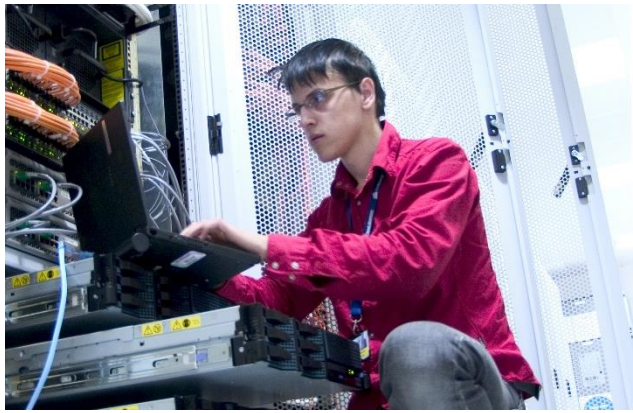
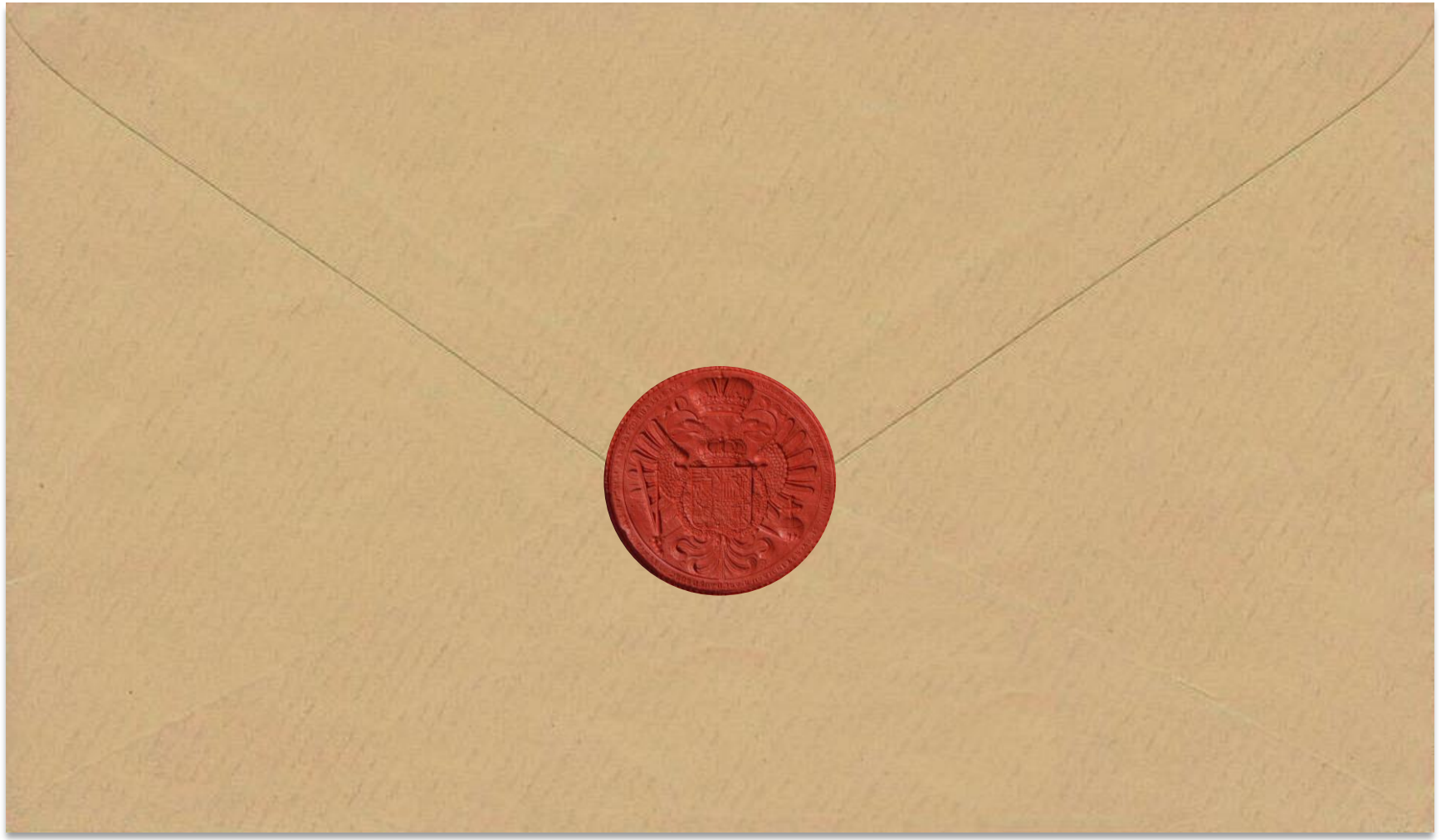
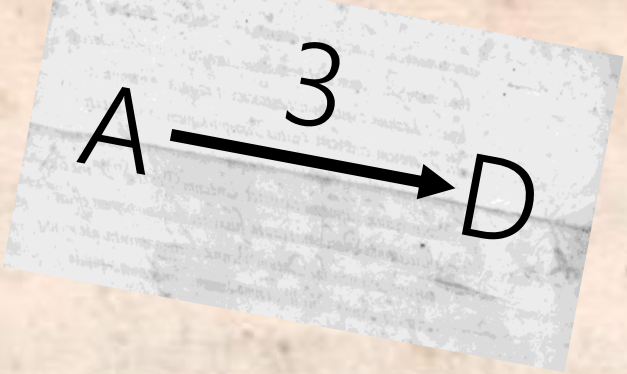
EMIL-VON-BEHRING GYMNASIUM

Codierung und Verschlüsselung
Symmetrische und Asymmetrische Verschlüsselung





YHQL, YLGL, YLFL!



