

Kryptografie

Kodieren, Verschleiern, Authentifizieren

Einstiege:

Text am Beamer entschlüsseln lassen

Stationenlernen zu historischen Verfahren

Material unter

Cryptool.de

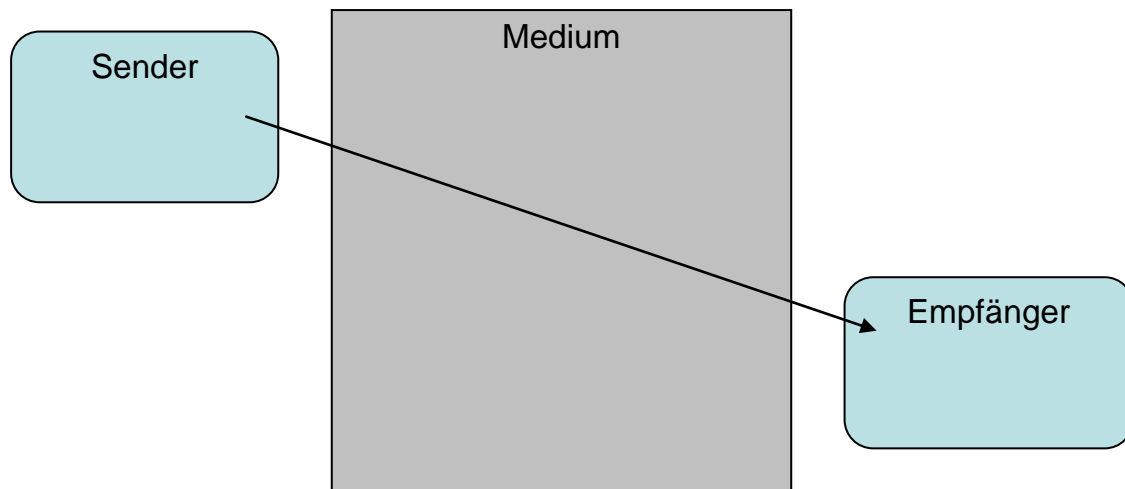
MathePrisma der Uni Wuppertal

1) Kodieren

Morsecode, ASCII – „Code“, Unicode, Codex, Codec, Encoder, Decoder.

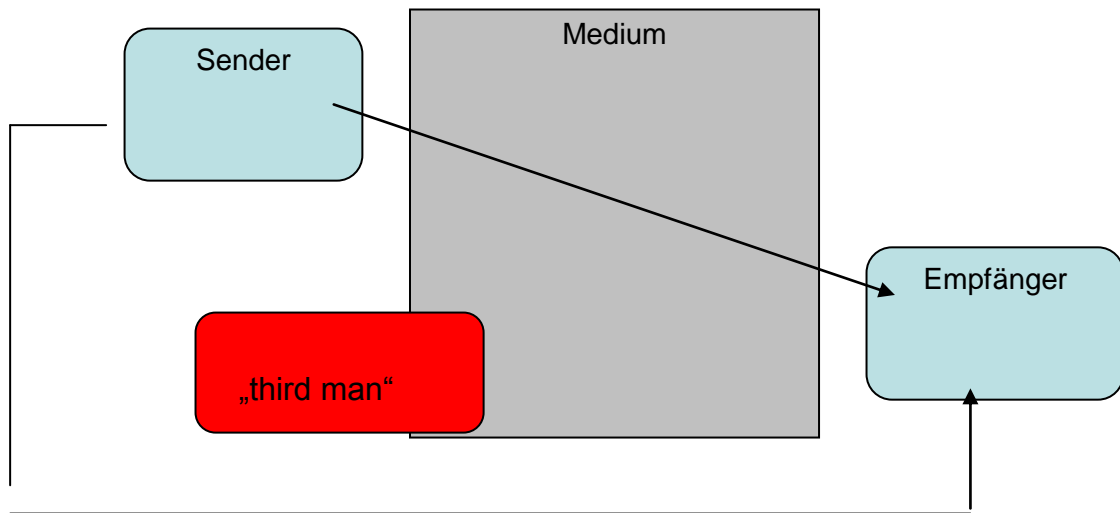
Ein **Code** (engl. von lat. codex Gesetzbuch) ist im weitesten Sinn eine *Vorschrift*, etwa für eine solche Umsetzung, aber auch für die Zuordnung einer Abkürzung, eines Schlüssels, bis hin zu Konventionen und Verhaltensregeln. In der deutschen Sprache versteht man darunter oft eine geheim gehaltene Vorschrift zur Verschlüsselung (Verschleierung) von Nachrichten.

Der Weg der Nachricht: „Ich komme heute zum Abendessen.“

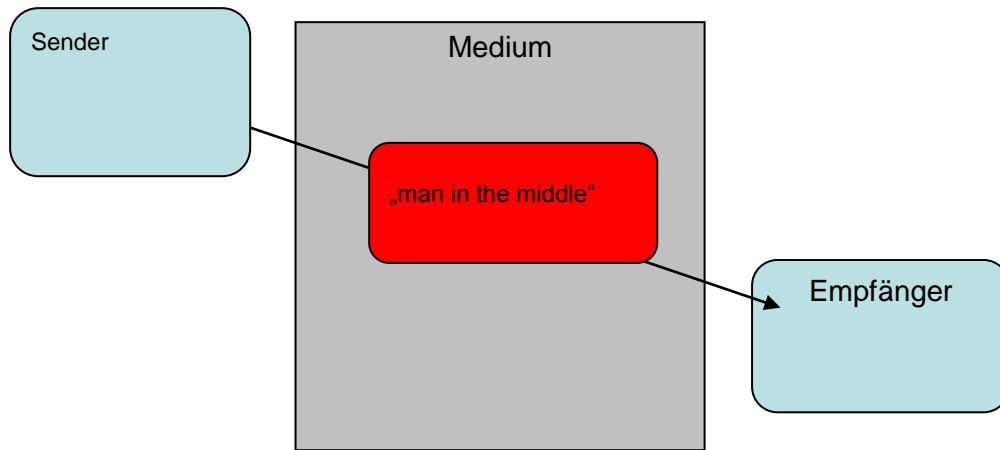


2) Verschleiern

Bei der Verschleierung von Informationen (fast immer von Informationen in Textform) ist die Situation ähnlich:



Der Schlüssel nimmt einen „Umweg“



ABCDEFGHIJKLMNOPQRSTUVWXYZ

symmetrische Verfahren**Name** **Cäsar****Idee** Buchstaben verschieben z.B. A → E, B → F ...**Bsp.:** „INFORMATIK“ um 3 verschieben:

RLYUSLOYWSHU → ?

