Computernetzwerke

Eine sehr kurze Einführung in die Kryptographie

Sebastian Bauer

Wintersemester 2022/2023

Computer Engineering Curriculum

Outline

Kryptographie

Begriff

kryptós (altgr.) – geheim
gráphein (altgr.) – schreiben

Konstruktion von Informationssysteme, die widerstandsfähig gegen

- Manipulation und
- unbefugtes Lesen

sind.

Ziele von Kryptosystemen

1. **Vertraulichkeit/Zugriffsschutz**. Daten nur von berechtigten Empfängern lesbar.
2. **Integrität/Änderungsschutz**. Daten müssen vollständig und unverändert sein.
3. **Authentizität/Fälschungsschutz**. Urheber der Daten eindeutig identifizierbar.
4. **Verbindlichkeit/Nichtabstreitbarkeit**. Daten können im Nachhinein vom Urheber nicht abgestritten werden.

Kein Ziel ist es, den Datenaustausch zu verbergen.

Outline

Symmetrische Verschlüsselungsverfahren

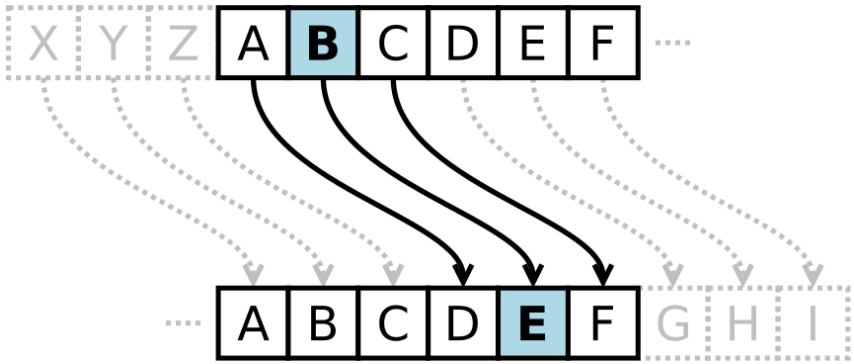
- Ein gemeinsames Geheimnis (Schlüssel), den Sender und Empfänger kennen
- Sender verschlüsselt Nachricht mit Schlüssel
- Empfänger entschlüsselt Nachricht mit Schlüssel
- Modell: Safe, in der Nachricht abgelegt wird

Aus *Kryptografie verständlich* (2016) von Paar und Pelzl

Caesar-Verschlüsselung

- Jedem Klartextbuchstaben, wird anderer kodierter Buchstabe zugeordnet
- Häufig durch Rotation definiert

$$\begin{aligned} \text{encrypt}_k(p) &= (p + k) \mod 26 \\ \text{decrypt}_k(c) &= (c - k) \mod 26 \end{aligned}$$



Beispiel für $k = 3$

Substitutionsverfahren

- Caesar-Verschlüsselung ist ein Substitutionsverfahren
- Allgemein: Zeichen oder auch Zeichengruppen werden durch andere Zeichen ersetzt

Beispiel

```
EIN SATZ MIT X, DAS WAR WOHL NIX

RVA FNGM ZVG K, QNF JNE JBUY AVK
```

Rotation, welches k?

Kryptoanalyse

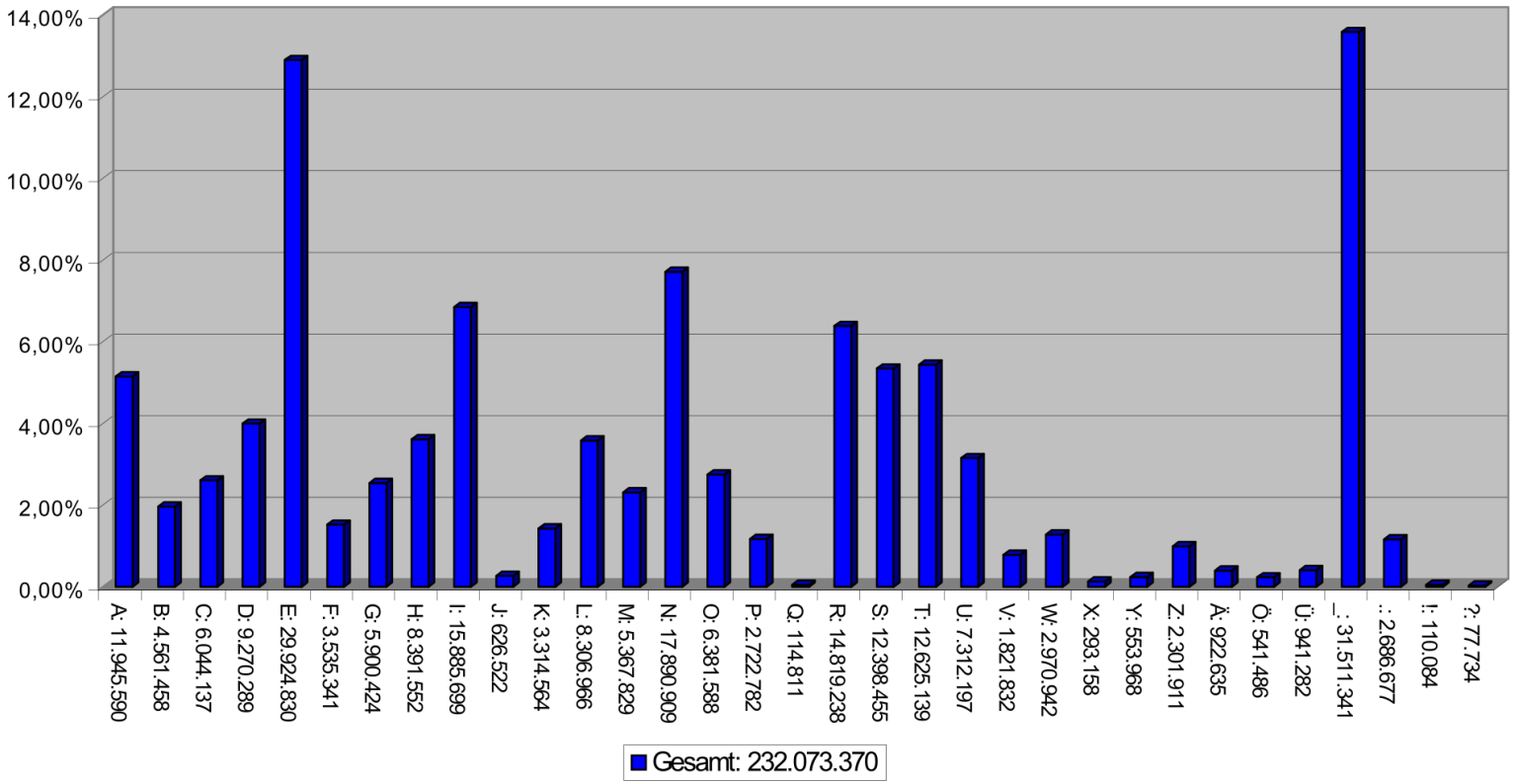
- Kryptoanalyse: Methoden, um Informationen aus verschlüsselten Texten zu extrahieren

$\Rightarrow$ Rückschlüsse auf Sicherheit des Verfahrens

- Eine Variante: Statistische Eigenschaften des verschlüsselten Texts ausnutzen
  - Häufigkeitsanalyse ermittelt relative Vorkommen der Buchstaben

Literarische Einführung in die Kryptoanalyse, insbesondere Häufigkeitsanalyse: The Gold-Bug von Edgar Allan Poe

Buchstabenverteilung



Von Arbeitsgruppe EBUSS (Potsdam, Mosbach) - selbst vektorisiert, Vorlage: Bitmap von Benutzer:Ebuss, GFDL
https://commons.wikimedia.org/w/index.php?curid=22474452

Kerckhoffs' Prinzip

Kerckhoffs' Prinzip (1883)

Die Sicherheit einer Verschlüsselung basiert allein auf der Geheimhaltung des Schlüssels (und nicht des Algorithmus).