XQVHU NDLVHU MXOLXV FDHVDU KDW
LP RVWHQ JURVVH HUIROJH
HUCLHKHQ NRHQQHQ.
QXU VHLQHU VWUDWHJLVFKHQ
IDHKIJNHLWHQ KDEHQ ZLU HV CX
YHUGDQNHQ, GDVV GLHVHU
JURVVDUWLJH VLHJ JHOLHQJHQ

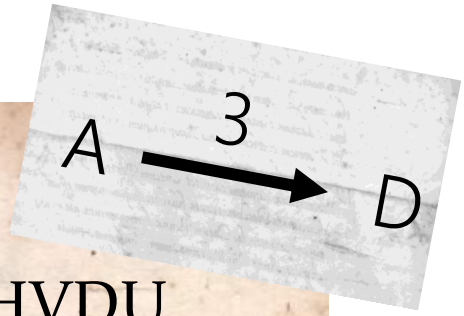
Aufgabe:

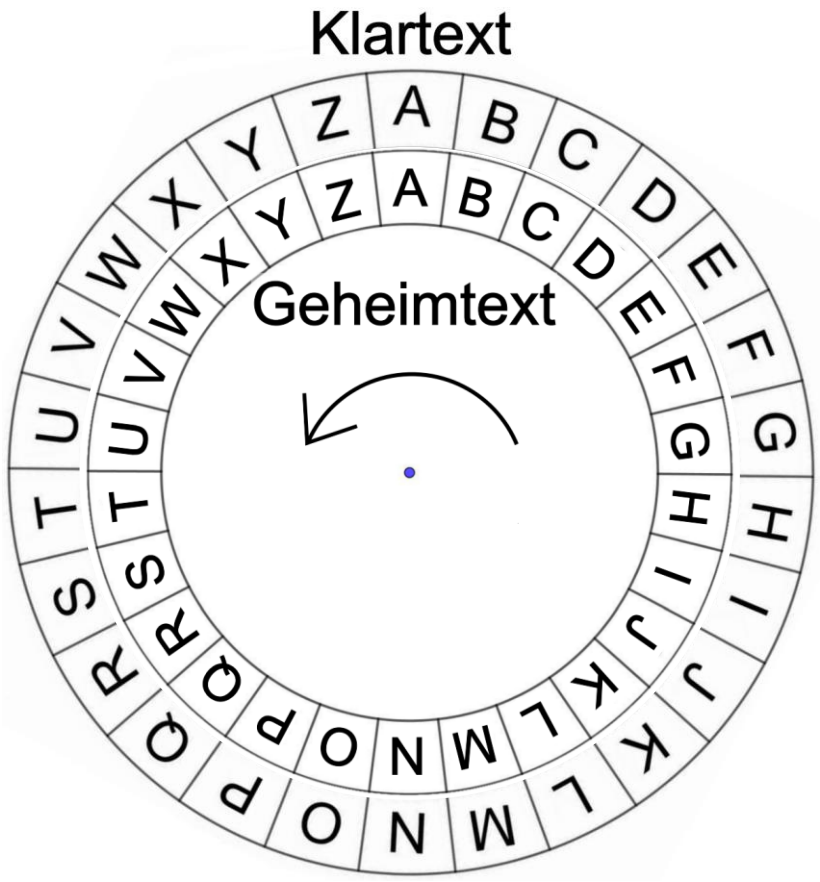
- Versuche mithilfe der Scheibe die erste Zeile des Briefs zu entziffern.

YHQL, YLGL, YLFL!

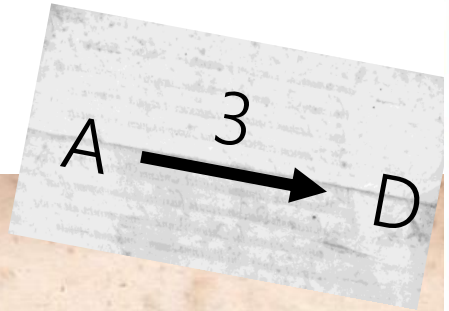
XQVHU NDLVHU MXOLXV FDHVDU
KDW LP RVWHQ JURVVH HUIROJH
HUCLHKHQ NRHQQHQ.

QXU VHLQHU VWUDWHJLVFKHQ
IDHKIJNHLWHQ KDEHQ ZLU HV CX
YHUGDQNHQ, GDVV GLHVHU
JURVVDUWLJH VLHJ JHOLHQJHQ
NRQQWH.





Ein kleiner Tipp…



YHQL, YLGL, YLFL!

XQVHU NDLVHU MXOLXV FDHVDU KDW LP RVWHQ JURVVH HUIROJH HUCLHKHQ NRHQQHQ.

QXU VHLQHU VWUDWHJLVFKHQ IDHKIJNHLWHQ KDEHQ ZLU HV CX YHUGDQNHQ, GDVV GLHVHU JURVVDUWLJH VLHJ JHOLHQJHQ NRQQWH.