Programmierung

(Nummer des Buchstabens im Alphabet +
Verschiebung) mod Alphabetlänge
ergibt Nummer des neuen Buchstabens

$$c = m + z \pmod{p}$$

Theorie

Restklasse (hier modulo 26)

Kompromittierung „Brute Force“ – max. 25 Versuche

```
for (int i=0;i< einString.length() ;i++ ) {  
    nr = alphabet.indexOf( einString.charAt(i));  
    if (nr > -1) {  
        nr = (nr + verschiebung) % alphabet.length();  
        ausString = ausString + alphabet.charAt(nr);  
    } // end of if  
} // end of for
```

Name	Substitution (allg. monoalph.)
Idee	Buchstaben permutieren; Schlüssel ist zweites Alphabet in anderer Reihenfolge
Programmierung	Nummer des Buchstabens im Alphabet neuer Buchstabe im Schlüssel an derselben Stelle ABCDEFGHIJKLMNOPQRSTUVWXYZ EXDBFGYULKMJNOTQRSAIPWCHZV
Beispiel	„INFORMATIK“ → LNGTSNEILM XLTJTYLF → ?

Kompromittierung

„Brute Force“ –

Fakultät der Alphabetlänge Versuche - geht nicht;

bei längeren Texten in bekannter Sprache

Statistik der Buchstaben

```
for (int i=0; i<einString.length() ;i++ ) {  
    nr = alphabet.indexOf( einString.charAt(i));  
    if (nr > -1) {  
        ausString = ausString + permAlph.charAt(nr) ;  
    } // end of if  
} // end of for
```

Name

Vigenère (polyalph. Substitution)

Idee

Schlüsselwort wird zum Verschieben benutzt

A +0
 B +1
 E +4
 R +17

Urtext	G	E	H	E	I	M	N	I	S
Schl.	A	B	E	R	A	B	E	R	A
Chiffre	G	F	L	V	I	N	R	Z	S

Programmierung

(Nummer des Buchstabens im Alphabet +
 Nummer des Schlüsselwortbuchstabens (-1))
 mod Alphabetlänge
 ergibt Nummer des neuen Buchstabens
 $c[i] = m[i] + z[i] \pmod{p}$

Bsp.:

„INFORMATIK“ mit ABER: IOJFRNEKIL

HFKYKE mit ABER → ?

Kompromittierung Schlüssellänge suchen
(Korrelation Text : verschobener Text)
dann Häufigkeitsanalyse

Verbesserte Verfahren ->AutoKey (KEYDIESISTEINGEHEIMERT), Vernam
(Schlüssellänge= Klartextlänge), OneTimePad (Vernam+zufälliger Schlüssel)

```
while (schluessel.length() < einString.length())  
    schluessel = schluessel + schluessel;  
  
for (int i=0; i < einString.length() ;i++ ) {  
  
    nr = alphabet.indexOf( einString.charAt(i));  
    nr2 = alphabet.indexOf( schluessel.charAt(i));  
    chiffre = (nr + nr2) % alphabet.length();  
  
    if (nr > -1) {  
        ausString = ausString + alphabet.charAt(chiffre);  
    } // end of if  
  
} // end of for
```

Vigenère-Quadrat

	Text																									
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
S c h l ü s s e l	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

G
e
h
e
i
m
t
e
x
t

Name **Transposition (Skytale, griech.)**

Idee Buchstaben bleiben – Position
verändern, zum Beispiel spaltenweise
eintragen und zeilenweise auslesen

d	i	e	t
i	s	r	e
e	t	u	x
s	d	r	t

→ dietisreetuxsdr

Programmierung Text in einem zweidimensionalen Array of char
ablegen; Schlüssel ist eine Permutation der Zahlen
1 ... n*m

Kompromittierung Wenn man von **einem** bekannten Quelltext die
Verschlüsselung kennt, kennt man auch den
Schlüssel

```
String einString, ausString;
char [] zeichen, zeichen2;
einString = ein.getZeile(0);
ausString = "";
zeichen2 = new char[16];

while (einString.length() % 16 !=0)
    einString = einString + " ";
while (einString.length() !=0) {
    zeichen = einString.substring(0,16).toCharArray();
    zeichen2[ 0] = zeichen[ 0];
    zeichen2[ 1] = zeichen[ 4];
    zeichen2[ 2] = zeichen[ 8];
    zeichen2[ 3] = zeichen[12];
    zeichen2[ 4] = zeichen[ 1];
    zeichen2[ 5] = zeichen[ 5];
    zeichen2[ 6] = zeichen[ 9];
    zeichen2[ 7] = zeichen[13];
    zeichen2[ 8] = zeichen[ 2];
    zeichen2[ 9] = zeichen[ 6];
    zeichen2[10] = zeichen[10];
    zeichen2[11] = zeichen[14];
    zeichen2[12] = zeichen[ 3];
    zeichen2[13] = zeichen[ 7];
    zeichen2[14] = zeichen[11];
    zeichen2[15] = zeichen[15];
    ausString = new String(zeichen2);
    aus.schreibeZeile(ausString);
    einString = einString.substring(16, einString.length());
}
```

Name	AES (Rijndael-Algorithmus)
Idee	Transposition und polyalphabetische Substitution wiederholen und mischen

Blockgröße von 128 Bit und Schlüsselgröße von 128, 192 oder 256 Bit.

Jeder Block wird zunächst in eine zweidimensionale [Tabelle](#) mit vier Zeilen und 4 Spalten geschrieben, deren Zellen ein [Byte](#) groß sind.

Jeder Block wird nun nacheinander bestimmten Transformationen unterzogen.

Aber anstatt jeden Block einmal mit dem Schlüssel zu [verschlüsseln](#), wendet AES verschiedene Teile des erweiterten Originalschlüssels nacheinander auf den Klartext-Block an. Die Anzahl r dieser Runden variiert und ist von der Schlüssellänge abhängig.

