv4 definiert keine integrierten Sicherheitskonzepte - Daten werden in Klartext übertragen, eine Authentifizierung/Integritätsprüfung ist nicht möglich. Jeder kann die Absenderadresse eines IP-Pakets fälschen und den Payload mitlesen und modifizieren.

Verschlüsselungs- und Authentifizierungsmechanismen wurden allerdings bei IPv6 spezifiziert. Es ist eine allgemein gehaltene Spezifikation, so dass kryptographische Algorithmen ausgetauscht werden können, ohne das Protokoll zu verändern. Die Spezifikation für IPv6 wurde auch für IPv4 nachimplementiert, um bereits für IPv4 Sicherheitsmechanismen bieten zu können.

# IP Security Protocol (IPSec)

- **Zwei Arten der Anwendung:**

- *Transport Mode*

- Aufbau einer sicheren Punkt-zu-Punkt-Verbindung zwischen zwei Rechnern

- *Tunnel Mode*

- Verbindung zweier “Security-Gateways”, d.h. ganzer Netze
    - Security-Gateways können mit den Zugangsroutern der Netze verbunden werden

Transport Mode

Tunnel Mode



Der prominentere der beiden Modi ist der Tunnel Mode.

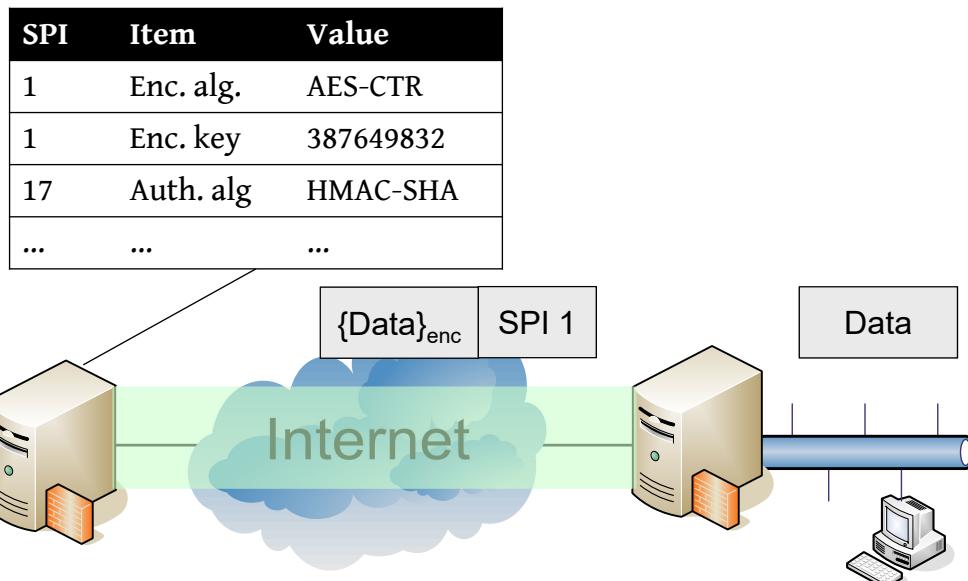
## Security Associations (SA)

- **IPSec legt keine kryptographischen Algorithmen fest**
  - ▶ **Security Association** – Vereinbarung der Sicherheitsmechanismen für eine Sitzung
    - Muss vor Beginn der Kommunikation ausgehandelt werden
    - Enthält z.B.
      - Authentifizierungsmechanismus und Schlüssel
      - Verschlüsselungsalgorismus und Schlüssel
      - Initialisierungsvektor (IV), falls benötigt
      - Gültigkeitsdauer der einzelnen Schlüssel und der gesamten SA
      - ...
  - ▶ **Security Parameter Index (SPI)** – Referenz auf eine SA
    - Muss jedes gesicherte Paket enthalten
    - Zeigt dem Empfänger, wie das Paket zu behandeln ist

# Security Associations und Security Parameter Index

- **Security Association Databases**

- ▶ SPI ist Index



Wurde eine SA ausgehandelt, speichern beide Kommunikationspartner die Vereinbarungen in einer lokalen Konfigurationsdatenbank; Über den SPI erfolgt der Zugriff auf eine SA in der Datenbank. Jedes durch IPsec gesicherte Paket muss also die SPI enthalten, um dem Empfänger mitzuteilen, mit welchen Algorithmen und Schlüsseln das Paket zu behandeln ist.

Eine SA ist immer unidirektional – für die Sicherung der gesamten Kommunikation muss jede Seite eine SA erstellen.

## IP Security Protocol (IPSec)

- **Ziel: Authentifizierung, Integrität und Verschlüsselung**

- ▶ *Authentication Header (AH)*: Informationen zur Authentifizierung des Senders und zur Integrität des IP-Pakets
- ▶ *Encapsulation Security Payload (ESP)*: Informationen zu Verschlüsselung, Integritätsprüfung
- ▶ Die Mechanismen können einzeln oder gemeinsam genutzt werden

IPSec unterscheidet zwei Teilbereiche der Sicherheit:

- (1) Authentifizierung des Senders
- (2) Verschlüsselung des Payloads

Beide dazu definierten Mechanismen unterstützen die Integritätsprüfung.

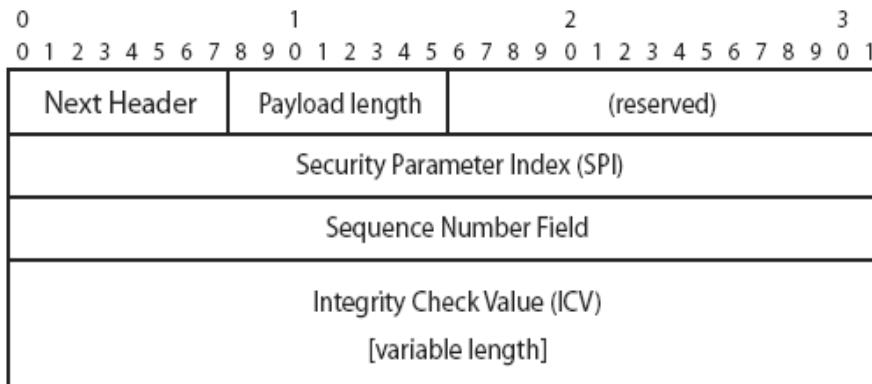
Sollen alle Sicherheitsaspekte unterstützt werden, können auch beide Mechanismen gemeinsam eingesetzt werden (für jeden muss allerdings eine eigene SA erstellt werden).

