Systeme I: Betriebssysteme

Kapitel 9 Sicherheit

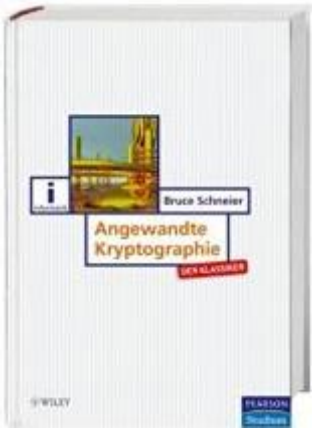
Wolfram Burgard

Version 24.01.2018

Quellen

Teile dieses Kapitels basieren auf

- dem Skript zur Vorlesung Sicherheit am Karlsruher Institut für Technologie,
- der Vorlesung „Systeme I“ von Prof. Schneider aus dem Wintersemester 2014/15 und
- dem Buch „Angewandte Kryptographie“ von Bruce Schneier.



Was ist Sicherheit?

- Betriebssicherheit (safety)
  - Sicherheit der Situation, die vom System geschaffen wird
  - Das System verhält sich entsprechend der Spezifikation fehlerfrei und gefährdet die Benutzerin / den Benutzer nicht
  - Annahme: Keine externen Akteure, keine Manipulation
  - z.B. Notausgangstüre
- Angriffssicherheit (security)
  - Sicherheit in Bezug auf äußere Manipulation
  - Nicht nur wahrscheinliche Fehlerszenarien müssen betrachtet werden
  - z.B. Türschloss, Wasserzeichen

Beispiel: Chrysler Hack

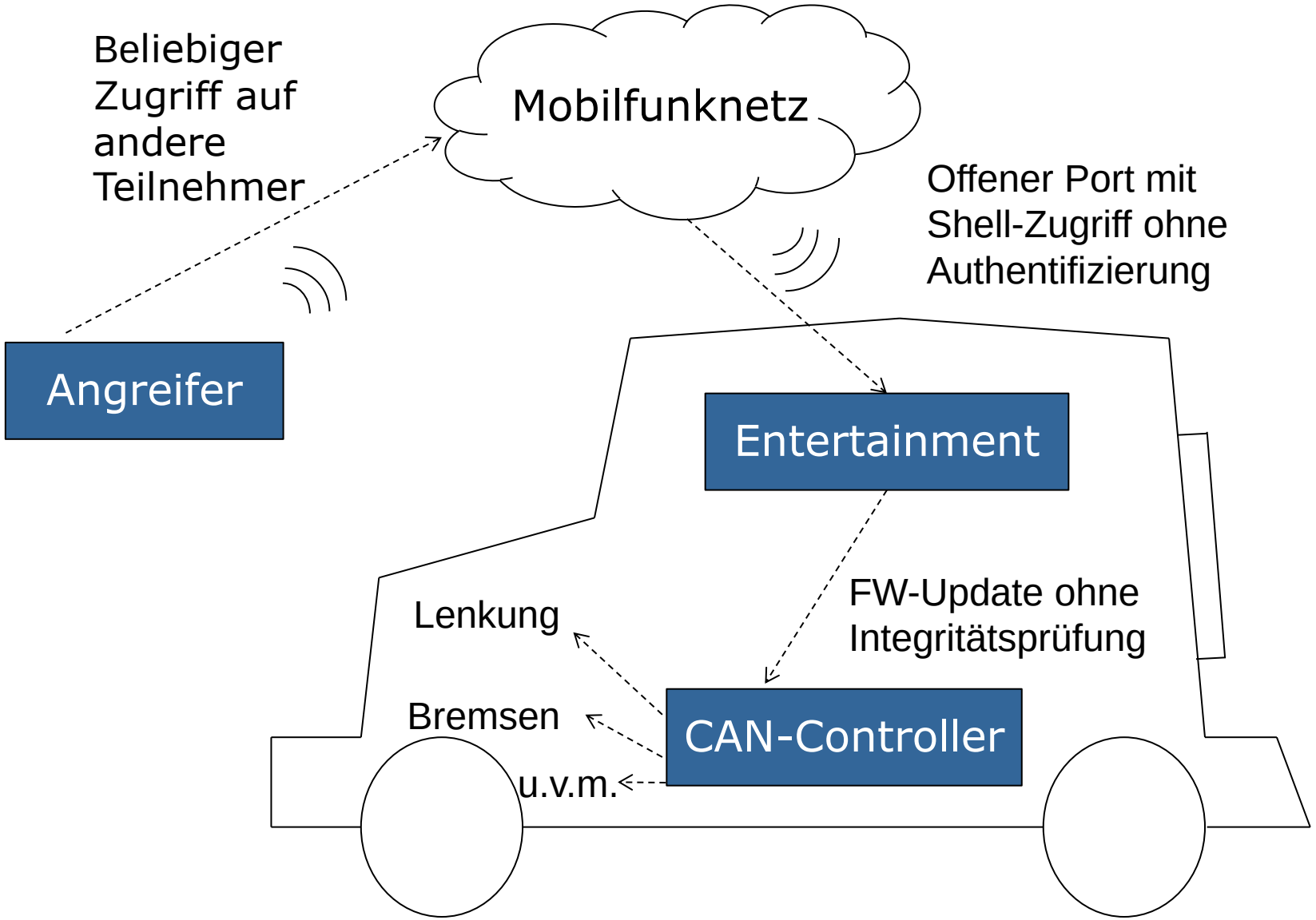
- „After Jeep Hack, Chrysler Recalls 1.4M Vehicles for Bug Fix“ (Wired, 24.07.15)
- Gravierende Sicherheits-lücken im Entertainment-System

Quelle: Andy Greenberg/Wired
Forscher: Charlie Miller
and Chris Valasek

→ Fahrzeuge über Mobilfunknetz fernsteuerbar

→ Mangelhafte Angriffssicherheit kann auch Betriebssicherheit beeinträchtigen

Chrysler Hack: Vorgehen



Chrysler Hack: Resultate

Sicherheitsziele

- Vertraulichkeit der Daten (data confidentiality)

  Geheime Daten sollen geheim bleiben.

  → Kryptographie
- Datenintegrität (data integrity)

  Unautorisierte Benutzer dürfen ohne Erlaubnis des Besitzers Daten nicht modifizieren.

  → Signaturen
- Systemverfügbarkeit (system availability)

  Niemand soll das System so stören können, dass es dadurch unbenutzbar wird.
- Datenschutz (privacy)

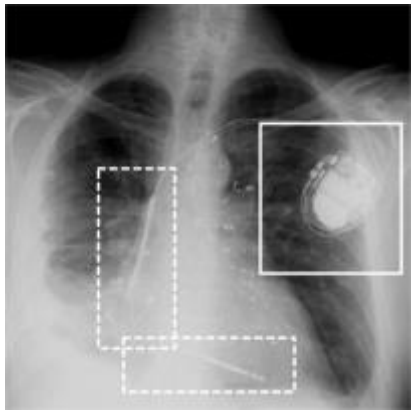
  Schutz von Personen vor dem Missbrauch ihrer persönlichen Daten

Beispiele für Sicherheitsziele

- Bluetooth-Zugriff auf Defibrillator

  Datenintegrität: Keine Konfiguration durch Fremde
  →Authentifizierung, Signaturen

  Systemverfügbarkeit: Nicht durch Verbindungsversuche Batterie leeren
  → Protokollentwurf



- Geruchssensor für Stoma-Träger

  Vertraulichkeit und Datenschutz: Nicht jeder soll die Messwerte mitlesen können
  →Verschlüsselung

- Konfigurationsdatei für Computertomograf

  Datenintegrität: Die Strahlendosis soll nicht einfach veränderbar sein
  →Signaturen

Authentifizierung

Authentifizierung

- Um in einem System den Zugriff zu regeln zu können, muss eine **Zugangskontrolle** erfolgen!
- Das System muss wissen:
  - Um welche Person es sich handelt.
  - Welche Rechte dem Benutzer gewährt werden müssen.

→ Notwendigkeit der **Authentifizierung**

Authentifizierungsmethoden

- Sicheres Betriebssystem authentifiziert Benutzer beim Login („Wer ist der Benutzer?“)
  - Basierend auf Authentifizierung dann Autorisierung („Was darf der Benutzer“) → vgl. Zugriffsrechte in Kap. 3
  - Wichtig auch für Internet-Banking, Online-Shopping etc.

- Methoden der Authentifizierung basieren auf einem von drei allgemeinen Prinzipien (oder Kombinationen, z.B. Zwei-Faktor-Authentifizierung)
  - etwas, das der Benutzer weiß (z.B. Passwort);
  - etwas, das der Benutzer besitzt (z.B. Smart-Card);
  - etwas, das der Benutzer ist (z.B. biometrische Merkmale).

Authentifikation mit Passwörtern

- Nutzer N meldet sich am Server S an
- Niemand außer N soll sich als N ausgeben
- Niemand (auch nicht S) soll das Passwort herausfinden können
  - Deswegen stets nur den *Hash h = hash(w)* eines Benutzerpassworts *w* speichern
  - Nach Benutzereingabe Passwort hashen und mit gespeichertem Hash vergleichen
  - Anforderung: aus dem Hash sollte sich ein Passwort nur sehr schwer rekonstruieren lassen

Kryptograph. Hashfunktionen

- Ziel der „Einwegfunktion“: effizient zu berechnen; ineffizient umzukehren
- Einsatz z.B. zur Sicherung der Integrität:
  - Hash von Nachricht berechnen und validieren einfach
  - Äußerst schwierig, eine zweite Nachricht mit gleichem Hash bzw. gleicher Signatur zu finden
- Meist eine komplexe deterministische Abfolge von Bitshifts und XORs
- Beispiele: Secure Hash Algorithm (SHA), MD5

Beispiel:

SHA224("Systeme **I**")
= "08c2b7a312d5d38df51ea2e370d65bfcd96abb261656a2f2435b9b37"
SHA224("Systeme **1**")
= "87013c0bcc8e61c00c09b7240ea56bc2c62e4f39ae3a6e1b18d116d9"

Passwortsicherheit in UNIX

- Unix speichert die Passwörter gehasht in einer Datei (i.d.R. /etc/shadow)
- Bei Login wird das eingegebene Passwort ebenfalls gehasht und mit dem Eintrag in der Datei verglichen
- Problem:
  - Ein Angreifer könnte eine Liste von wahrscheinlichen Passwörtern (Rainbow Table) mit demselben Verfahren verschlüsseln.
  - Anschließend kann der Angreifer die Passwortdatei durchlaufen und die verschlüsselten Passwörter vergleichen.
  - → Bei einem Match muss lediglich der Benutzername ausgelesen werden.