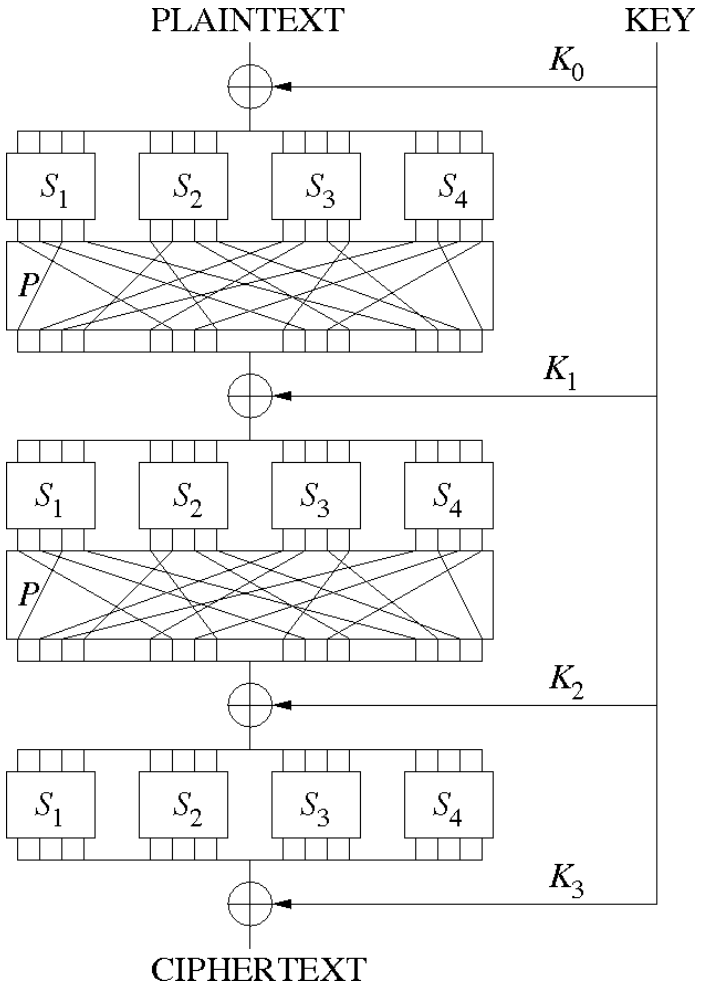
Einige Gründe, warum das auch heute noch gilt:

- Geheimhaltung eines Algorithmus ist schwierig
- Ersetzen eines kompromittierten Algorithmus ist schwierig
- Peer-Review von öffentlichen Algorithmen auf Fehler
- Hintertüren in geheimen Verschlüsselungsverfahren

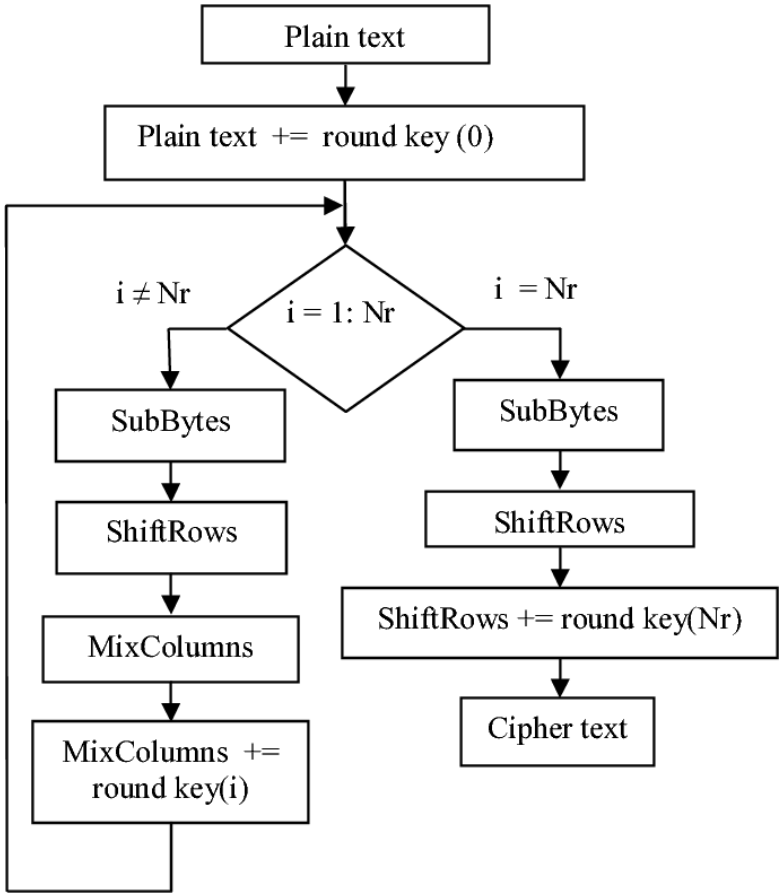
AES – Advanced Encryption Standards

- 1997 schrieb *National Institute of Standards and Technology* (NIST) Contest aus: 15 Teilnehmer
- Gewählt wurde 2000 das Verfahren von Rijndael[1]
- Basiert auf einem Substitutions-Permutations-Netzwerk
- Änderung eines Eingabebits bewirkt Änderung im kompletten verschlüsselten Text

[1] Joan Daemen, Vincent Rijmen



AES – Übersicht



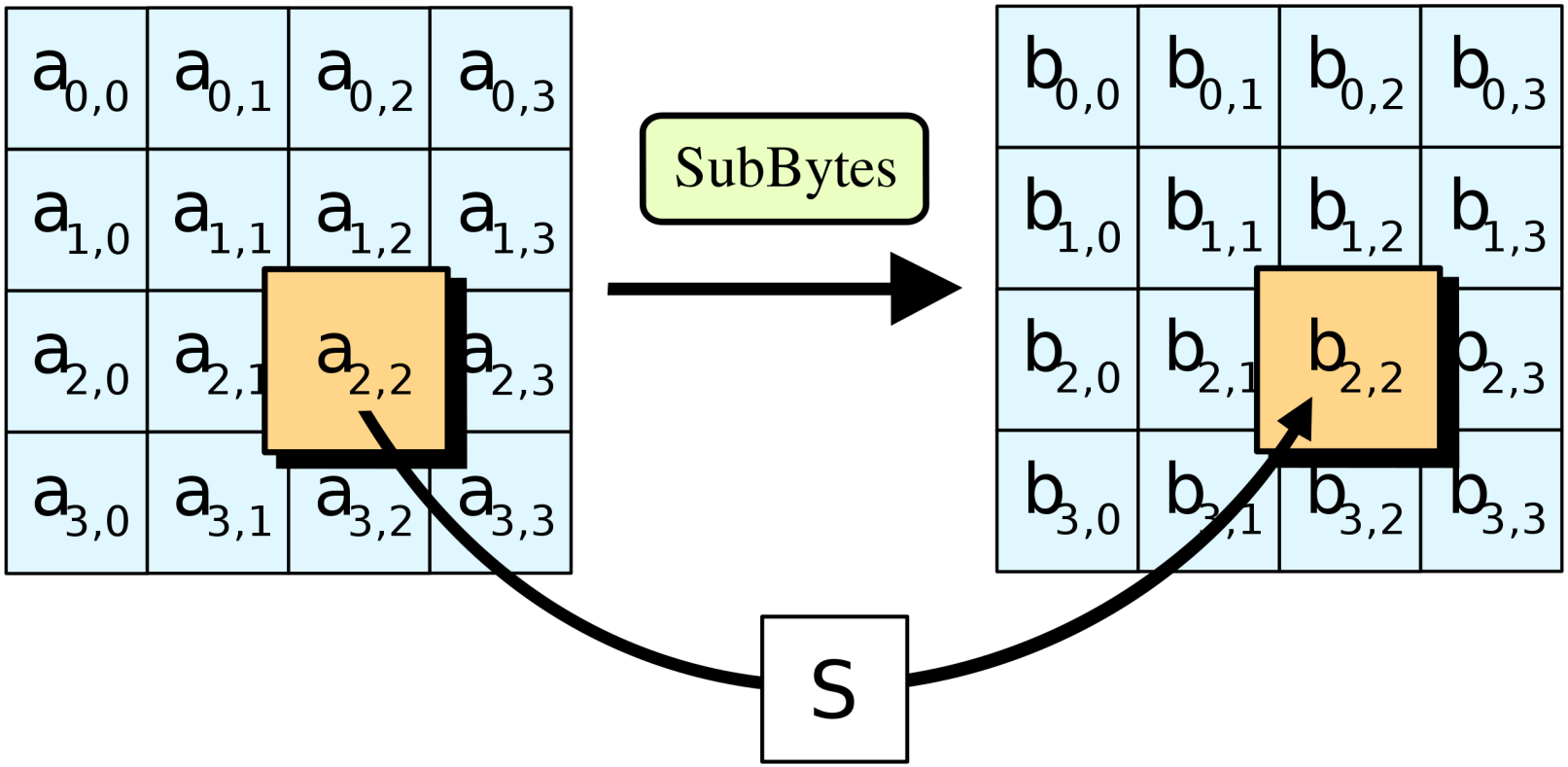
von Yenuguvanilanka, Elkeelany (2008), *Performance evaluation of hardware models of...*

- Blockgröße: 128 Bit
- Klartext $b_0 b_1 \ldots b_{15}$ wird blockweise in vierzeiliger Matrix spaltenweise angeordnet

