Public-Key-Verschlüsselung mit RSA

Elmar Tischhauser

AG IT-Sicherheit

Inhaltsübersicht

Inhaltsübersicht

Modulare Arithmetik

Definition

Für $0 < m \in \mathbb{Z}$ vereinbaren wir

$$\mathbb{Z}_m = \{0, 1, \ldots, m-1\}.$$

Arithmetik in $\mathbb{Z}_m$

Für $a, b \in \mathbb{Z}_m$ erklären wir

- Addition: $(a + b) \bmod m$,
- Subtraktion: $a - b = (a + (m - b)) \bmod m$,
- Multiplikation: $(a \cdot b) \bmod m$.

Resultat jeweils wieder in $\mathbb{Z}_m$

Was ist mit **Division?**

- z.B. $1/3 \equiv 1 \cdot 3^{-1} \equiv 3 \pmod 4$, denn $3 \cdot 3 \equiv 1 \pmod 4$,
- aber $2x \not\equiv 1 \pmod 4$ für alle $x \in \mathbb{Z}_4$,
  also existiert $2^{-1} \bmod 4$ nicht!

Existenz des multiplikativen Inversen

Theorem

Sei $a \in \mathbb{Z}_m$. Die Gleichung

$$ax \equiv 1 \pmod{m}$$

hat genau dann eine (eindeutige) Lösung $x \in \mathbb{Z}_m$ wenn $\gcd(a, m) = 1$.

Definition

Dieses x nennt man **das multiplikative Inverse von a modulo m**, auch geschrieben als a^{-1} mod m.

Wie können wir a^{-1} mod m berechnen?

Ein hilfreiches Lemma

Lemma (Bézout ≈1750)

Seien $a, b \in \mathbb{Z}$ ungleich Null. Dann existieren Ganzzahlen s und t so dass

$$sa + tb = \gcd(a, b).$$

Warum hilft uns das weiter?

- Angenommen $\gcd(a, m) = 1$, dann gibt es $s, t \in \mathbb{Z}$ mit

$$sa + tm = 1 = \gcd(a, m).$$

- Das heißt jedoch

$$sa + tm \equiv sa \equiv 1 \pmod{m}$$

- Also ist $s = a^{-1} \bmod m$.

Noch offen: wie bekommen wir die Koeffizienten s, t?

Wiederholung: Euklidischer Algorithmus

Euklids Idee:

falls $b = 0$ dann $\gcd(a, b) = |a|$

falls $b \neq 0$ dann $\gcd(a, b) = \gcd(|b|, a \bmod |b|)$

Algorithmus:

1. $r_{-1} := |a|, r_0 := |b|$
2. **for** $k = 1, \ldots$:
   - 2.1 $r_k := r_{k-2} \bmod r_{k-1}$

   until $r_k = 0$

Beispiel 1: $\gcd(30, 24)$

k	r_k
-1	30
0	24

Beispiel 2: $\gcd(100, -35)$

k	r_k
-1	100
0	$+35$

Wiederholung: Euklidischer Algorithmus

Euklids Idee:

falls $b = 0$ dann $\gcd(a, b) = |a|$

falls $b \neq 0$ dann $\gcd(a, b) = \gcd(|b|, a \bmod |b|)$

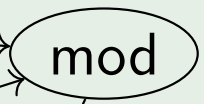
Algorithmus:

1. $r_{-1} := |a|, r_0 := |b|$
2. **for** $k = 1, \ldots$:
   - 2.1 $r_k := r_{k-2} \bmod r_{k-1}$

   until $r_k = 0$

Beispiel 1: $\gcd(30, 24)$

k	r_k
-1	30
0	24
1	6



Beispiel 2: $\gcd(100, -35)$

k	r_k
-1	100
0	$+35$

Wiederholung: Euklidischer Algorithmus

Euklids Idee:

falls $b = 0$ dann $\gcd(a, b) = |a|$

falls $b \neq 0$ dann $\gcd(a, b) = \gcd(|b|, a \bmod |b|)$

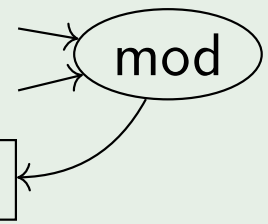
Algorithmus:

1. $r_{-1} := |a|, r_0 := |b|$
2. **for** $k = 1, \ldots$:
   2.1 $r_k := r_{k-2} \bmod r_{k-1}$

   until $r_k = 0$

Beispiel 1: $\gcd(30, 24)$

k	r_k
-1	30
0	24
1	6
2	0



Beispiel 2: $\gcd(100, -35)$

k	r_k
-1	100
0	$+35$

Wiederholung: Euklidischer Algorithmus

Euklids Idee:

falls $b = 0$ dann $\gcd(a, b) = |a|$

falls $b \neq 0$ dann $\gcd(a, b) = \gcd(|b|, a \bmod |b|)$

Algorithmus:

1. $r_{-1} := |a|, r_0 := |b|$
2. **for** $k = 1, \ldots$:
   - 2.1 $r_k := r_{k-2} \bmod r_{k-1}$

   until $r_k = 0$

Beispiel 1: $\gcd(30, 24)$

k	r_k
-1	30
0	24
1	**6**
2	0

Beispiel 2: $\gcd(100, -35)$

k	r_k
-1	100
0	$+35$

Wiederholung: Euklidischer Algorithmus

Euklids Idee:

falls $b = 0$ dann $\gcd(a, b) = |a|$

falls $b \neq 0$ dann $\gcd(a, b) = \gcd(|b|, a \bmod |b|)$

Algorithmus:

1. $r_{-1} := |a|, r_0 := |b|$
2. **for** $k = 1, \ldots$:
   - 2.1 $r_k := r_{k-2} \bmod r_{k-1}$

   until $r_k = 0$

Beispiel 1: $\gcd(30, 24)$

k	r_k
-1	30
0	24
1	**6**
2	0

Beispiel 2: $\gcd(100, -35)$

k	r_k
-1	100
0	$+35$

Wiederholung: Euklidischer Algorithmus

Euklids Idee:

falls $b = 0$ dann $\gcd(a, b) = |a|$

falls $b \neq 0$ dann $\gcd(a, b) = \gcd(|b|, a \bmod |b|)$

Algorithmus:

1. $r_{-1} := |a|, r_0 := |b|$
2. **for** $k = 1, \ldots$:
   - 2.1 $r_k := r_{k-2} \bmod r_{k-1}$

   until $r_k = 0$

Beispiel 1: $\gcd(30, 24)$

k	r_k
-1	30
0	24
1	**6**
2	0

Beispiel 2: $\gcd(100, -35)$

k	r_k
-1	100
0	$+35$
1	30

Wiederholung: Euklidischer Algorithmus

Euklids Idee:

falls $b = 0$ dann $\gcd(a, b) = |a|$

falls $b \neq 0$ dann $\gcd(a, b) = \gcd(|b|, a \bmod |b|)$

Algorithmus:

1. $r_{-1} := |a|, r_0 := |b|$
2. **for** $k = 1, \ldots$:
   - 2.1 $r_k := r_{k-2} \bmod r_{k-1}$

   until $r_k = 0$

Beispiel 1: $\gcd(30, 24)$

k	r_k
-1	30
0	24
1	**6**
2	0

Beispiel 2: $\gcd(100, -35)$

k	r_k
-1	100
0	$+35$
1	30
2	5

Wiederholung: Euklidischer Algorithmus

Euklids Idee:

falls $b = 0$ dann $\gcd(a, b) = |a|$

falls $b \neq 0$ dann $\gcd(a, b) = \gcd(|b|, a \bmod |b|)$

Algorithmus:

1. $r_{-1} := |a|, r_0 := |b|$
2. **for** $k = 1, \ldots$:
   - 2.1 $r_k := r_{k-2} \bmod r_{k-1}$

   until $r_k = 0$

Beispiel 1: $\gcd(30, 24)$

k	r_k
-1	30
0	24
1	**6**
2	0

Beispiel 2: $\gcd(100, -35)$

k	r_k
-1	100
0	$+35$
1	30
2	**5**
3	0

Erweiterung von Euklids Algorithmus

Euklidischer Algorithmus

1. $r_{-1} := |a|, r_0 := |b|$
2. **for** $k = 1, \ldots$:
   2.1 $r_k := r_{k-2} \bmod r_{k-1}$
   until $r_k = 0$

$q_k = \lfloor r_{k-2} / r_{k-1} \rfloor$

Wir sehen:

$r_{-1} = a = \mathbf{1} \cdot a + \mathbf{0} \cdot b$ $\quad \mathbf{s_{-1}}, \mathbf{t_{-1}}!$

Erweiterung von Euklids Algorithmus

Euklidischer Algorithmus

1. $r_{-1} := |a|, r_0 := |b|$
2. **for** $k = 1, \ldots$:
   2.1 $r_k := r_{k-2} \bmod r_{k-1}$

   until $r_k = 0$

$$q_k = \lfloor r_{k-2} / r_{k-1} \rfloor$$

Wir sehen:

$$\begin{aligned} r_{-1} &= a = \mathbf{1} \cdot a + \mathbf{0} \cdot b \\ r_0 &= b = \mathbf{0} \cdot a + \mathbf{1} \cdot b \end{aligned}$$

$\mathbf{s_{-1}}, \mathbf{t_{-1}}$!

$\mathbf{s_0}, \mathbf{t_0}$!

Erweiterung von Euklids Algorithmus

Euklidischer Algorithmus

1. $r_{-1} := |a|, r_0 := |b|$
2. **for** $k = 1, \ldots$:
   2.1 $r_k := r_{k-2} \bmod r_{k-1}$
   until $r_k = 0$

$$q_k = \lfloor r_{k-2} / r_{k-1} \rfloor$$

Wir sehen:

$$\begin{aligned}
r_{-1} &= a = \mathbf{1} \cdot a + \mathbf{0} \cdot b && \mathbf{s_{-1}, t_{-1}}! \\
r_0 &= b = \mathbf{0} \cdot a + \mathbf{1} \cdot b && \mathbf{s_0, t_0}! \\
r_1 &= r_{-1} - q_1 r_0 \\
&= s_{-1} a + t_{-1} b - q_1 (s_0 a + t_0 b) \\
&= (\mathbf{s_{-1}} - \mathbf{q_1 s_0}) \cdot a + (\mathbf{t_{-1}} - \mathbf{q_1 t_0}) \cdot b && \mathbf{s_1, t_1}! \\
&\vdots && \vdots
\end{aligned}$$

Also: $\mathbf{s_k} = \mathbf{s_{k-2}} - \mathbf{q_k s_{k-1}}$ und $\mathbf{t_k} = \mathbf{t_{k-2}} - \mathbf{q_k t_{k-1}}$!

Der erweiterte Euklidische Algorithmus

Erweiterter Euklid

1. $r_{-1} := |a|, r_0 := |b|$
2. $s_{-1} := 1, s_0 := 0$
3. $t_{-1} := 0, t_0 := 1$
4. **for** $k = 1, \ldots$:
   - 4.1 $r_k := r_{k-2} \bmod r_{k-1}$
   - 4.2 $q_k := \lfloor r_{k-2}/r_{k-1} \rfloor$
   - 4.3 $s_k := s_{k-2} - q_k s_{k-1}$
   - 4.4 $t_k := t_{k-2} - q_k t_{k-1}$

   until $r_k = 0$

- Es gilt stets $r_k = s_k a + t_k b$ (für alle $k \geq -1$)
- Letztes $r_k \neq 0$ liefert $s := s_k$ und $t := t_k$ mit

$$sa + tb = \gcd(a, b) = r_k$$

Erweiterter Euklidischer Algorithmus: Beispiel

Erweiterter Euklid

1. $r_{-1} := |a|, r_0 := |b|$
2. $s_{-1} := 1, s_0 := 0$
3. $t_{-1} := 0, t_0 := 1$
4. **for** $k = 1, \ldots:$
   - 4.1 $r_k := r_{k-2} \bmod r_{k-1}$
   - 4.2 $q_k := \lfloor r_{k-2}/r_{k-1} \rfloor$
   - 4.3 $s_k := s_{k-2} - q_k s_{k-1}$
   - 4.4 $t_k := t_{k-2} - q_k t_{k-1}$

   until $r_k = 0$

Beispiel: gcd(63, 22)