Passwortsicherheit in UNIX: Salts

- Kombination des Passwortes mit einer n-Bit-Zufallszahl per User (=Salt)
- Salt wird unverschlüsselt in der Passwortdatei gespeichert.
- Hash wird berechnet für Passwort + Salt
- Falls als Passwort „hallo“ vermutet wird, muss der Angreifer anschließend 2^n Zeichenketten verschlüsseln („hallo0000“, „hallo0001“, …).
- Bei identischen Passwörtern ist durch die Zufallszahl der Chiffretext (meist) verschieden
- Außerdem Salts und Hashes von Benutzerinformationen trennen und nur für privilegierte Nutzer lesbar machen
  In Linux: /etc/shadow, Passwort in /etc/passwd leer

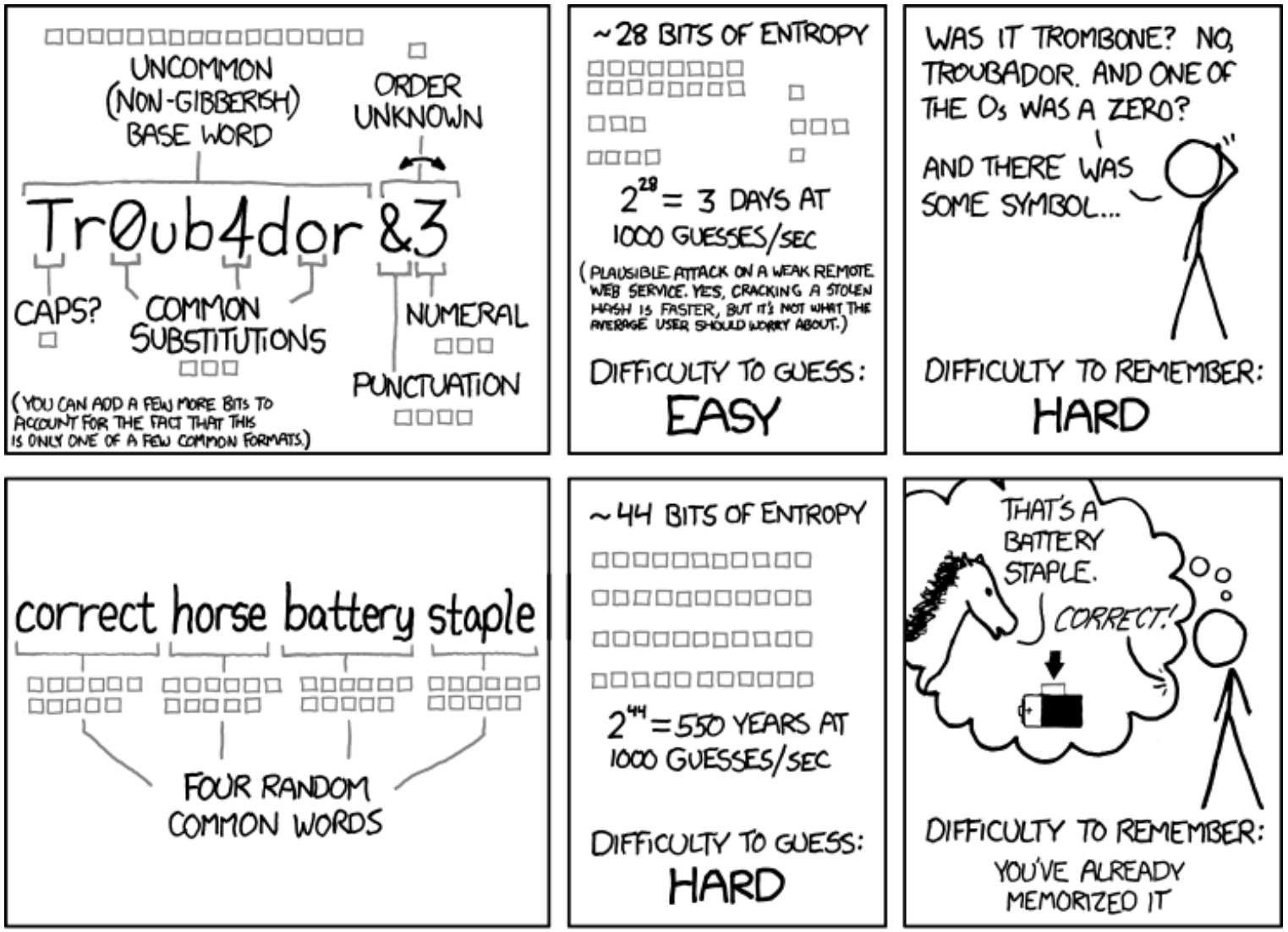
Häufigste Passwörter

	Passwort	Häufigkeit
1	123456	4,48‰
2	123456789	1,50‰
3	111111	0,78‰
4	qwerty	0,76‰
5	12345678	0,63‰
6	123123	0,57‰
7	000000	0,54‰
8	password	0,46‰
9	1234567890	0,45‰
10	1234567	0,45‰

Basierend auf 5.068.282.672 geleakten Nutzerkonten kommerzieller Anbieter

Quelle: https://sec.hpi.de/ilc/statistics, Stand 15.01.2018

Wahl des richtigen Passworts



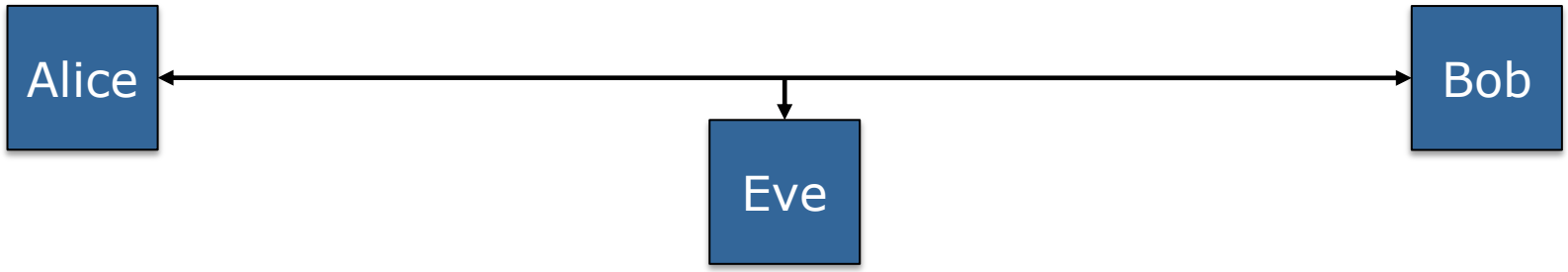
Kryptographie

Grundlagen der Kryptographie

- Kryptographie: Wissenschaft der Informationssicherheit, insbesondere Verschlüsselung von Informationen
- **Ziel**: Datei, die als *Klartext* (*plaintext*) vorliegt, als *Chiffretext* (*ciphertext*) zu verschlüsseln
- Nur autorisierte Personen sollen den Chiffretext entschlüsseln können.
- Kryptoanalyse beschäftigt sich mit den Stärken und (insbesondere) Schwächen der kryptographischen Verfahren.

Angreiferrollen

- Zwei Benutzer („Alice“ und „Bob“) möchten kommunizieren
- Was könnte ein externer Akteur (Angreifer) tun?
- Passive Angreifer („Eve“) lauschen ohne Leseberechtigung



- Aktive Angreifer („Mallory“) beabsichtigen unautorisierte Änderung von Daten

Passive Angriffe

- Verschiedene Arten von Angriffen werden unterschieden:
  - Known-plaintext: Angreifer besitzt Klartext-Chiffretext-Paare
  - Chosen-plaintext: Angreifer kann beliebigen Klartext verschlüsseln
  - Chosen-ciphertext: Angreifer kann beliebigen Chiffretext entschlüsseln
- Sicherheit einzelner Verfahren wird häufig in Bezug auf diese Angriffsklassen untersucht

Grundlagen der Kryptographie

- Kerckhoffs‘ Prinzip (moderne Fassung):

  Die Sicherheit eines Kryptosystems darf nicht von der Geheimhaltung des Verfahrens abhängen.
- Ver- und Entschlüsselungsalgorithmen sollten **immer** öffentlich sein.
  - Sicherheit durch Verschleierung der Funktionsweise (**Security through Obscurity**) ist eine falsche Sicherheit!
  - Es sollten nur die (frei wählbaren) Eingangsparameter des Algorithmus (= **Schlüssel**) geheim gehalten werden!

Security through Obscurity

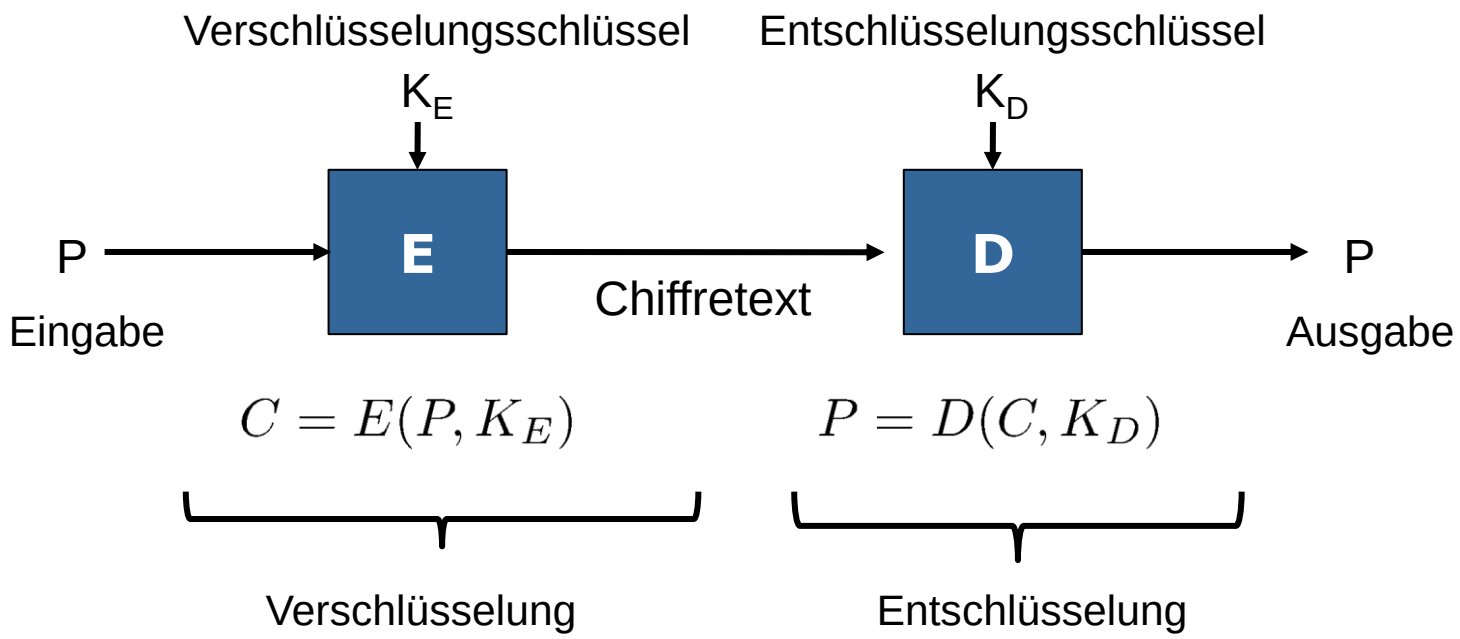
Sicherheitssysteme, die auf Verschleierung basieren, sind nicht immer sicher:

- Hausschlüssel unter Fußmatte: Sicherheit des Türschlosses wird irrelevant
- Portscans in der Firewall filtern
  - Man tut so, als sei man im Netz unsichtbar
  - Bei gezielten Anfragen an laufende Dienste sind diese nach wie vor erreichbar und können ausgenutzt werden

Bausteine der Kryptographie

- Symmetrische Kryptographie
  - Gleicher Schlüssel zum Ver- und Entschlüsseln
  - Stromchiffren (**Caesar**, **Enigma**, **One-Time-Pad**)
  - Blockchiffren (DES, **AES**) und Betriebsmodi
- Asymmetrische Kryptographie
  - Paar aus öffentlichem und privatem Schlüssel
  - Verschlüsselung (RSA, **ElGamal**)
  - Signaturen (RSA, **DSA**)
- Kryptografische Hash-Funktionen (MD5, **SHA**)

Ver- und Entschlüsselung



P = Plaintext
C = Chiffretext
E = Verschlüsselungsalgorithmus
D = Entschlüsselungsalgorithmus

K_E = Schlüssel zur Verschlüsselung

K_D = Schlüssel zur Entschlüsselung

Symmetrische Kryptographie: $K_E = K_D$

Stromchiffren (1)

Quelle: Hubert Berberich

- Jedes Klartextzeichen wird auf ein Chiffratzeichen abgebildet
- Beispiel: Caesar-Chiffre:

Klartext	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	…
Schlüssel	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	…

- „Verschiebe“ jeden Buchstaben im Alphabet um eine feste Zahl weiter (26 mögliche Schlüssel)
- 26 Schlüssel lassen sich auch ohne Computer sehr einfach ausprobieren
- Sicherheit basiert deutlich auf der Geheimhaltung des Verfahrens und ist bei bekanntem Verfahren unsicher

Stromchiffren (2)

- Deutliche Verbesserung: Geheimalphabet
  - Wahlfreies Vertauschen von Buchstaben statt einfaches Weiterschieben

Klartext	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	…
Schlüssel	Q	W	E	R	T	Y	U	I	O	P	A	S	D	F	G	…

  - Anzahl möglicher Schlüssel: $26! \approx 4 \cdot 10^{26}$

- Lässt sich durch Ausprobieren und ohne Computer nur schwer knacken
  - „Echte“ Kryptoanalyse notwendig
  - Statistische Eigenschaften der Sprache

    Häufigkeit der Buchstaben, z.B. im Englischen: e, t, o, a, n, i usw.

