Systeme I: Betriebssysteme

Kapitel 9 Sicherheit

Wolfram Burgard

Version 07.02.2017

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
  - Keine externen Akteure, keine Manipulation
- Angriffssicherheit (security)
  - Sicherheit in Bezug auf äußere Manipulation
  - Nicht nur wahrscheinliche Fehlerszenarien müssen betrachtet werden
  - Z.B. Türschloss, Wasserzeichen

Beispiel: Chrysler Hack

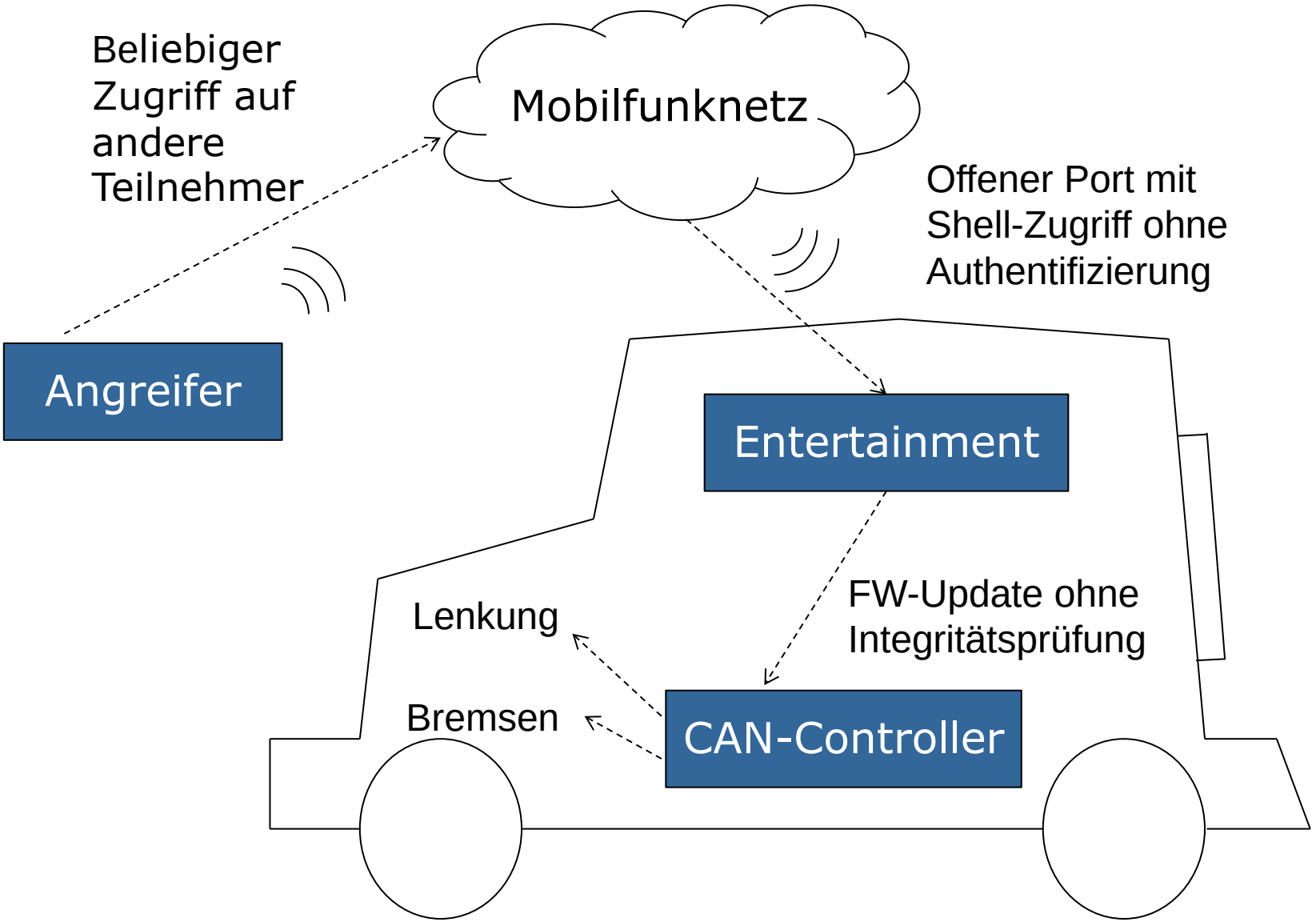
- „After Jeep Hack, Chrysler Recalls 1.4M Vehicles for Bug Fix“ (Wired, 24.07.15)
- Gravierende Sicherheits-lücken im Entertainment-System

Quelle: Andy Greenberg/Wired

→ Fahrzeuge über Mobilfunknetz fernsteuerbar

→ Mangelhafte Angriffssicherheit kann auch Betriebssicherheit beeinträchtigen

Chrysler Hack: Vorgehen



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)

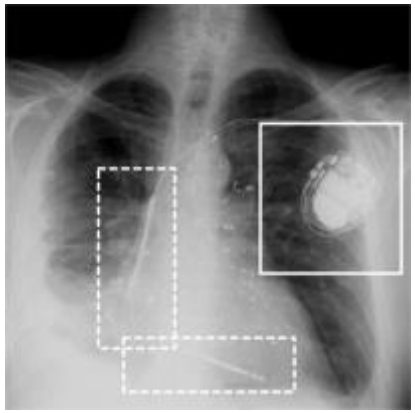
  Schutz von Personen vor dem Missbrauch ihrer persönlichen Daten

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: Nicht jeder soll die Messwerte mitlesen können
  →Verschlüsselung

- Konfigurationsdatei für Computertomograf

  Datenintegrität: Die Strahlendosis soll nicht einfach veränderbar sein
  →Signaturen