(Wikipedia)

S-Box

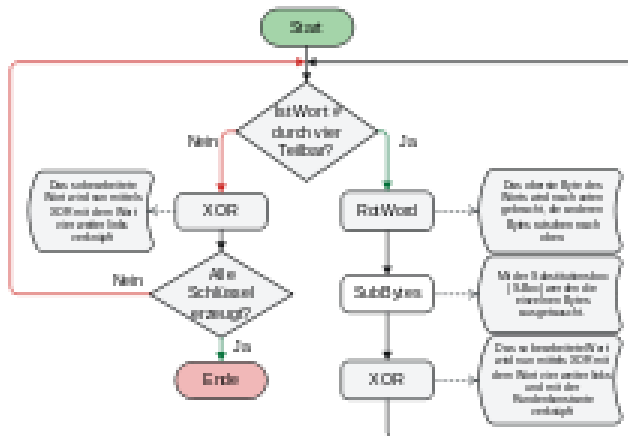
Eine [Substitutionsbox](#) (S-Box) dient als Basis für eine [monoalphabetische Verschlüsselung](#). Sie ist meist als [Array](#) implementiert und gibt an, wie in jeder Runde jedes Byte eines Blocks durch einen anderen Wert zu ersetzen ist. Typischerweise wird die S-Box in Blockchiffren eingesetzt, um die Beziehung zwischen Klar- und Geheimtext zu verwischen (in der kryptologischen Fachsprache *Konfusion* genannt). Die S-Box des AES setzt auch teilweise das [Shannon'sche](#) Prinzip der [Diffusion](#) um. Die Werte der S-Box und inversen S-Box können dynamisch berechnet werden um Speicher zu sparen oder vorberechnet sein und in einem Array gespeichert werden. Die S-Box besteht aus 256 Bytes, die konstruiert werden, indem zunächst jedes Byte außer der Null, aufgefasst als Vertreter des [endlichen Körpers](#) \mathbb{F}_{2^8} , durch sein multiplikatives Inverses ersetzt wird. Die Konstruktion der S-Box unterliegt Designkriterien, die die Anfälligkeit für die Methoden der linearen und der differentiellen Kryptoanalyse sowie für algebraische Attacken minimieren sollen.

Ablauf

- Schlüsselexpansion
- Vorrunde
 - AddRoundKey(Rundenschlüssel[0])
- Verschlüsselungsrunden ($r = 1$ bis $R-1$)
 - SubBytes()
 - ShiftRows()
 - MixColumns()
 - AddRoundKey(Rundenschlüssel[r])
- Schlussrunde
 - SubBytes()
 - ShiftRows()
 - AddRoundKey(Rundenschlüssel[R])

(Die Schlussrunde zählt auch als Runde, also $R = \text{Anzahl Verschlüsselungsrunden} + 1$ Schlussrunde)

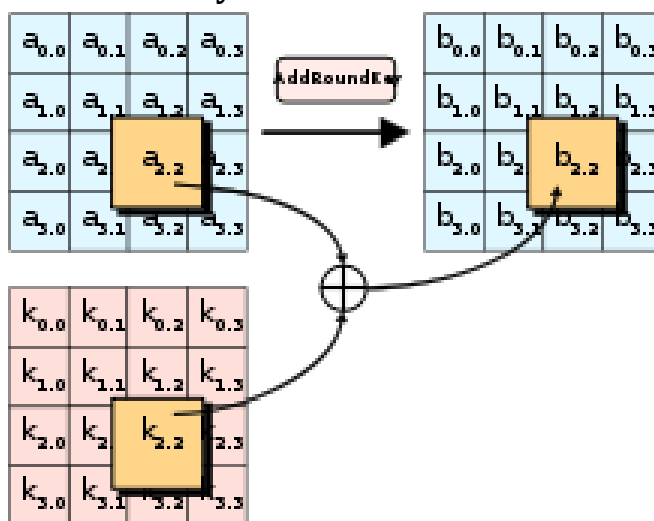
Schlüsselexpansion



Prinzip der Schlüsselexpansion bei AES

Zunächst müssen aus dem Schlüssel $R + 1$ Teilschlüssel (auch Rundenschlüssel genannt) erzeugt werden. Die Rundenschlüssel müssen die gleiche Länge wie die Blöcke haben. Somit muss der Benutzerschlüssel auf die Länge $b * (R + 1)$ expandiert werden, wobei b die Blockgröße angibt. Der Schlüssel wird in eine zweidimensionale Tabelle mit vier Zeilen und Zellen der Größe 1 Byte abgebildet. Die ersten Spalten der Tabelle werden mit dem Benutzerschlüssel gefüllt. Die weiteren Spalten werden wie folgt [rekursiv](#) berechnet: Um die Werte für die Zellen in der nächsten Spalte zu erhalten, werden die Spalten, welche je nach Blockgröße ein Vielfaches der vierten, sechsten oder achten Spalte sind, nach links rotiert ($[a_0, a_1, a_2, a_3]$ wird zu $[a_1, a_2, a_3, a_0]$) und mit Hilfe der S-Box verschlüsselt. Im Anschluss wird der „vorderste“ Wert der Spalte mit der rcon-Tabelle [XOR](#) verknüpft und abschließend die gesamte Spalte mit der um eine Schlüssellänge zurückliegenden Spalte XOR verknüpft. Die rcon-Tabelle ist ähnlich wie die S-Box eine Tabelle in Form eines Arrays, das konstante Werte, in diesem Fall die Zweierpotenzen, enthält. Jede andere Spalte wird aus einer XOR-Verknüpfung mit der Spalte eine Schlüssellänge vorher gebildet. Eine Besonderheit bildet AES-256. Dort wird jede 4. Spalte (also eine Spalte, welche normalerweise ohne Rotation und so weiter auskommt) durch die S-Box ersetzt und dann mit der Spalte eine Schlüssellänge zuvor XOR verknüpft.

AddRoundKey



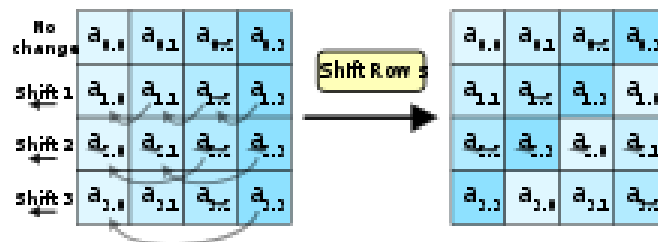
Bitweise XOR-Verknüpfung zwischen dem Block und dem aktuellen Rundenschlüssel

In der Vorrunde und am Ende jeder weiteren Verschlüsselungsrunde wird die KeyAddition ausgeführt. Hierbei wird eine bitweise XOR-Verknüpfung zwischen dem Block und dem aktuellen Rundenschlüssel vorgenommen. Dies ist die einzige Funktion in AES, die den Algorithmus vom Benutzerschlüssel abhängig macht.

SubBytes

Im ersten Schritt jeder Runde wird für jedes Byte im Block ein Äquivalent in der S-Box gesucht. Somit werden die Daten monoalphabetisch verschlüsselt.