Caesar-Verschlüsselung

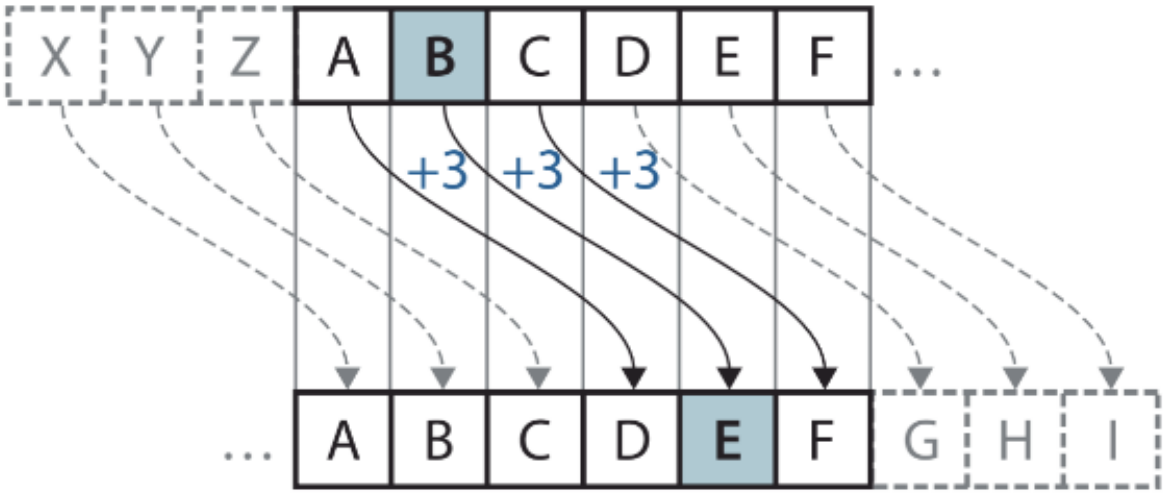
- Verschlüsseln ist das Codieren eines Klartextes mithilfe eines Schlüssels in einen Geheimtext, um Informationen geheim zu speichern oder zu übertragen.
- Die Caesar-Verschlüsselung ist ein einfaches Verschlüsselungsverfahren, bei dem man eine Zahl als Schlüssel wählt und jeden Buchstaben der Nachricht um diesen Wert im Alphabet verschiebt.

- Beispiel:

  Klartext: VENI, VIDI, VICI.

  Schlüssel: 3

  Geheimtext: YHQL, YLGL, YLFL.



Übung Caesar-Verschlüsselung

1) Kaiser Augustus nutzte die Caesar-Verschlüsselung mit Schlüssel 7. Verschlüssele folgenden Klartext mithilfe der Caesar-Scheibe: AUGUSTUS
2) Verschlüssele dein Lieblingstier mit einem beliebigen Schlüssel. Notiere den Schlüssel sowie den Geheimtext auf dem Notizzettel.
3) Diskutiere mit deinem Nachbarn, wie die Verschiebung eines Buchstabens in Java implementiert werden kann.

ASCII-Codierung	Zeichen
…	…
65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O
80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X
89	Y
90	Z
…	…

Implementierung Caesar-Verschlüsselung

Vereinfachung: erstmal nur Großbuchstaben

- Schritt 1: Codierung des Zeichens als Zahl nach ASCII-Codierung
- Schritt 2: Verschlüsselung: Addieren des Schlüssels zur Zahl

  Entschlüsselung: Subtrahieren des Schlüssels von der Zahl
- Schritt 3: Achtung: Wir müssen im ASCII-Bereich 65 – 90 bleiben!
- Schritt 4: Dekodierung der Zahl als Zeichen

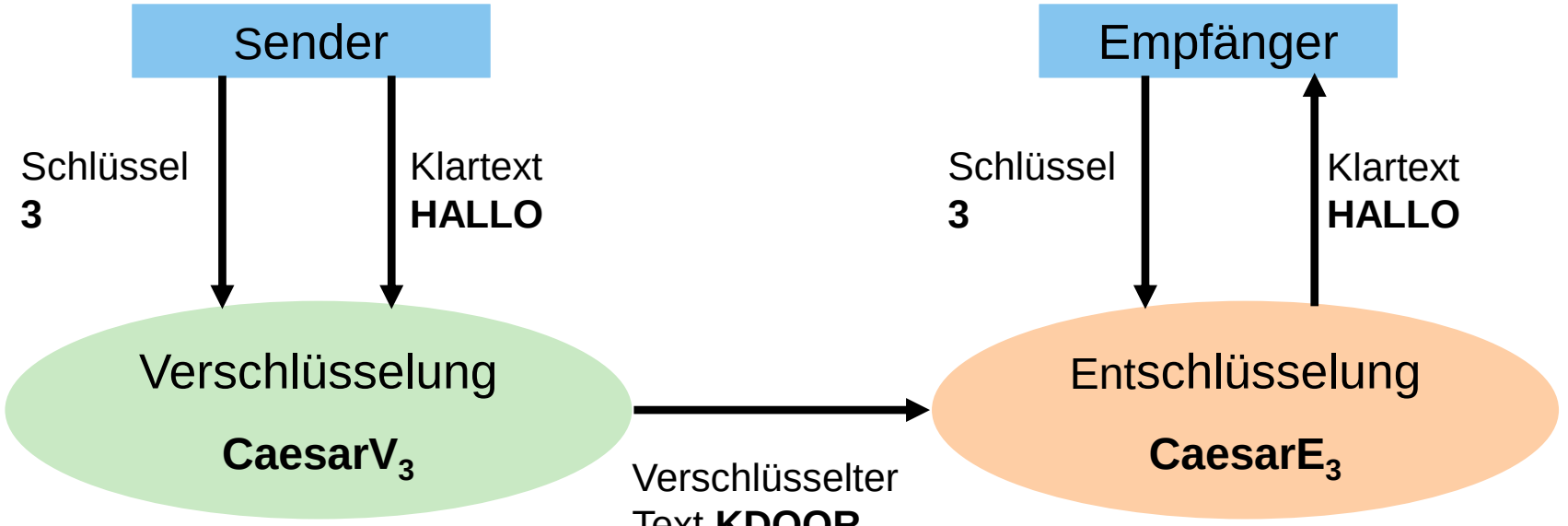
Aufgabe:
Entschlüssele das Wort auf deinem Zettel mit deinem Code.