Authentifizierung

Authentifizierung

- Damit ein System ansatzweise sicher sein kann, 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 der Kombination) von
  - 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 kein Passwort 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

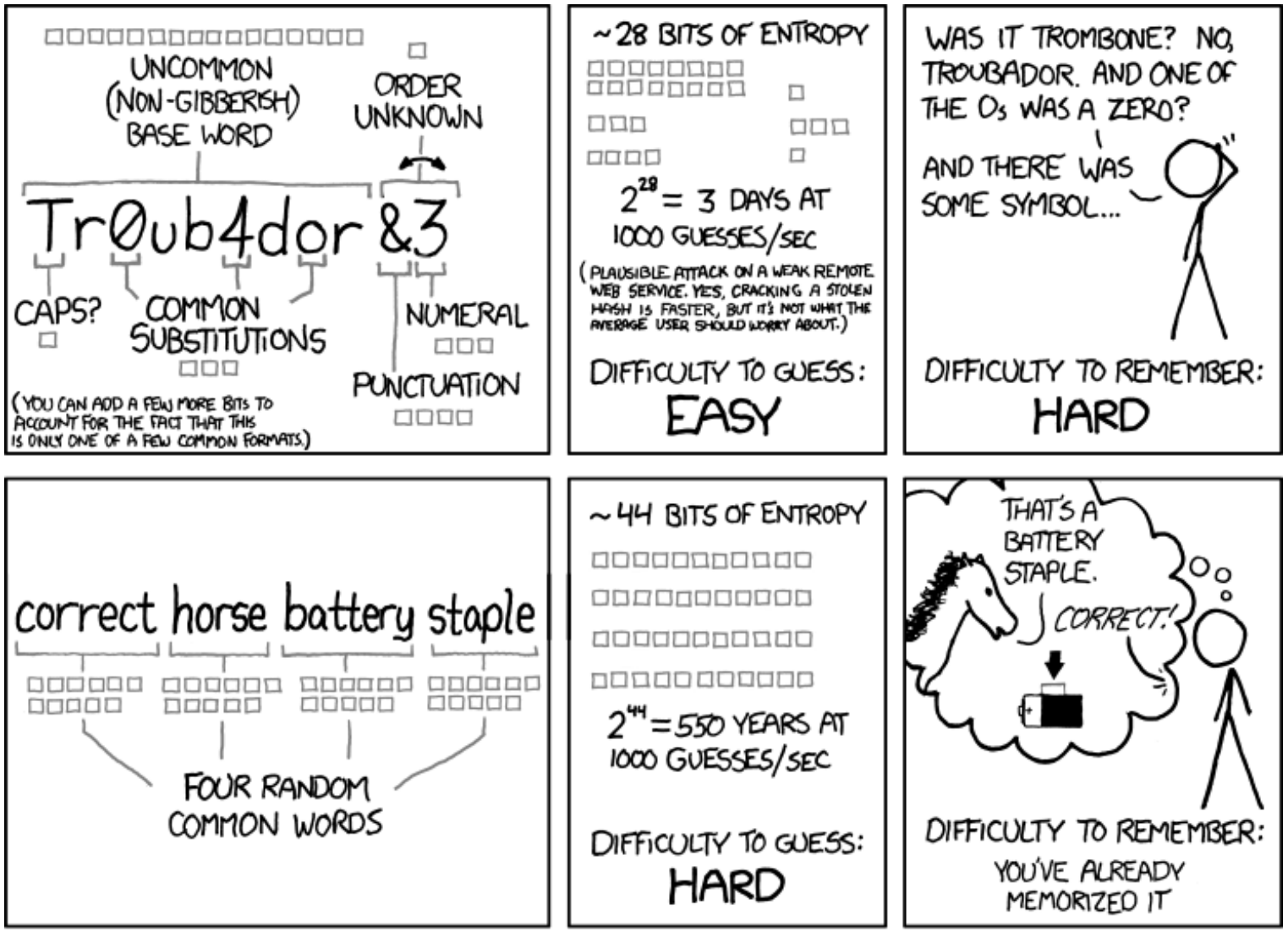
- Unix speichert die Passwörter gehasht in einer für Benutzer lesbaren Datei (/etc/passwd)
- Bei Login wird das eingegebene Passwort ebenfalls gehasht und mit dem Eintrag in der Datei verglichen
- Problem:
  - Ein Angreifer könnte eine Liste von wahrscheinlichen Passwörtern (Rainbow Table) mit demselben Verfahren verschlüsseln.
  - Anschließend muss der Angreifer die öffentlich einsehbare 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 (=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 priveligierte Nutzer lesbar machen
  In Linux: /etc/shadow, Passwort in /etc/passwd leer

Häufigste Passwörter

Wahl des richtigen Passworts



Klausur

- Freitag, dem 10.3.2017 um 13 Uhr
- Audimax und Hörsaal 2004 im Kollegiengebäude II (KG II, Innenstadt)
- Bearbeitungsdauer: 90 min
- Keine Hilfsmittel zugelassen
- Weitere Infos und Raumeinteilung:

http://ais.informatik.uni-freiburg.de/teaching/ws16/systems1/

Fragestunde vor der Klausur

- Montag, den 6.3.2017 von 14 bis 15 Uhr
- Geb. 082, Hörsaal 00-006 (Kinohörsaal)
- Übungsleiter und Tutoren beantworten Fragen in kleiner Runde