ShiftRows



Zeilen werden um eine bestimmte Anzahl von Spalten nach links verschoben

Wie oben erwähnt, liegt ein Block in Form einer zweidimensionalen Tabelle mit vier Zeilen vor. In diesem Schritt werden die Zeilen um eine bestimmte Anzahl von Spalten nach links verschoben. Überlaufende Zellen werden von rechts fortgesetzt. Die Anzahl der Verschiebungen ist zeilen- und blocklängenabhängig:

Für den AES sind nur die fett markierten Werte relevant

Je nach Blocklänge b und Zeile in der Datentabelle wird die Zeile um 1 bis 4 Spalten verschoben

r

Zeile 1

Zeile 2

Zeile 3

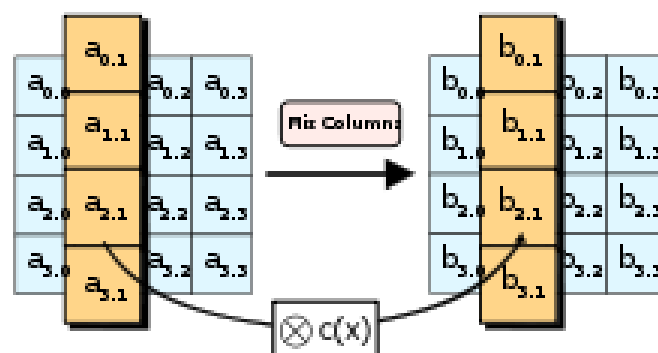
Zeile 4

$b=128$ $b=160$ $b=192$ $b=224$ $b=256$

0	0	0	0	0
1	1	1	1	1
2	2	2	2	3
3	3	3	4	4

MixColumns

→ Hauptartikel: [Rijndael MixColumns](#)^[4]



Die Spalten werden vermischt

Schließlich werden die Daten innerhalb der Spalten vermischt. Zur Berechnung eines Bytes b_j der neuen Spalte wird jedes Byte a_j der alten mit einer Konstanten (1, 2 oder 3) multipliziert. Dies

geschieht modulo des irreduziblen Polynoms $x^8 + x^4 + x^3 + x + 1$ im Galois-Körper $GF(2^8)$.
Dann werden die Ergebnisse XOR-verknüpft:

$$\begin{aligned} b_0 &= (a_0 \cdot 2) \oplus (a_1 \cdot 3) \oplus (a_2 \cdot 1) \oplus (a_3 \cdot 1) \\ b_1 &= (a_0 \cdot 1) \oplus (a_1 \cdot 2) \oplus (a_2 \cdot 3) \oplus (a_3 \cdot 1) \\ b_2 &= (a_0 \cdot 1) \oplus (a_1 \cdot 1) \oplus (a_2 \cdot 2) \oplus (a_3 \cdot 3) \\ b_3 &= (a_0 \cdot 3) \oplus (a_1 \cdot 1) \oplus (a_2 \cdot 1) \oplus (a_3 \cdot 2) \end{aligned}$$

In Matrixschreibweise:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix}$$

Nach den Rechengesetzen in diesem Galois-Körper gilt für die Multiplikation:

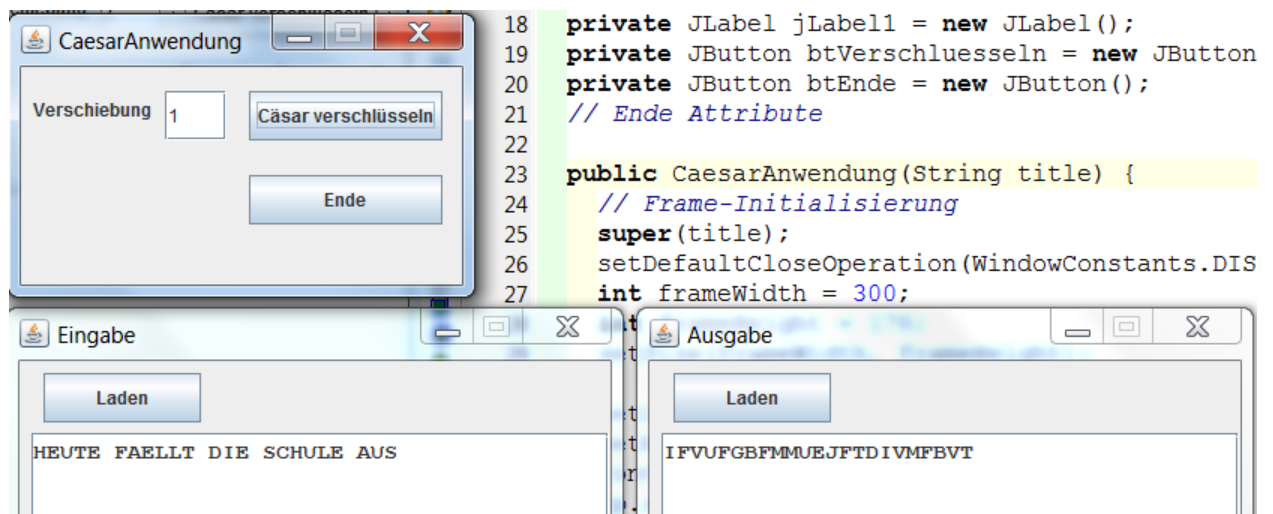
- $a \cdot 1 = a$
- $a \cdot 2 = \begin{cases} 2a & \text{wenn } a < 2^7 \\ 2a \oplus (11b)_{\text{hex}} & \text{wenn } a \geq 2^7 \end{cases}$
- $a \cdot 3 = (a \cdot 2) \oplus a$

Dabei bezeichnet $2a$ die normale Multiplikation von a mit 2 und \oplus die bitweise XOR-Verknüpfung.

Entschlüsselung

Bei der Entschlüsselung von Daten wird genau rückwärts vorgegangen. Die Daten werden zunächst wieder in zweidimensionale Tabellen gelesen und die Rundenschlüssel generiert. Allerdings wird nun mit der Schlussrunde angefangen und alle Funktionen in jeder Runde in der umgekehrten Reihenfolge aufgerufen. Durch die vielen XOR-Verknüpfungen unterscheiden sich die meisten Funktionen zum Entschlüsseln nicht von denen zum Verschlüsseln. Jedoch muss eine andere S-Box genutzt werden (die sich aus der originalen S-Box berechnen lässt) und die Zeilenverschiebungen erfolgen in die andere Richtung.

AUFGABE



1)

- Schreiben Sie das Programm **CaesarAnwendung1**, das einen vorgegebenen Klartext durch eine Verschiebung um n Zeichen ($1 \leq n < 26$) verschlüsselt (Alphabet aus Großbuchstaben).
- Schreiben Sie das Programm **CaesarAnwendung2**, welches zusätzlich einen durch eine Verschiebung verschlüsselten Text (Caesar-Verschlüsselung) wieder entschlüsselt (alle ASCII-Zeichen).

