Hochschule München

Fakultät für Informatik und Mathematik

Implementierung eines Verschlüsselungstools

Bachelorarbeit im Studiengang Scientific Computing

Verfasst von: Philipp Alexander Durry
Matrikelnummer: 04657713

Version vom: 20. April 2018

Prüfer: Prof. Dr. Josef Hörwick

Erklärung

Hiermit erkläre ich, dass ich die Bachelorarbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

München, 20. April 2018 Unterschrift

Zusammenfassung

Diese Bachelorarbeit im Studiengang Scientific Computing befasst sich mit der Implementierung von Verschlüsselungsverfahren. Die Arbeit besteht aus einem praktischen sowie einem theoretischen Teil. Für den praktischen Teil wurde ein Verschlüsselungstool mit graphischer Benutzeroberfläche (GUI) in Java programmiert. Dort stehen mehrere Verschlüsselungsverfahren zur Auswahl, die in der GUI benutzerfreundlich angewendet und ausprobiert werden können. Mit den kryptographischen Algorithmen können Textnachrichten ver- und entschlüsselt werden.
Der theoretische Teil umfasst die vorliegende schriftliche Ausfertigung und beschreibt und erklärt die implementierten Verschlüsselungsverfahren und dokumentiert deren Anwendung.
Die Auswahl der behandelten Verschlüsselungsverfahren deckt einige der wichtigsten Konzepte der Kryptologie ab. Darunter befinden sich sowohl symmetrische als auch asymmetrische Verfahren, alte Verfahren der klassischen Kryptographie sowie moderne aktuell verwendete Verfahren.

Die ausführbare Software sowie deren Programmcode befinden sich auf der beiliegenden CD-ROM. Um das Verschlüsselungstool verwenden zu können, muss Java 8 oder höher installiert sein. Mit einem Doppelklick auf die Datei EnDeCrypt.jar lässt sich das Programm samt GUI starten.

Inhaltsverzeichnis

1 Einführung

Kryptographie (aus dem Griechischen kryptós: „geheim“ und gráphein: „schreiben“ [**Gem65**]), die Wissenschaft der Geheimschriften ist aus unserem heutigen Leben nicht mehr wegzudenken. Sie ist unerlässlich für Militär, Geheimdienste und Regierungsapparate, schützt Wirtschaftsunternehmen vor Industriespionage und ermöglicht jedem Bürger geschützte private Kommunikation. Ohne Kryptographie wären zahlreiche Innovationen der letzten Jahre, vor allem im Bereich des Internets, nicht möglich gewesen, wie z.B. Online-Banking, Online-Shopping oder Kryptowährungen.
Die Kryptographie ist eine schon mehrere tausend Jahre alte Wissenschaft und verbindet die Mathematik mit der Informatik. Hinter den meisten Verschlüsselungsverfahren verbergen sich Algorithmen, die mathematische Konzepte nutzen. Um verschlüsselte Nachrichten bzw. Daten wieder lesbar zu machen, braucht es zu jedem Verschlüsselungsverfahren eine Umkehrfunktion, die den Ursprungstext wiederherstellen kann. Zum Ver- und Entschlüsseln benötigt man in der Regel einen geheimen Schlüssel. Versucht z.B. ein Angreifer eine Nachricht ohne Kenntnis des Schlüssels zu „knacken“, so spricht man von **Kryptanalyse**.

Da ich mich für Kryptographie begeistere und das Verschlüsseln von Informationen faszinierend finde, wollte ich meine Bachelorarbeit in diesem Bereich schreiben. In einer Vorlesung hatte ich bereits die theoretische Seite kennengelernt und wollte mich nun mit Kryptographie in der Praxis beschäftigen. Dazu habe ich ausgewählte Verschlüsselungsverfahren selbst programmiert und diese anschließend angewandt. Dabei war mir wichtig, neben älteren, klassischen Verfahren, auch modernere, aktuell praxisrelevante Verfahren zu implementieren. Dadurch konnte ich einen tiefen Einblick in die Architektur und Konzeption von Verschlüsselungsverfahren erhalten.
In den folgenden Kapiteln erläutere ich meine praktische Arbeit und stelle die relevanten Verschlüsselungsverfahren vor.

2 Das Verschlüsselungstool - EnDeCrypt

Um die implementierten Verschlüsselungsverfahren bequem ausprobieren bzw. anwenden zu können, wurde ein Anwendungsprogramm mit graphischer Benutzeroberfläche (GUI) entwickelt. Als Programmiersprache diente **Java**, sowohl für die GUI, als auch für die einzelnen Verschlüsselungsverfahren.
Der Name „**EnDeCrypt**“ setzt sich zusammen aus den englischen Wörtern für verschlüsseln und entschlüsseln: „encrypt“ und „decrypt“.

Mit diesem Programm lassen sich Texte mit einem zuvor gewählten Verschlüsselungsverfahren sowohl ver- als auch entschlüsseln. Verfügen zwei oder mehr Kommunikationspartner über das Programm, so ist eine verschlüsselte Kommunikation möglich, indem man die Nachrichten sowie evtl. Schlüssel über ein weiteres externes Programm, z.B. per E-Mail versendet.

2.1 Die graphische Benutzeroberfläche (GUI)

Abbildung 2.1: Das Willkommensfenster von **EnDeCrypt**

2.1.1 Aufbau und Anwendung

In der Menüleiste kann der Benutzer unter „Encryption Algorithm“ aus dem Drop-down-Menü ein Verschlüsselungsverfahren wählen. Nach der Auswahl teilt sich das Fenster in zwei Bereiche:
Der linke Bereich ist abhängig vom jeweils gewählten Verfahren und dient dazu, die erforderlichen Parameter anzugeben bzw. einzustellen. Meist muss hier ein Schlüssel angegeben bzw. automatisch erzeugt werden. Der größere rechte Bereich ist unabhängig vom gewählten Verfahren stets gleich aufgebaut. Er enthält zwei Textfelder, in welche Text per Hand eingetippt oder über Copy-Paste eingefügt werden kann. Das obere Plaintext-Textfeld ist für Klartext, also unverschlüsselten Text vorgesehen, wohingegen das Ciphertext-Textfeld für verschlüsselten Text bestimmt ist.
Durch Betätigen des Encrypt-Buttons wird Klartext aus dem Plaintext-Textfeld mit dem ausgewählten kryptographischen Verfahren verschlüsselt und der Geheimtext anschließend in das Ciphertext-Textfeld geschrieben. Analog bewirkt der Decrypt-Button das Entschlüsseln eines Geheimtexts und schreibt das Ergebnis in das Plaintext-Textfeld. Zum

Ver- und Entschlüsseln wird stets der im linken Bereich aktuell eingetragene Schlüssel verwendet.