ASCII-Codierung	Zeichen
…	…
65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O
80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X
89	Y
90	Z
…	…

Implementierung Caesar-Verschlüsselung

- Implementiere die Methode zeichenVerschlüsseln().
- Diskutiere die Funktionsweise der Methode wortEntschlüsseln() mit deinem Nachbarn.
- Implementiere die Methode wortVerschlüsseln().
- Bonus: Implementiere, dass Leerzeichen nicht ver- oder entschlüsselt werden, sondern unverändert an die Variable klartext angehängt werden.
- Bonus: Implementiere, dass alle Zeichen außerhalb des ASCII-Bereichs der großen Buchstaben unverändert an die Variable klartext angehängt werden.
- Bonus: Implementiere das Ver- und Entschlüsseln von groß- und kleingeschriebenen Buchstaben.

Caesar als Beispiel für symmetrische Verschlüsselung



$CaesarV_3$(“HALLO“) = “KDOOR“

$CaesarE_3$(“KDOOR“) = “HALLO“

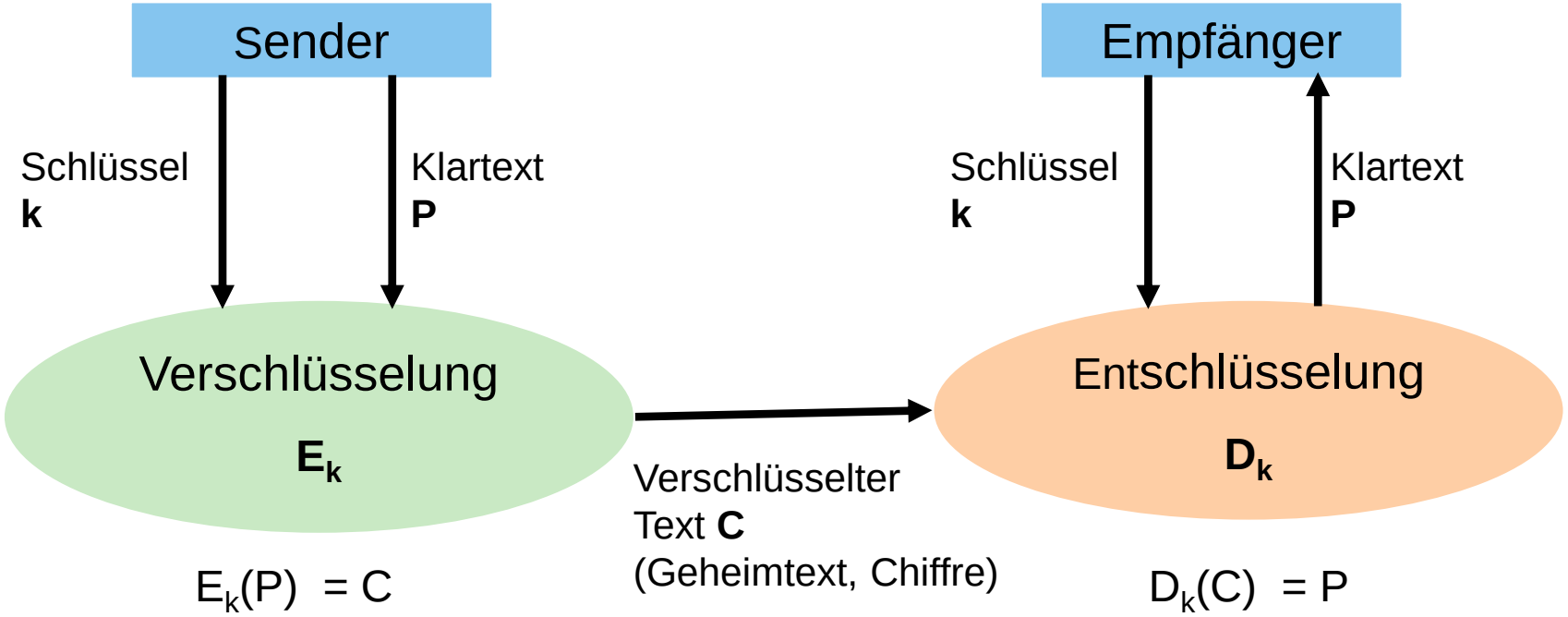
Gleicher Schlüssel für Verschlüsselung und Entschlüsselung
→ symmetrische Verschlüsselung

→ Alternativ könnte man auch mit $CaesarV_{23}$ entschlüsseln, sofern der Überlauf passend behandelt wird.

Entschlüsselungsgleichung::

$CaesarE_3(CaesarV_3$(“HALLO“)) = “HALLO“

Symmetrische Verschlüsselung



P = plain text (Klartext)

C = cipher text (Geheimtext, Schlüsseltext)

k = key (Schlüssel für die Ver-/Entschlüsselungsfunktion)

E = encrypt (Verschlüsselungsfunktion)

D = decrypt (Entschlüsselungsfunktion)

Entschlüsselungsgleichung::

$$D_k(E_k(P)) = P$$

Unter dem Begriff Symmetrische Verschlüsselung fasst man alle Kryptosysteme zusammen, bei welchen zur Ver- und Entschlüsselung derselbe Schlüssel verwendet wird.

Angriff auf die Caesar-Verschlüsselung

1. Möglichkeit: Brute-Force-Methode → 26 Möglichkeiten lassen sich schnell durchprobieren → Implementierung in Java

- Brute-Force-Methode (Methode der rohen Gewalt oder erschöpfende Methode) ist eine Lösungsmethode für informatische Probleme durch das systematische Ausprobieren aller Möglichkeiten/Fälle.