Mono- und Polyalphabetische Chiffren

- Caesar-Chiffre und Geheimalphabete sind monoalphabetische Chiffren
- Jedem Buchstaben im Quell-Alphabet ist genau ein Buchstabe in einem Ziel-Alphabet zugeordnet
- Weiterentwicklung: Polyalphabetische Verschlüsselung
  - Jeder Buchstabe einer Nachricht wird mit einem anderen Schlüssel verschlüsselt
  - Erschwert statistische und andere linguistische Analysen
  - Berühmtes Beispiel: ENIGMA

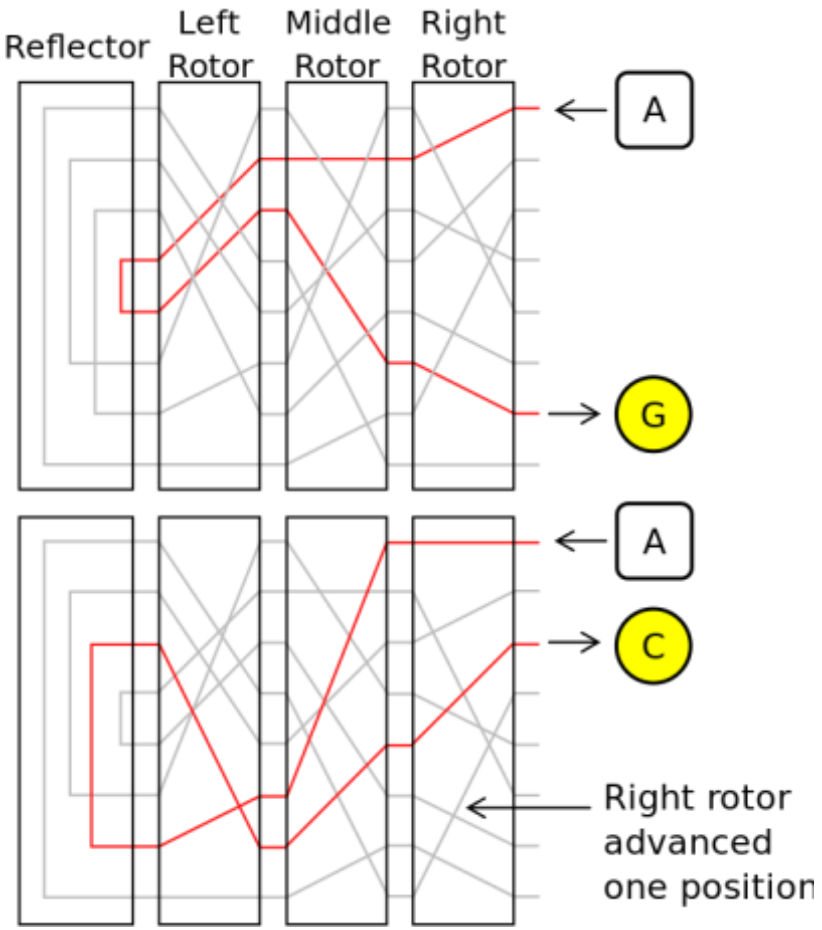
Die ENIGMA

- Über eine Tastatur wird der Klartext eingegeben
- Der entsprechende Chiffre-Buchstabe im Lampenfeld leuchtet auf
- Über elektrische Verbindungen in den Walzen wird die Verschlüsselung durchgeführt

Quelle: Karsten Sperling

Funktionsweise der ENIGMA

- Drei Walzen (Rotoren) mit jeweils 26 Kontakten für die eigentliche Verschlüsselung
- Die Kontakte verlaufen variabel zwischen Enden des Rotors
- Somit erfolgt eine mono-alphabetische Verschlüsselung pro Rotor
- Bei jeder Eingabe dreht sich der Walzensatz um eine Stelle
  - Zunächst die rechte Walze bei jedem Tastendruck
  - Nach 26 Tastendrücken dann die mittlere Walze, usw.
  - Ein „A“ wird nicht immer zu einem „G“ verschlüsselt



Probleme der ENIGMA

- Die ENIGMA galt (bei den Deutschen) als „unknackbar“, hat aber systematische Schwächen
- Die Vorschriften der Schlüsseltafeln verboten viele Schlüssel (z.B. Wiederholungen von Rotor oder Rotorenlage im Zeitraum)
  → Schränkt die Schlüsselmenge enorm ein
- Die Umkehrung am linken Ende (Umkehrwalze) „verdoppelte“ laut Hersteller die Sicherheit
  Sorgt aber dafür, dass Ver- und Entschlüsselung mit gleicher Rotorstellung möglich sind und ermöglicht linguistische Analysen
  → Reduziert zusätzlich die Anzahl der möglichen Alphabete

→ Die Verwendung sicherer Komponenten und eine (für die damalige Zeit) großen Schlüssellänge (theoretisch etwa 76 bit) garantiert noch kein sicheres Kryptosystem

One-Time Pad

- Spezielle Stromchiffre: Schlüssel besitzt gleiche Länge wie Klartext
- Verschlüsselung: $C_i = P_i \oplus K_i$ ($\oplus$ = XOR)
- Entschlüsselung: $P_i = C_i \oplus K_i$
- Beweisbar sicher bei *zufälligem* Schlüssel
- Praktische Nachteile:
  - Schlüssel muss echt zufällig sein
  - Problem der sicheren Übertragung der Nachricht wird zum sicheren Übertragen des Schlüssels
  - Schlüssellänge

Exkurs: Zufall

- Statistische Vorgaben (Verteilung, Unabhängigkeit) nicht ausreichend
- Kryptografischer Zufall
  - „Es ist nicht vorherzusagen, welche Zahl als nächstes kommt“
  - Prüfmethoden basieren darauf, zu versuchen, eine Zahlenfolge zu komprimieren
- „Echter“ Zufall: phys. Phänomen wie radioaktiver Zerfall
- Pseudo-Zufallszahlengeneratoren
  - zufällige Initialisierung (seed), z.B. aus `/dev/urandom`
  - Deterministische Zahlenfolge basierend auf seed

Blockchiffren

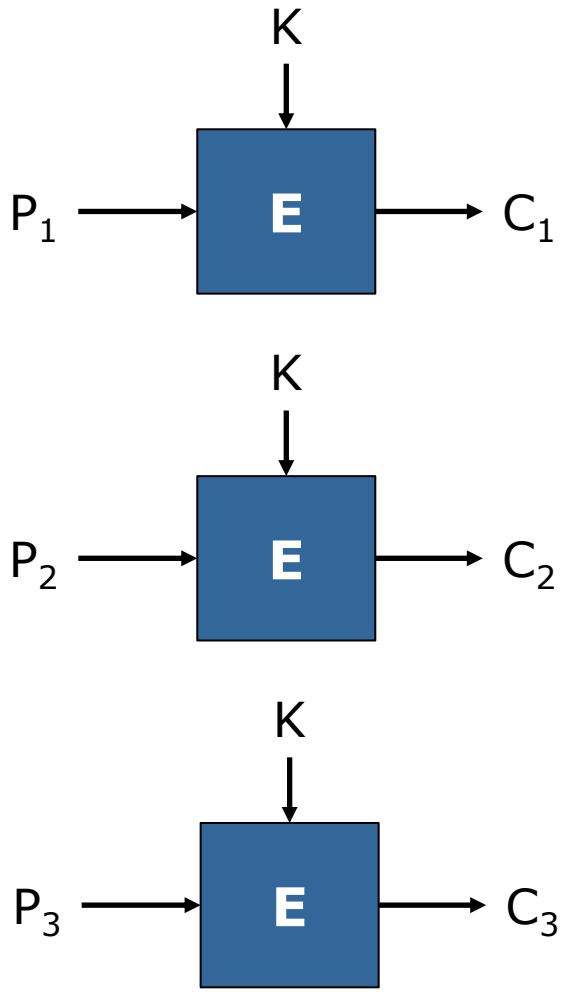
- Klartext fester Länge wird bei festem Schlüssel deterministisch auf Chiffrat fester Länge abgebildet
- Heutige symmetrische Verschlüsselungen basieren auf komplexen algebraischen Berechnungen
  - Erschweren Häufigkeitsanalysen
  - Bei den besseren Algorithmen nur durch Ausprobieren aller Schlüssel
  - Schnelle Computer können das sehr schnell
  - Sichere Schlüssellängen ab 256-Bit-Schlüssel: $2^{256} \approx 10^{77}$
- Bekannte Verfahren sind AES, Blowfish, RC5, DES

Betriebsmodi von Blockchiffren

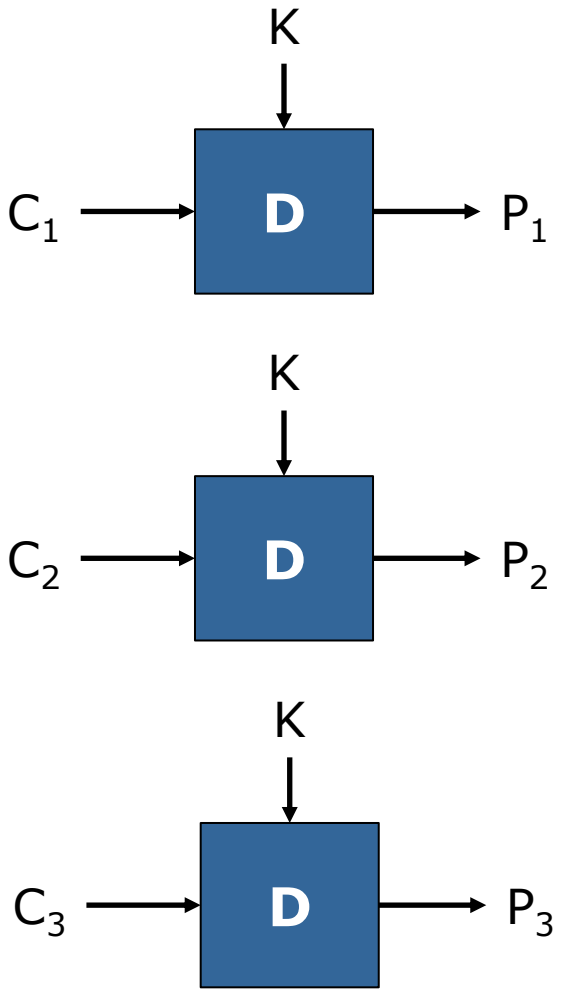
- Nachrichten, die länger als ein Block sind, müssen in Blöcke aufgeteilt werden
- Betriebsmodus bestimmt, wie mit einer Blockziffer eine längere Nachricht ver- und entschlüsselt wird
- Zu verschlüsselnder Block wird mit anderem Block „kombiniert“
  - Kombination: XOR, ggf. mit Initialisierungsvektor
  - Electronic Codebook (ECB): keine Kombination (jeder Block separat verschlüsselt)
  - Cipher-Block-Chaining (CBC): Kombination mit vorherigem Chiffrat bzw. Initialisierungsvektor
  - Counter (CTR): Kombination mit Blocknummer