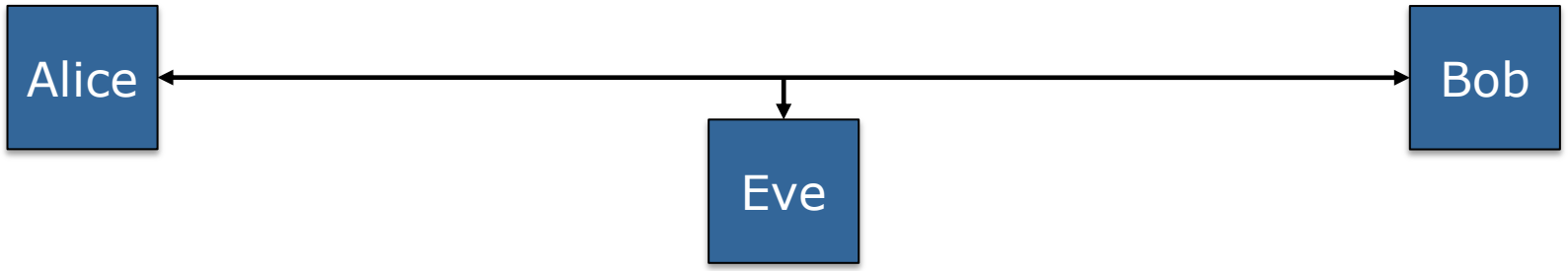
Kryptographie

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

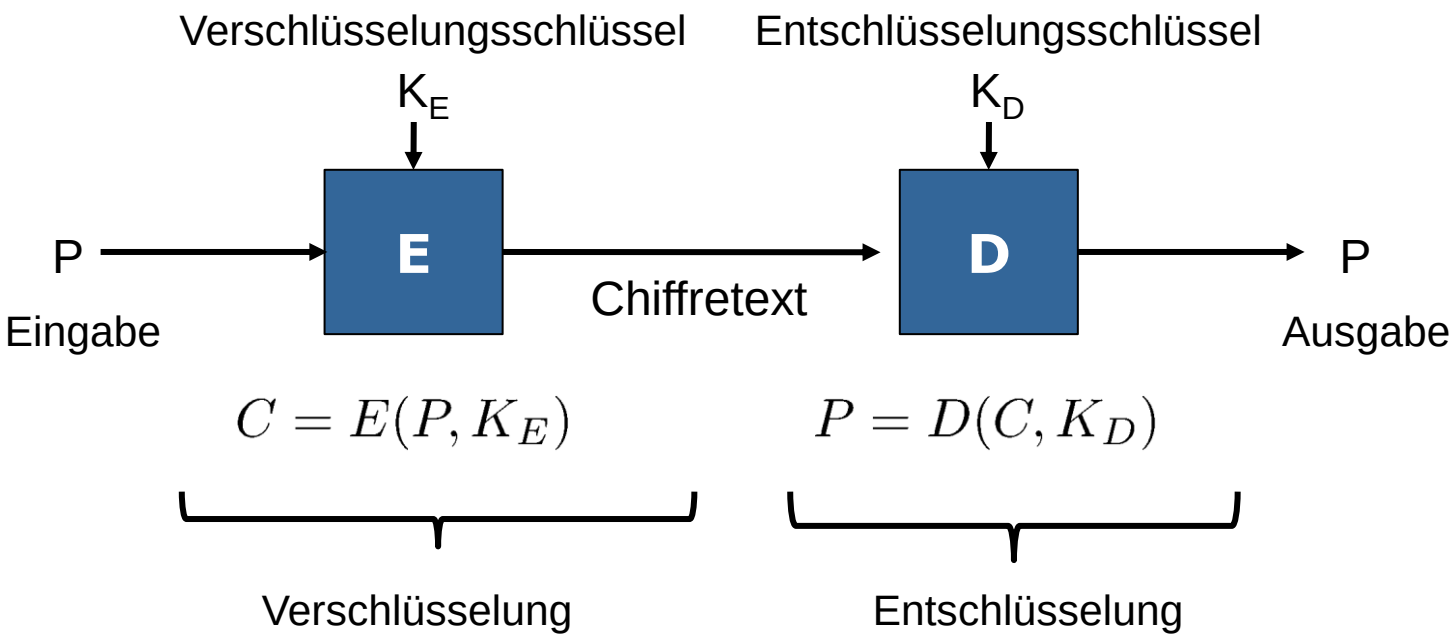
Sicherheitssysteme, die auf Verschleierung basieren, sind nicht immer sicher:

- 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
  - Stromchiffren (**Caesar**, **Enigma**, **One-Time-Pad**)
  - Blockchiffren (DES, **AES**) und Betriebsmodi
- Asymmetrische Kryptographie
  - 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)