In der Praxis findet allerdings üblicherweise eine Authentifizierung schon während der Schlüsselaushandlung statt, so dass während einer gesicherten Kommunikation nur noch ESP verwendet werden muss.

## Authentication Header (AH)

- AH: Authentifizierung des Senders und Datenintegrität für IP-Pakete durch Hinzufügen einiger Header-Felder

- ▶ Bei IPv6: Erweiterungs-Header
- ▶ Bei IPv4: entsprechender Eintrag im Protocol-Feld



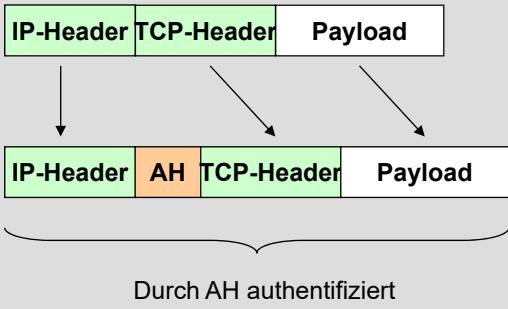
Authentizität und Integrität eines IP-Pakets können durch die vorher vorgestellten Algorithmen erreicht werden. IPSec definiert einen Erweiterungsheader AH, der dazu dient, dem Empfänger des Pakets alle zur Prüfung der Authentizität nötigen Informationen zukommen zu lassen:

- Next Header: Identifikation des Datenteils nach dem AH (IP, TCP, UDP, ...) – im IP-Paket selber wird im Protocol-Feld „AH“ eingetragen, die sonst dort befindliche Angabe wird im Transport-Modus nach hier übertragen, im Tunnel-Modus folgt das ursprüngliche IP-Paket.
- Payload Length: Länge des Integrity Check Value (ICV) in 32-Bit-Worten
- Reserved: für zukünftige Verwendung, aktuell werden Nullen eingefüllt
- SPI: Index der verwendeten SA, damit der Empfänger das Paket korrekt verarbeiten kann (verwendete Algorithmen und Schlüssel identifizieren kann).
- Sequence number: Eindeutige Nummer für jedes Paket, um gegen das Wiedereinspielen alter Pakete durch einen Angreifer zu schützen.
- ICV: Wert zur Sicherstellung der Integrität und Authentifizierung, berechnet nach dem in der SA festgehaltenen Verfahren, z.B. Verwendung von SHA-2 und Berechnung eines 128-Bit-Wertes  $\text{sha-2}(k_{\text{IPSec}} \mid \text{IP-Paket} \mid k_{\text{IPSec}})$ .

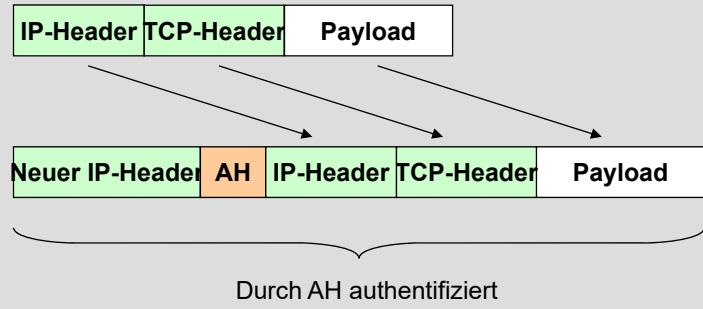
Ein Problem würde bei der Überprüfung des ICV auf Empfängerseite auftauchen: einige Felder des IP-Headers ändern sich während der Übertragung, z.B. TTL. Der Empfänger würde daher den ICV nicht über exakt das gleiche Paket berechnen wie der Sender, so dass die Authentizitätsprüfung fehlschlagen würde. Dies wird einfach dadurch umgangen, dass für die veränderlichen Felder auf beiden Seiten vor der ICV-Berechnung Default-Werte eingesetzt werden.

# Verwendung des Authentication Header

## Transport Mode



## Tunnel Mode



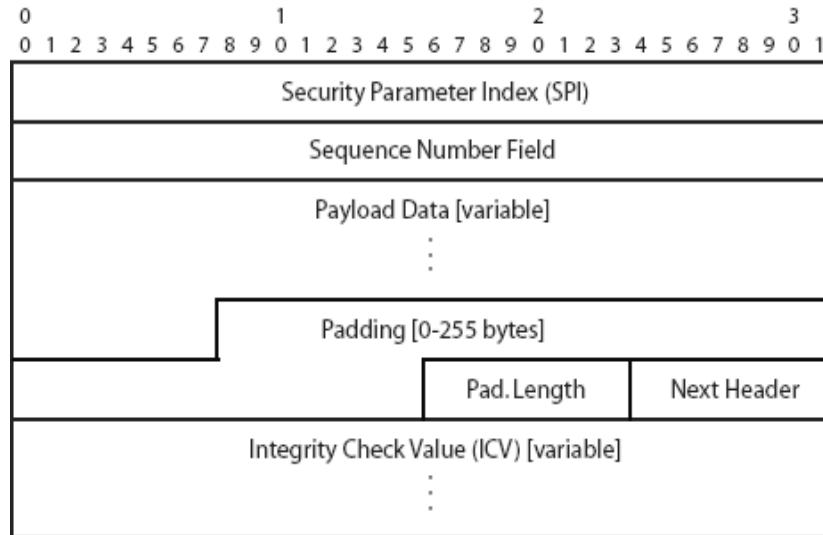
- AH kann in beiden Betriebsarten verwendet werden:

- ▶ Transport Mode: der AH wird an den IP-Header angehängt
- ▶ Tunnel Mode: da das Paket durch Security-Gateways weitergeleitet wird, wird ein neuer IP-Header benötigt, in dem Absender und Empfänger die jeweiligen Security-Gateways sind

## Encapsulating Security Payload (ESP)

- **ESP: Vertraulichkeit durch Verschlüsselung von IP-Paketen**

- ▶ Erweiterungsheader mit Angaben zur Verschlüsselung

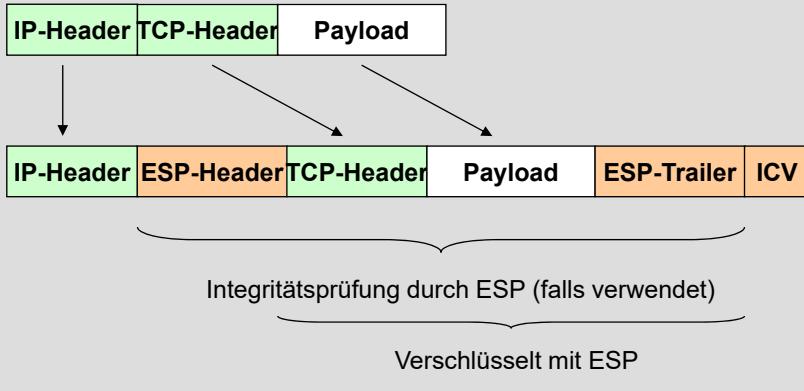


Auch zur Verschlüsselung von IP-Paketen werden wieder ein paar zusätzliche Headerangaben benötigt:

- SPI, Sequence Number, Next Header, ICV: wie in AH
- ICV ist optional, falls auch die Integrität geprüft werden soll.
- Payload Data: verschlüsselter Payload des IP-Pakets mit einem in der SA vereinbarten Verfahren.
- Padding: Zufallszahlen, zum Auffüllen des Payloads für das Verschlüsselungsverfahren (Vielfaches der Blocklänge nötig).
- Padding Length: Anzahl der eingefügten Padding-Bytes

# ESP im Transport Mode

## Transport Mode

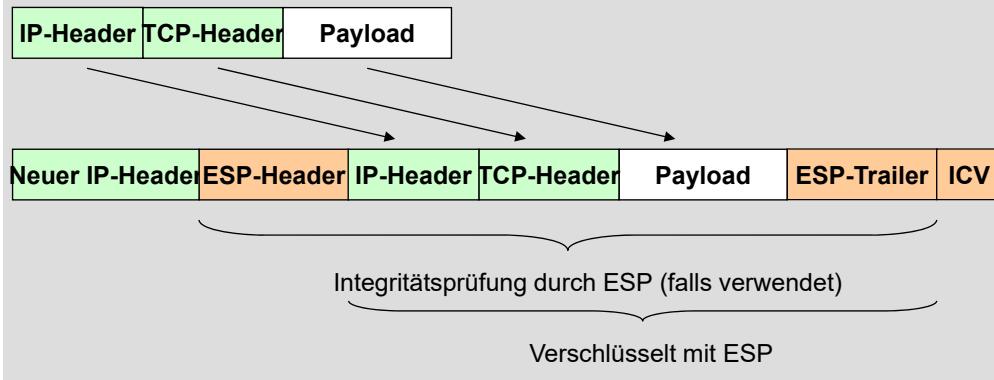


- **Einsatz im Transport Mode**

- ▶ Gesicherter Tunnel zwischen zwei Endpunkten
- ▶ Payload ist verschlüsselt und integritätsgeprüft

# ESP im Tunnel Mode

## Tunnel Mode

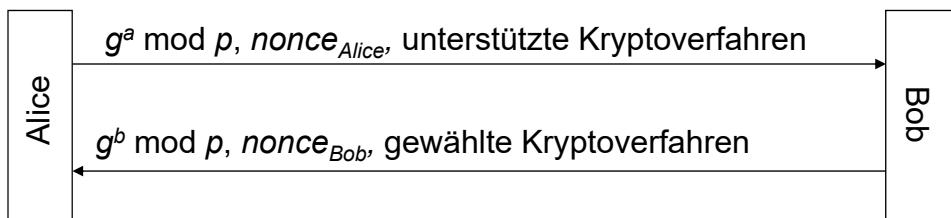


- **Einsatz im Tunnel Mode**

- ▶ z.B. zur Verbindung zweier physikalischer Netze in ein logisches LAN
- ▶ Auch: *Virtual Private Network (VPN)*: Einsatz des Tunnel Mode zur Verbindung einzelner Rechner mit einem LAN

# Internet Key Exchange (IKE)

- AH und ESP: einzeln oder gemeinsam einsetzen
  - ▶ Beide benötigen einen Mechanismus zur Aushandlung der SAs (dies beinhaltet Schlüsselaustausch!)
  - ▶ *Internet Key Exchange* (IKE) zur Vereinbarung von SAs über UDP
- Schritt 1: Schlüsselaustausch (Diffie-Hellman)
  - ▶ Berechnung eines geheimes Schlüssels  $S_{IKE} = g^{ab} \text{ mod } p$
  - ▶ Vereinbarung einer SA: Initiator listet die von ihm unterstützten Verfahren auf, Partner wählt ein von ihm unterstütztes Verfahren aus



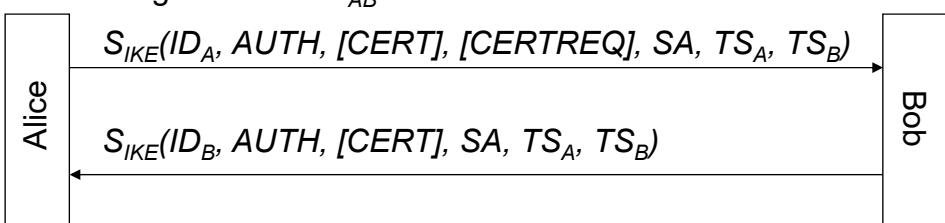
IPSec definiert nicht, wie ein Schlüssel zustande kommt, es werden existierende Schlüssel vorausgesetzt. Eine Möglichkeit, einen Schlüsselaustausch vorzunehmen, ist das Internet-Key-Exchange-Protokoll; der Schlüsselaustausch basiert hier auf Diffie-Hellman.