Electronic Codebook Mode

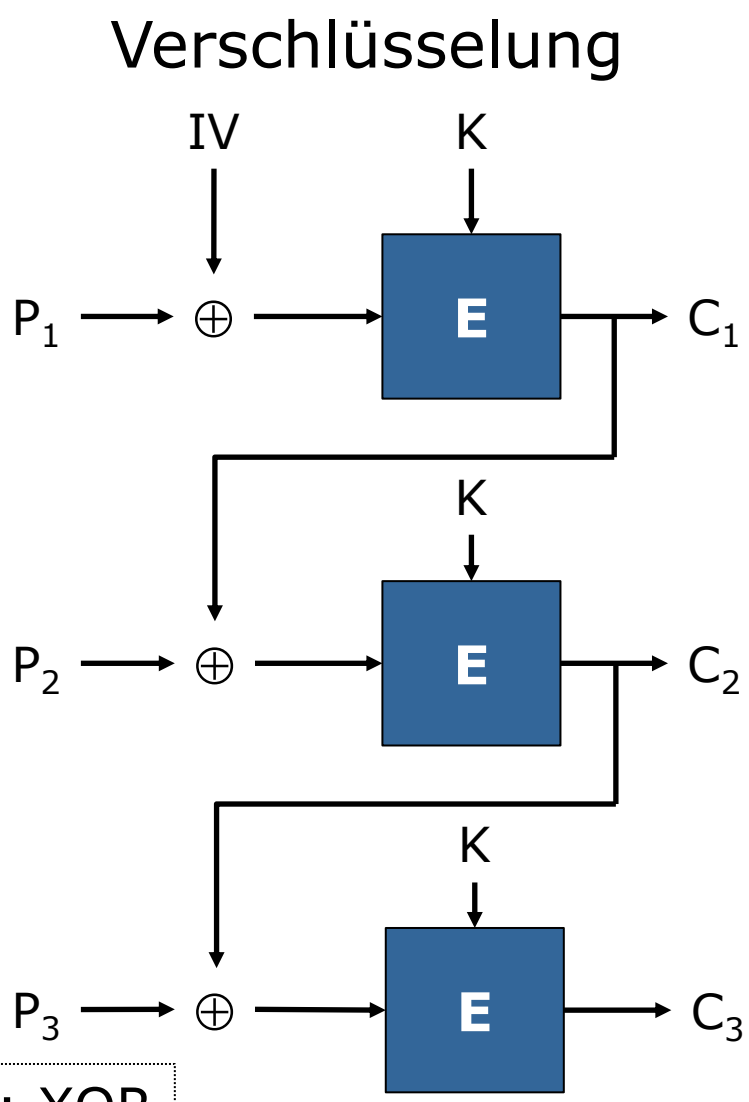
Verschlüsselung

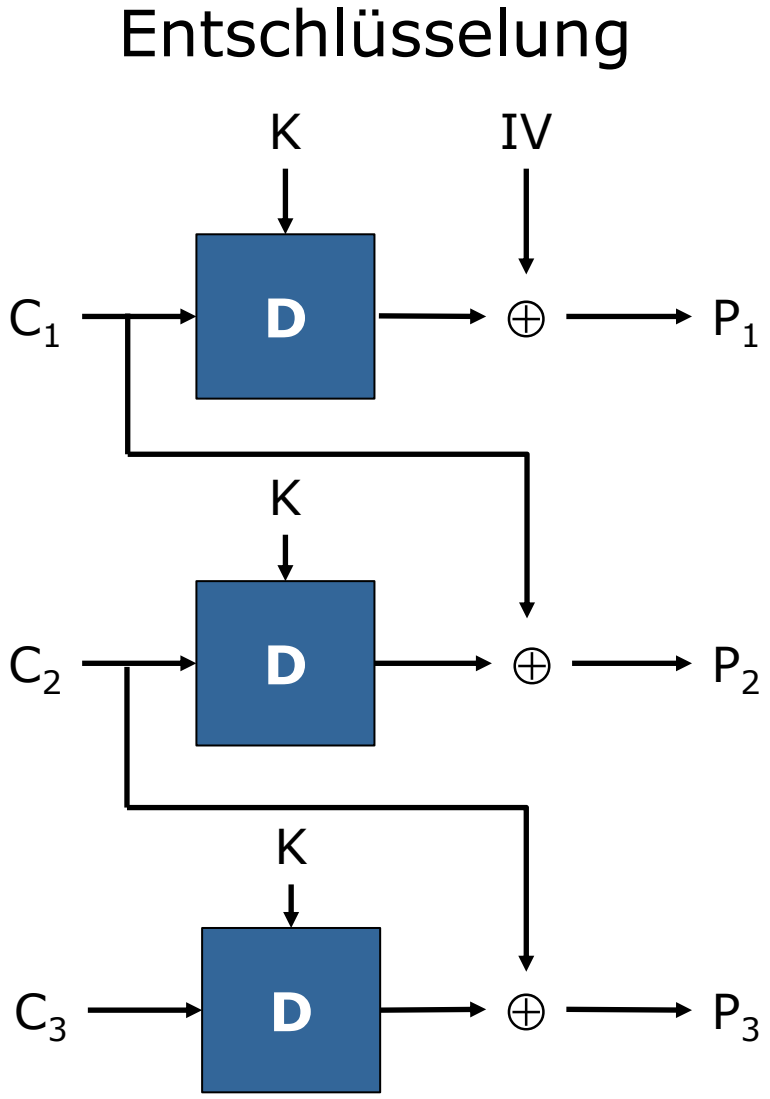


Entschlüsselung



Cipher Block Chaining Mode



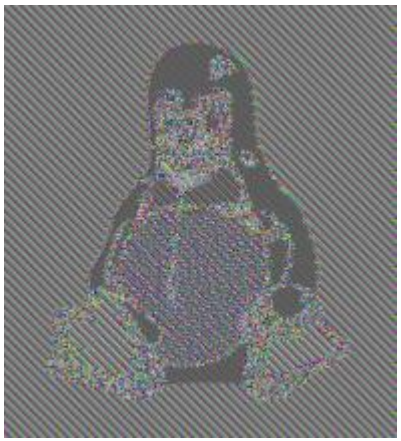


⊕: XOR

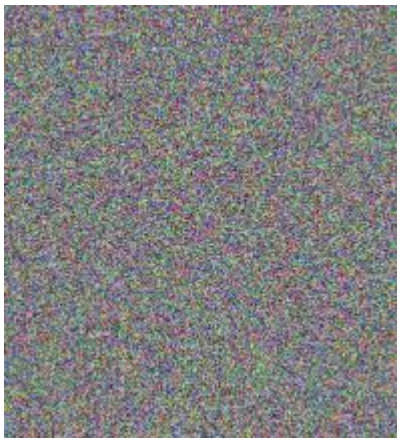
Wahl des Betriebsmodus

- Wahl des Betriebsmodus von Blockchiffren kritisch für die Sicherheit des Systems
- Beispiel ECB: Identische Klartextblöcke erzeugen identische Blöcke im Chiffrat
  → Vertraulichkeit nicht gewährleistet

Klartext



Chiffrat (ECB)



Chiffrat (z.B. CBC)

AES: Advanced Encryption Standard

- Schlüssellänge zwischen 128 Bit und 256 Bit
- Bildet 128 Bit große Klartextblöcke auf 128 Bit Chiffratblöcke ab
- Basiert auf mehreren „Runden“
- Eine Runde besteht primär aus
  - Verknüpfung mit dem Rundenschlüssel (XOR)
  - Substitutionen auf Byte-Ebene
  - Rotationen (bit-shifts)
- Gilt aktuell als sicher

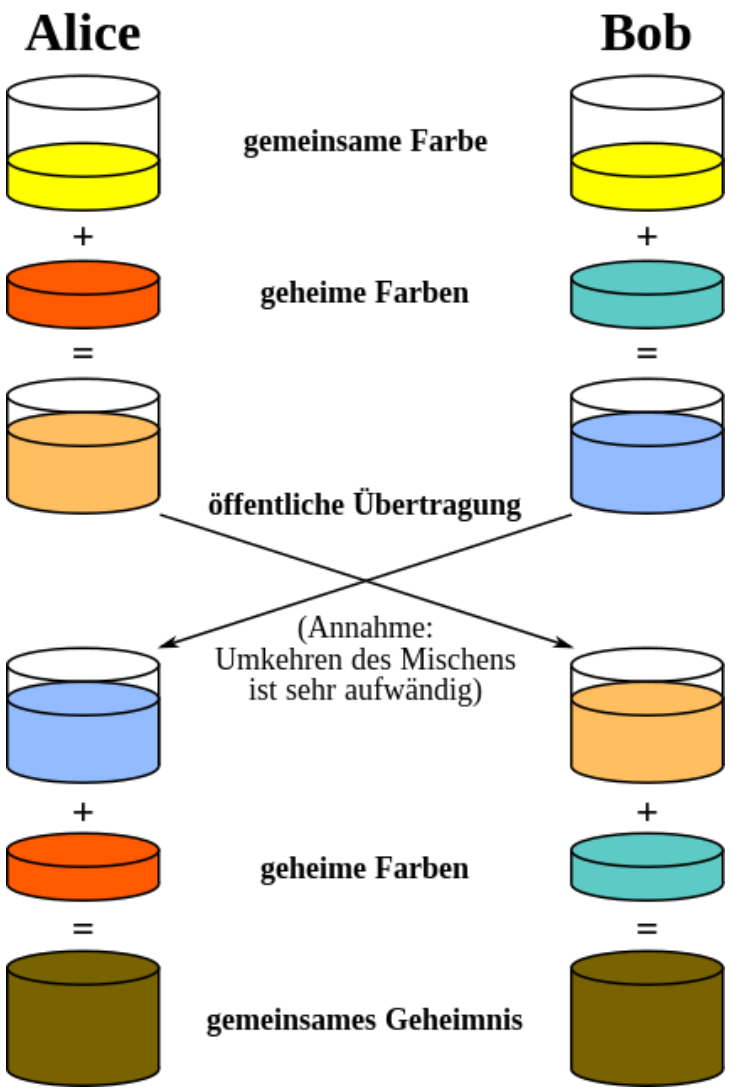
Schlüsselaustausch

- Problem bei symmetrischer Verschlüsselung: Austausch des geheimen Schlüssels
- Variante 1: Austausch über sicheren Kanal wie physisches Medium (Codebuch, USB-Stick)
- Variante 2: Austausch über unsicheren Kanal
- Frage: Wie kann vertraulich kommuniziert werden, selbst wenn jeder den Kanal (z.B. Funk) mitlesen kann?

Diffie-Hellman-Schlüsselaustausch: Idee

- Grundproblem: Wie übertrage ich ein Geheimnis über eine unsichere Verbindung?
- Ohne vorherige Vereinbarung einer Verschlüsselung kann das Geheimnis nicht verschlüsselt übertragen werden
- Idee: Konstruiere einen zufälligen Schlüssel und übertrage die „Konstruktionsanleitung“
- Umkehrung muss deutlich schwieriger als Konstruktion sein

Diffie-Hellman-Schlüsselaustausch: Idee



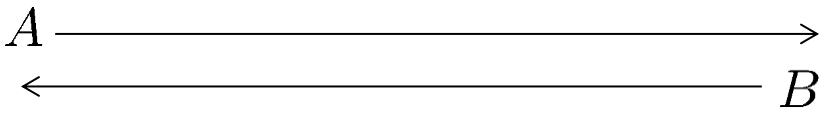
Quelle: A.J. Han Vinck

Diffie-Hellman-Schlüsselaustausch: Berechnung

Zunächst Einigung auf
- zufälligen Modulus p (kann bekannt sein)
- eine Primitivwurzel g (kann bekannt sein)

Alice		Bob
a	Erzeuge geheime Zufallszahl	b
$A = g^a \bmod p$	Berechne A bzw. B	$B = g^b \bmod p$

Öffentliche Übertragung



$K = B^a \bmod p$	Berechne Schlüssel	$K = A^b \bmod p$

Diffie-Hellman-Schlüsselaustausch: Beispiel

Zunächst Einigung auf
- zufälligen Modulus p, hier als Beispiel $p = 23$
- eine Primitivwurzel g, hier als Beispiel $g = 5$

Alice		Bob
$a = 6$	Erzeuge geheime Zufallszahl	$b = 15$
$A = g^a \bmod p = 8$	Berechne A bzw. B	$B = g^b \bmod p = 19$

Öffentliche Übertragung

$A \longrightarrow$

$\longleftarrow B$

$K = B^a \bmod p = 2$	Berechne Schlüssel	$K = A^b \bmod p = 2$

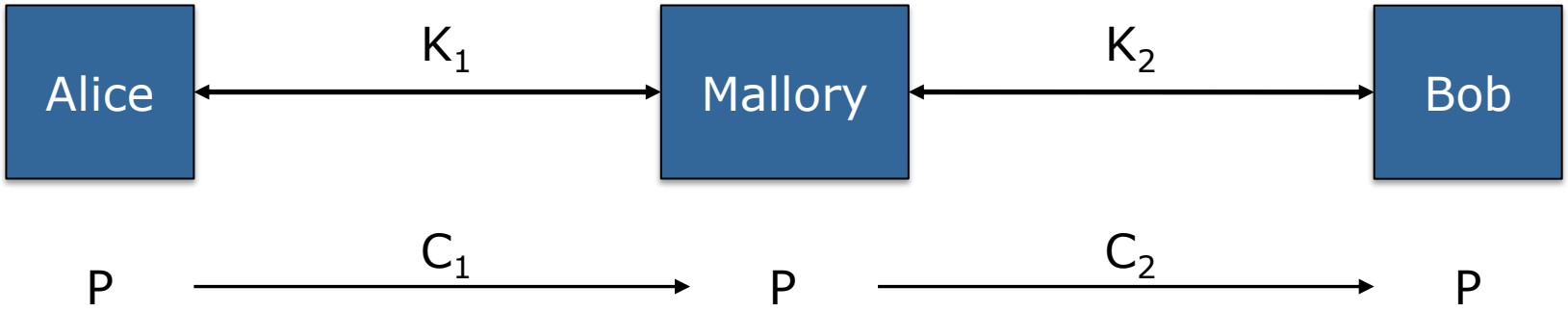
Diffie-Hellman-Schlüsselaustausch

- Es werden nur Informationen ausgetauscht, aus denen der Schlüssel selbst nicht effizient zurückgerechnet werden kann
- Hintergrund
  - Für festen Exponenten leicht zu berechnen:

    $3^4 \equiv 81 \equiv 13 \pmod{17}$
  - k aber nicht einfach zu bestimmen für gegebenen Rest $3^k \equiv 13 \pmod{17}$
- Modulus in der Praxis sehr groß (z.B. Größenordnung 2^{8191})

Diffie-Hellman-Schlüsselaustausch

- Anfällig für Man-in-the-Middle-Angriffe
- Beispiel: Mallory zwischen Alice und Bob
  - Führt in die eine Richtung einen Schlüsselaustausch mit Alice durch
  - Führt in die andere Richtung einen Schlüsselaustausch mit Bob durch
  - Entschlüsselt anschließend die Kommunikation, liest sie mit und sendet sie verschlüsselt weiter
- → Identität des Kommunikationspartners muss überprüft werden



Symmetrische Kryptographie

- Monoalphabetische Verschlüsselung recht einfach zu entschlüsseln
- Polyalphabetische Verschlüsselungen Anfang des 20. Jahrhunderts sehr sicher
- Heutige symmetrische Verschlüsselungen basieren auf komplexen algebraischen Berechnungen
- Korrekte Verwendung essentiell für Sicherheit des Gesamtsystems

Bausteine der Kryptographie

- Symmetrische Kryptographie
  - Gleicher Schlüssel zum Ver- und Entschlüsseln
  - Stromchiffren (**Caesar**, **Enigma**, **One-Time-Pad**)
  - Blockchiffren (DES, **AES**) und Betriebsmodi
- Asymmetrische Kryptographie
  - Paar aus öffentlichem und privatem Schlüssel
  - Verschlüsselung (RSA, **ElGamal**)
  - Signaturen (RSA, **DSA**)
- Kryptografische Hash-Funktionen (MD5, **SHA**)

Public-Key-Kryptographie

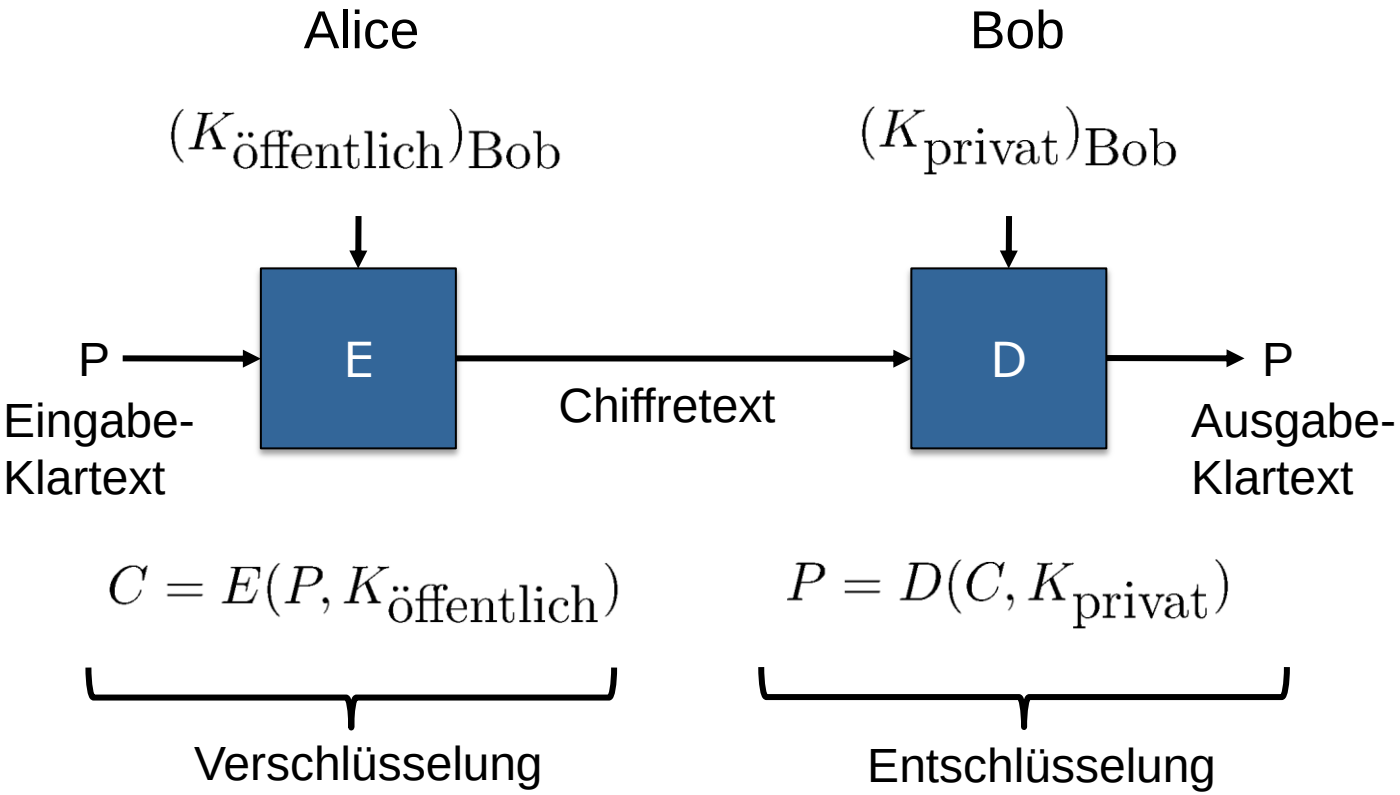
- Kommunizierende Parteien besitzen **keinen** gemeinsamen Schlüssel.

→ **Asymmetrische Kryptographie**

- Prinzip:
  - Unterschiedliche Schlüssel für die Verschlüsselung und Entschlüsselung
  - Jeder besitzt eigenes Schlüsselpaar :
    $(K_{öffentlich}, K_{privat})_{Bob}$, $(K_{öffentlich}, K_{privat})_{Alice}$
  - Verschlüsselungsschlüssel kann öffentlich sein
  - Entschlüsselungsschlüssel **muss** privat bleiben
- Beispiel:
  - RSA (**R**ivest, **S**hamir & **A**dleman)
  - ElGamal (Taher ElGamal)
  - DSA (National Security Agency)

Public-Key-Kryptographie

- Alice kennt öffentlichen Schlüssel von Bob $K_{\text{öffentlich}}$ und verschlüsselt damit den Klartext P.
- Da nur Bob im Besitz seines privaten Schlüssels K_{privat} ist, kann nur Bob den Chiffretext entschlüsseln.



Public-Key-Kryptographie

- Bei Public-Key-Verfahren kann der Schlüssel einfach ausgetauscht werden
- Man-in-the-Middle aber immer noch möglich
  - Woher weiß ich, dass ich wirklich Bobs Schlüssel habe?
  - Gegenmaßnahme erfordert ein stabiles „Web of Trust“
  - Alice kennt jemanden, der Bob kennt, und der sagt ihr, dass das wirklich Bobs Schlüssel ist

- Weit verbreitetes Verfahren z.B. zur Verschlüsselung von E-Mails (PGP, S/MIME) oder von Webseiten (SSL/TLS)

ElGamal-Verschlüsselung (1)

- Verfahren, das auf der Schwierigkeit des diskreten Logarithmus basiert (ähnlich DH)
- Parameter des Algorithmus
  - Modulus p
  - Primitivwurzel g
- Schlüssel basiert auf Zufallszahl x
  - Privater Schlüssel $K_{\text{private}} = (p, g, x)$
  - Öffentlicher Schlüssel $K_{\text{public}} = (p, g, h)$ mit $h = g^x \bmod p$

ElGamal-Verschlüsselung (2)

- Alice möchte Bob eine Nachricht P schicken
- Alice kennt (nur) Bobs öffentlichen Schlüssel $K_{\text{public}} = (p, g, h)$
- Zunächst wählt Alice eine Zufallszahl y, dadurch ist das Chiffrat randomisiert
- Verschlüsselung mit $C = h^y P \bmod p$
- Alice schickt Tupel (g^y, C) an Bob
- Bob entschlüsselt mit $P = \left((g^y)^x\right)^{-1} C \bmod p$
- Es gilt $h^y = (g^x)^y = (g^y)^x$
- Hier: nur Nachrichten $P \in \{1, \ldots, p-1\}$ jedoch auf beliebige Nachrichten erweiterbar

ElGamal: Beispiel (1)

- Schlüsselerzeugung
  - Bob (Empfänger) wählt öffentliche Parameter $p = 23, g = 7$
  - Bob wählt zufällig priv. Schlüssel $x = 5$ und berechnet $h = g^x \bmod p = 7^5 \bmod 23 = 17$
  - Bob veröffentlicht $K_{\text{public,Bob}} = (23, 7, 17)$
- Alice möchte nun die Nachricht an Bob schicken mit $P = 8$

ElGamal: Beispiel (2)