k	q_k	r_k	s_k	t_k
-1		63	1	0
0		22	0	1

Erweiterter Euklidischer Algorithmus: Beispiel

Erweiterter Euklid

1. $r_{-1} := |a|, r_0 := |b|$
2. $s_{-1} := 1, s_0 := 0$
3. $t_{-1} := 0, t_0 := 1$
4. **for** $k = 1, \ldots:$
   - 4.1 $r_k := r_{k-2} \bmod r_{k-1}$
   - 4.2 $q_k := \lfloor r_{k-2}/r_{k-1} \rfloor$
   - 4.3 $s_k := s_{k-2} - q_k s_{k-1}$
   - 4.4 $t_k := t_{k-2} - q_k t_{k-1}$

   until $r_k = 0$

Beispiel: gcd(63, 22)

k	q_k	r_k	s_k	t_k
-1		63	1	0
0		22	0	1
1	2	19	1	-2

Erweiterter Euklidischer Algorithmus: Beispiel

Erweiterter Euklid

1. $r_{-1} := |a|, r_0 := |b|$
2. $s_{-1} := 1, s_0 := 0$
3. $t_{-1} := 0, t_0 := 1$
4. **for** $k = 1, \ldots:$
   - 4.1 $r_k := r_{k-2} \bmod r_{k-1}$
   - 4.2 $q_k := \lfloor r_{k-2} / r_{k-1} \rfloor$
   - 4.3 $s_k := s_{k-2} - q_k s_{k-1}$
   - 4.4 $t_k := t_{k-2} - q_k t_{k-1}$

   until $r_k = 0$

Beispiel: gcd(63, 22)

k	q_k	r_k	s_k	t_k
-1		63	1	0
0		22	0	1
1	2	19	1	-2
2	1	3	-1	3

Erweiterter Euklidischer Algorithmus: Beispiel

Erweiterter Euklid

1. $r_{-1} := |a|, r_0 := |b|$
2. $s_{-1} := 1, s_0 := 0$
3. $t_{-1} := 0, t_0 := 1$
4. **for** $k = 1, \ldots:$
   - 4.1 $r_k := r_{k-2} \bmod r_{k-1}$
   - 4.2 $q_k := \lfloor r_{k-2}/r_{k-1} \rfloor$
   - 4.3 $s_k := s_{k-2} - q_k s_{k-1}$
   - 4.4 $t_k := t_{k-2} - q_k t_{k-1}$

   until $r_k = 0$

Beispiel: gcd(63, 22)

k	q_k	r_k	s_k	t_k
-1		63	1	0
0		22	0	1
1	2	19	1	-2
2	1	3	-1	3
3	6	1	7	-20

Erweiterter Euklidischer Algorithmus: Beispiel

Erweiterter Euklid

1. $r_{-1} := |a|, r_0 := |b|$
2. $s_{-1} := 1, s_0 := 0$
3. $t_{-1} := 0, t_0 := 1$
4. **for** $k = 1, \ldots:$
   - 4.1 $r_k := r_{k-2} \bmod r_{k-1}$
   - 4.2 $q_k := \lfloor r_{k-2}/r_{k-1} \rfloor$
   - 4.3 $s_k := s_{k-2} - q_k s_{k-1}$
   - 4.4 $t_k := t_{k-2} - q_k t_{k-1}$

   until $r_k = 0$

Beispiel: gcd(63, 22)

k	q_k	r_k	s_k	t_k
-1		63	1	0
0		22	0	1
1	2	19	1	-2
2	1	3	-1	3
3	6	$\boxed{\mathbf{1}}$	**7**	**-20**
4	3	0		

Folglich:

$\gcd(63, 22) = \mathbf{1} = \mathbf{7} \cdot 63 + (\mathbf{-20}) \cdot 22$

Inhaltsübersicht

Kryptografie: Wissenschaft sicherer Kommunikation

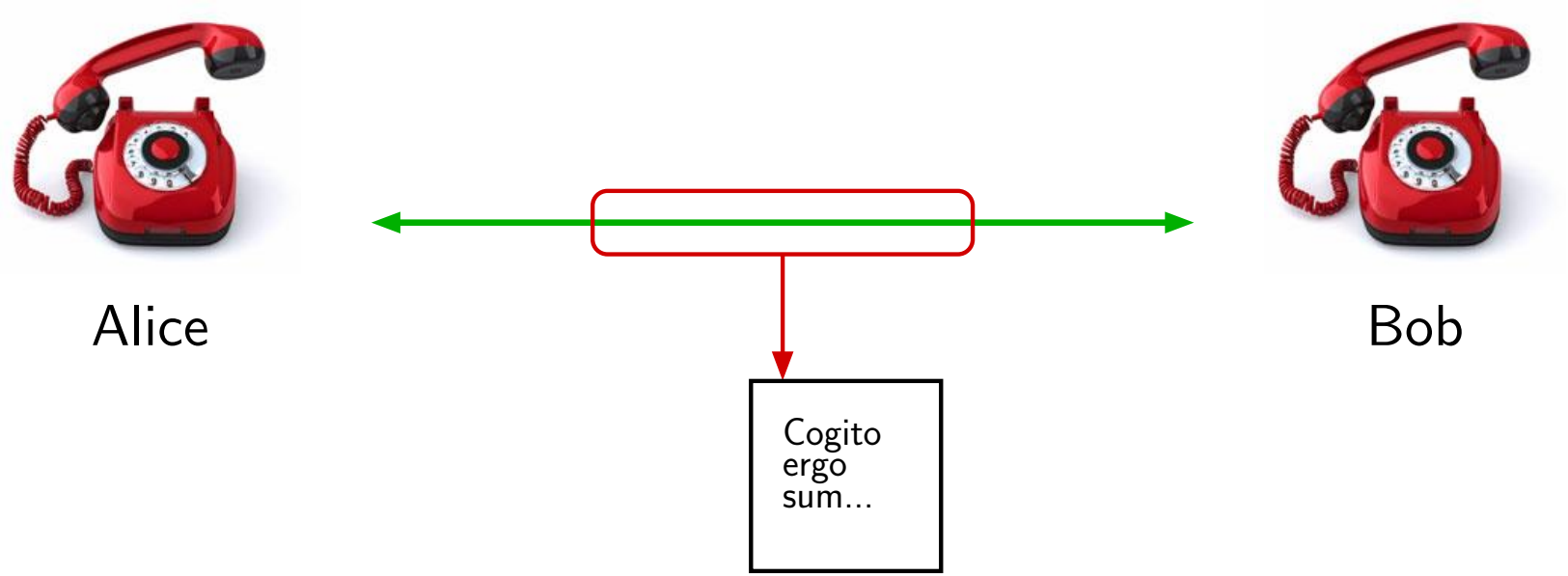
Algorithmen und Protokolle für Sicherheitsziele