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

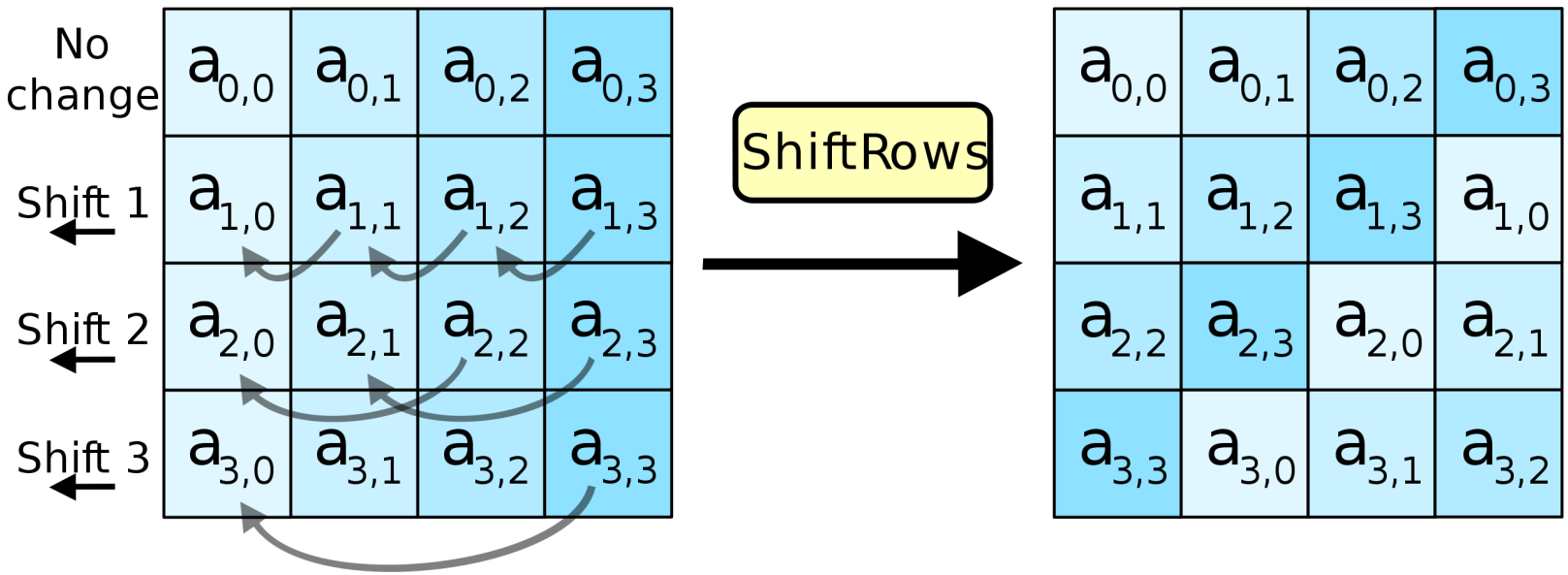
- Matrix wird durch mehrere Schritte mehrere Runden (10 bei AES-128) transformiert

AES – SubBytes

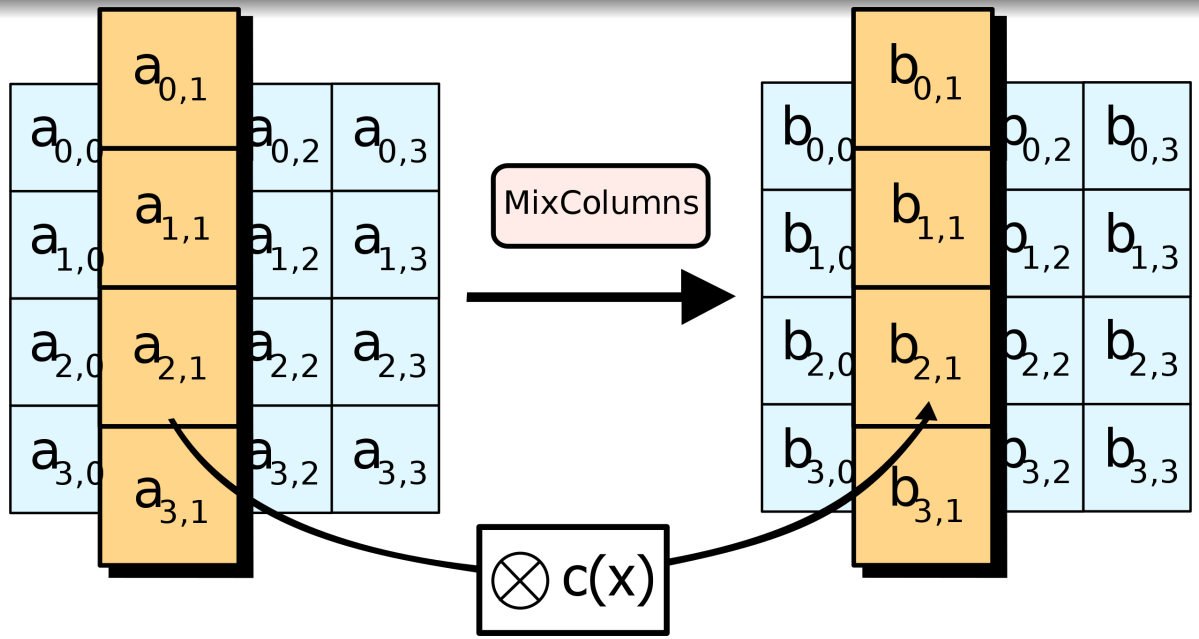


$$s(a_{i,j}) = (31a_{i,j} \mod 257) \otimes 99$$

AES – ShiftRows



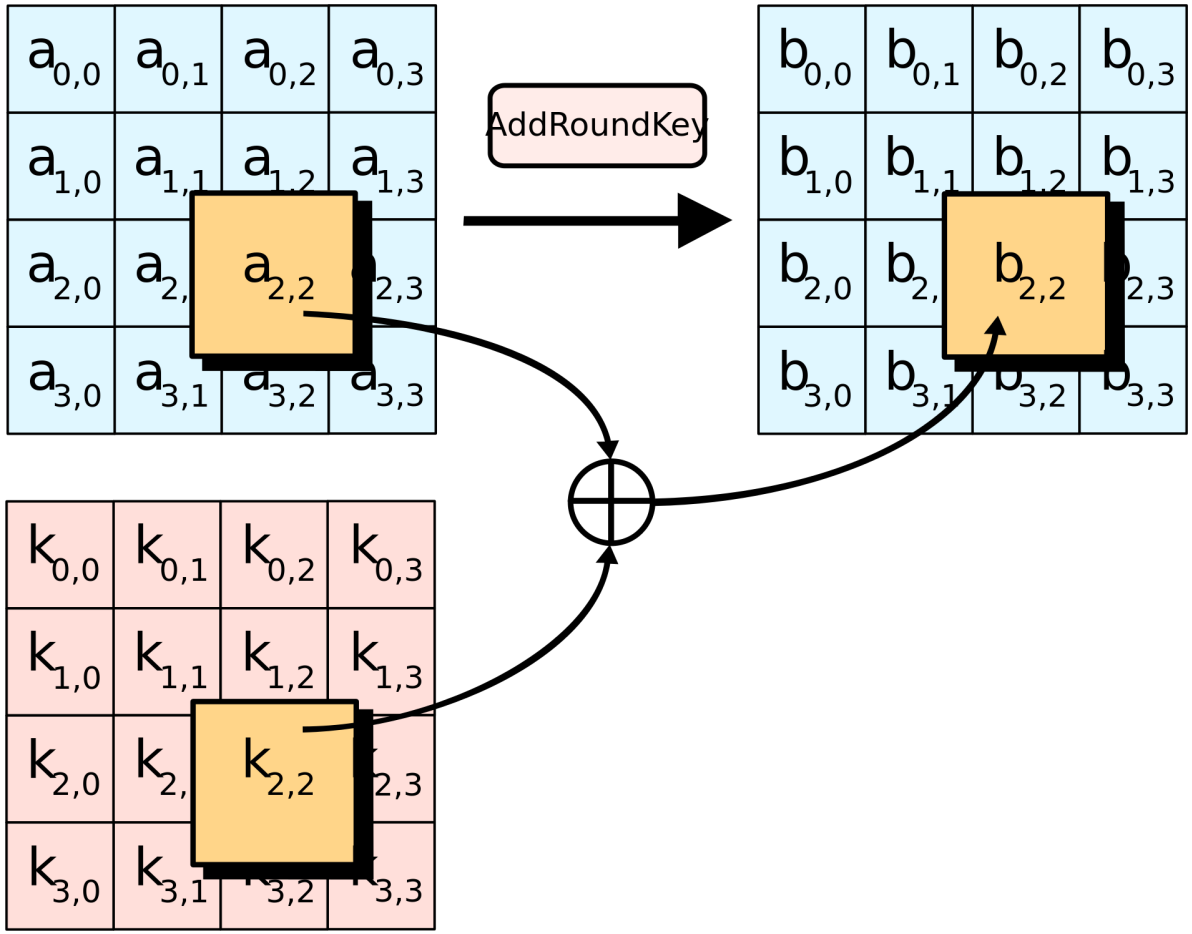
AES – MixColumns



Elemente werden mit als Polynom 7. Grades mit Koeffizienten aus $\mathbb{F}_2$ aufgefasst. Multiplikation modulo $x^8 + x^4 + x^3 + x + 1$.