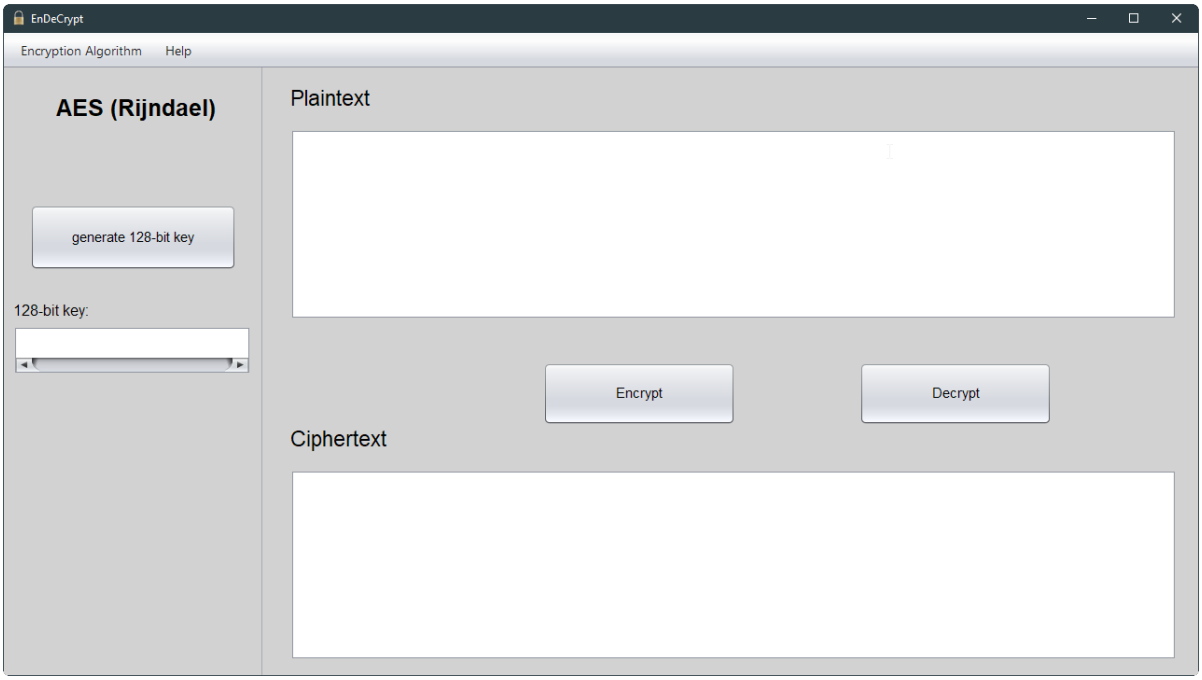

Abbildung 2.2: Die graphische Benutzeroberfläche

2.1.2 Die Verschlüsselungsverfahren

Im Auswahlmenü steht eine Auswahl an symmetrischen und asymmetrischen Verschlüsselungsverfahren zur Verfügung. Diese werden in *Kapitel 3 Symmetrische Verschlüsselung* und *Kapitel 4 Asymmetrische Verschlüsselung* beschrieben. Hier eine Übersicht über die implementierten Verfahren und deren charakterisierende Eigenschaften:

Verschlüsselungsverfahren		
Cäsar Verschlüsselung	symmetrisch	monoalphabetisch
Vigenère Verschlüsselung	symmetrisch	polyalphabetisch
One Time Pad	symmetrisch	polyalphabetisch/bitweise
AES (Rijndael)	symmetrisch	bitweise
RSA	asymmetrisch	bitweise

2.2 Die Softwarearchitektur

Das Verschlüsselungstool ist nach dem „**Model View Controller**“ Architekturmuster strukturiert. Die Software besteht aus insgesamt sieben Java Klassen, wovon die Klassen **Caesar**, **Vigenere**, **OneTimePad**, **AES** und **RSA** den Model-Teil implementieren, die Klasse **GUI** für den View-Teil zuständig ist sowie die Klasse **Controller** den Controller-Teil übernimmt.
Wie die Klassennamen bereits vermuten lassen, enthält der Model-Teil also die Logik der einzelnen Verschlüsselungsverfahren. Jedes Verfahren steht somit in seiner eigenen Klasse. Jede dieser Klassen enthält u.a. die statischen Methoden *encrypt()* sowie *decrypt()*. Die Methode *encrypt()* implementiert den kryptographischen Algorithmus und erhält als Übergabeparameter meist den Klartext und den Schlüssel, wohingegen die Methode *decrypt()* die dazugehörige Umkehrfunktion implementiert und als Parameter den Geheimtext sowie ebenfalls den Schlüssel erhält.
Der View-Teil der Anwendung, das bedeutet also die graphische Benutzeroberfläche, wird von der Klasse **GUI** beschrieben. Dort sind das Layout, alle Menüs, Textfelder und Buttons usw. festgelegt.
Die Klasse **Controller** agiert als Bindeglied und verbindet die Benutzeroberfläche mit der Logik der Verschlüsselungsverfahren und steuert den Ablauf.
Der Vorteil dieses modularen Aufbaus ist, dass einzelne Module bzw. Klassen leicht durch neue ausgetauscht werden können, ohne dass das gesamte Programm umgearbeitet werden muss. So könnte man beispielsweise die GUI durch eine neue GUI mit unterschiedlichem Design ersetzen, ohne die anderen Klassen verändern zu müssen.