Achtung: es hat hier keine Authentifizierung stattgefunden, d.h. wir wissen nicht, ob sich ein Man-in-the-Middle in den Schlüsselaustausch eingeklinkt hat.

# Internet Key Exchange (IKE)

## • Schritt 2: gegenseitige Authentifizierung

- ▶ Mehrere Varianten definiert
  - Pre-Shared Secrets oder Zertifikate
  - Verschlüsselung von Authentifizierungsnachrichten mit  $S_{IKE}$
- ▶ z.B. durch Signierung der Nachrichten mit privaten Schlüsseln
  - AUTH: signierter Hash der zuvor ausgetauschten Nachrichten
  - CERT, CERTREQ: Zertifikate bzw. Zertifikatsanfrage (optional)
  - TS<sub>A</sub>, TS<sub>B</sub>: IP-Adressbereiche, auf die die SA angewendet werden soll
  - SA: Security Association für die folgende Kommunikation inklusive Sitzungsschlüssel  $S_{AB}$



Erst im folgenden Schritt kommt die Authentifizierung. Zu diesem Zeitpunkt haben wir bereits einen symmetrischen Schlüssel vereinbart. Schlägt die Authentifizierung fehl, wird die Kommunikation allerdings abgebrochen und der ausgehandelte Schlüssel nicht weiter verwendet.

An dieser Stelle wird auch ein Sitzungsschlüssel  $S_{AB}$  für die folgende Kommunikation gewählt (Teil der SA). Der zuvor per Diffie-Hellman erstellte Schlüssel ist nur für IKE selbst, um einen gesicherten Kanal zu erstellen, über den Schlüssel für IPsec ausgehandelt werden können. Über den einen IKE-Kanal können nun Schlüssel für beliebig viele Security Associations ausgetauscht werden.

## Internet Key Exchange (IKE)

- **Schritt 3 (optional): Aushandlung mehrerer SAs wie in Schritt 1 mit neuen Sitzungsschlüsseln  $S_{AB}$** 
  - ▶ Soll mehr als eine Kommunikationssitzung ausgehandelt werden, können jetzt noch weitere SAs für andere TS (IP-Adressbereiche) ausgetauscht werden
  - ▶ Zweck: Generierung von Sitzungsschlüsseln, damit  $S_{IKE}$  möglichst selten benutzt wird und dadurch weniger Angriffsfläche bietet
- **Verwendung von IKE:**
  - ▶ IKE wird oft verwendet, ist aber kein fester Bestandteil von IPSec
  - ▶ Jedes beliebige Protokoll zur Schlüsselvereinbarung und zur Aushandlung von Kryptoalgorithmen kann verwendet werden

# Kapitel 6: Internet-Protokolle und Sicherheit

- **Grundlagen der Sicherheit**

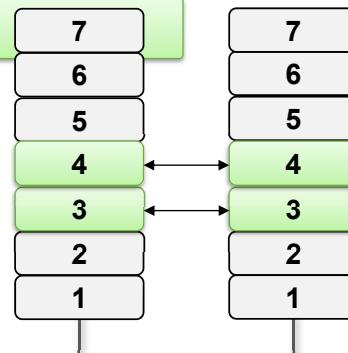
- ▶ Sicherheitsprobleme und -ziele
- ▶ Symmetrische und asymmetrische Verschlüsselung
- ▶ Authentifizierung

- **Sichere Internet-Protokolle**

- ▶ Vermittlungsschicht: IPSec, IKE
- ▶ Transportschicht: TLS

- **Schutz von lokalen Netzen**

- ▶ Firewalls: Packet Filter, Application Gateways



## Transport Layer Security (TLS)

- **Bietet Authentifizierung, Integrität und Verschlüsselung zwischen zwei *Anwendungen***

- ▶ TLS baut auf TCP auf, de-facto Standard für sicherheitssensitive Anwendungen im Internet
  - Wichtigste Anwendung: gesicherter HTTP-Verkehr (HTTPS) z.B. für Online-Banking
- ▶ Vorgänger: Secure Socket Layer (SSL)
- ▶ Spezielle Port-Nummern bei der Kommunikation über TLS, z.B.
  - HTTP über TLS = HTTPS nutzt Port 443
  - Telnet über TLS = Telnets nutzt Port 992

# Transport Layer Security (TLS)

- TLS umfasst vier Unterprotokolle:

- *Handshake Protocol*

- Authentifizierung, Aushandlung von kryptographischen Algorithmen und Schlüsseln

- *Record Protocol*

- Verschlüsselung und Integritätsschutz (und Kompression)