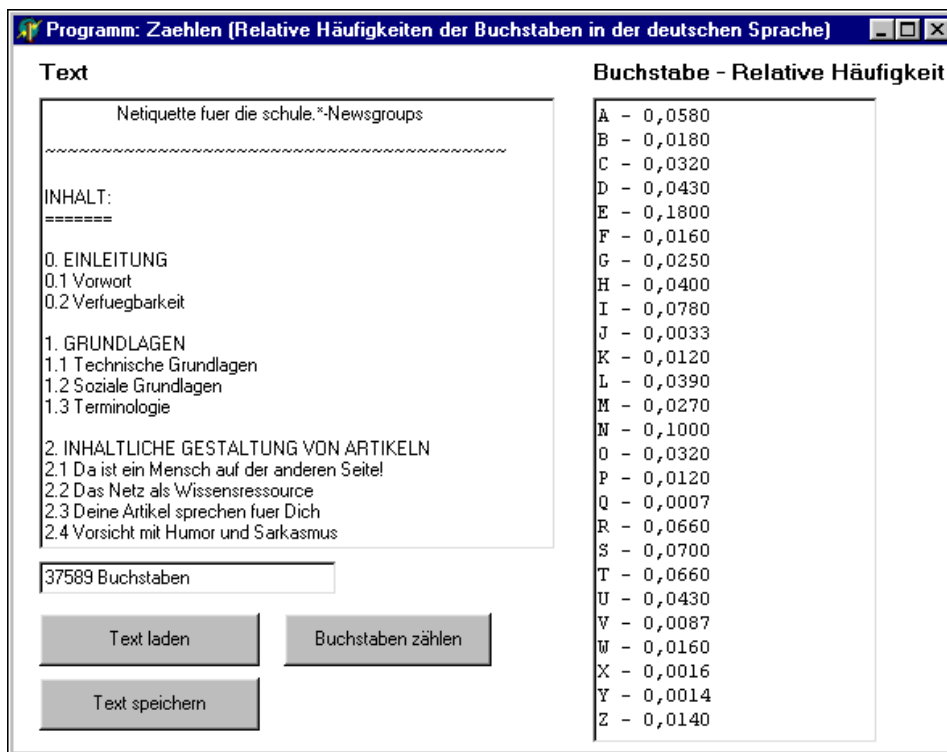
28.4 Ermitteln des Schlüssels

Um solche Geheimtexte zu „knacken“, kann man im Geheimtext die Häufigkeit der Buchstaben zählen und daraus Rückschlüsse auf die Kodierung ziehen. Durch Auszählen langer und verschiedenartiger Text erhält man eine Übersicht über die Wahrscheinlichkeiten für das Auftreten der einzelnen Buchstaben in der deutschen Sprache:

Buchstabe:	Wahrscheinlichkeit:	Buchstabe:	Wahrscheinlichkeit:
A	0,0433	P	0,0050
B	0,0160	Q	0,0001
C	0,0267	R	0,0686
D	0,0439	S	0,0539
E	0,1470	T	0,0473
F	0,0136	U	0,0319
G	0,0267	V	0,0074
H	0,0436	W	0,0142
I	0,0638	X	0,0001
J	0,0016	Y	0,0002
K	0,0096	Z	0,0142
L	0,0293	Ä	0,0049
M	0,0213	Ö	0,0025
N	0,0884	Ü	0,0058
O	0,0136	Leerzeichen	0,1515

Überprüfen Sie die Werte der Tabelle mit einem Programm **ZaehlenAnwendung**, mit dem Sie einen längeren Text (mindestens 1000 Buchstaben) in eine Textarea-Komponente einlesen und analysieren können. Lassen Sie

die Buchstaben sowie die relativen Häufigkeiten, mit denen sie im Text auftreten, in einer zweiten Memokomponente ausgeben.



(Abb. Relative Häufigkeit der Buchstaben A .. Z in einem deutschen Text)

Aufgabe

Schreiben Sie ein Programm **Knacken**, das für einen verschlüsselten (längeren) Text eine entsprechende Tabelle erzeugt. Versuchen Sie mithilfe dieses Programms den verschlüsselten Text zu entschlüsseln, indem Sie

- vom Programm eine Entschlüsselung durchführen lassen gemäß den ermittelten relativen Häufigkeiten der Buchstaben im chiffrierten Text,
- vom Programm ein von Ihnen vorgegebenes Schlüsselzeichen gegen das vermutete „Originalzeichen“ austauschen lassen.

Die hier angesprochenen Verschlüsselungsverfahren gehören zu den einfachen, monoalphabetischen Verschlüsselungen. Auch für diese Verschlüsselungstechnik gibt es weitere interessante Varianten (z. B. Chiffrierung durch Multiplikation statt durch Addition).

asymmetrische Verfahren

RSA1 Addition in Restklassen

Caesar-Verfahren

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	w	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
→												→													
									+5															+5	

$$\begin{aligned}
 \mathbf{0} &= \{ \dots -26, 0, 26, 52, 78, \dots \} & k * 26 \\
 \mathbf{1} &= \{ \dots -25, 1, 27, 53, \dots \} & k * 26 + 1 \\
 &\dots & \\
 \mathbf{5} &= \{ \dots -21, 5, 31, 57, \dots \} & k * 26 + 5 \\
 &\dots & \\
 \mathbf{8} &= \{ \dots -18, 8, 34, 60, \dots \} & k * 26 + 8 \\
 &\dots & \\
 \mathbf{25} &= \{ \dots -1, 25, 51, 77, \dots \} & k * 26 + 25
 \end{aligned}$$

Verschlüsselung: $\mathbf{c = m + z \pmod{p}}$

(Dabei ist m die Nachricht (Zahl zu dem Buchstaben), z die Schlüsselzahl und c die verschlüsselte Nachricht (Chiffre))