2.3 ASCII Zeichensatz

Damit **EnDeCrypt** nicht nur Buchstaben, sondern auch Leerzeichen und Sonderzeichen verschlüsseln kann, verwendet die Software als zugrunde liegendes Alphabet den ASCII Zeichensatz. Als kryptographisches Alphabet werden in der Literatur häufig lediglich die 26 Buchstaben von A bis Z verwendet, womit praxistaugliche Textnachrichten, die z.B. Satzzeichen enthalten, aber nicht verschlüsselt werden können. Der ASCII Zeichensatz beinhaltet die folgenden Zeichen, welche **EnDeCrypt** demnach korrekt ver- und entschlüsseln kann:

Dez	Hex	Okt	ASCII	Dez	Hex	Okt	ASCII	Dez	Hex	Okt	ASCII	Dez	Hex	Okt	ASCII
0	0x00	000	NUL	32	0x20	040	SP	64	0x40	100	@	96	0x60	140	`
1	0x01	001	SOH	33	0x21	041	!	65	0x41	101	A	97	0x61	141	a
2	0x02	002	STX	34	0x22	042	"	66	0x42	102	B	98	0x62	142	b
3	0x03	003	ETX	35	0x23	043	#	67	0x43	103	C	99	0x63	143	c
4	0x04	004	EOT	36	0x24	044	$	68	0x44	104	D	100	0x64	144	d
5	0x05	005	ENQ	37	0x25	045	%	69	0x45	105	E	101	0x65	145	e
6	0x06	006	ACK	38	0x26	046	&	70	0x46	106	F	102	0x66	146	f
7	0x07	007	BEL	39	0x27	047	'	71	0x47	107	G	103	0x67	147	g
8	0x08	010	BS	40	0x28	050	(	72	0x48	110	H	104	0x68	150	h
9	0x09	011	HT	41	0x29	051	)	73	0x49	111	I	105	0x69	151	i
10	0x0A	012	LF	42	0x2A	052	*	74	0x4A	112	J	106	0x6A	152	j
11	0x0B	013	VT	43	0x2B	053	+	75	0x4B	113	K	107	0x6B	153	k
12	0x0C	014	FF	44	0x2C	054	,	76	0x4C	114	L	108	0x6C	154	l
13	0x0D	015	CR	45	0x2D	055	-	77	0x4D	115	M	109	0x6D	155	m
14	0x0E	016	SO	46	0x2E	056	.	78	0x4E	116	N	110	0x6E	156	n
15	0x0F	017	SI	47	0x2F	057	/	79	0x4F	117	O	111	0x6F	157	o
16	0x10	020	DLE	48	0x30	060	0	80	0x50	120	P	112	0x70	160	p
17	0x11	021	DC1	49	0x31	061	1	81	0x51	121	Q	113	0x71	161	q
18	0x12	022	DC2	50	0x32	062	2	82	0x52	122	R	114	0x72	162	r
19	0x13	023	DC3	51	0x33	063	3	83	0x53	123	S	115	0x73	163	s
20	0x14	024	DC4	52	0x34	064	4	84	0x54	124	T	116	0x74	164	t
21	0x15	025	NAK	53	0x35	065	5	85	0x55	125	U	117	0x75	165	u
22	0x16	026	SYN	54	0x36	066	6	86	0x56	126	V	118	0x76	166	v
23	0x17	027	ETB	55	0x37	067	7	87	0x57	127	W	119	0x77	167	w
24	0x18	030	CAN	56	0x38	070	8	88	0x58	130	X	120	0x78	170	x
25	0x19	031	EM	57	0x39	071	9	89	0x59	131	Y	121	0x79	171	y
26	0x1A	032	SUB	58	0x3A	072	:	90	0x5A	132	Z	122	0x7A	172	z
27	0x1B	033	ESC	59	0x3B	073	;	91	0x5B	133	[	123	0x7B	173	{
28	0x1C	034	FS	60	0x3C	074	<	92	0x5C	134	\	124	0x7C	174	|
29	0x1D	035	GS	61	0x3D	075	=	93	0x5D	135	]	125	0x7D	175	}
30	0x1E	036	RS	62	0x3E	076	>	94	0x5E	136	^	126	0x7E	176	~
31	0x1F	037	US	63	0x3F	077	?	95	0x5F	137	_	127	0x7F	177	DEL

Abbildung 2.3: Der ASCII Zeichensatz zitiert aus [**Bil18c**]

Die obige Tabelle listet alle 128 ASCII Zeichen auf und war für die Erstellung dieser Bachelorarbeit unerlässlich. Jedes Zeichen ist einem eindeutigen Zahlenwert zugeordnet, der jeweils in den drei Zahlensystemen Dezimal, Hexadezimal und Oktal angegeben ist. Davon sind die Zeichen mit den Indizes 0 bis 31 und das Zeichen mit Index 127 nicht druckbare Steuerzeichen. Diese können also nicht als Text dargestellt werden und sind für uns somit nicht relevant. Lediglich das Steuerzeichen „LF“ mit Index 10 ist wichtig. Es steht für engl. „line feed“ und steht für einen Zeilenvorschub. Dieses Zeichen wird benötigt um Texte auch über mehrere Zeilen schreiben zu können. In **EnDeCrypt** können also die 95 druckbaren Zeichen mit den Indizes 32 bis 126 sowie der Zeilenvorschub korrekt ver- und entschlüsselt werden.

Da es genau 128 Zeichen sind, reichen sieben Bit aus, um alle Symbole kodieren zu können. Der Computer speichert die Zeichen in der Binärdarstellung ab. Der Buchstabe „A“ beispielsweise mit dem dezimalen Zahlenwert 65 lautet als Binärzahl demnach „1000001“. Um Verschlüsselungsverfahren programmieren zu können, welche Text auf Bit-Ebene verschlüsseln, ist es notwendig die Binärdarstellung der Zeichen zu kennen.

3 Symmetrische Verschlüsselung

Die meisten Verschlüsselungsalgorithmen verwenden einen geheimen Schlüssel um Nachrichten zu verschlüsseln und entschlüsseln. Man spricht dann auch von **Secret-Key-Verfahren** oder symmetrischer Verschlüsselung. Im Gegensatz dazu steht die asymmetrische Verschlüsselung bzw. **Public-Key-Verfahren**, die in Kapitel 4 behandelt werden [**Sch13**].

Für mathematische Beschreibungen wird in der Literatur für den Klartext häufig der Buchstabe m (engl. message) verwendet. Analog wird der Geheimtext als c (engl. ciphertext) bezeichnet und der Schlüssel mit k (engl. key). Im mathematischen Sinne sind Verschlüsseln und Entschlüsseln Funktionen, die mit den Buchstaben e (engl. encrypt) und d (engl. decrypt) abgekürzt werden. So kann man den Zusammenhang von Klartext, Geheimtext und Schlüssel folgendermaßen mathematisch formulieren [**Sch13**]:

$$e(m, k) = c$$

$$d(c, k) = m$$

Mit dieser Struktur wurden auch auch die Funktionen *encrypt()* und *decrypt()* der einzelnen Verschlüsselungsklassen implementiert. **Abbildung 3.1** verdeutlicht die Funktionsweise des Schlüssels graphisch.