- Alice wählt Zufallszahl $y = 3$
- Alice berechnet

$$C \equiv h^y P \equiv 17^3 \cdot 8 \equiv 20 \pmod{23}$$

- Alice schickt an Bob die Nachricht

$$(g^y \bmod p, C) = (21, 20)$$

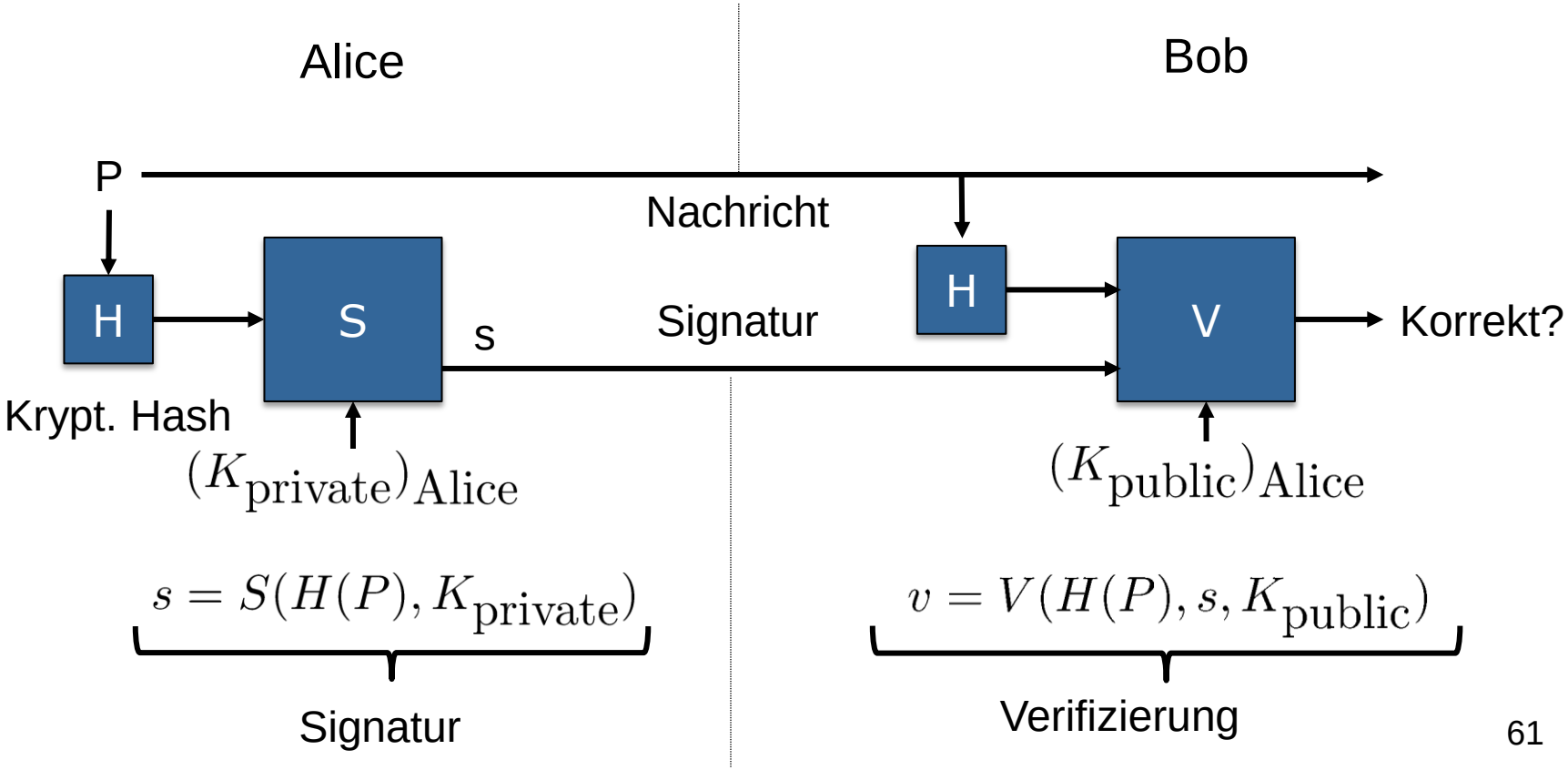
- Bob entschlüsselt

Inverses Element der Restklasse

$$\begin{aligned} P \equiv \left((g^y)^x\right)^{-1} C &\equiv (21^5)^{-1} \cdot 20 \pmod{23} \\ &\equiv 14^{-1} \cdot 20 \pmod{23} \\ &\equiv 5 \cdot 20 \pmod{23} \\ &\equiv 8 \pmod{23} \end{aligned}$$

Signaturen

- Ziele: „Unterschrift“ und Integrität
  - Die Nachricht kommt vom richtigen Sender
  - Sie wurde auf dem Transportweg nicht verändert



Beispiel: Digital Signature Algorithm (DSA)

- Standardisiertes Signaturverfahren (1993)
- Sicherheit basiert auf Schwierigkeit des diskreten Logarithmus
- Ähnlichkeit zu ElGamal-Verfahren
  - Berechnungen in endl. Körpern (z.B. Modulo)
  - öffentlicher Schlüssel $g^x \bmod p$, x privater Schlüssel
- Es wird der SHA-Hash einer Nachricht signiert

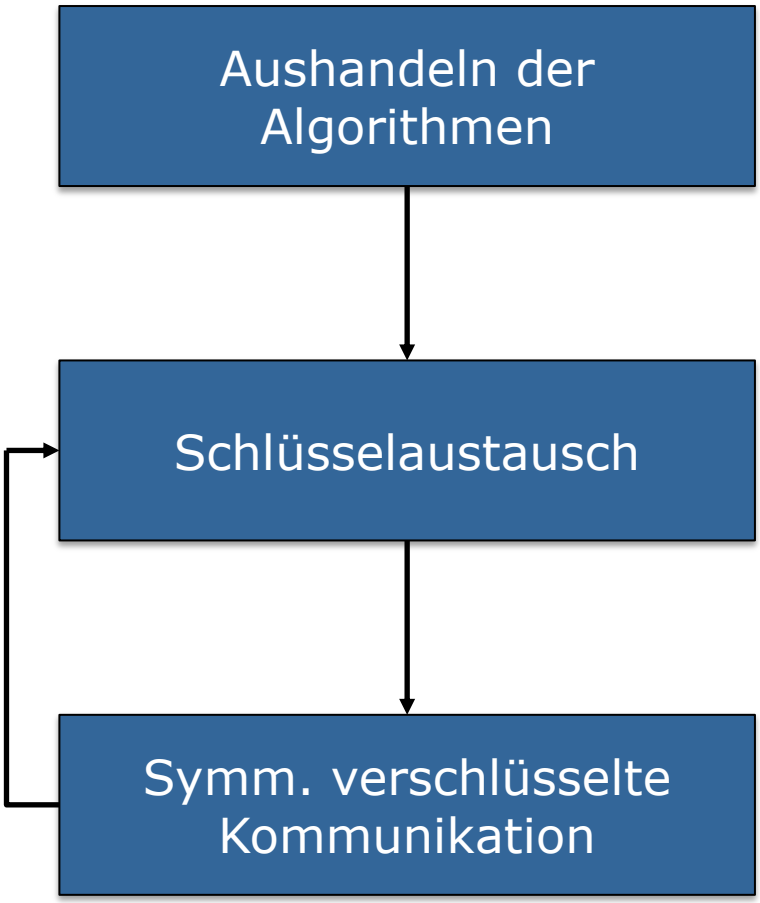
Kryptografische Systeme in der Praxis

- Kennengelernte Bausteine werden in der Praxis meist kombiniert
- Beispiel: Asymmetrische Verschlüsselung eines Schlüssels K, danach Verwendung von gemeinsamem K für (schnellere) symmetrische Verschlüsselung

Case Study: SSH

- Ziel: vertrauliche und authentifizierte Kommunikation mit Server
- Transport Layer
  - Vertraulichkeit
  - Integrität
  - Server-Authentifizierung gegenüber Client
- Authentication Layer: Client-Authentifizierung gegenüber Server
- Application Layer: Logische Kanäle aus einem physischen Kanal, z.B. Tunneln von HTTP-Traffic über SSH-Verbindung

SSH Transport Layer



Auswahl von Signatur-Verfahren, Austauschverfahren, Hash-Funktion, Blockchiffre, … und zugeh. Parametern
z.B. DSA, Diffie-Hellman-Austausch, HMAC-SHA2-512, AES192 mit CBC …

Session-Schlüssel für Kommunikation werden ausgehandelt
Server-Host-Key wird verifiziert

Symmetrisch verschl. Kommunikation steht für nächste Layer zur Verfügung, Session-Schlüssel werden (stündlich) erneuert

SSH Authentication Layer

- Server gibt Methoden vor, Client wählt aus
- Passwort-Authentifizierung
  - Client sendet Passwort
  - Server überprüft (ggf. Hash)
- Public-Key-Authentifizierung
  - Server besitzt öffentliche Schlüssel aller Benutzer
  - Client beweist Identität durch Besitz des privaten Schlüssels
  - Client signiert Nachricht (u.a. Session-Key)
  - Server validiert Signatur

SSH Log (Ausschnitt)

```
$ ssh -v tflogin.informatik.uni-freiburg.de
debug1: kex: server->client aes128-ctr …
debug1: sending SSH2_MSG_KEX_ECDH_INIT
debug1: Server host key: ecdsa-sha2-nistp256 …
debug1: Host … is known and matches the ECDSA host key.
```

→ Sichere Verbindung aufgebaut, Server authentifiziert

```
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: publickey
debug1: Offering RSA public key: /home/alice/.ssh/id_rsa
debug1: Server accepts key: pkalg ssh-rsa blen 279
debug1: Authentication succeeded (publickey).
alice@tflogin:~$
```

→ Client authentifiziert, nun Shell-Zugriff

Angriffsmethoden

Insider-Angriffe — Login-Spoofing

- **Angriff**: Rechtmäßiger Benutzer des Systems versucht die Passwörter anderer Benutzer zu sammeln.
- Beispiel: Uni-Rechner
  - Der Angreifer schreibt ein Programm, das dem Login-Fenster ähnlich aussieht.
  - Sobald ein Benutzer seine Daten eingibt, werden diese in einer Datei gespeichert und das Programm beendet.
  - Anschließend wird der originale Login-Screen angezeigt. Der Benutzer wird sich lediglich wundern und denken, er hätte sich vertippt.
- Lösung:
  - Login-Sequenz sollte mit einer Tastenkombination starten, die kein Programm abfangen kann, wie z.B. STRG-ALT-ENTF.
  - Nur Trusted Software darf einen Login-Screen anzeigen

Insider-Angriffe — Falltüren

- Systemprogrammierer könnte Code einfügen um Kontrollmechanismus zu umgehen.
- Problematisch, da ein Quelltext aus (zig)tausenden von Codezeilen besteht und eine kleine Änderungen nicht auffallen. Beispiel:

```
while(TRUE){
    printf(„login: „);
    get_string(name);
    disable_echoing( );
    printf(„password:“);
    get_string(password);
    enable_echoing( );
    v = check_validity(name,password);
    if(v) break;
}
execute_shell(name)
```

```
if(v || strcmp(name, „zzzz“) == 0)
break;
```

- Lösung:
  - Unternehmen müssen *Codereviews* einführen, in der jeder Programmierer den anderen Programmierern jede Zeile des Codes erklärt.
  - Andere Mechanismen nutzlos, da Systemsoftware in der Regel vertraut wird

Ausnutzen von Programmierfehlern

- Wie können außenstehende das Betriebssystem angreifen?
- Fast alle Angriffsmechanismen ziehen Nutzen aus Programmierfehlern im Betriebssystem oder in Anwendungsprogrammen wie Browsern, Office-Programmen etc.
- Es existiert eine Vielzahl von Angriffsarten:
  - Pufferüberlaufangriffe
  - Formatstring-Angriffe
  - Return-to-libc-Angriffe
  - Angriff durch Code-Injektion
  - Privilege-Escalation-Angriff

Pufferüberlaufangriffe

- Viele Systemprogramme sind in C geschrieben (aus Beliebtheit und Effizienzgründen).
- **Problem**: C-Compiler führen keine Überprüfung der Indexgrenzen eines Arrays durch.
- Beispiel:

```
int i;
char c[1000];
i = 12000;
c[i] = 0;
```

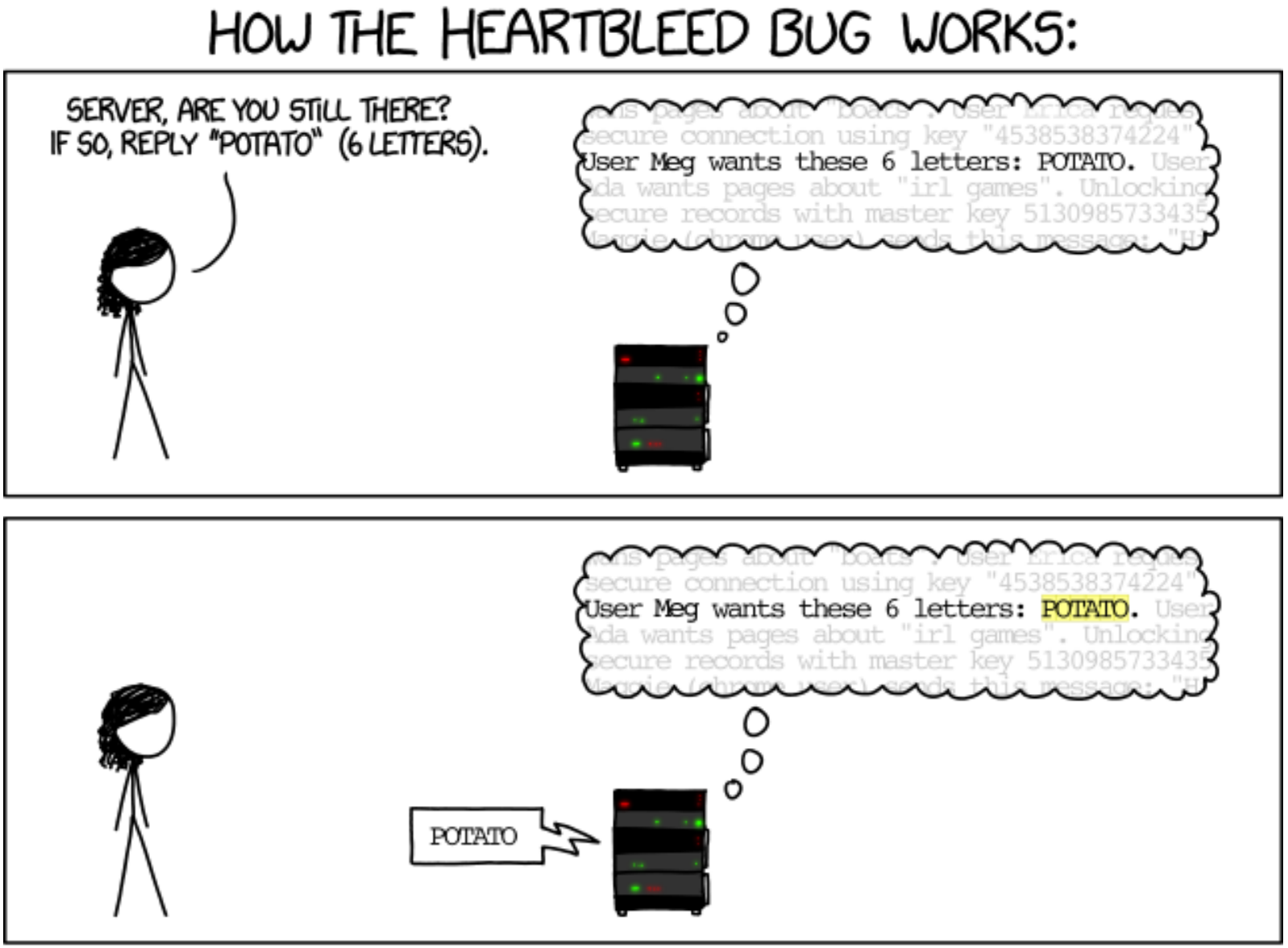
  - Dies führt dazu, dass 11.000 Bytes entfernt vom Ende des Arrays c ein Byte überschrieben wird.
  - Durch einen Pufferüberlauf kann so auch (fremder) Programmcode verändert und für eigene Zwecke verwendet werden