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} a_{0,i} \\ a_{1,i} \\ a_{2,i} \\ a_{3,i} \end{pmatrix} = \begin{pmatrix} b_{0,i} \\ b_{1,i} \\ b_{2,i} \\ b_{3,i} \end{pmatrix}$$

AES – AddRoundKey



AES – Beispiel

Verschlüsselung eines kurzen Textes

```
$ key=12345678123456781234567812345678
$ echo "Hallo" | openssl aes-128-ecb -K ${key} | hexdump -C | cut -c 11-
78 58 f2 c4 33 8a 38 77  5b ba 48 66 14 d2 15 c8  |xX..3.8w[.Hf....|

$ echo "HAllo" | openssl aes-128-ecb -K ${key} | hexdump -C | cut -c 11-
b8 bc c5 7c 94 1d ea 90  8e 2f 8a 56 c6 5a 9b 94  |...|...../.V.Z..|
```

AES – Beispiel

Verschlüsselung eines kurzen Textes

```
$ key=12345678123456781234567812345678
$ echo "Hallo" | openssl aes-128-ecb -K ${key} | hexdump -C | cut -c 11-
78 58 f2 c4 33 8a 38 77  5b ba 48 66 14 d2 15 c8  |xX..3.8w[.Hf....|

$ echo "HAllo" | openssl aes-128-ecb -K ${key} | hexdump -C | cut -c 11-
b8 bc c5 7c 94 1d ea 90  8e 2f 8a 56 c6 5a 9b 94  |...|...../.V.Z..|
```

Verschlüsselung eines Bitmap-Bildes

```
$ convert Tux.jpg rgb:- |
 openssl aes-128-ecb -K ${key} |
 convert -depth 8 -size 196x216 rgb:- png:- |
 display
```

AES – Beispiel

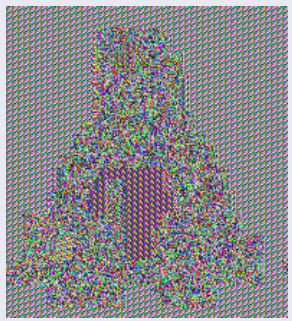
Verschlüsselung eines kurzen Textes

```
$ key=12345678123456781234567812345678
$ echo "Hallo" | openssl aes-128-ecb -K ${key} | hexdump -C | cut -c 11-
78 58 f2 c4 33 8a 38 77  5b ba 48 66 14 d2 15 c8  |xX..3.8w[.Hf....|

$ echo "HAllo" | openssl aes-128-ecb -K ${key} | hexdump -C | cut -c 11-
b8 bc c5 7c 94 1d ea 90  8e 2f 8a 56 c6 5a 9b 94  |...|...../.V.Z..|
```

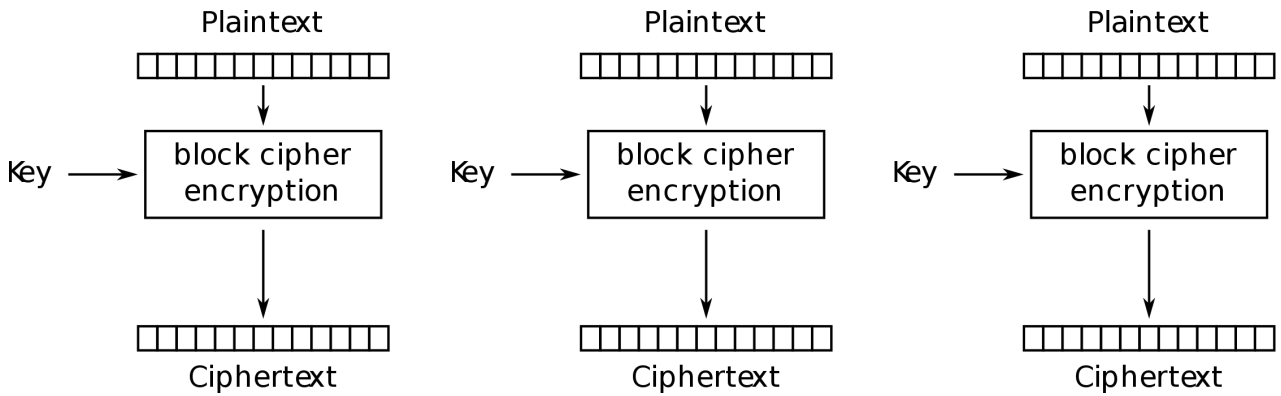
Verschlüsselung eines Bitmap-Bildes

```
$ convert Tux.jpg rgb:- |
  openssl aes-128-ecb -K ${key} |
  convert -depth 8 -size 196x216 rgb:- png:- |
  display
```



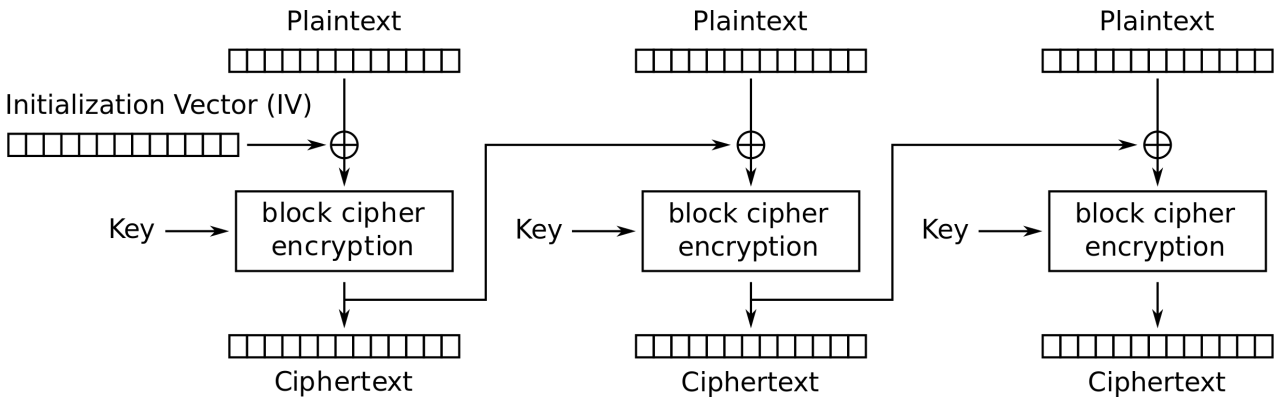
AES – ECB Betriebsart

- Bisher wurden Blöcke unabhängig voneinander chiffriert
- Diese Betriebsart heißt Electronic Code Book Mode



AES – CBC Betriebsart

- Eine Alternative ist Cipher Block Chaining Mode
- Verknüpfung des Textes vor Verschlüsselung mit dem vorherigen verschlüsselten Block
- Initialisierungsvektor für den ersten Block notwendig



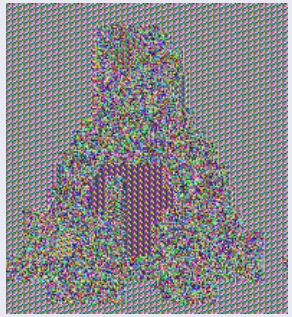
Nachteile:

1. Nicht mehr parallelisierbar
2. POODLE-Angriff

AES – CBC und Tux

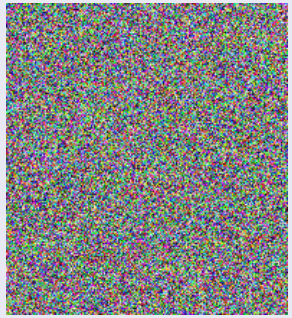
Verschlüsselung mit AES und EBC

```
$ convert Tux.jpg rgb:- |
openssl aes-128-ecb -K ${key} |
convert -depth 8 -size 196x216 rgb:- png:- |
display
```



Verschlüsselung mit AES und CBC

```
$ convert Tux.jpg rgb:- |
openssl aes-128-cbc -K ${key} -iv 0 |
convert -depth 8 -size 196x216 rgb:- png:- |
display
```



AES – Anwendungen

- SSH
- WLAN
- WPA2
- IPsec

Aufgabe

Aufgabe

Wie viele Schlüssel werden benötigt, wenn es n Teilnehmer gibt und jeder Teilnehmer mit jedem Teilnehmer verschlüsselt Nachrichten austauschen können soll?