Da bei der symmetrischen Kryptographie beide Kommunikationspartner den geheimen Schlüssel brauchen, müssen sie sich auf einen gemeinsamen Schlüssel einigen bzw. den Schlüssel auf irgendeinem Weg austauschen. Die größte Schwäche der symmetrischen Verschlüsselung besteht im Schlüsselaustausch, denn ein unverschlüsselt übertragener Schlüssel kann abgehört werden und ist somit nicht mehr geheim. Um das Schlüsselaustauschproblem zu lösen, gibt es eigens dafür entwickelte Verfahren wie z.B. den Diffie-Hellman-Schlüsselaustausch, aber auch der RSA Algorithmus aus Kapitel 4 kann dafür verwendet werden.

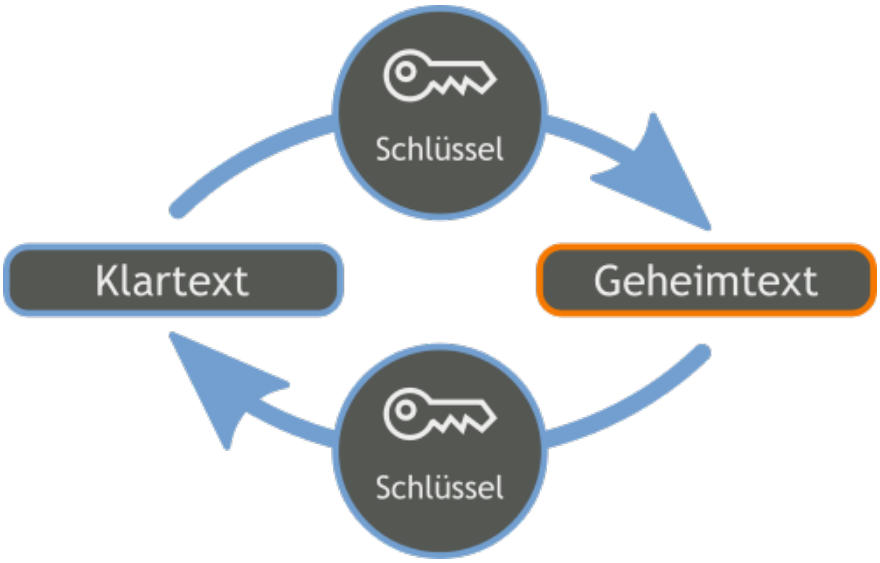

Abbildung 3.1: Darstellung des Schlüsselprinzips, Bildquelle: [**Bil18d**]

3.1 Cäsar Verschlüsselung

Das erste symmetrische Verschlüsselungsverfahren dieser Bachelorarbeit ist die Cäsar Verschlüsselung oder Cäsar Chiffre. Dieses wurde wohl bereits von Julius Cäsar verwendet, um militärische Nachrichten zu übermitteln [**Ert12**].
Dieses Verfahren gehört zu den monoalphabetischen Substitutionschiffren, was bedeutet, dass dabei Buchstaben bzw. Zeichen durch andere ausgetauscht (= substituiert) werden und gleiche Buchstaben aus dem Klartext immer auf gleiche andere Buchstaben im Geheimtext abgebildet werden. Im Gegensatz dazu stehen die polyalphabetischen Substitutionschiffren, bei denen gleiche Buchstaben im Klartext auf verschiedene Buchstaben im Geheimtext abgebildet werden können, was die Sicherheit eines Verschlüsselungsverfahrens stark erhöht. Dies wird im Folgenden an einem Beispiel nochmals genauer erläutert [**Sch13**].

Funktionsweise

Da für alle 128 ASCII Zeichen der Platz nicht ausreicht und es viel zu unübersichtlich würde, wird die Funktionsweise anhand des lateinischen Alphabets mit 26 Buchstaben erklärt. Die Konzepte lassen sich jedoch auf beliebig große andere Alphabete übertragen.

Die Cäsar Verschlüsselung verwendet eine Alphabetverschiebung um Klartextbuchstaben auf Geheimtextbuchstaben abzubilden. Dazu wird jeder Buchstabe des Geheimtextalphabets um einen konstanten Wert verschoben, wobei die Verschiebung der geheime Schlüssel ist.

Beispiel mit Verschiebung = 5:

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
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

In der Tabelle ist das untere Alphabet um 5 Stellen zum oberen verschoben. Der Klartext „HALLO“ würde damit demnach auf den Geheimtext „MFQQT“ abgebildet werden. Da es sich wie oben bereits erwähnt um eine monoalphabetische Verschlüsselung handelt, wird der Buchstabe „L“ beide Male auf den gleichen Buchstaben „Q“ abgebildet. Bei einer polyalphabetischen Verschlüsselung könnte das „L“ auf unterschiedliche Buchstaben abgebildet werden, wodurch der Klartext offensichtlich stärker verschleiert würde.

Weist man den Zeichen eines Alphabets Indizes von 0 bis $n-1$ zu, wobei n die Größe des Alphabets sei, kann man die Cäsar Chiffre auch mathematisch formulieren. Seien außerdem b der Index eines Buchstaben im Klartextalphabet, c der Index eines Buchstaben im Geheimtextalphabet und k der Schlüssel (Verschiebung), so gilt für das Verschlüsseln [**Ert12**]:

$$b \mapsto (b + k) \mod n$$

sowie für das Entschlüsseln:

$$c \mapsto (c - k) \mod n$$

Für das Alphabet der ASCII Zeichen gilt $n = 95$.