- 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

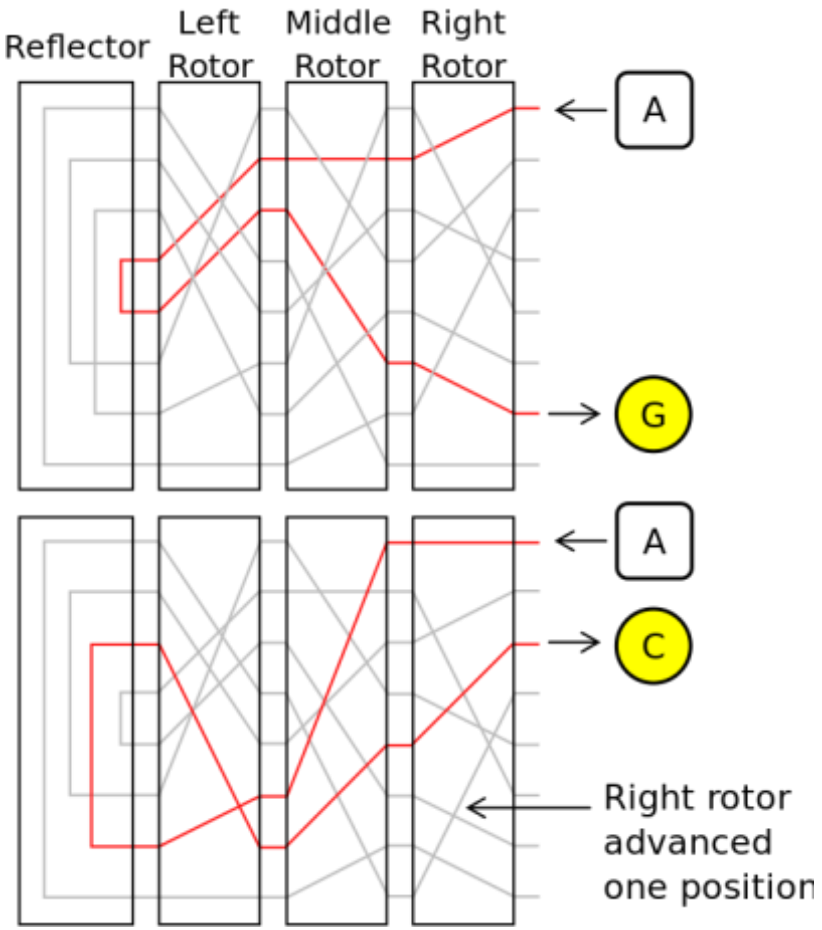
Die ENIGMA

- 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) große 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

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

AES: Advanced Encryption Standard

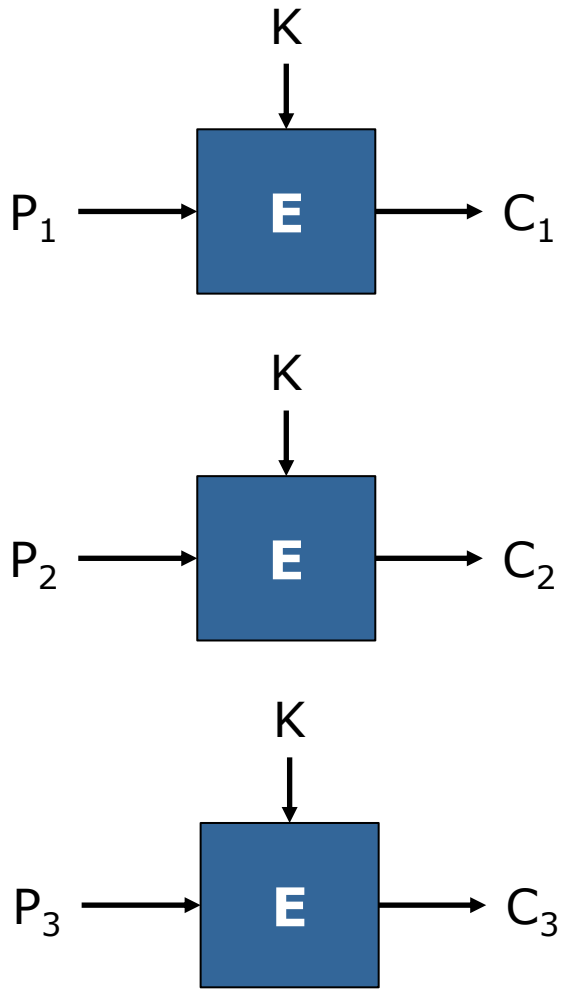
- Schlüssellänge zwischen 128 Bit und 256 Bit
- Bildet 128 Bit große Klartextblöcke auf 128 Bit Chiffratblöcke ab
- Basiert auf mehreren „Runden“
- Eine Runde besteht primär aus
  - Verknüpfung mit dem Rundenschlüssel (XOR)
  - Substitutionen auf Byte-Ebene
  - Rotationen (bit-shifts)
- Gilt aktuell als sicher (für Schlüssellängen ab 192bit)

Betriebsmodi von Blockchiffren

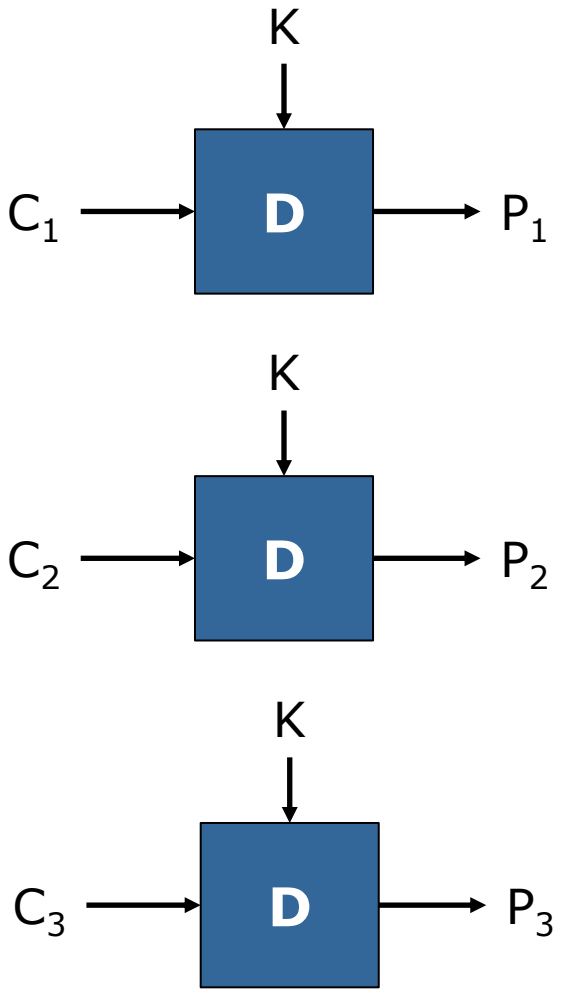
- Nachrichten, die länger als ein Block sind, müssen in Blöcke aufgeteilt werden
- Betriebsmodus bestimmt, wie mit einer Blockziffer eine längere Nachricht ver- und entschlüsselt wird
- Zu verschlüsselnder Block wird mit anderem Block „kombiniert“
  - Kombination: XOR, ggf. mit Initialisierungsvektor
  - 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