- *Change Cipher Spec*

- Signalisiert Beginn der verschlüsselten Kommunikation

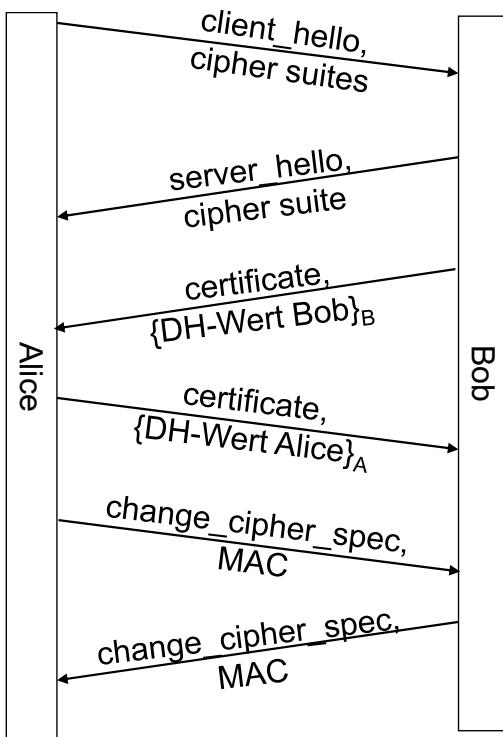
- *Alert Protocol*

- Reaktion auf Ausnahmesituationen

## Handshake Protocol

- Wie IKE: Aushandlung von Sicherheitsparametern
  - ▶ Aushandlung für eine *Sitzung*, d.h. eine Assoziation zwischen einem Client und einem Server
    - Kann mehrere Verbindungen umfassen
  - ▶ Beinhaltet *Master Secret* für alle Verbindungen der Sitzung (zur Generierung von Schlüsseln für einzelne Verbindungen)
- Aufgaben:
  - ▶ Aushandeln eines *Verschlüsselungsalgorithmus*\*
  - ▶ Gegenseitige *Authentifizierung*
  - ▶ *Schlüsselaustausch*
- Ähnlich zu IKE, aber kein eigenständiges Protokoll, sondern fester Bestandteil von TLS

## Etablierung einer Sitzung



### • Ablauf von *Authentifizierung und Schlüsselaustausch*

- ▶ Einigung auf Algorithmen in Hello-Phase
- ▶ Bob authentifiziert sich und schickt signierten DH-Wert zum Schlüsselaustausch an Alice
- ▶ Alice authentifiziert sich und schickt signierten DH-Wert zum Schlüsselaustausch an Bob
- ▶ Abschließend Sicherstellung, dass sich Alice und Bob auf gemeinsamen Schlüssel geeinigt haben: Austausch eines Message Authentication Code über das Master-Secret und einen Hash der vorherigen Nachrichten

#### Phase 1: Wahl eines Verschlüsselungsalgorithmus

Die Hello-Nachricht von Alice beinhaltet Listen der von Alice unterstützten Verschlüsselungs- und Kompressionsalgorithmen.

Bob wählt aus den Listen jeweils einen Algorithmus aus und teilt ihn Alice mit.

Zudem beinhalten beide Nachrichten noch einen Zufallswert.

#### Phase 2: authentifizierter Schlüsselaustausch (Serverseite)

Zum Schlüsselaustausch sind verschiedene Verfahren definiert. Empfohlen wird die Verwendung von Diffie-Hellman mit neu gewählten Zufallswerten. Bei diesem Verfahren wählt Bob einen Geheimwert und berechnet seinen dazugehörigen öffentlichen Diffie-Hellman-Wert. Diesen signiert er und sendet ihn zusammen mit seinem Zertifikat an Alice. Hier geschehen zwei Dinge gleichzeitig:

- Schlüsselaustausch nach Diffie-Hellman
- Authentifizierung des Servers durch Anwendung des zertifizierten Schlüssels auf den DH-Wert.

#### Phase 3: authentifizierter Schlüsselaustausch (Clientseite)

Analog zu oben schickt Alice ihren signierten DH-Wert an Bob.

Basierend auf den ausgetauschten DH-Werten und den Zufallswerten der Hello-Nachrichten können Alice und Bob nun ein Master-Secret berechnen.

Die Einbeziehung der zusätzlichen Zufallswerte hat den Zweck, eine einfachere Neuberechnung eines Master-Secrets bei Aufbau einer weiteren Verbindung zu ermöglichen: es müssen nur noch die Hello-Nachrichten ausgetauscht werden, die DH-Werte vom vorherigen Schlüsselaustausch können weiterverwendet werden. Dies beschleunigt die Etablierung einer weiteren Verbindung.

#### Phase 4: Abschluss des Handshakes

Durch ChangeCipherSpec signalisieren beide Seiten, dass der Handshake beendet ist. Um sicherzugehen, dass beide Seiten sich auf den gleichen Schlüssel geeinigt haben, wird ein Message Authentication Code beigefügt, der über das Master-Secret und einen Hash der gesamten vorherigen Handshake-Nachrichten berechnet wird.

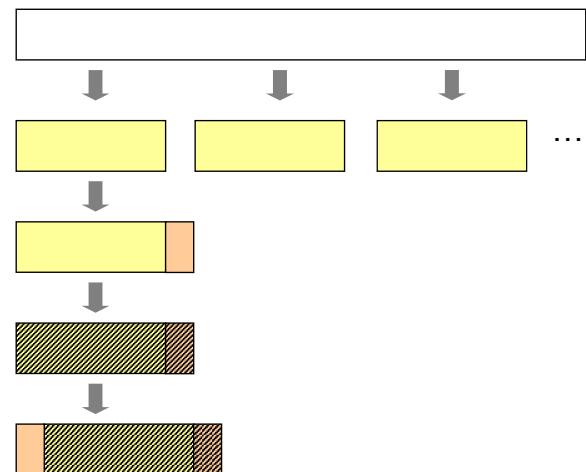
In Phase 2 und 3 können auch drei andere Varianten verwendet werden:

- Anonymer DH-Schlüsselaustausch ohne Zertifikate. Bitte Vorsicht mit Man-in-the-Middle-Attacken.
- Austausch vorzertifizierter DH-Werte. Dies hat allerdings zur Folge, dass man immer den gleichen DH-Wert verwendet, was dem Schlüsselaustausch eine Zufallskomponente nimmt.
- Wahl des Schlüssels durch Alice und mit RSA verschlüsselte Übertragung an Bob.