2. Möglichkeit: Häufigkeitsanalyse des Textinhalts

- Das Zeichen e ist mit Abstand der häufigste Buchstabe im Deutschen. Damit sollte der am häufigsten vorkommende Zeichen im Schlüsseltext auf das e im Klartext darstellen. Daraus lässt sich die Verschiebung bestimmen.

- Weitere Ideen: Erster Buchstabe in einem Dreizeichenwort → D, E am Ende oft gefolgt von einem N, Wenige Buchstaben kommen doppelt vor (N, M, T)

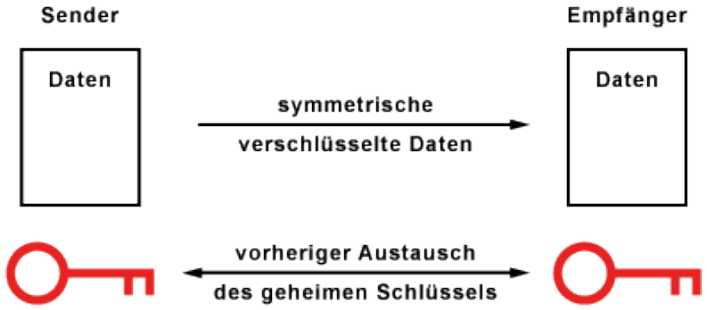
Zeichen	englisch	deutsch	Zeichen	englisch	deutsch
A	8.04	6.47	N	7.09	9.84
B	1.54	1.93	O	7.60	2.98
C	3.06	2.68	P	2.00	0.96
D	3.99	4.83	Q	0.11	0.02
E	12.51	17.48	R	6.12	7.54
F	2.30	1.65	S	6.54	6.83
G	1.96	3.06	T	9.25	6.13
H	5.49	4.23	U	2.71	4.17
I	7.26	7.73	V	0.99	0.94
J	0.16	0.27	W	1.92	1.48
K	0.67	1.46	X	0.19	0.04
L	4.14	3.49	Y	1.73	0.08
M	2.53	2.58	Z	0.09	1.14

Symmetrische Verschlüsselung

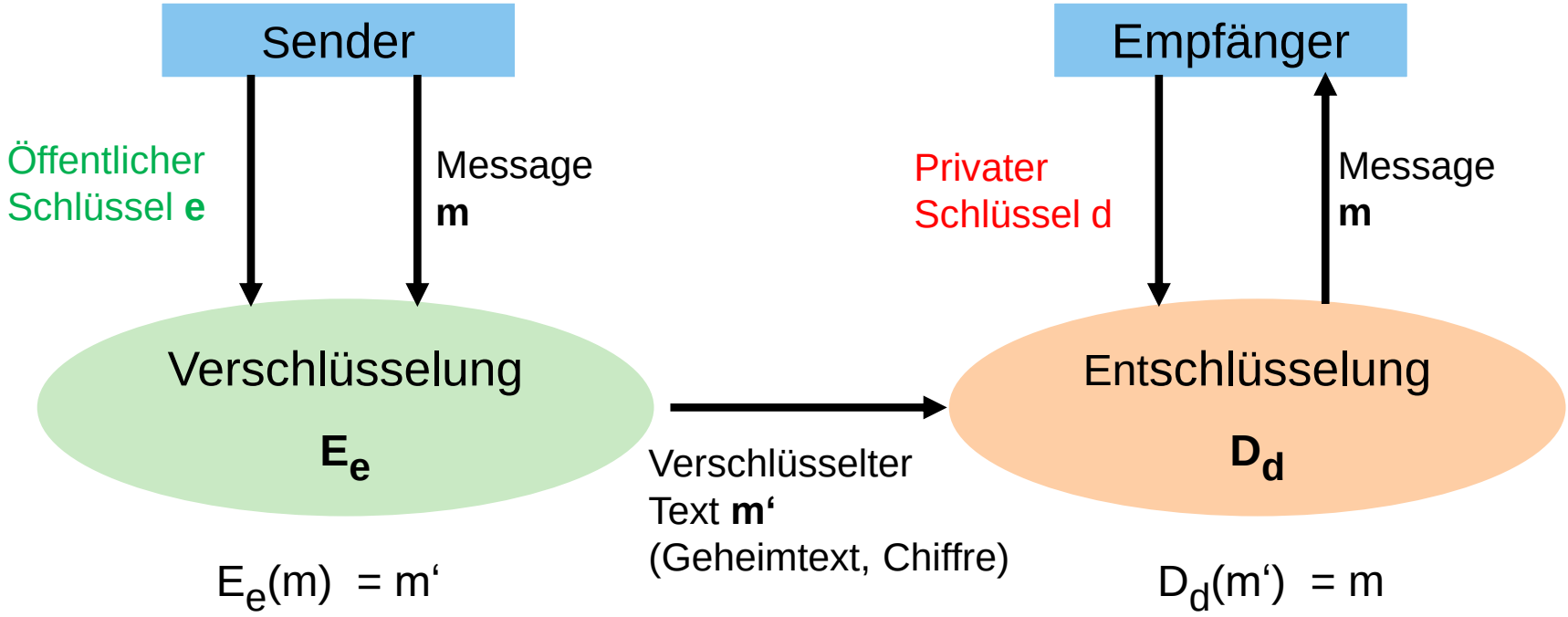
- Es gibt natürlich auch sichere symmetrische Verschlüsselungen! → Advanced Encryption Standard (z. B. AES-256)
- Aber auch bessere/komplexere/sichere symmetrische Kryptosysteme haben Nachteile:
- Der Sender verschlüsselt also die Nachricht mit dem gemeinsamen Schlüssel. Der Geheimtext wird anschließend übermittelt und der Empfänger dekodiert diesen mit demselben Schlüssel. Das bedeutet auch, dass dieser Schlüssel ebenfalls übermittelt, aber unbedingt geheim bleiben muss. Darin liegt auch ein Hauptproblem: Sender und Empfänger müssen sich auf einen gemeinsamen Schlüssel einigen, ohne dass ein Dritter auf den Schlüssel zugreifen kann.