Demonstration

Im Folgenden wird für jedes kryptographische Verfahren dessen Implementierung demonstriert und dazu stets der selbe Klartext verschlüsselt. So lassen sich im Geheimtext anschaulich die Unterschiede der verschiedenen Verfahren erkennen.
Der Klartext, welcher als Beispiel dienen soll, lautet:

```
Ludwig Bechstein
Deutsches Maerchenbuch, 1847
Der Hase und der Fuchs

Ein Hase und ein Fuchs reisten beide mit einander.
Es war Winterszeit, gruente kein Kraut, und auf dem Felde kroch weder Maus noch Laus.
"Das ist ein hungriges Wetter," sprach der Fuchs zum Hasen, "mir schnurren alle Gedaerme zusammen."
```

Dieser Textausschnitt eignet sich gut, da er sowohl Klein- und Großbuchstaben enthält, als auch Zahlen, Sonderzeichen, Leerzeichen und Zeilenvorschübe.
Der mit Verschiebung = 34 in **EnDeCrypt** erzeugte Geheimtext lautet wie folgt:

```
n8':,*Bd(&+67(,1
f(876&+(6Bo$(5&+(1%8&+NBSZVY
f(5Bj$6(B81'B'(5Bh8&+6

g,1Bj$6(B81'B(,1Bh8&+6B5(,67(1B%(,'(B0,7B(,1$1'(5P
g6B:$5By,17(56=(,7NB*58(17(B.(,1Bm5$87NB81'B$8)B'(0Bh(/'(B.52&+B:('(5Bo$86B12&+Bn$86P
Df$6B,67B(,1B+81*5,*(6By(77(5NDB635$&+B'(5Bh8&+6B=80Bj$6(1NBD0,5B6&+1855(1B$//(Bi('$(50(B=86$00(1PD
```

Man kann gut erkennen wie alle Zeichen (inklusive Leerzeichen) des Klartextes ersetzt wurden. Die Struktur des Ausgangstextes bleibt aber weitgehend erhalten und birgt ein Sicherheitsrisiko.
Dieses Verschlüsselungsverfahren ist auch deshalb sehr unsicher, da ein Angreifer lediglich die Verschiebungen 1 bis $n - 1$ durchprobieren muss, um den Text zu entschlüsseln. Die Verschiebung um n Stellen entspricht einer Verschiebung von 0 und bewirkt daher keine Verschlüsselung.

Eine weitere Angriffsmethode, die bei allen monoalphabetischen Chiffren funktioniert, ist die sog. Häufigkeitsanalyse, die das Vorkommen der einzelnen Zeichen in einem gegebenen Geheimtext zählt. Mithilfe der Buchstabenhäufigkeiten einer Sprache, können so Rückschlüsse auf die Buchstaben des Klartextes gezogen werden. Beispielsweise sind in der deutschen Sprache die Buchstaben E und N die häufigsten mit 17,4% sowie 9,78%. Je länger ein Geheimtext ist, umso aussagekräftiger wird die Häufigkeitsanalyse [**Ert12**].

3.2 Vigenère Verschlüsselung

Die Vigenère Verschlüsselung stammt aus dem 16. Jahrhundert und hat große Ähnlichkeiten zur Cäsar Chiffre, ist aber keine monoalphabetische sondern eine polyalphabetische Verschlüsselung. Dadurch können Klartextbuchstaben auf unterschiedliche Geheimtextbuchstaben abgebildet werden, was die Sicherheit des Geheimtextes stark erhöht. Allerdings ist auch dieses Verfahren für die moderne Kryptographie nicht mehr geeignet und findet in der Praxis keine Anwendung [**Ert12**].

Funktionsweise

Wie bereits die Cäsar Verschlüsselung verschiebt die Vigenère Verschlüsselung Buchstaben im Alphabet. Dazu verwendet sie als Verschiebeschlüssel aber nicht nur eine Zahl, sondern ein ganzes Wort. Das Vorgehen soll auch hier wieder anhand des lateinischen Alphabets mit 26 Buchstaben erklärt, und anschließend die Verschlüsselung mit dem ASCII Alphabet demonstriert werden. Am besten lässt sich das Verfahren direkt an einem **Beispiel** demonstrieren:
Man nehme an, es solle der Text „Das ist ein Klartext“ mit dem Schlüsselwort „Schluessel“ verschlüsselt werden. Dazu wird das Schlüsselwort Buchstabe für Buchstabe über den Klartext geschrieben und gegebenenfalls wiederholt bis der gesamte Text abgedeckt ist. Anschließend wird für jedes Buchstabenpaar aus Schlüssel und Klartext eine Cäsar Verschlüsselung durchgeführt. Man verschiebt also einen Klartextbuchstaben um den darüberstehenden Schlüsselbuchstaben. Der Wert der Verschiebung ist dabei die Stelle des Schlüsselbuchstabens im Alphabet. Dazu werden den Buchstaben A bis Z wieder die Zahlen 0 bis 25 zugewiesen [**Ert12**].

Schlüsselwort:	S	C	H	L	U	E	S	S	E	L	S	C	H	L	U	E	S
Klartext:	D	A	S	I	S	T	E	I	N	K	L	A	R	T	E	X	T
Geheimtext:	V	C	Z	T	M	X	W	A	R	V	D	C	Y	E	Y	B	L

Der Geheimtext lautet nun also „VCZ TMX WAR VDCYEYBL“.

Alternativ kann man auch das sog. Vigenère-Quadrat verwenden. Dabei sucht man einen Klartextbuchstaben in seiner Spalte und den Schlüsselbuchstaben in den Zeilen. Der Schnittpunkt ergibt dann ebenfalls den Geheimtextbuchstaben [**Ert12**].

Vigenère-Quadrat

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

Demonstration

Es soll wieder der selbe Klartext wie in Kapitel 3.1 als Beispiel dienen, um die Vigenère Verschlüsselung auf dem ASCII Alphabet zu demonstrieren. Mit dem Schlüsselwort „Passwort123“ ergibt sich folgender Geheimtext:

```
|WXkaWr7vu{DVY]f
7Z"'4JYgw=TZ$u{6PVi[X~tBJGg
8Yjo;V%w3FPXs\UetW(v9U

=YatYs'6aib\oX^ 2YFE\gwbX^%'x?aVYaTXt~{(PG]bY^WZ$@
uUskYbrLz!(6Tgn]Yg!1y&FGbh]o^Zz!3{TUil{rj v32WZs\U`tWw 5Gs_j_V]1*x5GfsEQhh1!#4Js@Yef#
4W2Us]kdrZz!39Wb[jYZZ%2j6VhYj{tt%#&2E\s\UetW(v9Usnm]r=r&x?msueYet%u{?Wff]^rV}~xP)YXYUebv2.FUUaeUa#3
```

Im Vergleich mit der Cäsar Verschlüsselung fällt auf, dass die Struktur des Ausgangstextes ebenfalls weitgehend erhalten bleibt. Allerdings werden gleiche Klartextbuchstaben nun auf unterschiedliche Geheimtextbuchstaben abgebildet. Dadurch lässt sich eine polyalphabetische Verschlüsselung nicht durch eine Häufigkeitsanalyse „knacken“ [**Ert12**].