Alice

Bob

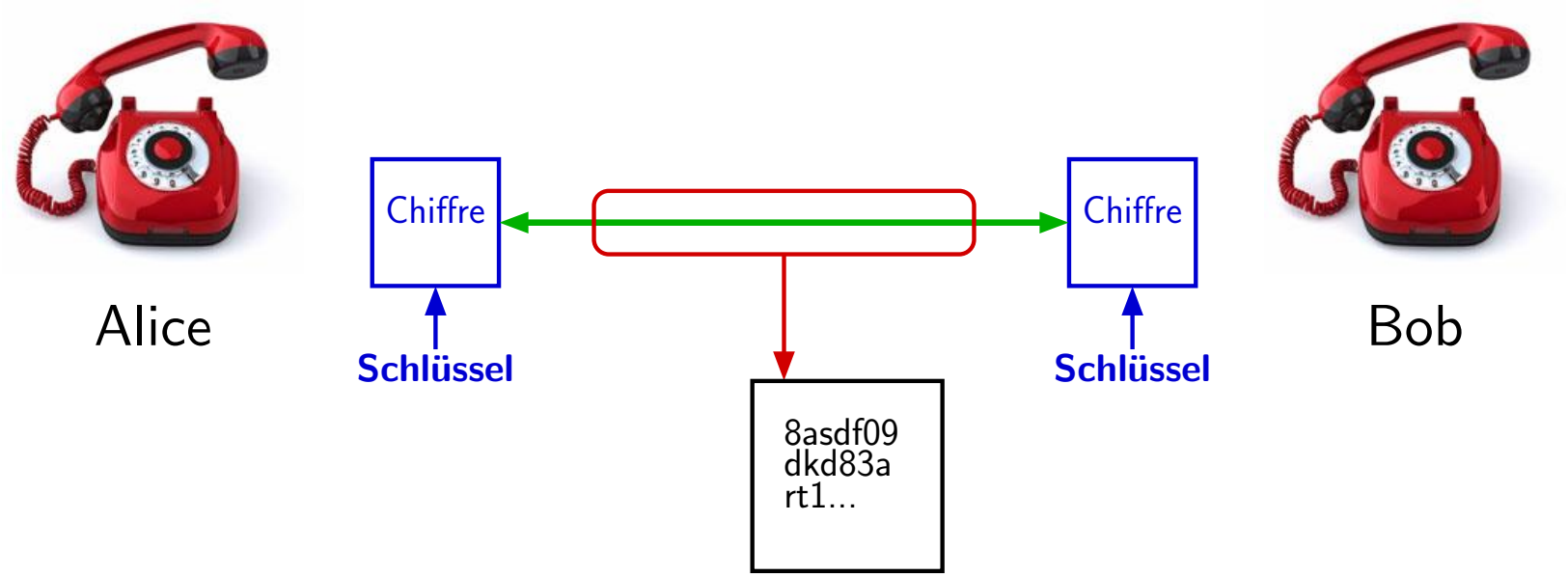
Kryptografie: Wissenschaft sicherer Kommunikation

Algorithmen und Protokolle für Sicherheitsziele



Kryptografie: Wissenschaft sicherer Kommunikation

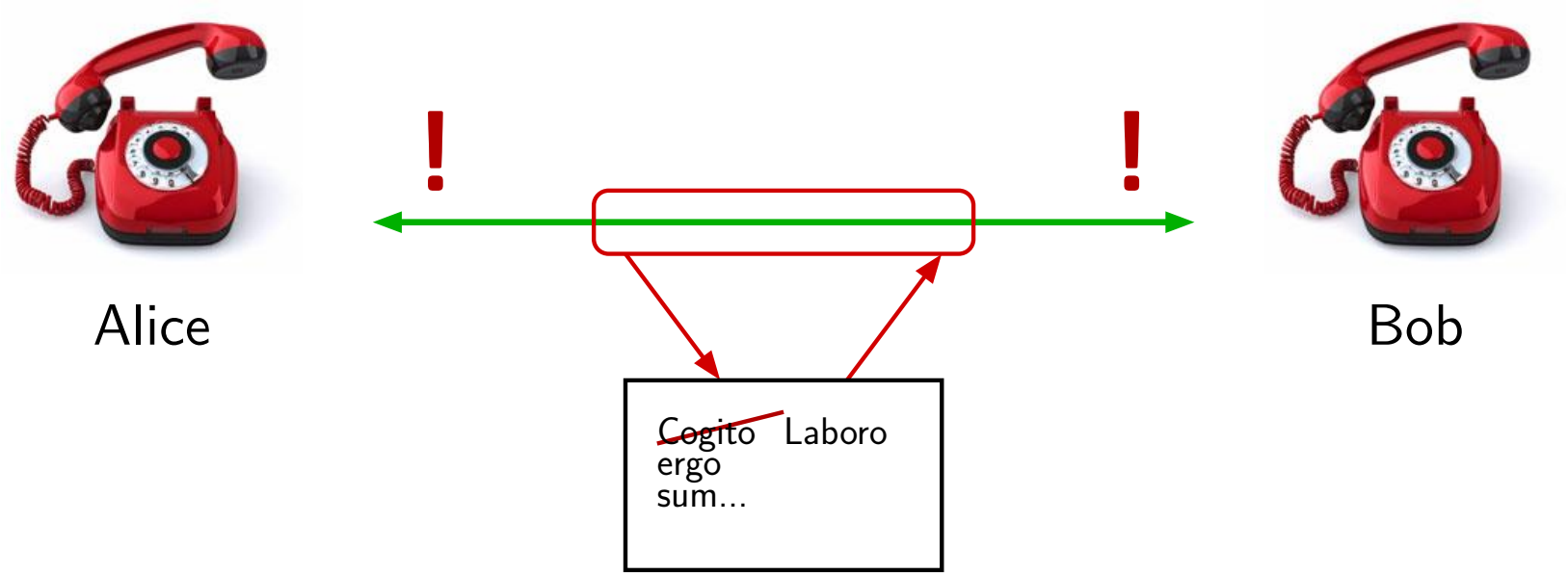
Algorithmen und Protokolle für Sicherheitsziele