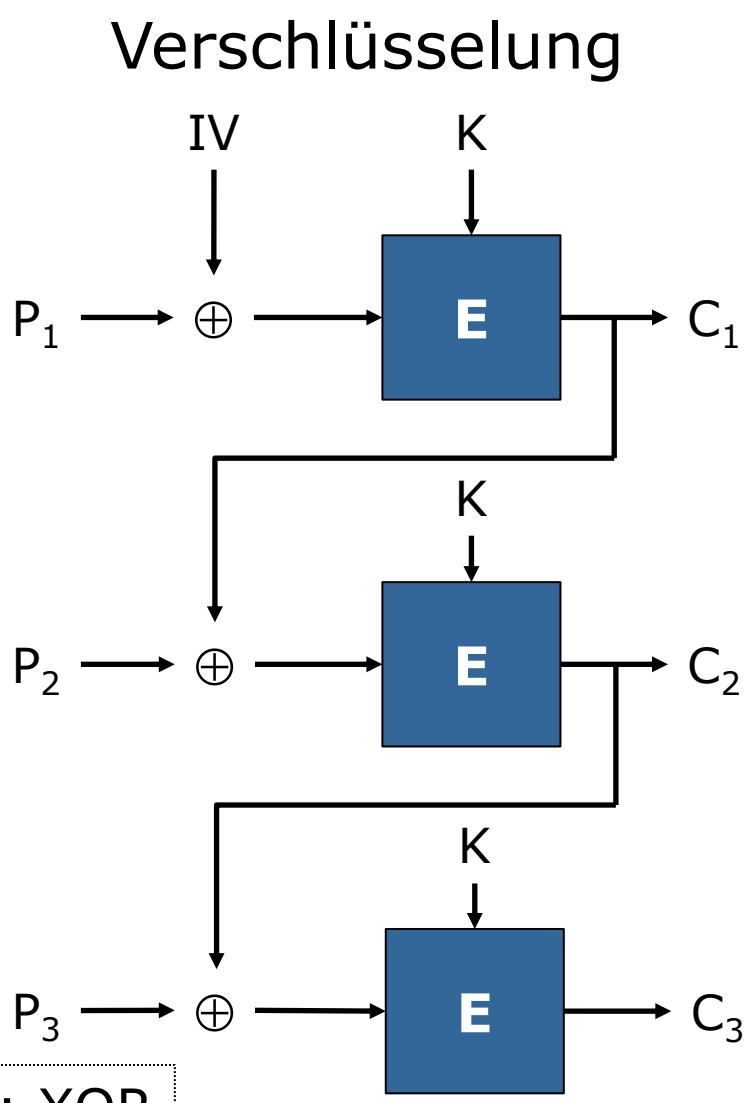
Verschlüsselung

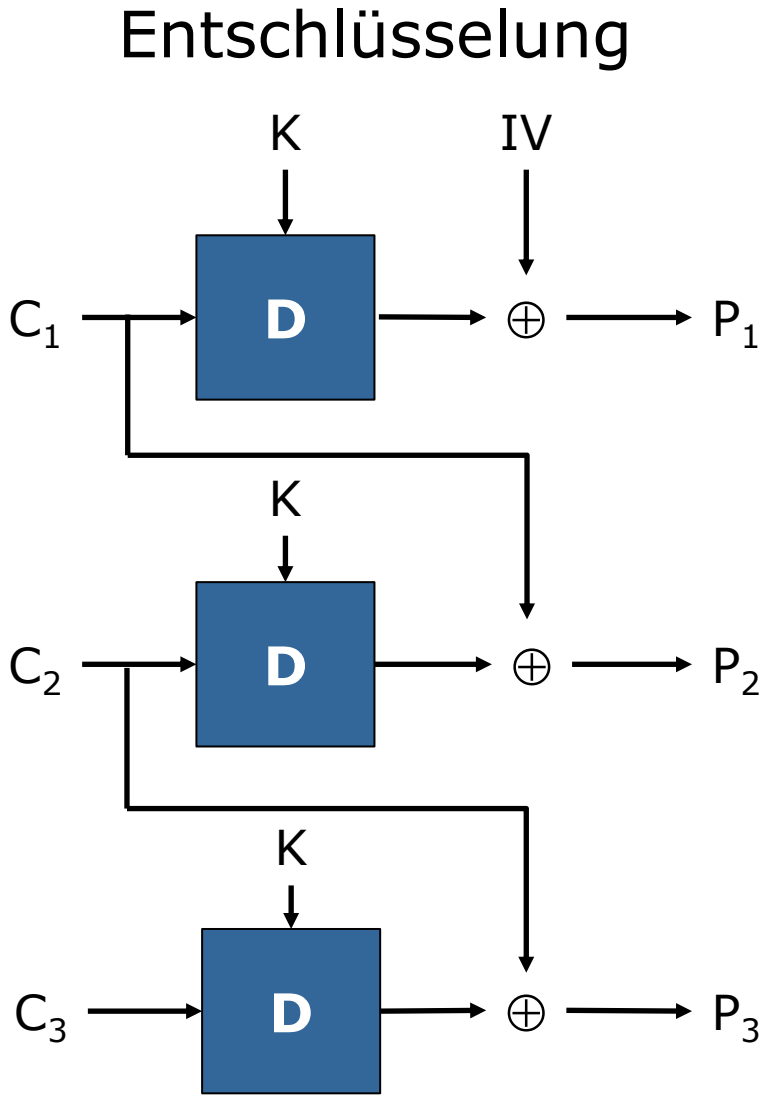


Entschlüsselung



Cipher Block Chaining Mode



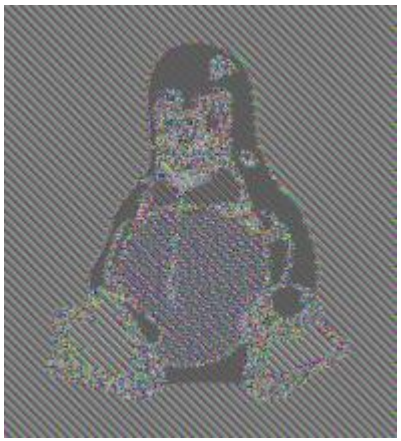


⊕: XOR

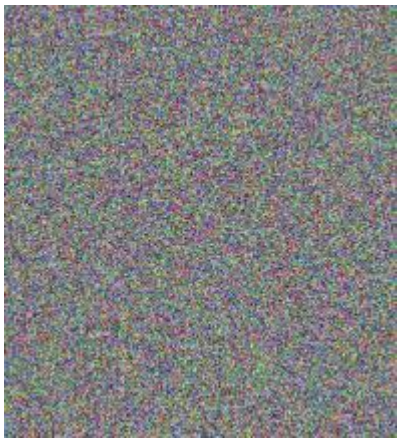
Wahl des Betriebsmodus

- 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)

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

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

Exkurs: Zufall

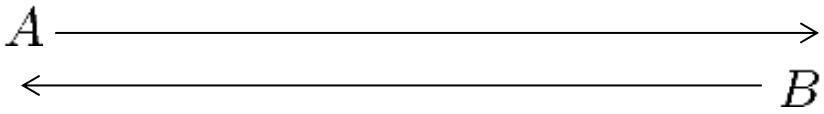
- Statistische Vorgaben (Verteilung, Unabhängigkeit) nicht ausreichend
- Kryptografischer Zufall
  - „Es ist nicht vorherzusagen, welche Zahl als nächstes kommt“
  - Prüfmethoden basieren darauf, zu versuchen, eine Zahlenfolge zu komprimieren
- „Echter“ Zufall: phys. Phänomen wie radioaktiver Zerfall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