3.3 One Time Pad

Beim One Time Pad, welches 1917 erfunden wurde, handelt es sich um eine polyalphabetische Verschlüsselung, die der Vigenère Verschlüsselung ähnelt. Die Besonderheit des One Time Pad ist allerdings, dass das Schlüsselwort genauso lang wie der Klartext ist. Der Schlüssel darf nicht wie bei der Vigenère Verschlüsselung wiederholt oder mehrfach verwendet werden [**Ert12**].

Pseudozufallszahlen und Schlüsselerzeugung

Dieses Verfahren kann wie die beiden vorherigen für ein beliebiges Alphabet implementiert werden. In der Praxis verschlüsselt man aber meist binär, weshalb die Implementierung in **EnDeCrypt** auf Bit-Ebene arbeitet. Dazu müssen die Zeichen des Klartextes zuerst in ihre Bitdarstellung umgeschrieben werden. Es gelten dabei die Kodierungen aus der ASCII Tabelle von Kapitel 2.3. Nehmen wir beispielsweise an, wir möchten die Nachricht „Text“ verschlüsseln. Deren Binärdarstellung entspricht der Bitfolge „01010100 01100101 01111000 01110100“, denn:

Klartextbuchstabe:	T	e	x	t
Dezimaldarstellung:	84	101	120	116
Binärdarstellung:	01010100	01100101	01111000	01110100

Nun brauchen wir einen Schlüssel, welcher genauso lang wie die Nachricht selbst ist. Beim binären One Time Pad ist der Schlüssel nun kein Wort, sondern eine Folge von Bits. Da es nicht sinnvoll wäre, sich eine Bitfolge selbst zu überlegen, überlässt man dies dem Computer. Dieser kann sehr schnell sehr lange Bitfolgen erzeugen und dabei noch eine weitere wichtige Anforderung an den Schlüssel erfüllen: Die Werte der Bits sollen zufällig gewählt werden. Hierfür kommt ein sog. Pseudozufallszahlengenerator zum Einsatz. Pseudo deshalb, weil der Computer keine rein zufälligen Zahlen erzeugen kann, die erzeugten Zahlen scheinen nur sehr zufällig zu sein. Die Pseudozufallszahlen eines guten Generators reichen für die Praxis aber völlig aus. In den Java Bibliotheken gibt es bereits implementierte Pseudozufallszahlengeneratoren, die man in seine eigenen Programme einbinden kann. Um die Verschlüsselungsverfahren kryptographisch so sicher wie möglich zu implementieren, wurde für die Erzeugung der Zufallsbits die Java Klasse

SecureRandom aus dem **java.security** Paket verwendet. Hierbei handelt es sich um einen speziell für kryptographische Zwecke entworfenen Pseudozufallszahlengenerator, dessen Zufallszahlen besonders wenige statistische Abhängigkeiten aufweisen. Je zufälliger der Schlüssel ist, desto robuster ist das Verfahren gegen Angriffe mit statistischen Methoden.

Verschlüsselung mit dem One Time Pad

Mit dem Schlüssel aus Zufallsbits können nun die Bits des Klartextes verschlüsselt werden. Dazu werden der Reihe nach jeweils ein Schlüsselbit und ein Klartextbit XOR - verknüpft. Das Ergebnis der XOR Operationen ist dann der Geheimtext. Die logische XOR Verknüpfung (exklusives Oder) zweier Bits ist wie folgt definiert:

0 XOR 0 = 0
0 XOR 1 = 1 symbolische Schreibweise: $\oplus$
1 XOR 0 = 1
1 XOR 1 = 0

Setzen wir nun unser oben begonnenes Beispiel fort und wenden die XOR Verknüpfung an, erhalten wir eine veränderte Bitfolge, welche die Bits des Geheimtextes sind:

Klartextbuchstabe:	T	e	x	t
Klartextbits:	01010100	01100101	01111000	01110100
Zufallsbits:	01101111	11100001	10000001	00111110
Ergebnis der XOR Verknüpfung:	00111011	10000100	11111001	01001010

Um den Geheimtext zu entschlüsseln, muss lediglich die XOR Verknüpfung auf den Geheimtextbits zusammen mit dem passenden Schlüssel erneut angewendet werden und damit ergeben sich wieder die Klartextbits. Das liegt daran, dass die Umkehrfunktion zur XOR Verknüpfung ebenfalls die XOR Verknüpfung ist. Die XOR Funktion ist also selbstinvers [**Ert12**].

Demonstration

Durch die bitweise Verschlüsselung entstehen im Geheimtext Bitfolgen, die nicht mehr als druckbare ASCII Zeichen dargestellt werden können. Aus diesem Grund werden als Geheimtext für die bitweisen Verschlüsselungsmethoden keine ASCII Zeichen mehr angegeben, sondern stattdessen Hexadezimal oder Dezimalzahlen. Bei den bitweisen Verfahren werden nun sogar Zeilenvorschübe mit verschlüsselt, wodurch die Struktur des Ursprungstextes vollständig verschleiert wird. Dies ist der mit dem One Time Pad verschlüsselte Klartext aus Kapitel 3.1 in hexadezimaler Zahlendarstellung:

```
46be93461424b33824a8d8c1e41c5dcd82b36cf4ea141c16fa467819084691adf25ae490d39acc4
9a30a8bbcfe82eee4c0de159aee94a9c5c0d0d035a72b55cba77f5b4ea0935fb74f8281d8e800b2
52d9c637c1e2ed094f4d9622461dab0eb113d6fa96ebe723ed06925d00c438465b8ff1c5a8d749b
3f9cb3e26527cf291aef43a64a60d5fd15f0465c14c0684b571ba79dc966ea03a66d35f3eaedd5c3
32acfab471a203f12420bf11b5f61e2bf9a7dda167e9302175531e0470a744f57aa431f87c9492f6a
16621f16b56dec548b5e99f66c207256039f6fb743d20ae9b8a0d8b6bc9acd6b1f5bd462d3390c01
1115e5c023124fbc4cf539290dee44a3c729fa27e1c17936392a74e8ccc193419df041b25ffab4cab
5b7386b59937f0df37428eda87567870fe11170c2b9db84861785
```

Der dazugehörige Schlüssel sieht ganz ähnlich aus, denn er muss ja exakt gleich lang sein wie der Klartext/Geheimtext.