## Record Protocol

- Übernimmt Verschlüsselung und Kompression aller Nachrichten nach der Change Cipher Spec:

- ▶ Segmentiere einen Datenstrom in Blöcke fester Länge
  - Kompression möglich
- ▶ Anhängen eines MAC
- ▶ Verschlüsselung von komprimierten Daten und MAC
- ▶ Hinzufügen eines TLS-Headers:
  - Content Type (Alert, Handshake, Anwendungsdaten)
  - Versionsnummer
  - Länge



## Alert Protocol

- Für Ausnahmesituationen – definiert Fehlermeldungen und Maßnahmen, eingeteilt in zwei Sicherheitslevel

- *Level 1: Warning*

- Ausnahmesituationen mit geringem Sicherheitsrisiko
    - Keine Maßnahmen definiert

- *Level 2: Fatal*

- Ausnahmesituationen, die auf einen Angriff hindeuten, z.B.:
      - Unexpected message
      - Bad Record MAC
      - Decryption/Decompression Failure
      - Handshake Failure
    - Verbindung wird geschlossen
    - Innerhalb der aktuellen Sitzung werden keine Verbindungen mehr geöffnet



## TLS 1.3

- **Aktuelle Version: TLS 1.3:**

- ▶ Nutzung vieler schwacher Verschlüsselungs- und Hashverfahren wird verboten, Aufnahme neuer Verfahren
- ▶ Perfect Forward Secrecy – Schlüsselaustausch nach der oben dargestellten Variante erfordert
- ▶ 1-RTT und 0-RTT Handshakes
  - Beschleunigung der Schlüsselaushandlung
  - Verwendet z.B. in QUIC – siehe Vorlesung “Communication Systems Engineering”
- ▶ Unterstützung obsoleter Features nicht fortgesetzt
  - Kompression, ChangeCipherSpec als eigenes Protokoll

## IPSec vs. TLS

IPSec	TLS
Network Layer	Transport Layer
Vor dem Benutzer verborgen	Interaktion mit dem Benutzer (z.B. Akzeptieren von Zertifikaten)
Kann automatisiert werden	
Zentrale Verwaltung	Verwaltung durch die Anwendung oder den Benutzer
Unabhängigkeit von speziellen Algorithmen (Verschlüsselung, Hash, Kompression...)	

- Zusammenfassend: es ist unmöglich zu sagen, dass IPSec besser ist als TLS oder umgekehrt – sie sind für verschiedene Nutzungsszenarien entwickelt**

Sollen ganze Netze oder einzelne Rechner sicher gekoppelt werden, bietet sich die Nutzung von IPSec an. Dies ist zum Beispiel bei VPNs der Fall.

Muss nur die Kommunikation zweier Anwendungen gesichert werden, ist die Nutzung von TLS deutlich einfacher.

# Kapitel 6: Internet-Protokolle und Sicherheit

- **Grundlagen der Sicherheit**

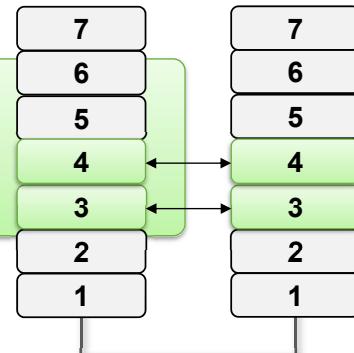
- ▶ Sicherheitsprobleme und -ziele
- ▶ Symmetrische und asymmetrische Verschlüsselung
- ▶ Authentifizierung

- **Sichere Internet-Protokolle**

- ▶ Vermittlungsschicht: IPSec, IKE
- ▶ Transportschicht: TLS

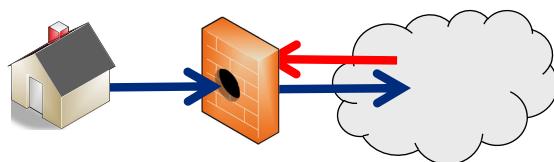
- **Schutz von lokalen Netzen**

- ▶ Firewalls: Packet Filter, Application Gateways



## Sicherung von Netzwerken: Firewall

- **IPSec/TLS: Verschlüsselung, Authentifizierung und Integritätsprüfung**
  - ▶ Aber: wie schützen wir unser Netz gegen unerwünschten oder sogar schädlichen Verkehr?
    - Eine *Firewall* ist eine Barriere zwischen einem privaten, geschützten Netzwerk (Intranet) und anderen Netzwerken (z.B. dem Internet)
    - Ziel: unerwünschte und unautorisierte Kommunikation in oder aus dem geschützten Netzwerk verhindern
    - Dazu nötig: Verarbeitung von Informationen ab Schicht 3



Funktionen einer Firewall können sein:

- Benutzerkontrolle - Nur autorisierte Benutzer haben Zugriff über eine Firewall
- Zugriffskontrolle - Einschränkung des Zugriffs auf bestimmte Dienste
- Verhaltenskontrolle - Nutzung der Applikationen ist definiert, z.B. Mail-Filter für Viren und bestimmte Dateianhänge
- Richtungskontrolle - Unterschiedliche Regeln für eingehende und ausgehende Daten
- Protokollierung - Zusammenfassung der Nutzung und der Zugriffe
- Verbergen des internen Netzes - Topologie, Adressen, ... sollen nach außen hin nicht sichtbar sein

## Firewall-Typen

- Generell zwei Typen von Firewalls:

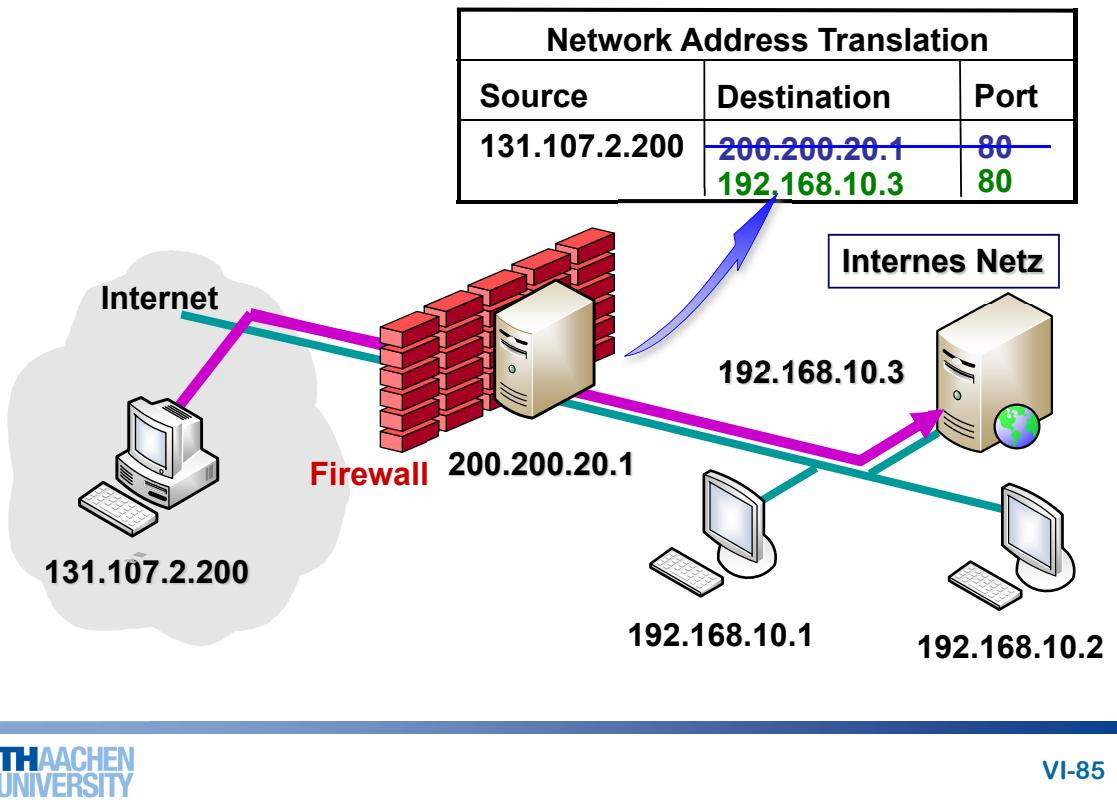
- *Paketfilter*

- Netzwerkverkehr wird anhand von Filterregeln analysiert: Quelladressen und Zieladressen (IP-Adresse und Port), Transportprotokoll, Verbindung
    - Billig und einfach (alle Verbindungen können kontrolliert werden), aber schwer, Filterregeln korrekt/vollständig zu definieren
    - Ggf. mit Intrusion-Detection-System zur Erkennung „böswilliger“ Ströme

- *Proxy Server (Application Gateway)*

- Kontrollierter Zugriff auf einen Dienst: die Firewall fängt einen Aufruf ab und entscheidet, ob er weitergeleitet werden soll
    - Proxy ist die nach außen hin einzige sichtbare Komponente des Netzes
    - Kontrolle basiert auf Benutzernamen, Protokollen und/oder Inhalt
    - Mehr Möglichkeiten (Loggen detaillierter Informationen, Authentifizierung, ...), aber für jedes Anwendungsprotokoll wird ein eigener Proxy benötigt

## Funktionen einer Firewall: Proxy



Basierend auf der IP-Adresse des Senders kann die Firewall die Adresse des Empfängers ändern und die Pakete zustellen. Damit bleibt das 'wahre' Aussehen des internen Netzes dem Sender verborgen. Diese Funktionalität bietet auch NAT.

Darüber hinaus bieten Firewalls allerdings noch deutlich mehr Funktionalität – sie erlauben eine gezieltere Filterung von Paketen.

## Paketfilter

- **Zwei mögliche Prinzipien:**

- ▶ Alles, was nicht ausdrücklich erlaubt ist, ist verboten
- ▶ Alles, was nicht ausdrücklich verboten ist, ist erlaubt
- ▶ Z.B. für den Mailserver auf Rechner 137.226.12.67 und Port 25:
  - From (IP \* ), (port \*)  
To (IP 137.226.12.67), (port 25)  
DENY
  - From (IP 137.226.12.67), (port 25)  
To (IP \* ), (port 25)  
ALLOW
- ▶ D.h.: aus dem Netz heraus darf unser Mailserver Mails an jeden externen Mailserver schicken, aber niemand darf von außerhalb Mails in das Netz hineinsenden

## Paketfilter

---

- **Eigenschaften:**

- ▶ Schnelle Verarbeitung von IP-Paketen, aber nur Kontrolle auf Adressierungsebene
- ▶ *Statische Paketfilter* haben nur einen festen Satz solcher Regeln
- ▶ *Dynamische Paketfilter* berücksichtigen Zustand einer Verbindung:
  - Lehne alle Pakete von außen ab
  - Erst nach einem Verbindungsauflauf von innen (gesetztes SYN-Flag), werden Antwort-Pakete von außen durchgelassen
    - Temporäres Anlegen von Filterregeln

# Proxy Server

---

- **Zwei Typen:**

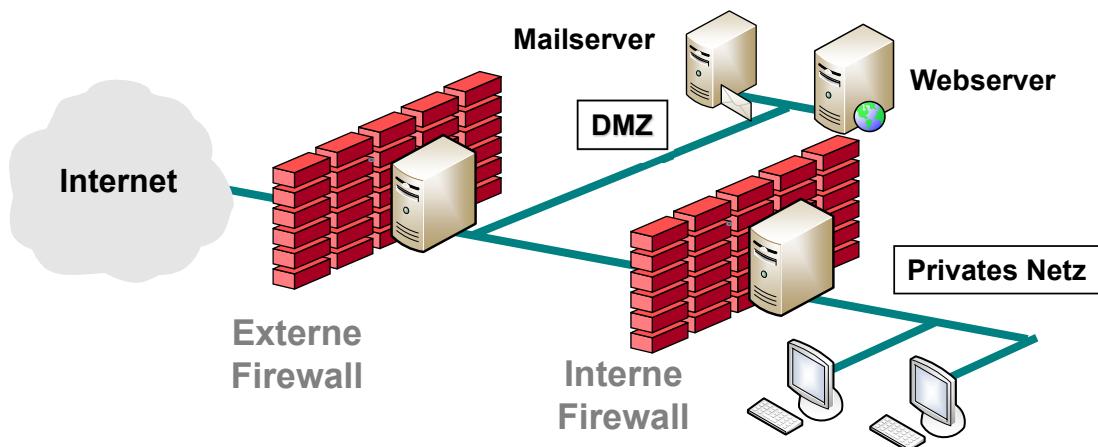
- ▶ *Circuit-Level Proxy*

- Arbeitet auf Schicht 3/4 (z.B. Port-Nummern)
    - Proxy, der von allen Anwendungen verwendet werden kann
    - Fängt alle Verbindungen ab, damit bleibt das interne Netz verborgen
    - Im Wesentlichen: NAT + Firewall