Einmalverschlüsselung (One-Time-Pad)

Schlüssel ist

- genauso lang wie der zu verschlüsselnde Text und ist
- zufällig entsprechend der Gleichverteilung erzeugt

Verschlüsselung:

- Zum Beispiel byteweise mit XOR oder Addition

$\Rightarrow$ Perfekte Sicherheit (Shannon, 1949)

Einmalverschlüsselung – Beispiel

Schlüssel	S	=	WZSLXWMFQUDMPJLYQOXXB
Klartext	K	=	ANGRIFFIMMORGENGRAUEN
Verschlüsselt	G	=	XNZDGCSODHSEWOZFIPSCP

Einmalverschlüsselung – Beispiel

Schlüssel	S = WZSLXWMFQUDMPJLYQOXXB
Klartext	K = ANGRIFFIMMORGENGRAUEN
Verschlüsselt	G = XNZDGCSODHSEWOZFIPSCP
Alt. Schlüssel	S' = AEOUQXOFCBJQSJLIZXLHV
Alt. Klartext	K' = WIKIPEDIAFINDENWIRGUT

Es gilt: K + S = G = K' + S'

Asymmetrische Verschlüsselungsverfahren

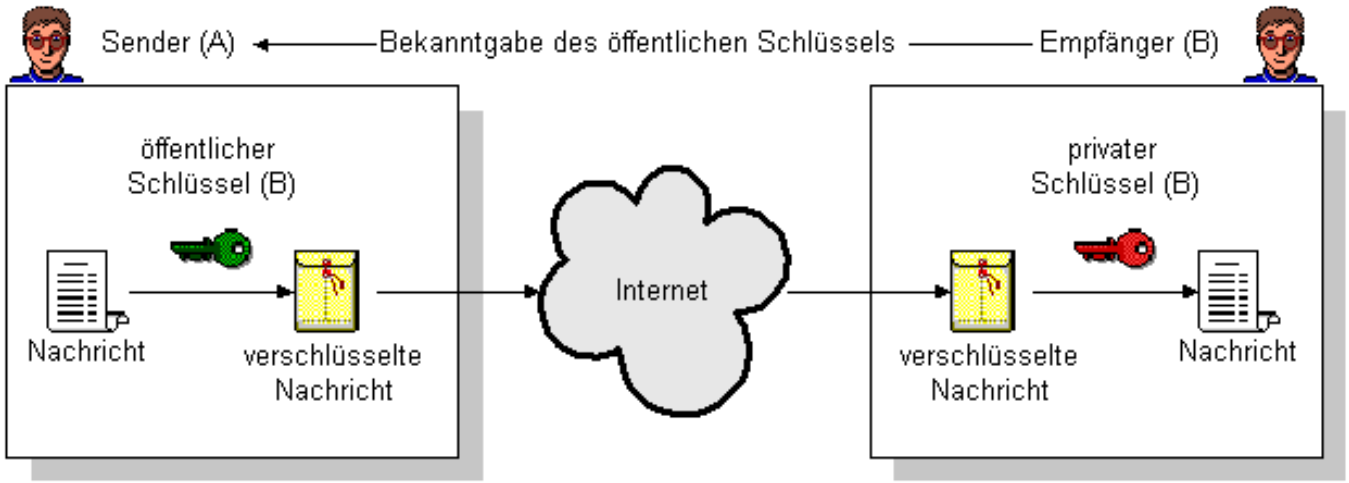
- Ein Schlüsselpaar: öffentlicher und privater Schlüssel
- Öffentlicher Schlüssel geht aus privatem hervor
- Erst seit den 1970ern bekannt

Man kann damit Nachrichten

- verschlüsseln
- signieren (deren Echtheit bestätigen)

Vertraulichkeit

- Sender verschlüsselt mit öffentl. Schlüssel des Empfängers
- Lesen der Nachricht nur mit passenden privaten Schlüssel
- Modell: Schlüssel-Schloss-Prinzip

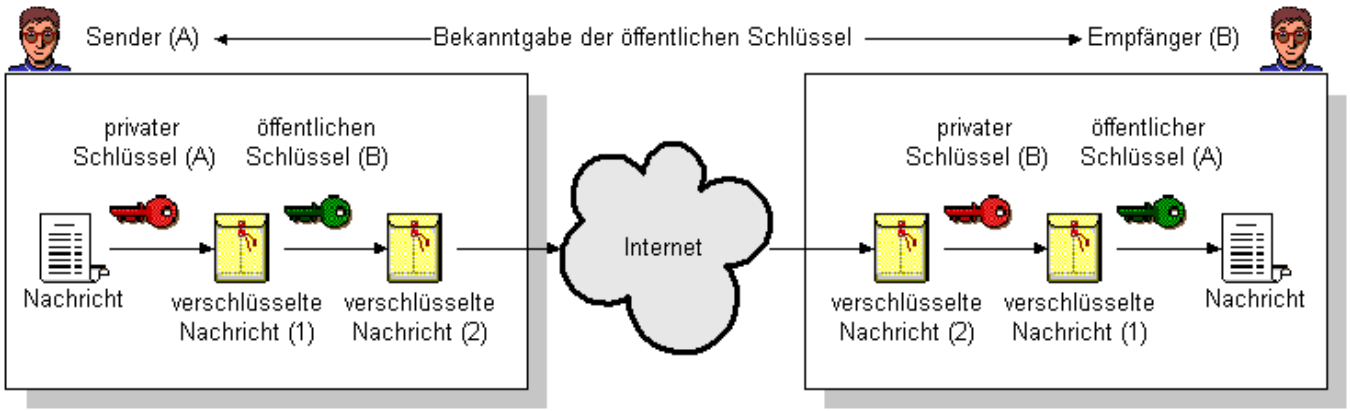


von http://ddi.cs.uni-potsdam.de/Lehre/e-commerce/elBez2-5/page06.html

Ver- und Entschlüsseln ist in der Regel wesentlich aufwendiger als bei symmetrischen Verfahren.

Signieren (Echtheit verifizieren)

- Sender verschlüsselt Text mit privatem Schlüssel
- Empfänger verifiziert Echtheit mit öffentlichen Schlüssel
- Kombination mit Vertraulichkeit möglich



von http://ddi.cs.uni-potsdam.de/Lehre/e-commerce/elBez2-5/page06.html

Primzahlerzeugung

Wichtiges Werkzeug ist die Wahl einer zufälligen Primzahl, aus der die Schlüssel resultieren:

1. Wähle zufällig eine Zahl n
2. Teste, ob Zahl n prim ist
3. Falls ja, dann gib n zurück, ansonsten gehe zurück zu 1

PRIMES

Der Primzahltest wird in theoretischen Informatik als *PRIMES* bezeichnet.

Zufall

Wir kennen bereits einen Pseudozufallsgenerator. Welchen?

Pseudozufallsgeneratoren eigenen sich aber nicht für unser Problem. Wieso nicht?

Zufall

Wir kennen bereits einen Pseudozufallsgenerator. Welchen?

Pseudozufallsgeneratoren eigenen sich aber nicht für unser Problem. Wieso nicht?

Wie geht es besser?

- Sammeln von Umgebungsgeräusche z. B. von Gerätetreibern (Tastatur, Interrupts, etc.)
- Ablegen in einem Entropiepool
- Jedes ausgelesene Bit hängt vom Zufallsbit des Entropie-Pools ab

Beispiel: Linux Trackpoint-Implementierung (`input_report_rel()` folgen bis `add_timer_randomness()`)

FYI: Zufall unter Linux

/dev/random: Daten aus dem Entropiepool

```
$ cat /proc/sys/kernel/random/entropy_avail
$ od -d /dev/random
```

Blockiert, wenn Entropie zur Neige geht

/dev/urandom: Seed aus dem Entropiepool

```
$ od -d /dev/urandom
```

Blockiert nicht, erneuert Seed gelegentlich, problematisch nur bei Early-Boot-Applications

getrandom(): Systemaufruf ab Linux 3.17

```
ssize_t getrandom(void *buf, size_t buflen, unsigned int flags);
```

Empfohlen, blockiert wenn Entropiepool noch nicht groß genug (Verhalten änderbar).

FYI2: Weiterführendes zu Zufall unter Linux

Mögliche neue Implementierung im Linux-Kernel:

- https://www.chronox.de/lrng.html,
- https://www.chronox.de/lrng/doc/lrng.pdf

Studie des BSIs:

- https://www.bsi.bund.de/DE/Publikationen/Studien/LinuxRNG/index_htm.html

Warnung

Erzeuge kritische kryptographische Schlüssel nur auf Systemen, über denen Du die Kontrolle hast. Kopiere die Schlüssel notfalls auf die Systeme, über die Du keine Kontrolle hast (Shared-Servers).

FYI3: RDRAND

- Dedizierte Instruktion, um echte Zufallzahlen zu erzeugen (ab Intel Ivy Bridge, neue AMD Prozessoren)
- Nutzt Hardware-Entropiequellen (z. B. thermisches Rauschen)
- Hoher Durchsatz: 6.4 Gbit/s

Hintertüren?