Pufferüberlaufangriffe

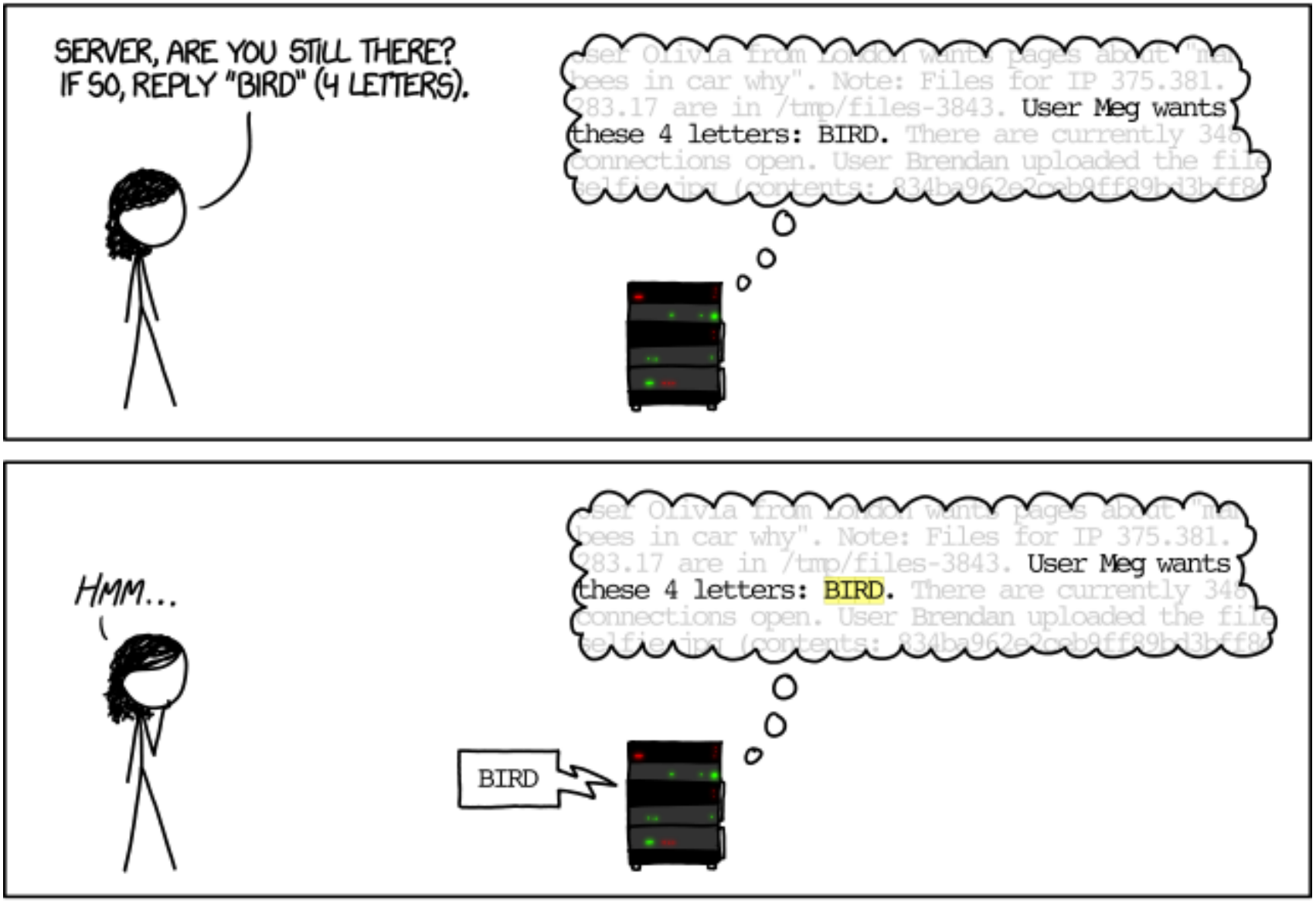
- Pufferüberlaufangriffe sind mit Abstand die häufigsten Programmierfehler, die aktiv ausgenutzt werden
  - Ping-of-Death: Frühe TCP/IP-Implementierungen waren anfällig für Buffer Overflows, die das System durch einen Netzwerk-Ping komplett lahmlegten (zuletzt 2007 in Solaris behoben)
  - Xbox-Hack 2003: Erlaubt das Spielen unlizensierter Spiele ohne Hardwaremodifikation
- Nicht nur schreibender Zugriff möglich, sondern auch lesender

  Sommer 2014: Heartbleed, erlaubt das Auslesen von bis zu 16kB Speicher aus TLS-Servern

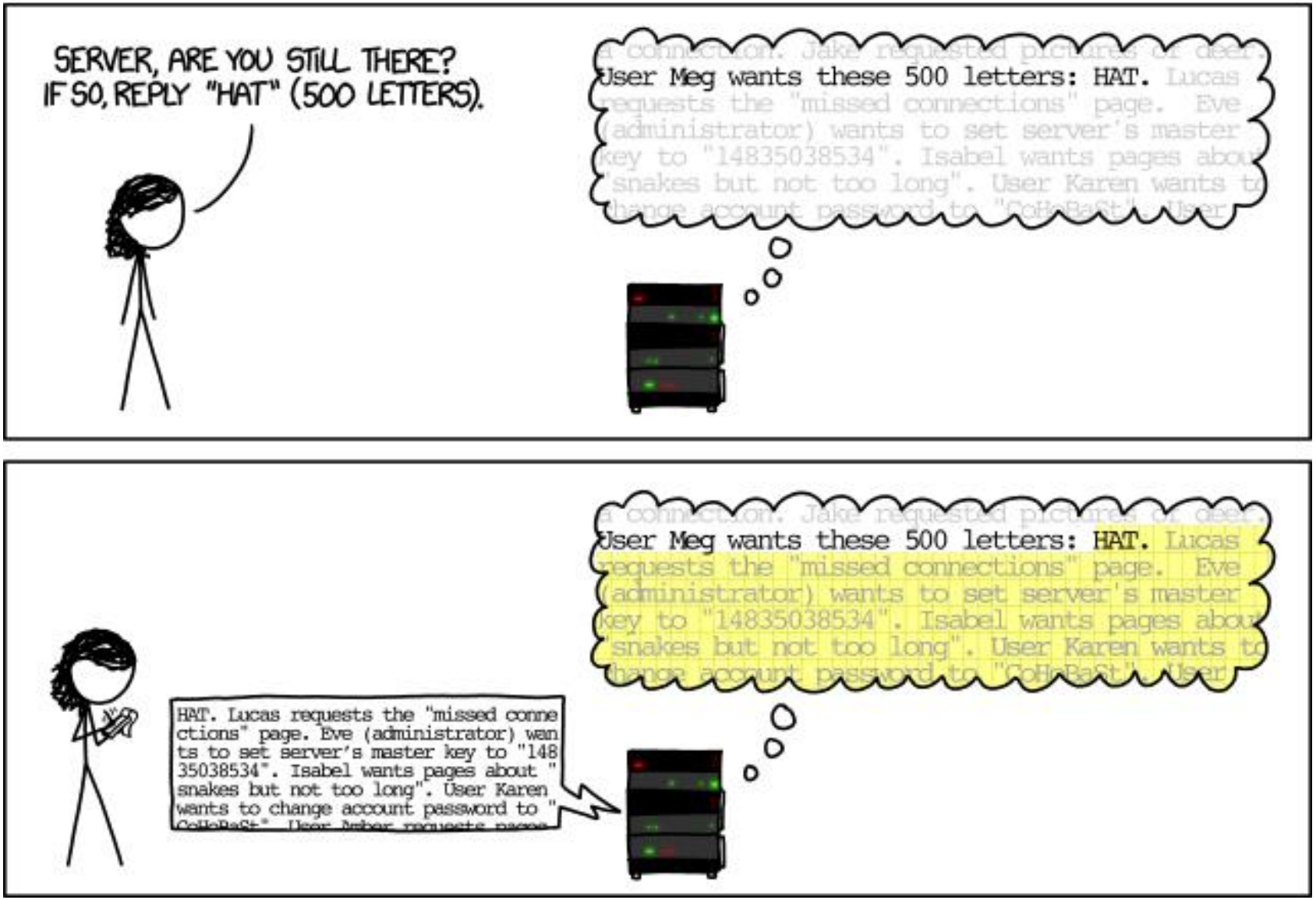
Heartbleed


HOW THE HEARTBLEED BUG WORKS:
SERVER, ARE YOU STILL THERE? IF SO, REPLY "POTATO" (6 LETTERS).
User Meg wants these 6 letters: POTATO.
POTATO

Heartbleed


SERVER, ARE YOU STILL THERE? IF SO, REPLY "BIRD" (4 LETTERS).
these 4 letters: BIRD.
HMM...
BIRD

Heartbleed


SERVER, ARE YOU STILL THERE? IF SO, REPLY "HAT" (500 LETTERS).
User Meg wants these 500 letters: HAT.
HAT. Lucas requests the "missed conne ctions" page. Eve (administrator) wan ts to set server's master key to "148 35038534". Isabel wants pages about " snakes but not too long". User Karen wants to change account password to "

Verdeckte Kanäle

- Modulation der CPU-Nutzung:
  - Server kann für eine gewisse Zeit hohe CPU-Last erzeugen, um das Bit 1 zu übertragen.
  - Um das Bit 0 zu übertragen, legt sich der Server für die gleiche Zeitspanne schlafen.
  - Kollaborateur muss Antwortzeit sorgfältig überwachen, um den Bit-Strom zu entdecken.
  - Problem: Verrauschter Kanal → fehlerkorrigierender Code (z.B. Hamming-Code)
- Modulation der Paging-Rate (Seitenfehler)
  - Viele Seitenfehler für das Bit 1
  - Keine Seitenfehler für das Bit 0

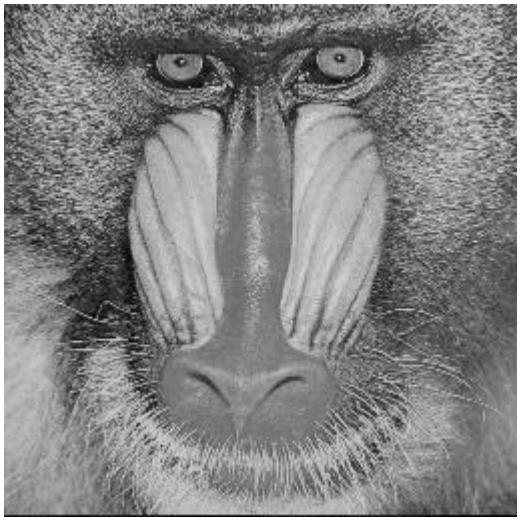
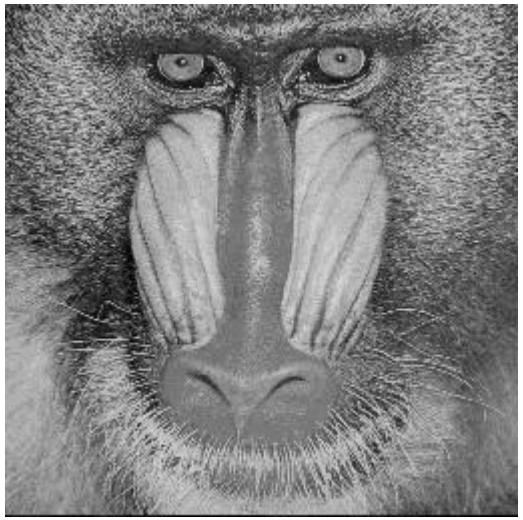
Verdeckte Kanäle

Es existieren noch viele weitere Möglichkeiten, Informationen aus einem System hinauszuschleusen, wie z.B. Belegung von Betriebsmitteln, Erzeugung von Dateien und deren Löschung, Steganographie, …

→ Es ist extrem schwierig, alle verdeckten Kanäle zu finden oder sie zu blockieren.