Vertraulichkeit

Kryptografie: Wissenschaft sicherer Kommunikation

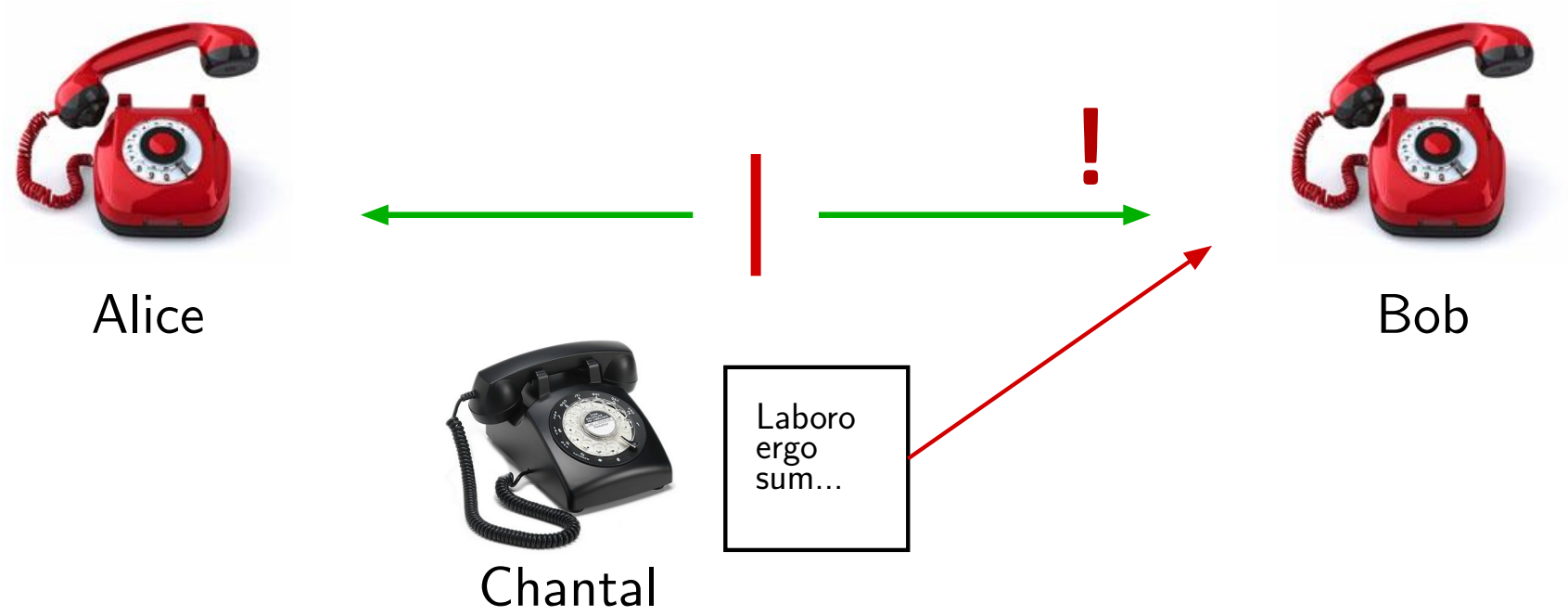
Algorithmen und Protokolle für Sicherheitsziele



Integrität

Kryptografie: Wissenschaft sicherer Kommunikation

Algorithmen und Protokolle für Sicherheitsziele



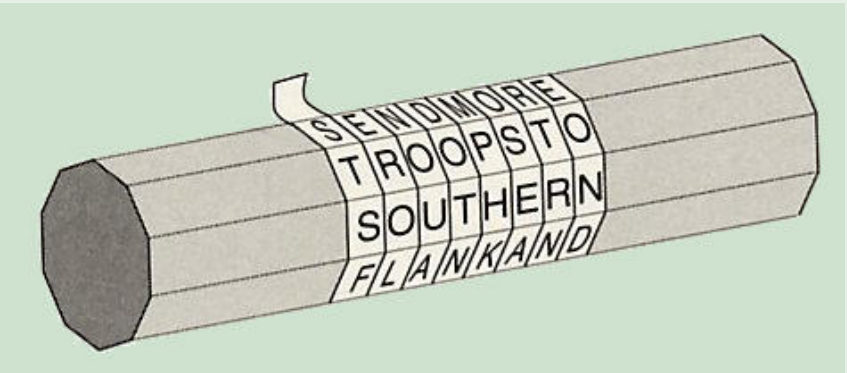
Authentizität

Sicherheit von Verschlüsselungen

Kerckhoffs' Prinzip (1883)

Alles über die Chiffre ist bekannt bis auf den geheimen **Schlüssel**.

Antikes Beispiel: Skytale



Schlüssel ist einziger unbekannter Parameter: muss lang genug sein, um Brute-Force-Angriffe zu verhindern.