$$\mathbf{c + z' = m + z + z' = m * (z + z') = m + 0 = m \pmod{p}}$$

Lösung: $\mathbf{z' = -z}$ Das ist die Restklasse von $\mathbf{p - z}$

RSA2 modulares Multiplizieren

$$c = m * z \pmod{p}$$

$$c * z' = m * z * z' = m * (z * z') = m * 1 = m \pmod{p}$$

$$\text{Inverse bezüglich der Multiplikation: } z * z' \equiv 1 \pmod{p}$$

Multiplikation modulo 26

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
2	2	4	6	8	10	12	14	16	18	20	22	24	0	2	4	6	8	10	12	14	16	18	20	22	24
3	3	6	9	12	15	18	21	24	1	4	7	10	13	16	19	22	25	2	5	8	11	14	17	20	23
4	4	8	12	16	20	24	2	6	10	14	18	22	0	4	8	12	16	20	24	2	6	10	14	18	22
5	5	10	15	20	25	4	9	14	19	24	3	8	13	18	23	2	7	12	17	22	1	6	11	16	21
6	6	12	18	24	4	10	16	22	2	8	14	20	0	6	12	18	24	4	10	16	22	2	8	14	20
7	7	14	21	2	9	16	23	4	11	18	25	6	13	20	1	8	15	22	3	10	17	24	5	12	19
8	8	16	24	6	14	22	4	12	20	2	10	18	0	8	16	24	6	14	22	4	12	20	2	10	18
9	9	18	1	10	19	2	11	20	3	12	21	4	13	22	5	14	23	6	15	24	7	16	25	8	17
10	10	20	4	14	24	8	18	2	12	22	6	16	0	10	20	4	14	24	8	18	2	12	22	6	16
11	11	22	7	18	3	14	25	10	21	6	17	2	13	24	9	20	5	16	1	12	23	8	19	4	15
12	12	24	10	22	8	20	6	18	4	16	2	14	0	12	24	10	22	8	20	6	18	4	16	2	14
13	13	0	13	0	13	0	13	0	13	0	13	0	13	0	13	0	13	0	13	0	13	0	13	0	13
14	14	2	16	4	18	6	20	8	22	10	24	12	0	14	2	16	4	18	6	20	8	22	10	24	12
15	15	4	19	8	23	12	1	16	5	20	9	24	13	2	17	6	21	10	25	14	3	18	7	22	11
16	16	6	22	12	2	18	8	24	14	4	20	10	0	16	6	22	12	2	18	8	24	14	4	20	10
17	17	8	25	16	7	24	15	6	23	14	5	22	13	4	21	12	3	20	11	2	19	10	1	18	9
18	18	10	2	20	12	4	22	14	6	24	16	8	0	18	10	2	20	12	4	22	14	6	24	16	8
19	19	12	5	24	17	10	3	22	15	8	1	20	13	6	25	18	11	4	23	16	9	2	21	14	7
20	20	14	8	2	22	16	10	4	24	18	12	6	0	20	14	8	2	22	16	10	4	24	18	12	6
21	21	16	11	6	1	22	17	12	7	2	23	18	13	8	3	24	19	14	9	4	25	20	15	10	5
22	22	18	14	10	6	2	24	20	16	12	8	4	0	22	18	14	10	6	2	24	20	16	12	8	4
23	23	20	17	14	11	8	5	2	25	22	19	16	13	10	7	4	1	24	21	18	15	12	9	6	3
24	24	22	20	18	16	14	12	10	8	6	4	2	0	24	22	20	18	16	14	12	10	8	6	4	2
25	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Die Tabelle zeigt, dass es zu allen **geraden** Zahlen keine multiplikativ **Inverse** gibt; in den jeweiligen Zeilen kommt die 1 als Ergebnis nicht vor.

Multiplikation modulo 23

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	Inverse
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	1
2	2	4	6	8	10	12	14	16	18	20	22	1	3	5	7	9	11	13	15	17	19	21	12
3	3	6	9	12	15	18	21	1	4	7	10	13	16	19	22	2	5	8	11	14	17	20	8
4	4	8	12	16	20	1	5	9	13	17	21	2	6	10	14	18	22	3	7	11	15	19	6
5	5	10	15	20	2	7	12	17	22	4	9	14	19	1	6	11	16	21	3	8	13	18	14
6	6	12	18	1	7	13	19	2	8	14	20	3	9	15	21	4	10	16	22	5	11	17	4
7	7	14	21	5	12	19	3	10	17	1	8	15	22	6	13	20	4	11	18	2	9	16	10
8	8	16	1	9	17	2	10	18	3	11	19	4	12	20	5	13	21	6	14	22	7	15	3
9	9	18	4	13	22	8	17	3	12	21	7	16	2	11	20	6	15	1	10	19	5	14	18
10	10	20	7	17	4	14	1	11	21	8	18	5	15	2	12	22	9	19	6	16	3	13	7
11	11	22	10	21	9	20	8	19	7	18	6	17	5	16	4	15	3	14	2	13	1	12	21
12	12	1	13	2	14	3	15	4	16	5	17	6	18	7	19	8	20	9	21	10	22	11	2
13	13	3	16	6	19	9	22	12	2	15	5	18	8	21	11	1	14	4	17	7	20	10	16
14	14	5	19	10	1	15	6	20	11	2	16	7	21	12	3	17	8	22	13	4	18	9	5
15	15	7	22	14	6	21	13	5	20	12	4	19	11	3	18	10	2	17	9	1	16	8	20
16	16	9	2	18	11	4	20	13	6	22	15	8	1	17	10	3	19	12	5	21	14	7	13
17	17	11	5	22	16	10	4	21	15	9	3	20	14	8	2	19	13	7	1	18	12	6	19
18	18	13	8	3	21	16	11	6	1	19	14	9	4	22	17	12	7	2	20	15	10	5	9
19	19	15	11	7	3	22	18	14	10	6	2	21	17	13	9	5	1	20	16	12	8	4	17
20	20	17	14	11	8	5	2	22	19	16	13	10	7	4	1	21	18	15	12	9	6	3	15
21	21	19	17	15	13	11	9	7	5	3	1	22	20	18	16	14	12	10	8	6	4	2	11
22	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	22

Die Inverse lässt sich immer mit dem „erweiterten Euklidischen Algorithmus“ berechnen.

Zum Beispiel: 15

Dass der größte gemeinsame Teiler von 15 und 23 1 ist, wird nicht verwundern – schließlich ist 23 eine Primzahl.

Bei der Berechnung schreiben wir aber jetzt mit, wie oft man 15 und 23 verwendet hat.

m	n	Berechnung	
23	15	$r = 23 - 1 \cdot 15 = 8$	$8 = 23 - 15$
15	8	$r = 15 - 1 \cdot 8 = 7$	$7 = 15 - 1(23 - 15)$ $= 2 \cdot 15 - 23$
8	7	$r = 8 - 7 = 1$	$1 = 8 - 7$ $= (23 - 15) - (2 \cdot 15 - 23)$ $= 2 \cdot 23 - 3 \cdot 15$

Aus der letzten Zeile $1 = 2 \cdot 23 - 3 \cdot 15$ entnimmt man, dass $(-3) \cdot 15 \equiv 1 \pmod{23}$ ist.
 $-3 \equiv 20 \pmod{23}$, denn $-3 + 23 = 20$, also ist 20 die gesuchte Inverse.