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	$B = g^b \bmod p = 19$

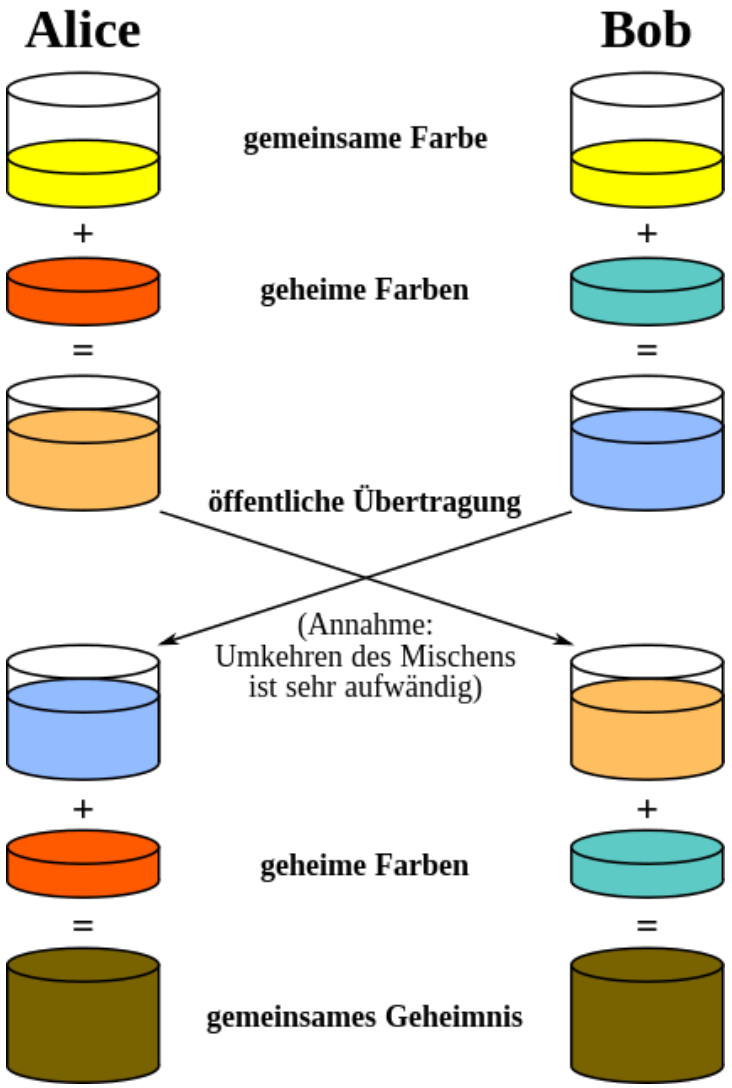
Öffentliche Übertragung

$A \longrightarrow$

$\longleftarrow B$

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

Diffie-Hellman-Schlüsselaustausch: Analogie