Sicherheitsniveaus

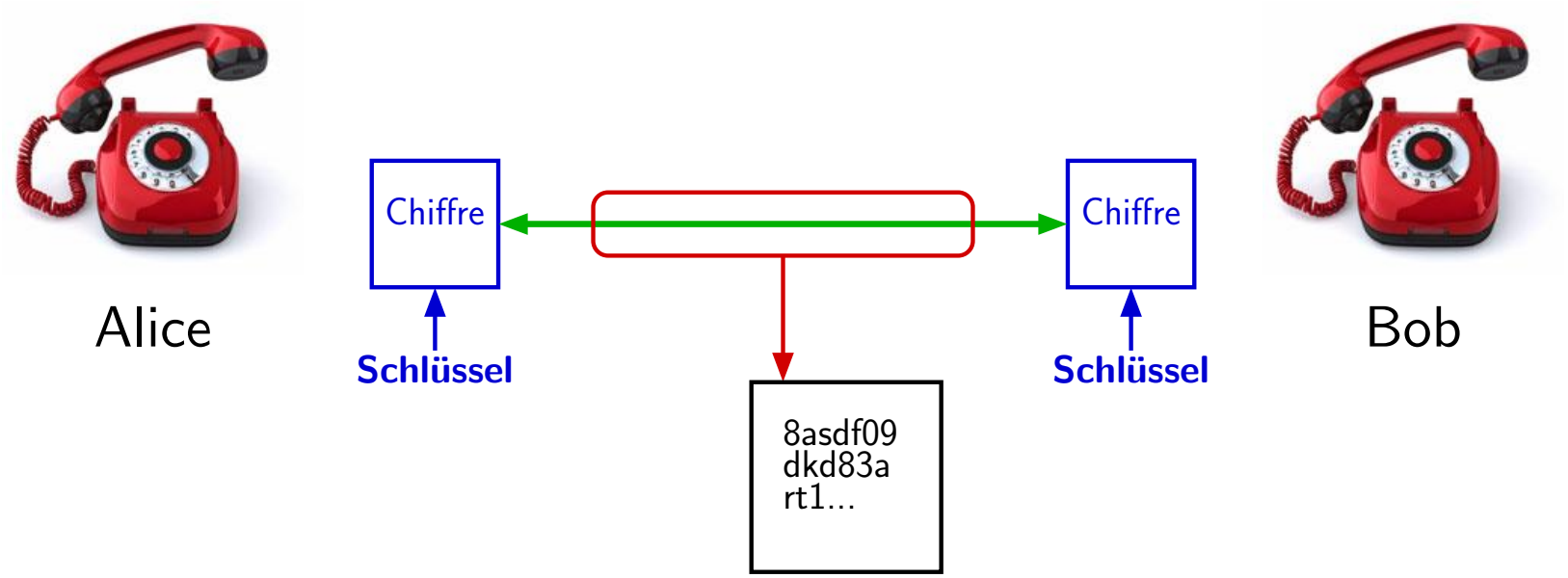
Das Sicherheitsniveau einer Chiffre wird ausgedrückt in der Anzahl Operationen, die ein Angreifer zum Brechen mindestens ausführen muss.

Sicherheitslevel [bits]	10^9 ops/s	100'000 Cores
16	0.07 ms	0.7 ns
32	4.29 s	0.004 ms
64	585 Jahre	5 h
128	10^{22} Jahre	10^{16} Jahre

Heutzutage: Sicherheitsniveau von 128 oder 256 Bits

Inhaltsübersicht

Vertraulichkeit durch Verschlüsselung

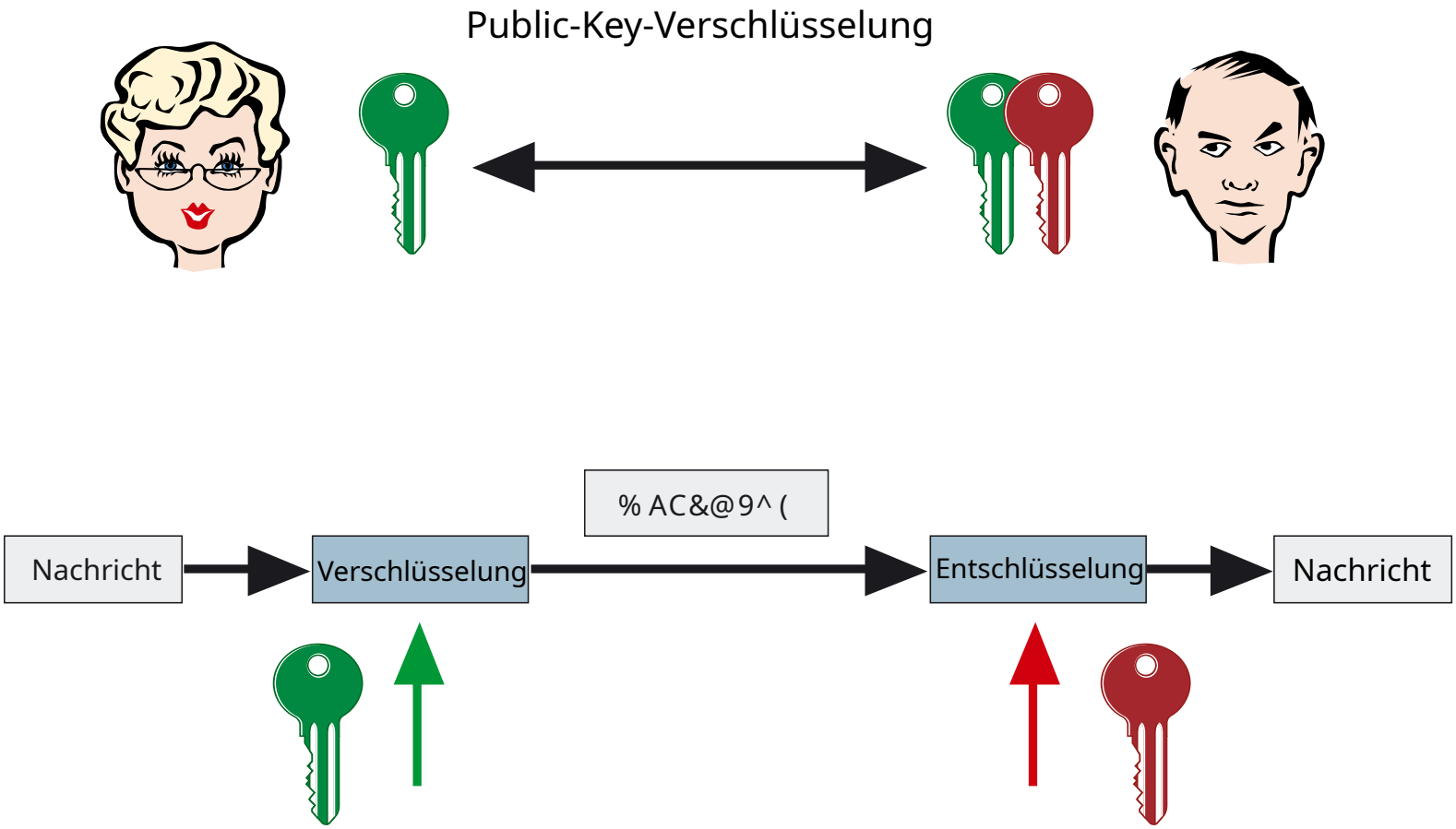


Symmetrische Verschlüsselung

- Ver- und Entschlüsselung benutzen den gleichen Schlüssel.
- Nutzer tauschen (paarweise) auf sichere Weise Schlüssel aus.
- Bei n Nutzern sind das $\approx n^2/2$ Schlüssel!