RSA3 modulares Potenzieren, erster Versuch

$$c = m^e \pmod{p}, \text{ wobei } m < p, e < p \text{ und } c < p.$$

$$c^d = (m^e)^d = m \pmod{p}$$

Wenn m und p keine gemeinsamen Teiler haben, reduziert sich die Gleichung zu $m^{e \cdot d - 1} = 1 \pmod{p}$

Es geht also zunächst um Zahlen mit $m^t = 1 \pmod{p}$

p = 11 Potenzieren modulo 11

Exponent >>	1	2	3	4	5	6	7	8	9	10	11
1	1	1	1	1	1	1	1	1	1	1	1
2	2	4	8	5	10	9	7	3	6	1	2
3	3	9	5	4	1	3	9	5	4	1	3
4	4	5	9	3	1	4	5	9	3	1	4
5	5	3	4	9	1	5	3	4	9	1	5
6	6	3	7	9	10	5	8	4	2	1	6
7	7	5	2	3	10	4	6	9	8	1	7
8	8	9	6	4	10	3	2	5	7	1	8
9	9	4	3	5	1	9	4	3	5	1	9
10	10	1	10	1	10	1	10	1	10	1	10

Rechnung exemplarisch für die **Zeile 5**: 5; dann $5 \cdot 5 = 25 = 2 \cdot 11 + 3$; $3 \cdot 5 = 15 = 11 + 4$; $4 \cdot 5 = 20 = 11 + 9$, und weiter mit 5 multiplizieren (modulo 11)

Verschlüsselung ist möglich mit den Exponenten 3, 7 und 9; es sind gerade die Zahlen, die **mit $p - 1 = 10$ teilerfremd** sind. Bei den anderen Exponenten erhält man für unterschiedliche Werte für m dasselbe Ergebnis, so dass eine Entschlüsselung nicht möglich ist.

Für $e = 7$ und $m = 5$ liefert die Tabelle den Wert 3 für c. Womit müssen wir potenzieren (mod 11), um wieder 5 zu erhalten? Die Tabelle zeigt uns die Lösungen 3 und 8.

Für $e = 7$ und $m = 8$ liefert die Tabelle den Wert 2 für c. Womit müssen wir potenzieren (mod 11), um wieder 8 zu erhalten? Die Tabelle zeigt uns diesmal nur eine Lösung: 3.

$p = 13$ **Potenzieren modulo 13**

Exponent >>	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	4	8	3	6	12	11	9	5	10	7	1	2
3	3	9	1	3	9	1	3	9	1	3	9	1	3
4	4	3	12	9	10	1	4	3	12	9	10	1	4
5	5	12	8	1	5	12	8	1	5	12	8	1	5
6	6	10	8	9	2	12	7	3	5	4	11	1	6
7	7	10	5	9	11	12	6	3	8	4	2	1	7
8	8	12	5	1	8	12	5	1	8	12	5	1	8
9	9	3	1	9	3	1	9	3	1	9	3	1	9
10	10	9	12	3	4	1	10	9	12	3	4	1	10
11	11	4	5	3	7	12	2	9	8	10	6	1	11
12	12	1	12	1	12	1	12	1	12	1	12	1	12

Beobachtung in beiden Beispielen: in der vorletzten Spalte (p-1) stehen lauter Einsen.
Dies lässt sich auch für weitere Primzahlen bestätigen und sogar allgemein beweisen:

Der kleine Satz von Fermat

Für eine Primzahl p und alle zu p teilerfremden ganzen Zahlen a gilt :

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

$$m^{e \cdot d} \equiv m \pmod{p}$$

Zykluslänge

Mit m^e befinden wir uns in der Zeile m; m^e weiter mit m^e multiplizieren (mod p) bedeutet, dass wir uns in dem dortigen Zyklus bewegen.

Zyklus \leftrightarrow multiplikative Gruppe

$$e \cdot d \equiv 1 \pmod{(p-1)}$$

RSA4

$c = m^e \pmod n$, wobei $n = p * q$ und p, q Primzahlen.

$$m^{e^* d} \equiv m \pmod{n}$$

$$e * d \equiv 1 \pmod{\text{(Zykluslänge)}}$$

Zykluslänge : Mächtigkeit der mult. Gruppe

Potenzieren modulo 21

[illegible]

Potenzieren modulo 21 (nur die Zeilen mit Einsen)

Exp	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	4	8	16	11	1	2	4	8	16	11	1	2	4	8	16	11	1	2	4	8
4	4	16	1	4	16	1	4	16	1	4	16	1	4	16	1	4	16	1	4	16	1
5	5	4	20	16	17	1	5	4	20	16	17	1	5	4	20	16	17	1	5	4	20
8	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8	1	8
10	10	16	13	4	19	1	10	16	13	4	19	1	10	16	13	4	19	1	10	16	13
11	11	16	8	4	2	1	11	16	8	4	2	1	11	16	8	4	2	1	11	16	8
13	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13	1	13
16	16	4	1	16	4	1	16	4	1	16	4	1	16	4	1	16	4	1	16	4	1
17	17	16	20	4	5	1	17	16	20	4	5	1	17	16	20	4	5	1	17	16	20
19	19	4	13	16	10	1	19	4	13	16	10	1	19	4	13	16	10	1	19	4	13
20	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20	1	20

Beobachtungen:

- Die Zeilennummern liefern genau die Zahlen, die zu 21 teilerfremd sind.
- Es sind gleichzeitig genau die Zahlen, die bez. 21 invertierbar sind.
- Die Zykluslänge ist hier 6, die kleinste Zahl s mit $m^s \equiv 1 \pmod{n}$ für alle m

Wie findet man (ohne Probieren) einen Wert s , so dass $m^s \equiv 1 \pmod{n}$?

Hier hilft der Satz von Fermat-Euler:

Wenn zwei ganze Zahlen m und n (≥ 2) teilerfremd sind, so gilt $m^{\varphi(n)} \equiv 1 \pmod{n}$.

Dabei liefert die Eulersche φ -Funktion die Anzahl der natürlichen Zahlen $\leq n$, die zu n teilerfremd sind.

Spezialfall: Wenn n das Produkt von zwei Primzahlen p und q ist, haben (nur) p , $2p$, $3p$, ..., qp und q , $2q$, $3q$, ..., pq gemeinsame Teiler mit n . Wenn p und q verschieden sind, kommt in dieser Aufzählung außer pq (= qp) keine Zahl doppelt vor. Also gibt es $p + q - 1$ nicht teilerfremde Zahlen unter den insgesamt n infrage kommenden Zahlen. Teilerfremd sind dann die restlichen Zahlen: $n - (p + q - 1) = p \cdot q - q \cdot 1 - p \cdot 1 + 1 \cdot 1 = (p - 1) \cdot (q - 1)$.