Quelle: A.J. Han Vinck

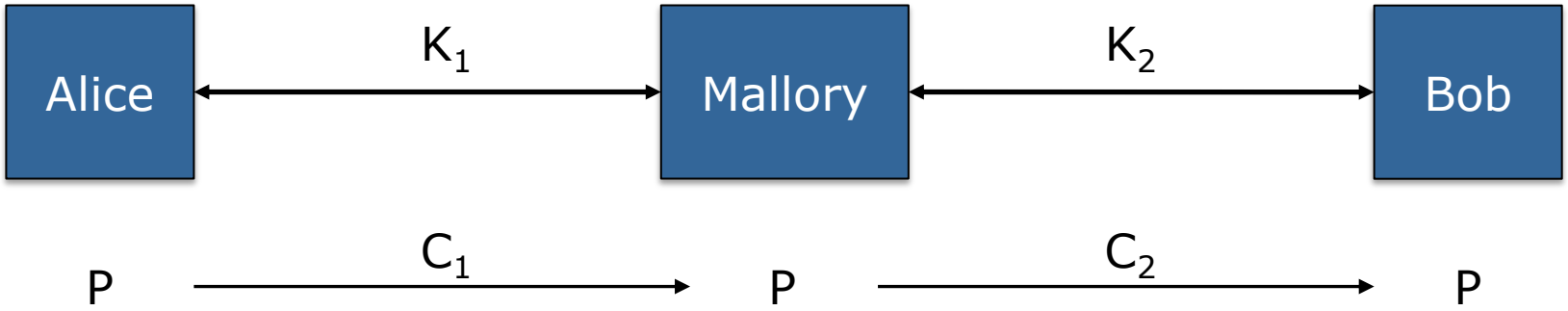
Diffie-Hellman-Schlüsselaustausch

- 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



Public-Key-Kryptographie

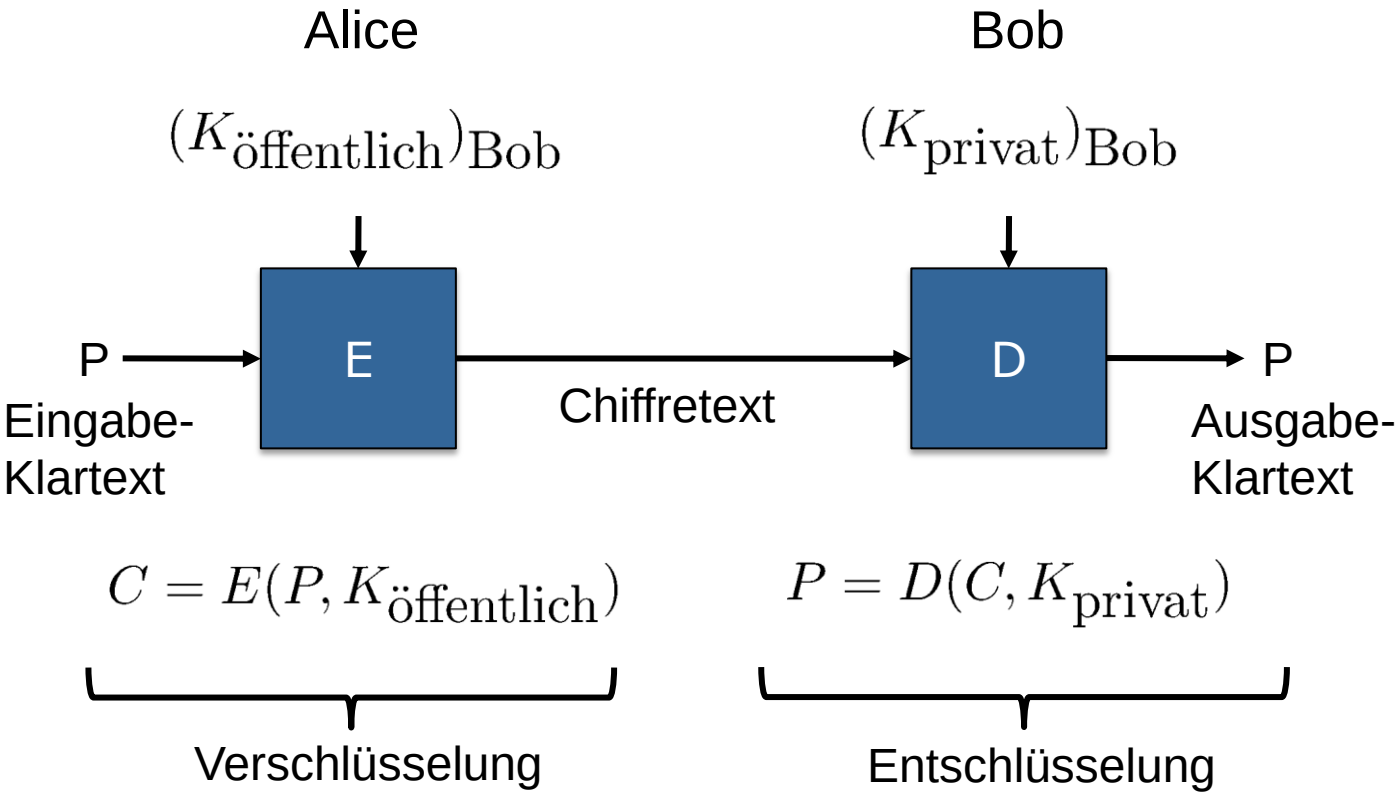
- Kommunizierende Parteien besitzen **keinen** gemeinsamen Schlüssel.