Merkle, Hellman, Diffie (1976)

Public-Key-Verschlüsselung I



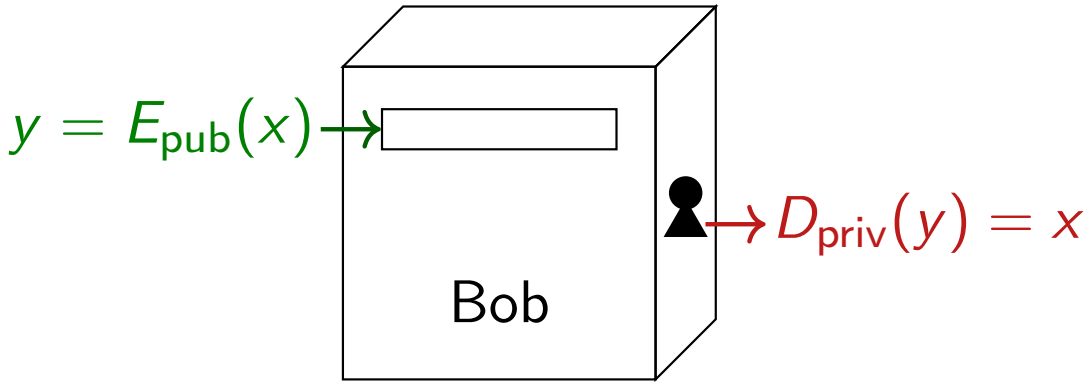
Public-Key-Verschlüsselung II

Eigenschaften

- Jeder Nutzer hat ein Schlüsselpaar bestehend aus privatem und öffentlichem Schlüssel
- Verschlüsselung mit öffentlichem, Entschlüsselung mit privatem Schlüssel
- Privater aus öffentlichem Schlüssel nicht berechenbar

Public-Key-Verschlüsselung: Analogien

Briefkasten:



Vorhängeschloss:

- Öffentlicher Schlüssel: 🔓, privater Schlüssel: passender 🔑
- Verschlüsselung: Nachricht in Kiste, diese mit Vorhängeschloss verschließen: 🔓 → 🔒
- Entschlüsselung: Schloss 🔒 mit 🔑 öffnen, Nachricht herausholen

Inhaltsübersicht

Rivest, Shamir, Adleman (1977)

"SRA"

Das RSA-Public-Key-Verschlüsselungsverfahren I

Schlüsselgenerierung

1. Wähle zwei zufällige Primzahlen $p \neq q$ und setze $n = pq$
2. Setze $\phi = (p-1)(q-1)$
3. Wähle eine kleine Zahl e mit $\gcd(e, \phi) = 1$
4. Berechne d aus

$$ed \equiv 1 \pmod{\phi},$$

also $d = e^{-1} \bmod \phi$.

Schlüsselpaar

- Der öffentliche Schlüssel ist nun (n, e),
- der private Schlüssel (p, q, d).

Das RSA-Public-Key-Verschlüsselungsverfahren II

Verschlüsselung

Die Verschlüsselung von $m \in \mathbb{Z}_n$ ist

$$c = m^e \bmod n.$$

Entschlüsselung

Die Entschlüsselung von $c \in \mathbb{Z}_n$ ist

$$m = c^d \bmod n.$$

→ Demo

Schlüssellänge für RSA

Empfohlene Schlüssellänge

- Modulus n von mindestens 2048 Bits (oft auch 4096)
- d.h. Primzahlen p, q von ungefähr 1024 Bits (resp. 2048)

Beispiel (2048-Bit RSA-Modulus)

25195908475657893494027183240048398571429282126204032027771378360436
62020707595556264018525880784406918290641249515082189298559149176184 5
02808489120072844992687392807287776735971418347270261896375014971824 6
91165077613379859095700097330459748808428401797429100642458691817195 1
18746121515172654632282216869987549182422433637259085141865462043576 7
98423387184774447920739934236584823824281198163815010674810451660377 3
06056201619676256133844143603833904414952634432190114657544454178424 0
20924616515723350778707749817125772467962926386356373289912154831438 1
67899885040445364023527381951378636564391212010397122822120720357

Warum so lange Schlüssel?

- Faktorisierung von n effizienter als Brute Force
- `en.wikipedia.org/wiki/RSA_Factoring_Challenge`

Korrektheit von RSA: Warum ist $m^{ed} \equiv m \pmod{n}$? I

Zur Erinnerung: $n = pq$, $\phi = (p-1)(q-1)$, $ed \equiv 1 \pmod{\phi}$

Lemma

Gilt für Primzahlen p, q, dass

$$x \equiv a \pmod{p} \quad \text{und} \quad x \equiv a \pmod{q},$$

dann gilt auch

$$x \equiv a \pmod{pq}.$$

Beweis.

$x \equiv a \pmod{p}$ und $x \equiv a \pmod{q}$ bedeutet
$p \mid (a-x)$ und $q \mid (a-x)$,
also $a - x = kp$ und $a - x = lq$ für $k, l \in \mathbb{Z}$.

Wegen Eindeutigkeit der Primfaktorzerlegung heißt das aber

$$a - x = kp = k'qp = k'pq = lq \quad \text{für } k' \in \mathbb{Z},$$

also $pq \mid (a-x)$ und damit $x \equiv a \pmod{pq}$. □

Korrektheit von RSA: Warum ist $m^{ed} \equiv m \pmod{n}$? II

Theorem (Kleiner Satz von Fermat)

Sei p prim. Für alle $0 < a \leq p - 1$ gilt

$$a^{p-1} \equiv 1 \pmod{p}.$$

Korrektheit von RSA: Warum ist $m^{ed} \equiv m \pmod{n}$? III

Beweis.