RDRAND steht in der Kritik, Hintertüren bereit zu halten.
Problematisch auch bei Mikrocode-Updates.
FreeBSD: „We cannot trust them anymore.“
Siehe auch RDRAND Bug bei AMD. Bekannt seit 2014 (Bugzilla) 2019 wiederentdeckt u.a. bei Systemd (GitHub).
Siehe auch Youtube.

PRIMES

Aufgabe

Entwerfe einen Algorithmus, der zu einer Zahl n korrekt entscheidet, ob sie eine Primzahl ist oder nicht.

PRIMES

Aufgabe

Entwerfe einen Algorithmus, der zu einer Zahl n korrekt entscheidet, ob sie eine Primzahl ist oder nicht.

Beobachtung bei Probedivision

Es genügt die Probedivision nur bis $\sqrt{n}$ auszuführen (Probedivision ist aber immer noch nicht praktikabel).

AKS-Primzahltest

- Seit 2002 ist bekannt: $PRIMES \in P$ (Agrawal-Kayal-Saxena-Primzahltest, AKS-Primzahltest)
- Theoretisch hoch interessant: Setzt nicht Richtigkeit der Riemannschen Vermutung voraus
- Dauert in der Praxis immer noch zu lang: probabilistische Algorithmen wie der Miller-Rabin-Test funktionieren besser.

Einwegfunktionen

Einwegfunktionen

Einwegfunktion $f(x)$

1. Berechnung $y = f(x)$ zu einem x erfolgt in Polynomialzeit
2. Berechnung des Urbilds x zu einem $y = f(x)$ ist „schwer“

Einwegfunktionen

Einwegfunktion $f(x)$

1. Berechnung $y = f(x)$ zu einem x erfolgt in Polynomialzeit
2. Berechnung des Urbilds x zu einem $y = f(x)$ ist „schwer“

Beispiele

1. Ausmultiplikation von Termen:
   $(x-1)(x-2)(x-3)(x-4)(x-5) =$
   $x^5 + 15x^4 + 85x^3 - 225x^2 + 274x - 120$
2. Hashfunktionen wie z.B. SHA-2-Familie
3. Multiplikation zweier Primzahlen
4. Modulares Potenzieren (z. B. $m^e \mod n$)

Kryptographische Hashfunktionen: SHA-2-Familie

- SHA ist Abkürzung von *secure hash algorithm*
- Definiert z. B. in RFC 6234
- Bildet auf Bitfolge der Länge 224, 256, 384 oder 512 ab
- Nachfolger von SHA-1
  - Wurde bereits 2005 als unsicher eingestuft
  - 2017 wurde erste echte Kollision bekannt: zwei PDFs produzieren denselben Wert

Beispiele

```
$ echo -n "Franz jagt im komplett verwahrlosten Taxi quer durch Bayern" | sha224sum
49b08defa65e644cbf8a2dd9270bdededabc741997d1dadd42026d7b
$ echo -n "Frank jagt im komplett verwahrlosten Taxi quer durch Bayern" | sha224sum
58911e7fccf2971a7d07f93162d8bd13568e71aa8fc86fc1fe9043d1
```

```
$ echo -n "Franz jagt im komplett verwahrlosten Taxi quer durch Bayern" | sha256sum
d32b568cd1b96d459e7291ebf4b25d007f275c9f13149beeb782fac0716613f8
$ echo -n "Frank jagt im komplett verwahrlosten Taxi quer durch Bayern" | sha256sum
78206a866dbb2bf017d8e34274aed01a8ce405b69d45db30bafa00f5eeed7d5e
```

Multiplikation zweier Primzahlen

Semiprimzahl

Multipliziert man zwei Primzahlen a, b mit $a \neq b$, erhält man eine Semiprimzahl (aka Fastprimzahl 2. Ordnung).

Aus welchen Faktoren besteht die Semiprimzahl 323?

Multiplikation zweier Primzahlen

Semiprimzahl

Multipliziert man zwei Primzahlen a, b mit $a \neq b$, erhält man eine Semiprimzahl (aka Fastprimzahl 2. Ordnung).

Aus welchen Faktoren besteht die Semiprimzahl 323?

- Derzeit kein effizienter Algorithmus auf klassischen Rechnern bekannt, der dieses Problem FACTORING effizient löst
- „Problematisch“ erst bei sehr großen Zahlen

Existenz von Einwegfunktionen

Es ist nicht bekannt, ob Einwegfunktionen überhaupt existieren! Deren Existenz ist gleichbedeutend mit $P \neq NP$.

Existenz von Einwegfunktionen

Es ist nicht bekannt, ob Einwegfunktionen überhaupt existieren! Deren Existenz ist gleichbedeutend mit $P \neq NP$.

Shor-Algorithmus

Der Shor-Algoithmus (1994) löst FACTORING effizient auf Quantencomputern.

Falltüren

Falltürfunktion

Eine Falltürfunktion ist eine Einwegfunktion, die mit Hilfe von Zusatzinformation leicht umkehrbar ist.

Vorhängeschlösser

Vorhängeschlösser sind Falltürfunktion: Es ist einfach, sie zu schließen. Öffnen ist ohne Schlüssel aber schwierig.

Potenzielle mathematische Falltürfunktion

- Faktorisierung von Semiprimzahlen, wenn ein Faktor bekannt ist

RSA-Kryptosystem

1. Finde zwei Primzahlen p und q, so dass $n = pq$ eine k-Bit Zahl (k ist Sicherheitsparameter)

RSA-Kryptosystem

1. Finde zwei Primzahlen p und q, so dass $n = pq$ eine k-Bit Zahl (k ist Sicherheitsparameter)
2. Wähle e mit $1 < e < \phi(n) = (p-1)(q-1)$ und $\gcd(e, (p-1)(q-1)) = 1$ (aka e ist teilerfremd zu $\phi(n)$)

Eulersche Funktion: $\phi(n)$

$\phi(n)$ gibt an, wie viele zu n teilerfremde Zahlen mit $m \leq n$ und $m, n \in \mathbb{N}$. Das spielt hier aber keine sonderlich große Rolle.

RSA-Kryptosystem

1. Finde zwei Primzahlen p und q, so dass $n = pq$ eine k-Bit Zahl (k ist Sicherheitsparameter)
2. Wähle e mit $1 < e < \phi(n) = (p-1)(q-1)$ und $\gcd(e, (p-1)(q-1)) = 1$ (aka e ist teilerfremd zu $\phi(n)$)
3. Berechne d mit $1 < d < (p-1)(q-1)$ und $de \mod (p-1)(q-1) = 1$

Eulersche Funktion: $\phi(n)$

$\phi(n)$ gibt an, wie viele zu n teilerfremde Zahlen mit $m \leq n$ und $m, n \in \mathbb{N}$. Das spielt hier aber keine sonderlich große Rolle.

RSA-Kryptosystem

1. Finde zwei Primzahlen p und q, so dass $n = pq$ eine k-Bit Zahl (k ist Sicherheitsparameter)
2. Wähle e mit $1 < e < \phi(n) = (p-1)(q-1)$ und $\gcd(e, (p-1)(q-1)) = 1$ (aka e ist teilerfremd zu $\phi(n)$)
3. Berechne d mit $1 < d < (p-1)(q-1)$ und $de \mod (p-1)(q-1) = 1$

$\rightarrow$ Privater Schlüssel: d

$\rightarrow$ Öffentlicher Schlüssel: (n, e)

Eulersche Funktion: $\phi(n)$

$\phi(n)$ gibt an, wie viele zu n teilerfremde Zahlen mit $m \leq n$ und $m, n \in \mathbb{N}$. Das spielt hier aber keine sonderlich große Rolle.

Beispiel zur Schlüsselgenerierung

1. Setze Sicherheitsparameter $k = 8$

Beispiel zur Schlüsselgenerierung

1. Setze Sicherheitsparameter $k = 8$
2. Finde $p = 11$ und $q = 23$

Beispiel zur Schlüsselgenerierung

1. Setze Sicherheitsparameter $k = 8$
2. Finde $p = 11$ und $q = 23$
3. $n = pq = 253$ (0b11111101)

Beispiel zur Schlüsselgenerierung

1. Setze Sicherheitsparameter $k = 8$
2. Finde $p = 11$ und $q = 23$
3. $n = pq = 253$ (0b11111101)
4. $\phi(n) = (p-1)(q-1) = 10 \cdot 22 = 220$

Beispiel zur Schlüsselgenerierung

1. Setze Sicherheitsparameter $k = 8$
2. Finde $p = 11$ und $q = 23$
3. $n = pq = 253$ (0b11111101)
4. $\phi(n) = (p-1)(q-1) = 10 \cdot 22 = 220$
5. Finde $e = 3$ und damit $d = 147$ (erweiterter Euklidischer Algorithmus)

Öffentlicher Schlüssel: $(n, e) = (253, 3)$
Privater Schlüssel: $d = 147$

Verschlüsselung

Verschlüsseln einer „Nachricht“ $m \in \{0, \ldots, n-1\}$:

$$c(m) = m^e \mod n$$

- Verschlüsseln also eigentlich nur Zahlen
- Geht aber, da wir alles auf Zahlen abbilden können

Verschlüsselung von $m = 165$

$n = 253, e = 3$, wie lautet $c(165)$?