Zusammenfassung:

- Wie kommt der Empfänger an den Schlüssel?
- Der Schlüssel muss unbedingt für immer geheim bleiben!



Asymmetrische Verschlüsselung



$$D_d(E_e(m)) = m$$

- Ein öffentlicher Schlüssel **e** zum Verschlüsseln (darf jeder bekommen)
- Ein privater Schlüssel **d** zum Entschlüsseln (bleibt geheim)
- → Wichtig: es soll quasi unmöglich sein, dass man d aus Kenntnis von e herleitet/berechnet!

RSA als Beispiel für eine asymmetrische Verschlüsselung

Das wohl bekannteste Verschlüsselungsverfahren RSA geht auf Rivest, Shamir und Adleman (1977) zurück – daher die Bezeichnung.

- **Wähle zwei ungerade Primzahlen p und q:**
- **Berechne N = pq**
- **Wähle eine natürliche Zahl e aus (e > 1), sodass ggT(e , (p-1)(q-1)) = 1**
- **Bestimme ein inverses Element d von e mod (p-1)(q-1), sodass ed = 1 mod (p-1)(q-1)**

→ Öffentlicher Schlüssel (n, e)

→ Privater Schlüssel (n, d)

RSA

→ Öffentlicher Schlüssel (n, e)

→ Privater Schlüssel (n, d)

- Verschlüsseln der Nachricht p:

$$m' = E_e(p) = m^e \bmod n$$

- Entschlüsseln der Nachricht c:

$$m = D_d(m') = (m')^d \bmod n$$

- Entschlüsselungsgleichung:

$$\mathbf{m = (m^e)^d \bmod n \quad (Modularer\ Kehrwert)}$$

Beispiel zur Schlüsselgenerierung bei RSA

1. Wähle zwei Primzahlen — **p=7** und **q=11**
2. Berechne — **n** = p * q = **77**
3. Wähle e so, dass e teilerfremd zu (p-1) * (q-1) **= 60** ist: — z. B. **e = 47**
4. Wähle d so, dass (e * d) mod (p-1) * (q-1) = 1 gilt: — → **d = 23**
5. Setze **e = (47, 77)** und **d = (23, 77)**

- Verschlüsseln der Nachricht m=**2** durch den Sender:

6. m‘ = $\mathbf{E_e(2)} = 2^{47} \bmod 77 = \mathbf{18}$

- Entschlüsseln der Nachricht m‘ durch den Empfänger:

7. $\mathbf{D_d(18)} = 18^{23} \bmod 77 = \mathbf{2} = m$

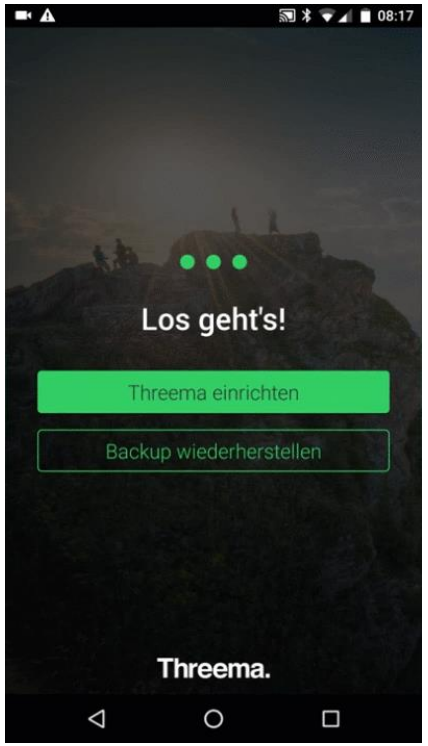
Für die Berechnung großer Potenzen benutzt man das effiziente Square-And-Multiply-Verfahren (binäre Exponentitation), welches die Potenz schrittweise berechnet. → Siehe RSA.xlsx

Übung

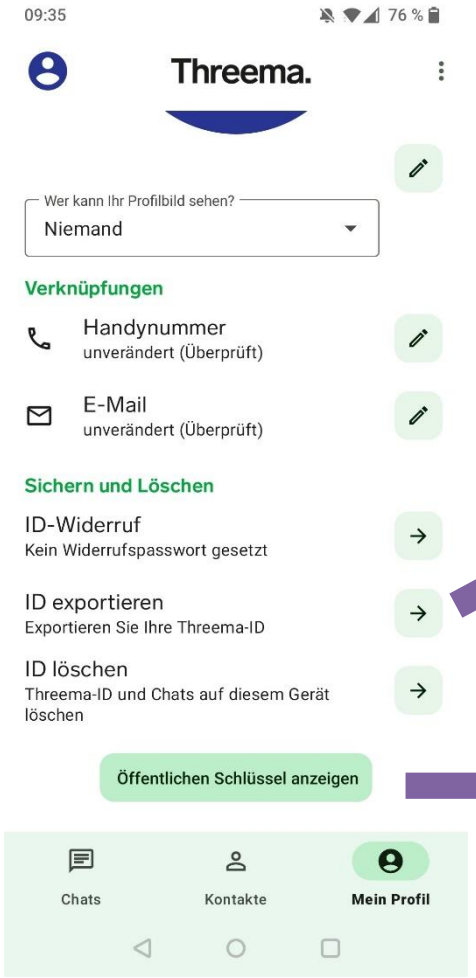
1. Öffne die Datei RSA.xlsx und generiere dir ein gültiges Schlüsselpaar.
2. Trage deinen öffentlichen Schlüssel in die zentrale Klassenliste ein.
3. Klicke auf das Tabellenblatt „Ver- und Entschlüsseln mit RSA“ in derselben Excel-Datei
4. Suche dir einen Partner/Empfänger deiner Nachricht und trage seinen öffentlichen Schlüssel in dem Tabellenblatt bei N und e ein. (N > e)
5. Überlege dir ein Wort mit vier Buchstaben und codiere es in ASCII und verschlüssele jeden Buchstaben einzeln mit dem Tabellenblatt.
6. Gib deinem Partner das verschlüsselte Wort!
7. Entschlüssele umgekehrt an dich übergebene Wörter mit deinem privaten Schlüssel N und d.