Betrachte die Menge $A = \{1, 2, \ldots, p-1\}$.

Multiplikation dieser Zahlen modulo p mit einem $0 < a \leq p-1$ ergibt die Menge $B = \{a, 2a, \ldots, (p-1)a\}$.
Alle Zahlen in B sind ungleich 0 mod p und paarweise verschieden (denn wegen $\gcd(a, p) = 1$ ist a modulo p invertierbar). Also gilt $A = B$.

Dann ist aber das Produkt aller Zahlen in A und B gleich und wir erhalten:

$$\begin{aligned} 1 \cdot 2 \cdot \cdots \cdot (p-1) &\equiv a \cdot 2a \cdot \cdots \cdot (p-1)a \\ &\equiv a^{p-1}(1 \cdot 2 \cdot \cdots \cdot (p-1)) \pmod{p}, \end{aligned}$$

also $1 \equiv a^{p-1} \pmod{p}$. □

Korrektheit von RSA: Warum ist $m^{ed} \equiv m \pmod{n}$? IV

Zur Erinnerung: $n = pq$, $\phi = (p-1)(q-1)$, $ed \equiv 1 \pmod{\phi}$, also $ed = k(p-1)(q-1) + 1$.

Beweisen nun $m^{ed} \equiv m \pmod{p}$:

1. $m \equiv 0 \pmod{p} \Rightarrow 0^{ed} = 0 \pmod{p}$, ✔
2. $m \not\equiv 0 \pmod{p} \Rightarrow m^{k(p-1)(q-1)+1} \equiv m \pmod{p}$. ✔

Analog für $m^{ed} \equiv m \pmod{q}$.

Insgesamt gilt also $m^{ed} \equiv m \pmod{n}$. □

Warum arbeiten wir im Exponenten modulo ϕ?

Zur Erinnerung: $n = pq$, $\phi = (p-1)(q-1)$, $ed \equiv 1 \pmod{\phi}$

Lemma

$$\phi = \#\{0 \leq x < pq \mid \gcd(x, pq) = 1\} = (p-1)(q-1)$$

Beweis.

Einzig mögliche Teiler von pq sind Primzahlen p und q.

- In $\mathbb{Z}_n$ sind nur $0, p, 2p, \ldots, (q-1)p$ nicht teilerfremd zu p.
- In $\mathbb{Z}_n$ sind nur $0, q, 2q, \ldots, (p-1)q$ nicht teilerfremd zu q.
- Also ist $\phi = n - q - p + 1 = pq - (p+q) + 1 = (p-1)(q-1)$

□

Modulo n haben also nur diese ϕ vielen Zahlen ein multiplikatives Inverses.

Effiziente Implementierung von RSA

Naive Berechnung von $z := x^y \bmod n$ erfordert y Multiplikationen.
d ist jedoch mindestens ungefähr 2^{2048}.

Idee:

Sei $y = y_\ell \dots y_0$ die Binärrepräsentation von y, also

$$y = 2^\ell y_\ell + \cdots + 2^2 y_2 + 2y_1 + y_0 \qquad \text{mit } y_i \in \{0, 1\}$$

dann

$$z := x^y = ((((((\cdots(x^{y_\ell})^2)x^{y_{\ell-1}})^2 \cdots)x^{y_1})^2 x^{y_0} \bmod n$$

Square-and-Multiply-Algorithmus

1. $z := 1$
2. **for** $i := \ell$ **downto** 0 **do**
   - 2.1 $z := z^2 \bmod n$
   - 2.2 **if** $y_i = 1$ **then** $z := z \cdot x \bmod n$

Beispiele für Square-and-multiply

	S	M	Rechensequenz	binärer Exponent
x^{16}	4	0	x, x^2, x^4, x^8, x^{16}	10000
x^{17}	4	1	$x, x^2, x^4, x^8, x^{16}, x^{17}$	10001
x^{18}	4	1	$x, x^2, x^4, x^8, x^9, x^{18}$	10010
x^{19}	4	2	$x, x^2, x^4, x^8, x^9, x^{18}, x^{19}$	10011
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
x^{31}	4	4	$x, x^2, x^3, x^6, x^7, x^{14}, x^{15}, x^{30}, x^{31}$	11111
x^{32}	5	0	$x, x^2, x^4, x^8, x^{16}, x^{32}$	100000

Für k-bit Exponenten: höchstens k Quadrierungen (S) und k Multiplikationen (M)

Faktorisierung von $n = pq$ bricht RSA

Kenntnis der Faktoren p, q ermöglicht Berechnung des geheimen Exponenten d:

1. Berechne $\phi = (p-1)(q-1)$
2. Mit e vom öffentlichen Schlüssel: berechne $d = e^{-1} \bmod \phi$ mit erweitertem Euklid

Faktorisierungsproblem allgemein

- Bester Algorithmus ist das Generalized Number Field Sieve (GNFS)
- Februar 2020: 829-Bit Modulus in 2700 Core-Jahren faktorisiert
- Quantencomputer können sehr effizient faktorisieren (Sekunden)

Wie faktorisiert man Zahlen?

Angenommen, wir wollen RSA-Modulus $n = pq$ faktorisieren.

Probedivision

Alle möglichen Teiler durchprobieren braucht $\sqrt{n}$ Operationen.

Faktorisierungsidee (Pollard-Rho)

Finde $x \neq x' \in \mathbb{Z}_n$ mit

$$x \equiv x' \pmod{p}.$$

Dann ist $p \leq \gcd(x - x', n) < n$ ein Faktor von n.

→ Demo

Braucht ungefähr $\sqrt{p} \leq \sqrt[4]{n}$ Operationen.
GNFS: mindestens 2^{95} Operationen für 2048-Bit n.

Verwendung von RSA

Verschlüsseln mit RSA

- RSA wird nur zum Verschlüsseln symmetrischer Schlüssel verwendet. Mit diesen verschlüsselt man dann Daten (z.B. mit dem Advanced Encryption Standard, AES).
- "Textbook" RSA wie hier beschrieben sollte nie verwendet werden (zuviel mathematische Struktur).
- Standards wie RSA-OAEP oder RSA-PSS lösen dieses Problem.