- ▶ *Application-Level Proxy*

- Verwendet auch Informationen von Schicht 7
    - Ein eigener Proxy für jedes Anwendungsprotokoll (SMTP, FTP, HTTP, ...) ist nötig
    - Benutzer können authentifiziert werden, bevor Daten durchgelassen werden
    - Mehr Möglichkeiten als ein Paketfilter, aber aufwändiger

## Sicherheitskonzept – DMZ

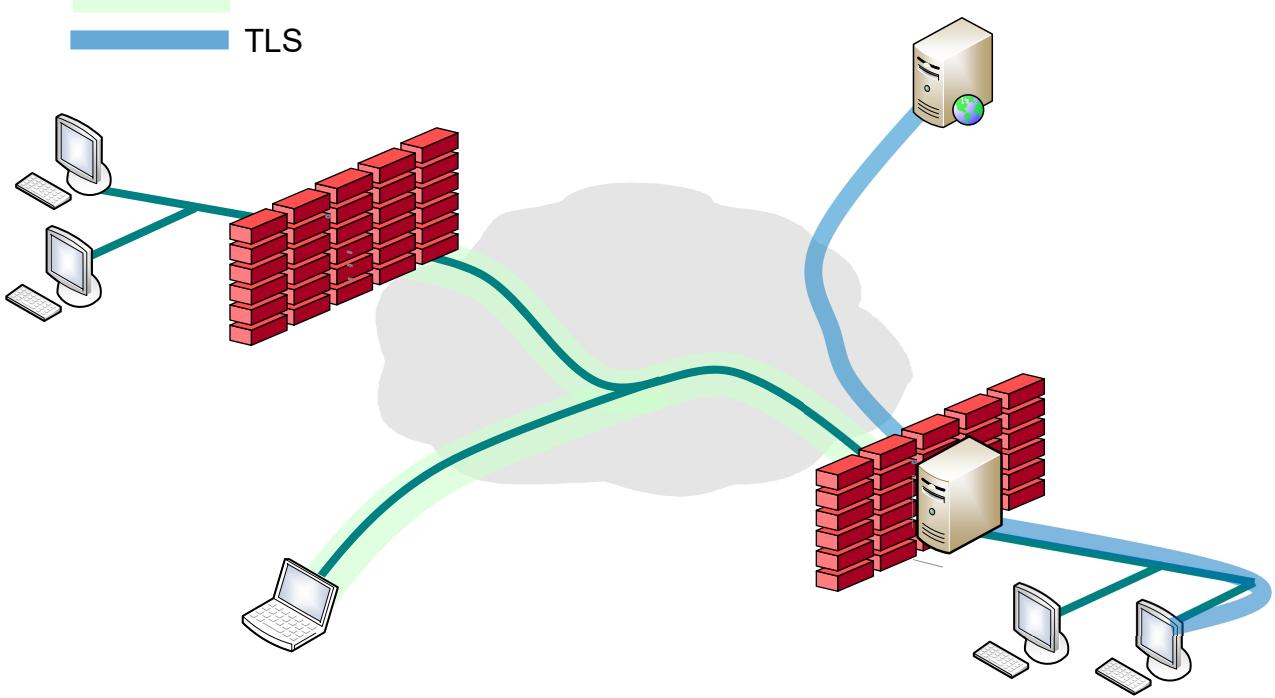


- ▶ Ressourcen, die vom Internet aus erreichbar sein sollen, in einem eigenen Netzwerksegment platzieren (*DMZ – Demilitarisierte Zone*)
- ▶ Dieses Segment ist durch eine (einfache) Firewall an das Internet angeschlossen (einfache Filterung für unkritische Systeme)
- ▶ Das private Netz ist durch eine weitere (starke) Firewall komplett abgeschottet

## Firewalls + IPsec + TLS

IPSec Tunnel Mode

TLS



## Zusammenfassung

---

- **Verschlüsselung und Authentifizierung**
  - ▶ Symmetrische vs. asymmetrische Kryptographie
    - Symmetrisch: schnell, Schlüsselaustausch notwendig
      - Schlüsselaustausch: Diffie-Hellman, Authentifizierung notwendig
    - Asymmetrisch: langsam, Zertifizierung öffentlicher Schlüssel notwendig
  - ▶ CAs als Ausgangspunkt zur Authentifizierung
- **Sichere Protokolle**
  - ▶ IPSec: Verschlüsselung und Authentifizierung auf Schicht 3
    - Zusatzprotokoll zum Schlüsselaustausch, z.B. IKE
  - ▶ TLS: Verschlüsselung, Authentifizierung und Schlüsselaustausch auf/über Schicht 4
  - ▶ Ähnlicher Funktionsumfang, unterschiedliche Anwendungsfälle
- **Weitere Sicherheitsmaßnahmen, z.B. Firewalls**

## Ausblick

---

- **Weitere Vorlesungen im Bereich Kommunikationssysteme bei COMSYS**
  - ▶ Communication Systems Engineering (WS)
  - ▶ Advanced Internet Technology (WS)
  - ▶ Mobile Internet Technology (SS)
- **Research Focus Classes**
  - ▶ Zu speziellen und höchst aktuellen Themen
- **Seminare & Praktika**
  - ▶ Internet Technology & Advanced Internet Technology (Seminar)
  - ▶ Communication Systems Lab