Vorteile, Nachteile und Sicherheit

Macht ein Anwender beim Verschlüsseln mit dem One Time Pad alles richtig, dann hat ein Angreifer keine Chance den Geheimtext ohne Kenntnis des Schlüssels zu knacken. Unter der Voraussetzung, dass echte Zufallszahlen und nicht nur Pseudozufallszahlen verwendet werden, ist das One Time Pad mathematisch nachweislich absolut sicher. Man spricht auch von einer **perfekten** Chiffre. Echte Zufallszahlen sind in der Praxis zwar nicht möglich, aber sehr gute Pseudozufallszahlen bieten ebenfalls eine fast perfekte Sicherheit. Trotz der enormen Sicherheit, bringt dieses Verfahren auch einen gravierenden Nachteil mit sich, der die Einsatzgebiete in der Praxis stark einschränkt. Das größte Problem besteht im Schlüsselaustausch, denn der Schlüssel muss ja mindestens so lang sein wie die Nachricht bzw. die Daten selbst und kostet damit viel Speicherplatz. So ein großer Schlüssel kann z.B.

nicht schnell und effizient über das Internet übertragen werden. Um das One Time Pad effizient verwenden zu können, sollten schon vor der Kommunikation mehrere Schlüssel vereinbart werden und zum Beispiel über eine CD oder einen USB Stick geteilt werden. Wichtig ist auch, dass jeder Schlüssel unbedingt nur einmal verwendet werden darf, da sonst die statistische Unabhängigkeit zwischen Schlüssel und Geheimtexten verloren ginge. Werden diese Dinge bei der Anwendung berücksichtigt, bietet das One Time Pad aber außerordentliche Sicherheit [**Ert12**] [**Sch13**].

3.4 AES (Rijndael)

Die Abkürzung **AES** steht für **Advanced Encryption Standard** und bezeichnet einen offiziellen Standard der US-amerikanischen Behörde National Institute of Standards and Technology. Dieses moderne Verschlüsselungsverfahren entstand im Jahre 1998 im Rahmen eines Wettbewerbs, bei dem ein neues, besseres Verfahren entwickelt werden sollte, um DES (Data Encryption Standard) abzulösen. Das Gewinnerverfahren des Advanced Encryption Standard Wettbewerbs wurde von den Erfindern eigentlich **Rijndael** getauft, wird aber meist als AES bezeichnet. Es handelt sich um ein Verfahren, das die hohen Anforderungen des Wettbewerbs an Sicherheit, Recheneffizienz und Flexibilität erfüllt und gilt aktuell als das bedeutendste symmetrische Verschlüsselungsverfahren und wird in der Praxis sehr häufig eingesetzt [**Sch13**].

Der AES ist eine **Blockverschlüsselung** oder **Blockchiffre**, was bedeutet, dass wie schon beim One Time Pad binär verschlüsselt wird. Die Bits eines Klartextes werden dabei in mehrere Blöcke bzw. Abschnitte eingeteilt, die dann einzeln verschlüsselt werden. Die Blocklänge des AES beträgt 128 Bits, es werden also Blöcke mit 128 Bits gebildet. Die Schlüssellänge ist mit entweder 128, 192 oder 256 Bits variabel [**AES01**]. Die Implementierung für **EnDeCrypt** beschränkt sich auf eine Schlüssellänge von 128 Bits, weshalb die folgenden Erklärungen für diese Variante gelten.

Ein Block $(a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3, c_0, c_1, c_2, c_3, d_0, d_1, d_2, d_3)$ wird beim AES spaltenweise in eine 4×4 - Matrix geschrieben, wobei jeder der 16 Einträge ein Byte (8 Bits) enthält:

$$\begin{matrix} a_0 & b_0 & c_0 & d_0 \\ a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \end{matrix}$$

Eine solche Matrix wird beim AES auch „**State**“ genannt.
Aus Platzgründen und für eine bessere Übersicht schreibt man die Bytes a_0 bis d_3 nicht in ihrer Binärdarstellung, sondern stets als Hexadezimalzahlen [**Sch13**].

3.4.1 Ablauf

Im Folgenden wird beschrieben wie mit dem AES ein einzelner Block bzw. State verschlüsselt wird. Abbildung 3.2 stellt den Ablauf des Algorithmus graphisch dar.