→ **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

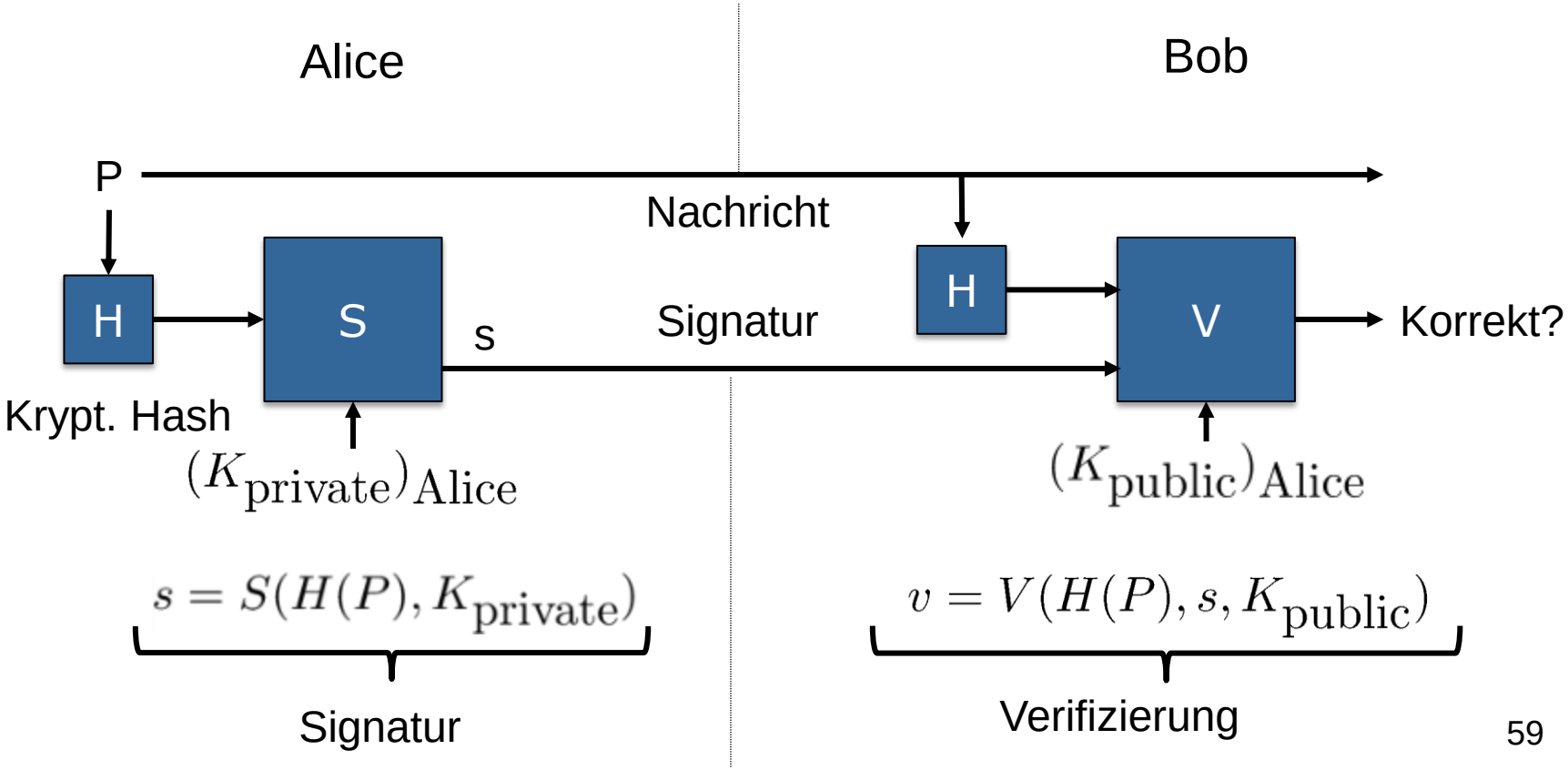
$$(g^y \bmod p, C) = (21, 20)$$

- Bob entschlüsselt

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

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

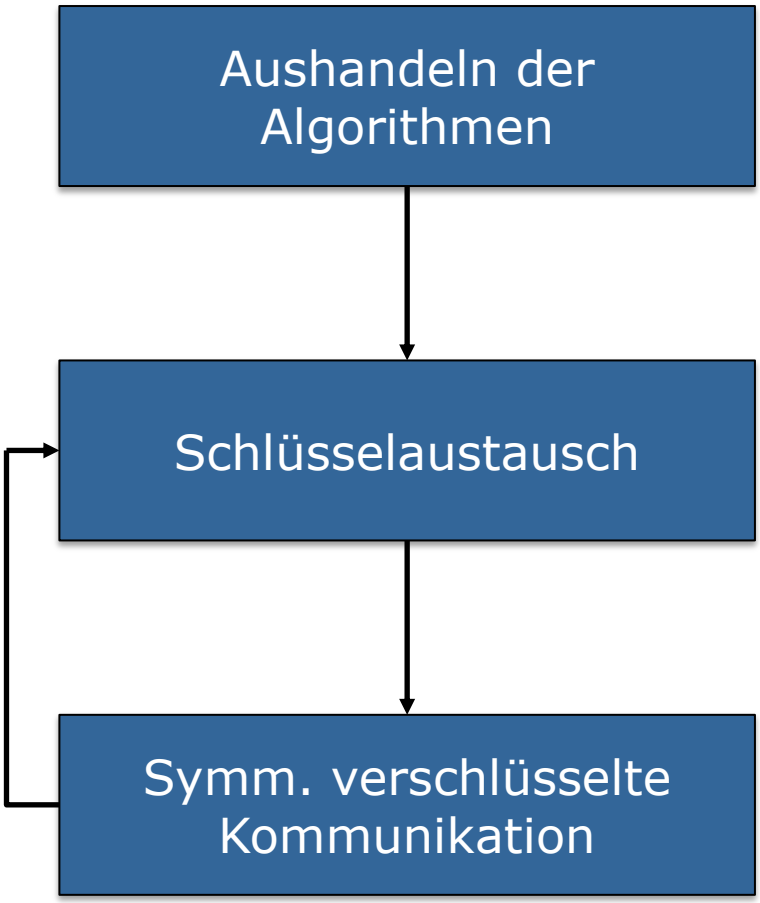
Kryptografische Systeme in der Praxis

- 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