Steganography

Was verbirgt sich hinter diesem Bilderpaar?


weitere Möglichkeiten: Text, Audio, ...

Quelle: CS 4953 The Hidden Art of Steganography

Denial of Service

Denial of Service (**DoS**): Angriff auf ein Host (Server) in einem Datennetz mit dem Ziel einen oder mehrere Dienste durch Überlast arbeitsunfähig zu machen.

→ Erfolgt der Angriff koordiniert von einer großen Anzahl an Rechnern (z.B. Botnet) spricht man von Distributed Denial of Service (**DDoS**)

Funktionsweise: Der Host wird mit einer größeren Anzahl an Anfragen belastet als er verarbeiten kann. Daraufhin stellt dieser den Dienst ein (z.B. durch Absturz) oder kann reguläre Anfragen nur langsam bearbeiten, so dass diese abgebrochen werden.

Entwurf sicherer Systeme

Multilevel-Sicherheit

- Die meisten Betriebssysteme ermöglichen es dem Benutzer festzulegen, wer seine Dateien lesen und schreiben darf.
  - **Benutzerbestimmbare Zugriffskontrolle** (Discretionary Access Control)
  - Für einfache Systeme ohne besonderen Anforderungen ist dies ausreichend
- Organisationen benötigen allerdings häufig stärkere Sicherheit.
  - Organisation möchte zusätzlich Zugriffskontrolle festlegen.
  - **Systembestimmbare Zugriffskontrolle** (Mandatory Access Control)
  - Kontrollieren den Informationsfluss, damit keine Informationen nach außen dringen.

Bell-LaPadula-Modell

- Verfahren zur Sicherung von Vertraulichkeit
- Dokumente (Objekte) haben Sicherheitsstufen:

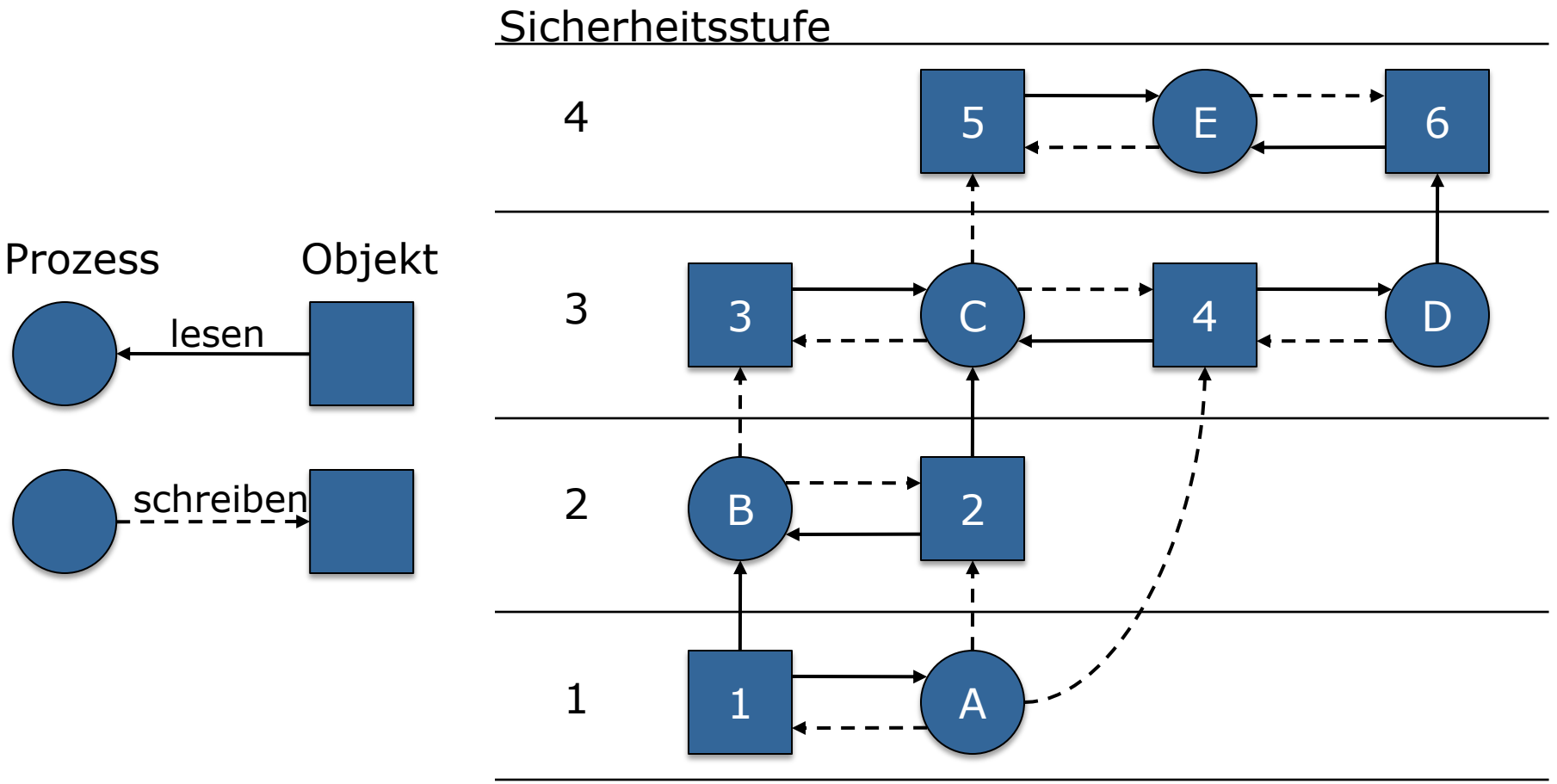
  Nicht klassifiziert, vertraulich, geheim, streng geheim, …
- Benutzer haben zugehörige Sicherheitsstufe.

  Beim Erstellen erhält das Dokument die Sicherheitsstufe des Benutzers.

- Regeln wie Informationen fließen können:
  - **Simple-Security-Regel**: Lesen ist nur auf gleicher oder von niedrigerer Stufe erlaubt.
  - ***-Regel**: Schreiben ist nur auf gleicher oder höherer Stufe erlaubt.

Bell-LaPadula-Modell

Ein Vorstandsmitglied kann von der unteren Sicherheitsstufe lesen, aber nur in gleiche oder höhere Stufe schreiben, um keine Informationen nach unten fließen zu lassen.



Bell-LaPadula-Modell

- Betriebssystem muss diese Regeln umsetzen
- Beispiel:
  - Sicherheitsstufe des Benutzers mit der UID oder der GID speichern.
  - Beim Login bekommt die Shell die Sicherheitsstufe des Users und vererbt diese an alle Kinderprozesse.
  - Wenn ein Prozess eine Datei / Objekt einer höheren Sicherheitsstufe öffnet, sollte das Betriebssystem dies zurückweisen.
  - Analog gilt dies für das Schreiben von Dateien in eine niedrigere Sicherheitsstufe.
- In der Regel nur wenige Stufen vorhanden (z.B. Administrator/Benutzer)

Bell-LaPadula-Modell

- Das Bell-LaPadula-Modell ist gut um die *Vertraulichkeit von Daten* zu wahren.

- **Nachteil**:
  Für Unternehmen nicht wirklich praktikabel
  - Beispiel: Da Informationen nur nach oben geschrieben werden können, kann der Hausmeister in wichtige Unternehmensdokumente (z.B. Strategie) schreiben.
  - → Anderes Modell notwendig, welches die *Integrität des Informationsflusses* sichert und umgekehrte Regeln besitzt (kein Aufwärtsschreiben und kein Abwärtslesen)

Trusted Computing

- Aus Systemsicht müssen daher auch die verwendeten Programme „authentifiziert“ werden.
  - Ist es wirklich der „echte“ Linux-Kernel, den ich gerade boote?
  - Wird das VBA-Macro in der Excel-Tabelle wirklich nicht die Systemstabilität beeinträchtigen?
- Sowohl aus System-, als auch aus Benutzersicht ist es daher von Vorteil, auch einzelner Software vertrauen zu können.

Trusted Computing

- Lösung: Digitale Signaturen
  - Grundlage ist häufig ein Public-Key-Kryptosystem
  - Vom Dokument wird ein (kryptographischer) Hash (MD5, SHA1, SHA256) berechnet (Quasi eine Quersumme)
  - Der Hash wird „verschlüsselt“ mit dem privaten Schlüssel
  - Die erhaltene Signatur kann mit dem öffentlichen Schlüssel und dem Dokument überprüft werden
- Nur der Inhaber des privaten Schlüssels kann Signaturen anfertigen, jeder kann die Authentizität überprüfen

Trusted Computing

- Bevor nun ein Programm gestartet wird, kann seine Signatur geprüft werden
- Z.B. Apps im AppStore
- Bei falscher Signatur muss von einer Manipulation ausgegangen werden und der Benutzer wird gewarnt oder die Ausführung komplett verhindert

Trusted Computing

- Dieses Verfahren kann man auch direkt auf den Kernel anwenden
  - SecureBoot von Microsoft bootet nur „autorisierte“ Windows-Versionen
  - Android-Telefone booten ohne weitere Tricks meist nur den Kernel des Herstellers und keinen anderen
- Dabei wird das System durch Hardware (Trusted Platform Module) unterstützt, das die Signaturen manipulationssicher in Hardware prüft
  - Das TPM stellt hierbei aber nur den Signaturmechanismus bereit
  - Die Policy (darf ausführen oder nicht) wird durch die Software des BIOS/UEFI vom Hardwarehersteller bestimmt!

Trusted Computing

- Theoretisch könnte man damit viele Sicherheitsprobleme auf einmal lösen
  - Keine Viren, Spyware, Botnetze, Werbesoftware, … mehr
  - Keine jugendgefährdenden, politisch unkorrekten, systemisch bedenklichen Inhalte mehr
- Deutliche Einschränkungen
  - Ich kann nur noch Programme ausführen, die der Hersteller mir erlaubt (Restriktive App-Stores)
  - Ich muss dem Hersteller trauen, dass er immer in meinem Interesse entscheidet und nie in seinem eigenen (oder dem anderer)
  - Ich muss bei Sicherheitslücken warten, bis eine signierte Version des Patches vorliegt

Zusammenfassung (1)

- Wir unterscheiden Betriebssicherheit und Angriffssicherheit.
- Passworte müssen sorgfältig gewählt werden, um Angriffe zu verhindern.
- Es gibt leicht zu merkende und schwer zu erratende Passwörter.
- Das Diffie-Hellman-Verfahren erlaubt es zwei Personen, sich über einen unsicheren Kanal auf einen gemeinsamen Schlüssel zu einigen.

Zusammenfassung (2)

- Asymmetrische Kryptographieverfahren erlauben es, ohne gemeinsamen Schlüssel zu kommunizieren.
- Ein Beispiel dafür ist die ElGamal-Verschlüsselung.
- Signaturen erlauben die Verifikation des Senders und basieren auf ähnlichen Prinzipien.
- Trusted Computing kann viele Sicherheitsprobleme lösen, führt aber zu substantiellen Restriktionen.