Erinnerung: (n, e) ist öffentlicher Schlüssel

Verschlüsselung

Verschlüsseln einer „Nachricht“ $m \in \{0, \ldots, n-1\}$:

$$c(m) = m^e \mod n$$

- Verschlüsseln also eigentlich nur Zahlen
- Geht aber, da wir alles auf Zahlen abbilden können

Verschlüsselung von $m = 165$

$n = 253, e = 3$, wie lautet $c(165)$? $c(165) = 165^3 \mod 253 = 110$

Erinnerung: (n, e) ist öffentlicher Schlüssel

Entschlüsselung

Entschlüsselung von $c \in \{0, \ldots, n-1\}$

$$m(c) = c^d \mod n$$

Entschlüsselung von $c = 110$

$n = 253, d = 147$, wie lautet $m(110)$?

Erinnerung: d ist privater Schlüssel

Entschlüsselung

Entschlüsselung von $c \in \{0, \ldots, n-1\}$

$$m(c) = c^d \mod n$$

Entschlüsselung von $c = 110$

$n = 253, d = 147$, wie lautet $m(110)$?
$m(110) = 110^{147} \mod 253 = 110$

Erinnerung: d ist privater Schlüssel

Entschlüsselung

Entschlüsselung von $c \in \{0, \ldots, n-1\}$

$$m(c) = c^d \mod n$$

Entschlüsselung von $c = 110$

$n = 253, d = 147$, wie lautet $m(110)$?
$m(110) = 110^{147} \mod 253 = 110$
(z. B. via `python -c "print pow(110,147,253)"`)

Erinnerung: d ist privater Schlüssel

Einschub: Zwei Sätze

Kleiner fermatscher Satz

Sei $a \in \mathbb{N}$ und p eine Primzahl und a kein Vielfaches von p:

$$a^{p-1} \mod p = 1$$

Chinesischer Restsatz Folgerung

Seien p und q teilerfremd und gelte:

$$x \mod p = y \mod p$$
$$x \mod q = y \mod q$$

Dann gilt auch:

$$x \mod (pq) = y \mod (pq)$$

RSA-Satz

Satz

Sei d privater und (n, e) öffentlicher Schlüssel. Es gilt:

$$m(c(m)) = (m^e \mod n)^d \mod n = m^{ed} \mod n = m$$

Beweis

1. Es gilt $de \mod (p-1)(q-1) = 1$ mit $pq = n$

RSA-Satz

Satz

Sei d privater und (n, e) öffentlicher Schlüssel. Es gilt:

$$m(c(m)) = (m^e \mod n)^d \mod n = m^{ed} \mod n = m$$

Beweis

1. Es gilt $de \mod (p-1)(q-1) = 1$ mit $pq = n$
2. Damit existiert ein l mit $de = 1 + l(p-1)(q-1)$

RSA-Satz

Satz

Sei d privater und (n, e) öffentlicher Schlüssel. Es gilt:

$$m(c(m)) = (m^e \mod n)^d \mod n = m^{ed} \mod n = m$$

Beweis

1. Es gilt $de \mod (p-1)(q-1) = 1$ mit $pq = n$
2. Damit existiert ein l mit $de = 1 + l(p-1)(q-1)$
3. $m^{de} = m^{1+l(p-1)(q-1)}$

RSA-Satz

Satz

Sei d privater und (n, e) öffentlicher Schlüssel. Es gilt:

$$m(c(m)) = (m^e \mod n)^d \mod n = m^{ed} \mod n = m$$

Beweis

1. Es gilt $de \mod (p-1)(q-1) = 1$ mit $pq = n$
2. Damit existiert ein l mit $de = 1 + l(p-1)(q-1)$
3. $m^{de} = m^{1+l(p-1)(q-1)} = m(m^{p-1})^{l(q-1)}$

RSA-Satz

Satz

Sei d privater und (n, e) öffentlicher Schlüssel. Es gilt:

$$m(c(m)) = (m^e \mod n)^d \mod n = m^{ed} \mod n = m$$

Beweis

1. Es gilt $de \mod (p-1)(q-1) = 1$ mit $pq = n$
2. Damit existiert ein l mit $de = 1 + l(p-1)(q-1)$
3. $m^{de} = m^{1+l(p-1)(q-1)} = m(m^{p-1})^{l(q-1)} = m(m^{q-1})^{l(p-1)}$

RSA-Satz

Satz

Sei d privater und (n, e) öffentlicher Schlüssel. Es gilt:

$$m(c(m)) = (m^e \mod n)^d \mod n = m^{ed} \mod n = m$$

Beweis

1. Es gilt $de \mod (p-1)(q-1) = 1$ mit $pq = n$
2. Damit existiert ein l mit $de = 1 + l(p-1)(q-1)$
3. $m^{de} = m^{1+l(p-1)(q-1)} = m(m^{p-1})^{l(q-1)} = m(m^{q-1})^{l(p-1)}$
4. Wenn m kein Vielfaches von p und $q \rightarrow$ kleiner Fermat

RSA-Satz

Satz

Sei d privater und (n, e) öffentlicher Schlüssel. Es gilt:

$$m(c(m)) = (m^e \mod n)^d \mod n = m^{ed} \mod n = m$$

Beweis

1. Es gilt $de \mod (p-1)(q-1) = 1$ mit $pq = n$
2. Damit existiert ein l mit $de = 1 + l(p-1)(q-1)$
3. $m^{de} = m^{1+l(p-1)(q-1)} = m(m^{p-1})^{l(q-1)} = m(m^{q-1})^{l(p-1)}$
4. Wenn m kein Vielfaches von p und $q \rightarrow$ kleiner Fermat
   1. $m^{de} \mod p = m(m^{p-1})^{l(q-1)} \mod p = m \mod p$

RSA-Satz

Satz

Sei d privater und (n, e) öffentlicher Schlüssel. Es gilt:

$$m(c(m)) = (m^e \mod n)^d \mod n = m^{ed} \mod n = m$$

Beweis

1. Es gilt $de \mod (p-1)(q-1) = 1$ mit $pq = n$
2. Damit existiert ein l mit $de = 1 + l(p-1)(q-1)$
3. $m^{de} = m^{1+l(p-1)(q-1)} = m(m^{p-1})^{l(q-1)} = m(m^{q-1})^{l(p-1)}$
4. Wenn m kein Vielfaches von p und $q \rightarrow$ kleiner Fermat
   1. $m^{de} \mod p = m(m^{p-1})^{l(q-1)} \mod p = m \mod p$
   2. $m^{de} \mod q = m(m^{q-1})^{l(p-1)} \mod q = m \mod q$

RSA-Satz

Satz

Sei d privater und (n, e) öffentlicher Schlüssel. Es gilt:

$$m(c(m)) = (m^e \mod n)^d \mod n = m^{ed} \mod n = m$$

Beweis

1. Es gilt $de \mod (p-1)(q-1) = 1$ mit $pq = n$
2. Damit existiert ein l mit $de = 1 + l(p-1)(q-1)$
3. $m^{de} = m^{1+l(p-1)(q-1)} = m(m^{p-1})^{l(q-1)} = m(m^{q-1})^{l(p-1)}$
4. Wenn m kein Vielfaches von p und $q \rightarrow$ kleiner Fermat
   1. $m^{de} \mod p = m(m^{p-1})^{l(q-1)} \mod p = m \mod p$
   2. $m^{de} \mod q = m(m^{q-1})^{l(p-1)} \mod q = m \mod q$
5. Somit $m^{de} \mod n = m$, da $n = pq$ und $m < n$

Anzahl der Schlüssel

Aufgabe

Wie viele Schlüssel werden benötigt, wenn es n Teilnehmer gibt?