Beispiel App Threema

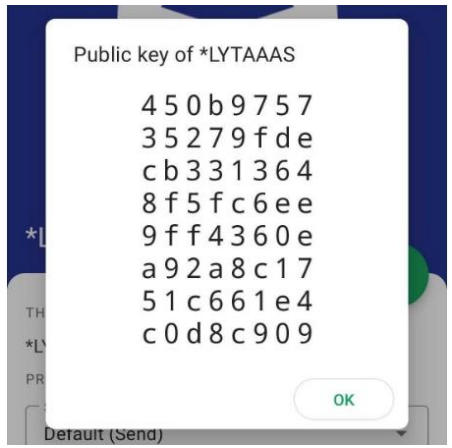
Generierung des Schlüsselpaars bei Einrichten der App



Einstellungen

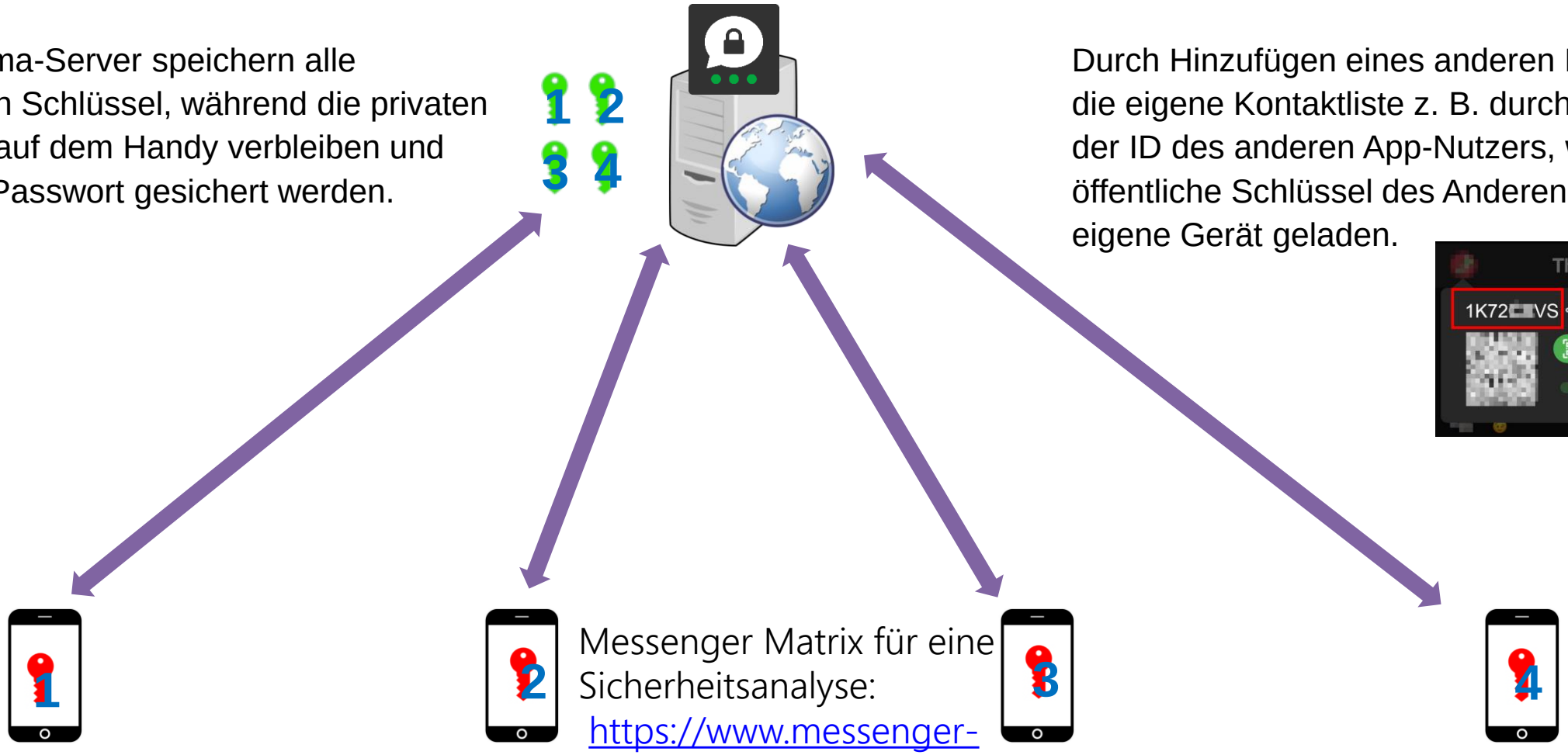


3. Nun werden Ihre ID-Daten (d.h. Ihr privater Schlüssel) mit dem Passwort verschlüsselt und als Text sowie als QR-Code angezeigt.