Nebenmodul

RSA als Kryptosystem

Man sucht zwei Primzahlen p und q und verwendet $n = p \cdot q$ als Modul.
Der Verschlüsselungsexponent e muss teilerfremd zu $p-1$ und $q-1$ sein.
Verschlüsselt wird durch Potenzieren modulo n .
Den Entschlüsselungsexponenten finden wir als Lösung von
$$e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$$
durch Anwenden des erweiterten Euklidischen Algorithmus.
Entschlüsselt wird dann wieder durch modulares Potenzieren mit d .

Aufgrund der bisherigen Beobachtungen und der genannten zahlentheoretischen Sätze wird dies gelingen.

Damit ist RSA ein funktionierendes Kryptosystem – wie auch die vorher betrachteten Systeme.

Wo ist die **Asymmetrie** ?

Bei kleinen Zahlen kann man ja alles noch durchprobieren, bei relevanten (=sicheren) Systemen verwendet man Primzahlen mit mehreren hundert Stellen. Die Multiplikation $p \cdot q = n$ ist eine einfache Rechnung.

Wie findet man aber p und q , wenn man n kennt?

Beispiel: Schreibe 220 als Produkt ($10 \cdot 22$)
 Schreibe 221 als Produkt ???

Man muss praktisch alle Zahlen bis zur kleineren der beiden (riesigen) Primzahlen durchprobieren, um p und damit auch q zu finden. Weil es keine weiteren Teiler gibt, kann man auch nicht im Laufe der Rechnung mit kleineren Zahlen weiter rechnen. In der Zeit (in Jahren !), die der schnellste Computer braucht, um diese Zerlegung durchzuführen, liegt die Sicherheit dieses Verfahrens.

Asymmetrie:

Ich kann einem Kommunikationspartner n und e verraten (öffentlicher Teil des Schlüssels).

Damit kann dieser Nachrichten an mich verschlüsseln.

Er kann aber solche Nachrichten nicht entschlüsseln. Auch andere, die n und e kennen, können nur ver- aber nicht entschlüsseln.

Ich selbst kenne den geheimen Teil des Schlüssels, nämlich p und q , und kann den Entschlüsselungsexponenten berechnen und Nachrichten entschlüsseln.
Damit sind Verschlüsselung und Entschlüsselung getrennt.

Authentifizierung:

Wenn ich eine Nachricht bekomme und diese mit meinem Teil des Schlüssels entschlüsseln kann, weiß ich, dass sie von jemandem stammt, der meinen öffentlichen Schlüssel bekommen hat.

Wenn ich etwa jedem Bekannten ein anderes $e - n$ - Paar schicke, kann ich sicher unterscheiden, von wem die Nachricht kommt.

Beispiele / Aufgaben für die Verschlüsselung:

1) Im Projekt „RAS einfach“ können Sie Ver- und Entschlüsselung mit kleinen Primzahlen nachvollziehen. Der Wertebereich des Delphi – Datentyps „Longint“ begrenzt die Größe der Zahlen.

2) Beachten Sie auch die Ausführungen unter www.vom-hau.de/Javascript oder im Verzeichnis „09 Javascript RSA“ zu RSA. Dort ist der Zahlbereich größer, der Entschlüsselungsexponent wird aber immer noch durch Probieren gefunden.

3) Zum Nachrechnen:

$p = 7$ und $q = 11$, also $n = 77$

Hilfreich ist hier wieder eine Excel-Tabelle, die aber so groß ist, dass sie hier nicht auf das Blatt passt. (Die zentrale kopierfähige Formel, mit der auch die bisher abgedruckten Tabellen entstanden sind, lautet: =REST((B2*\$A2); 77))

Wenn man in dieser Tabelle die Zeilen mit den Vielfachen von 7 und den Vielfachen von 11 löscht (nicht teilerfremd zu 77), erkennt man Spalten mit lauter Einsen bei 30 und 60. $60 = (7-1)*(11-1)$ ist der nach dem Fermat-Euler-Satz erwartete Wert, 30 der optimale Wert für die Phasenlänge (nehmen wir hier).

Der Verschlüsselungsexponent muss dann zu 30 teilerfremd sein: $e = 7$?

a	b	Berechnung	
30	7	$r = 30 - 7 = 23$	$23 = 30 - 7$
23	7	$r = 23 - 3*7 = 2$	$2 = (30 - 7) - 3*7$ $= 30 - 4*7$
7	2	$r = 7 - 3*2 = 1$	$1 = 7 - 3*(30 - 4*7)$ $= -3 * 30 + 13 * 7$

13 ist also die Inverse von 7 modulo 30.

Eine Zahl (< 77) zum Verschlüsseln: etwa $m = 31$; die Tabelle liefert für $31^7 \bmod 77$ die Zahl $c = 59$

Entschlüsseln: $59^{13} \bmod 77$ liefert 31.

4) Unter „08 RSA Python“ finden Sie ein Python-Script – getestet unter ActivePython 2.5 – das die Schritte von zwei Primzahlen bis zur Entschlüsselung nachvollzieht. Der Vorteil von Python liegt hier darin, dass es mit beliebig großen ganzen Zahlen umgehen kann.

Schlüsseltausch nach Diffie-Hellmann

$$\left(\text{Basis}^{\text{Exponent1}} \right)^{\text{Exponent2}} = \left(\text{Basis}^{\text{Exponent2}} \right)^{\text{Exponent1}} \pmod{p}$$

Es gibt – außer aufwändigem Durchprobieren – keine Möglichkeit, vom Wert der Potenz auf den Exponenten zu schließen (Problem des diskreten Logarithmus).

In der Tabelle auf Seite 5 können wir die Werte nachschauen; für die Verschlüsselung mit festem Exponent waren die Spalten interessant, jetzt sind es die Zeilen: als g verwendbar sind Zahlen, in deren Zeile kein Wert doppelt vorkommt (und damit alle Zahlen < p vorkommen): 2, 6, 7 und 8. Man nennt diese Zahlen **Primitivwurzeln** der Primzahl p.

Alice	ein „Horcher“	Bob
Einer muss anfangen: p=11 g=6		
wählt geheimes a=3		
rechnet $A=6^3 \bmod 11$		
sendet A = 7	hört p=11, g=6, A=7	empfängt p, g und A
		wählt geheimes b=5
		rechnet $B=6^5 \bmod 11$
	hört B=10	sendet B = 10
empfängt B=10		
rechnet $S=B^a \bmod 11$ $= 10^3 \bmod 11 = 10$		rechnet $S=A^b \bmod 11$ $= 7^5 \bmod 11 = 10$
verwendet S=10 als Schlüssel	kennt also A und B;	verwendet S=10 als Schlüssel
	um S zu finden, muss er daraus a oder b berechnen, was bei den hier verwendeten kleinen Zahlen sehr leicht geht, bei großen Primzahlen aber unrealistisch lange dauert.	