Schüssellänge

Äquivalenz der Schlüssellänge in Bezug auf Sicherheitsniveau (aktueller Kenntnisstand)

AES	RSA	ECC[1]
80	1024	160
112	2048	224
128	3072	256
256	15360	512

Siehe Recommendation for Key Management: Part 1 – General (von NIST)

[1] Neueres asymmetrisches Verfahren: Elliptic Curves Cryptography

Hybride Verschlüsselungsverfahren

Laufzeit der asymmetrischen Verfahren:

- RSA Schlüssellänge: heute mind. 2048 Bit, besser mehr

→ Verschlüsseln dauert sehr lange

Hybride Verschlüsselungsverfahren

1. Generiere Schlüssel für symmetrisches Verfahren
2. Übertrage Schlüssel mit asymmetrischen Verfahren
3. Nutze für die eigentliche Nachricht symmetrisches Verfahren

Disclaimer

Disclaimer

1. Es konnten nur wenige Grundkenntnisse der Kryptographie vermittelt werden

Disclaimer

1. Es konnten nur wenige Grundkenntnisse der Kryptographie vermittelt werden
2. Diese reichen überhaupt nicht aus, um eigene sichere Systeme zu bauen

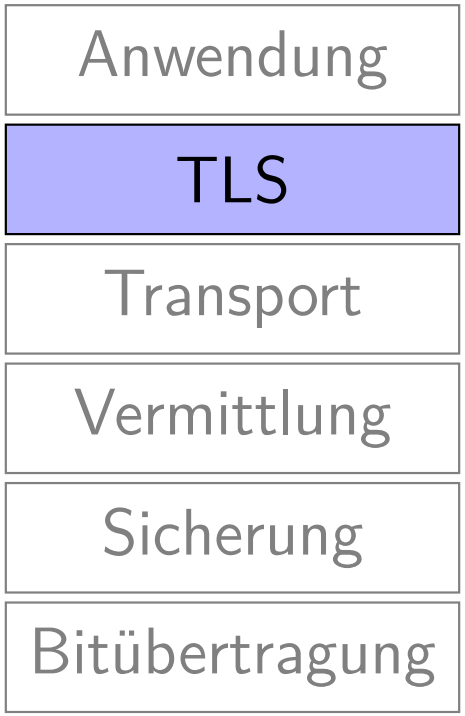
Outline

Anwendungen

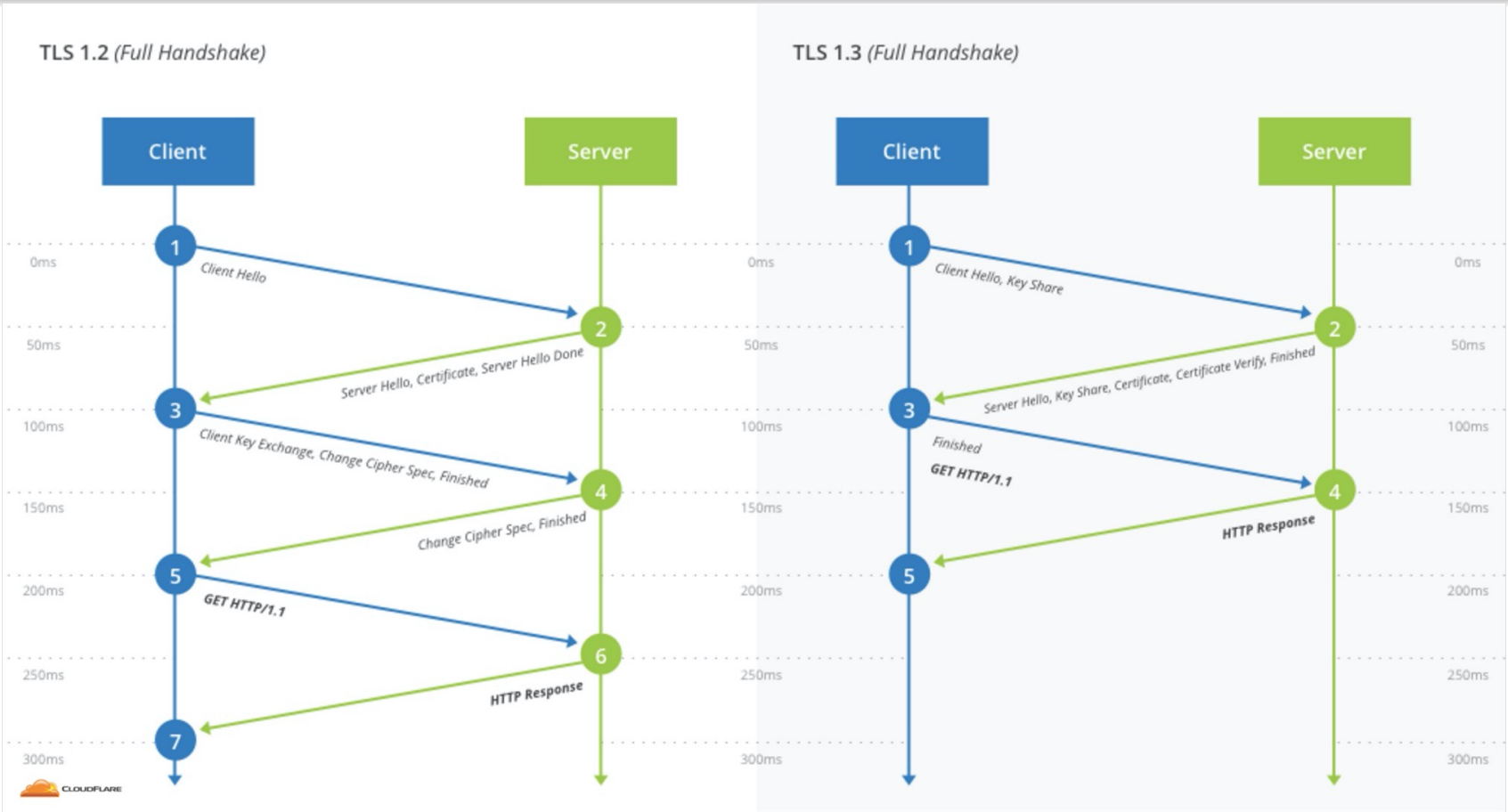
Beispiele:

- Gesicherte Netzwerkverbindung: TLS (früher SSL)
- Dateien verschlüsseln und signieren: GNU Privacy Guard (gpg)
- Auf entfernten Rechner einloggen: Secure Shell (ssh)
- Vertrauen in der DNS-Datenbank: DNSSEC

TLS


Anwendung
TLS
Transport
Vermittlung
Sicherung
Bitübertragung

TLS – Connection Handshake



The Illustrated TLS Connection:

- TLS 1.2 (2008): https://tls12.ulfheim.net/
- TLS 1.3 (2018): https://tls13.ulfheim.net/

GPG

- OpenSource Variante von PGP; patentfreie Algorithmen
- Implementiert RFC 4880
- Kommandozeilenwerkzeug
- Seit Version sind Core-Funktionialitäten nach libgcrypt ausgelagert
- Neben Nachrichtenverschlüsselung auch genutzt für Software-Signierung (z. B. Debian-Paketverwaltung)

GPG – Schlüsselgenerierung

```
$ gpg --generate-key
...
pub   rsa3072 2020-01-16 [SC] [expires: 2022-01-15]
4BA9B1E87EE262697AE013F30CC270130041E94C
uid   Sebastian Bauer <Sebastian.Bauer@htw-berlin.de>
sub   rsa3072 2020-01-16 [E] [expires: 2022-01-15]
```

GPG – Verschlüsselung

```
$ echo "Geheim" | gpg --encrypt --armor --recipient Sebastian.Bauer@htw-
berlin.de | tee secret.gpg
-----BEGIN PGP MESSAGE-----

hQGMA5dthHR6q2A+AQv/XtI778n6KtRFrHoAGUzi5DdAvY97+JW3rgjOldjJobTQ
xKPnUP6zFRopq1HSsvZnNE2olpk2u8X3FuJ3pg2OBXOK6p9qBBp9kwgBSYdjsvOI
qwCP9U/75livVqzLgZiJXMMIZlPVNymRZ5XZtUKuhK/EzrGgb8cJCKWZ4zMifPHI
ZG4igvtlIgOaztoANIRh7sFMY1omi5NDcZE5j/A1PlsPbeaJh8nY6igE03+lw/sV
Jo3ksenVZe/bE2g3ccSxHOl+5bewEtqXTlrUimwZsBQ7949xO2MXa+vYgVjzhPjU
BqshyLBgAWU61IG3Bzx3N5v4l+d8oDHjy9RRkmWUrqOph7uZhHsPjOk5KTZA7/py
a/7c0xJ9EzqgivcJkWyzHRVHtUYrCzeW/DAk1dxij3BIJ27yDnYxryP+2kQjmFP6
FNlQEDYNfmEXuQ7LMPekquPkd2CtaGMYCDkswTMtgcBkPl4jtfHOVfMicXSEaYwW
2+lNGVPswjCJgNhUvAoEOkIBJUUzgNJ9DCcyswHFM/PZ6Y8pop+NeAUFhQWIhpYc
DFTU4rB+2fztfFDYqDFY6mbsrd1UEelDhux6zW3onjAiNX8=
=CLaQ
-----END PGP MESSAGE-----
```

GPG – Entschlüsselung

```
$ cat secret.gpg | gpg --decrypt
gpg: encrypted with 3072-bit RSA key, ID 976D84747AAB603E, created 2020-
01-16
"Sebastian Bauer <Sebastian.Bauer@htw-berlin.de>"
Geheim
```

Vielen Dank für die Aufmerksamkeit!