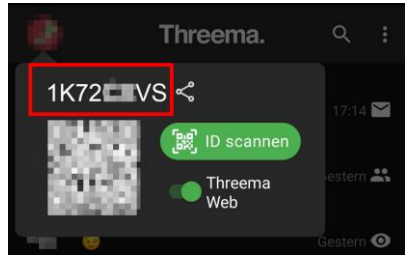


Funktionsweise eines asym. Systems

Die Threema-Server speichern alle öffentlichen Schlüssel, während die privaten Schlüssel auf dem Handy verbleiben und durch ein Passwort gesichert werden.



Durch Hinzufügen eines anderen Nutzers in die eigene Kontaktliste z. B. durch Scannen der ID des anderen App-Nutzers, wird der öffentliche Schlüssel des Anderen auf das eigene Gerät geladen.



Messenger Matrix für eine Sicherheitsanalyse: https://www.messenger-matrix.de/messenger-matrix.html

Sicherheit RSA – 1. Angriffsmöglichkeit

- Öffentlicher Schlüssel (n,e) ist bekannt → Finde p,q, um dann wie bei der Schlüsselgenerierung den privaten Schlüssel d zu berechnen ($e*d = 1 \bmod (p-1)(q-1)$)
- Finde eine Zerlegung von n mit n = p * q (p,q sind Primzahlen) → Primfaktorzerlegung
  - Es existiert kein effizienter Algorithmus zur Primfaktorzerlegung → bleibt nur Brute-Force-Methode
  - Probiere Primzahlen für p, dass q ganzzahlig ist q = n / p
  - ÜBUNG: Finde p und q für n = 39073 und e = 53 → RSA.xlsx Tabellenblatt: private key erraten
- Überlegung: warum reicht es aus, alle Primzahlen bis $\sqrt{39073}$ zu betrachten?
- Da kein effizientes Verfahren für eine Primfaktorzerlegung existiert, heißt es, dass man p und q nur groß genug wählen muss, damit es in absehbarer Zeit nicht möglich ist, n = p * q in die Primfaktoren p und q zu zerlegen.
- Was ist groß genug?

Große Primzahlen

- p und q: Wie groß ist denn sehr groß, sodass RSA kryptographisch sicher ist?
- Lesen: https://de.wikipedia.org/wiki/RSA-Kryptosystem#Schwierigkeit_des_Faktorisierungsproblems
- 3000-Bit-Schlüssel bedeutet, dass für diese Zahl mindestens ein Speicherplatz von 3000 Bit = 375 Byte genutzt werden soll. → nicht so viel Speicherplatz
- Das heißt die Zahl 2^{3000} ist zwar keine Primzahl, aber eine Zahl gültiger Größe.

2^{3000}

2^{3000}=1230231922161117176931558813276752514640713895736833715766118
029160058800614672948775360067838593459582429649254051804908512
884180898236823585082482065348331234959350355845017413023320111
360666922624728239756880416434478315693675013413090757208690376
793296658810662941824493488451726505303712916005346747908623702
673480919353936813105736620402352744776903840477883651100322409
301983488363802930540482487909763484098253940728685132044408863
734754271212592471778643949486688511721051561970432780747454823
776808464180697103083861812184348565522740195796682622205511845 5
120805520103100502558015893496459280011337454742207150136834139 0
754277906375983387610135423518424509667004216072062941158150237 1
248008430447184842098610320580417992206662247328722122088513643
683907670360209162653670641130936997002170500675501374723998766
005827579300723253474890612250135171889174899079911291512399773 8
72178519018229989376

- Diese Zahl besitzt 904 Ziffern. (natürlich per Hand gezählt ;-))

Probleme bei praktischer Umsetzung

- Mit solchen Zahlen soll nun noch der kodierte Text (z.B. ASCII) potenziert werden. Also $65^{2^{3000}} = \ldots$
- 1. Problem: Der Datentyp int in Java besitzt einen Wertebereich von -2.147.483.648 … 2.147.483.647
  - → Nutung des Datentyps BigInteger in Java. Dieser besitzt einen unbeschränkten Wertebereich (solange die Zahl im Arbeitsspeicher Platz findet ;-))
- 2. Problem: Lässt sich solche Potenzen überhaupt schnell genug berechnen?
  - → Ja, Square-And-Multiply ist sehr effizient!
- 3. Problem: Es müssen Primzahlen sein. Eine Zahl ist prim, wenn sie bei der Primfaktorzerlegung nur die 1 und sich selbst als Teiler hat. Und wir haben gerade gelernt, dass die Primfaktorzerlegung nicht effizient ist.
  - → Ja, aber es gibt probabilistischer Primzahltests, wie z. B. den Miller-Rabin-Test. Das heißt, er gibt mit einer gewissen Wahrscheinlichkeit an, ob eine gegebene Zahl eine Primzahl ist. Das heißt, dass der Miller-Rabin-Test nicht endgültig sagen, ob eine Zahl wirklich prim ist. Als Ausgabe kann der Miller-Rabin-Test lediglich sagen: „wahrscheinlich prim“ und „sicher nicht prim“. Dieser Algorithmus ist effizient.

Sicherheit RSA – 2. Angriffsmöglichkeit

- Wenn man schon nicht an den privaten Schlüssel gelangt, vielleicht kann man ja die Nachricht ohne diesen entschlüsseln?
- Verschlüsselung von m: $m' = m^e \bmod n$
- → Umstellung der Formel: $m = \sqrt[e]{m'} \bmod n$ damit gelöst?
- → Nein
- Es gibt bei kleinen Zahlen für e oder d oder n Möglichkeiten dies zu lösen, aber bei großen Zahlen geht man davon aus, das dies mathematisch vermutlich mindestens so schwer zu lösen ist, wie das Faktorisieren von n.