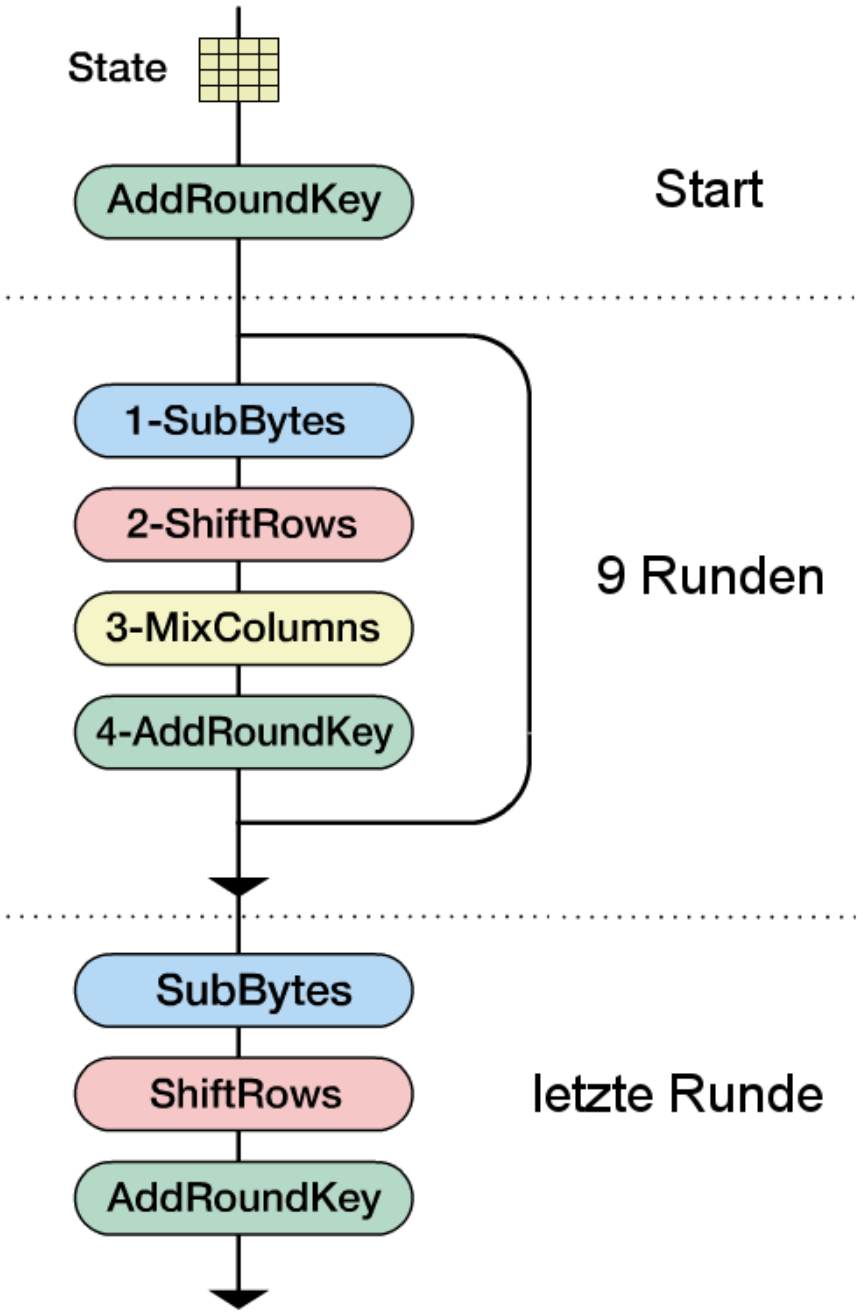

Abbildung 3.2: Ablauf des AES Algorithmus, Bildquelle: [**Bil18a**]

Der Algorithmus des AES verwendet die vier Funktionen **SubBytes**, **ShiftRows**, **MixColumns** und **AddRoundKey**, um eine sichere Verschlüsselung zu erreichen. Diese Funktionen werden mehrmals hintereinander in sog. Runden ausgeführt. Die Anzahl der zu durchlaufenden Runden ist abhängig von der Schlüssellänge, bei einer Schlüssellänge von 128 Bits beträgt die Rundenanzahl = 10.
Jede der vier Funktionen erhält als Übergabeparameter den zu verschlüsselnden State, verändert diesen und liefert den veränderten State als Rückgabeparameter zurück. Zu Beginn des Algorithmus durchläuft der State die Funktion AddRoundKey, anschließend folgen die Runden 1 bis 9 mit den Funktionen SubBytes, ShiftRows, MixColumns und AddRoundKey. Diese werden neun mal in einer Schleife ausgeführt, bevor abschließend die zehnte Runde folgt, die MixColumns nicht mehr beinhaltet. Das Ergebnis ist dann ein vollständig verschlüsselter State bzw. ein Block des Klartextes [**AES01**].

3.4.2 SubBytes, ShiftRows, MixColumns und AddRoundKey

Um die Funktionen erklären zu können, soll der hexadezimale 128 Bits Block
19 3d e3 be a0 f4 e2 2b 9a c6 8d 2a e9 f8 48 08
als Beispiel dienen. Er wird spaltenweise in einen State geschrieben:

$$\begin{array}{cccc} 19 & a0 & 9a & e9 \\ 3d & f4 & c6 & f8 \\ e3 & e2 & 8d & 48 \\ be & 2b & 2a & 08 \end{array}$$

Im Folgenden soll nun dieser State durch die vier Funktionen aufeinanderfolgend verändert werden.

SubBytes

Die Funktion SubBytes stellt eine sog. **S-Box** (engl. substitution box) dar. Mithilfe einer Substitutionstabelle werden alle Bytes eines States durch andere ausgetauscht. Indem man für ein Byte dessen erste Stelle als x-Koordinate (bestimmt die Zeile) und die zweite Stelle als y-Koordinate (bestimmt die Spalte) interpretiert, erhält man an der entsprechenden Position in der Tabelle das Substitutions Byte [**AES01**].

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Abbildung 3.3: S-Box der Funktion SubBytes [**AES01**]

Das Byte mit dem Wert 19 aus dem Beispiel State, müsste demnach durch das Byte d4 ersetzt werden, welches in der Tabelle in Zeile 1 und Spalte 9 steht. Führt man diese Substitution nun für alle Bytes aus, verändert sich der State folgendermaßen:

$$\begin{matrix} 19 & a0 & 9a & e9 \\ 3d & f4 & c6 & f8 \\ e3 & e2 & 8d & 48 \\ be & 2b & 2a & 08 \end{matrix} \longrightarrow \begin{matrix} d4 & e0 & b8 & 1e \\ 27 & bf & b4 & 41 \\ 11 & 98 & 5d & 52 \\ ae & f1 & e5 & 30 \end{matrix}$$

ShiftRows

Die Funktion ShiftRows verändert einen State indem sie die Elemente innerhalb einer Zeile rotiert:

- Die 1. Zeile wird nicht verändert.
- Das Element, welches ganz links in der 1. Spalte der 2. Zeile steht, wird in derselben Zeile in die 4. Spalte nach rechts außen geschrieben. Die restlichen Elemente in den Spalten 2, 3 und 4 werden jeweils um eine Stelle nach links verschoben.
- Analog wird anschließend die 3. Zeile bearbeitet, allerdings „springen“ nun die beiden Elemente aus Spalte 1 und 2 nach rechts in die Spalten 3 und 4. Die übrigen beiden Elemente rotieren dann um zwei Stellen nach links.
- In der letzten 4. Zeile „springen“ nun die drei Elemente der Spalten 1, 2 und 3 nach rechts in die Spalten 2, 3 und 4. Das Element an Stelle 4 verschiebt sich um drei Stellen nach links [**AES01**].

Für das Beispiel ergibt sich schließlich folgende Transformation:

$$\begin{array}{cccc} d4 & e0 & b8 & 1e \\ 27 & bf & b4 & 41 \\ 11 & 98 & 5d & 52 \\ ae & f1 & e5 & 30 \end{array} \longrightarrow \begin{array}{cccc} d4 & e0 & b8 & 1e \\ bf & b4 & 41 & 27 \\ 5d & 52 & 11 & 98 \\ 30 & ae & f1 & e5 \end{array}$$

MixColumns

Die Funktion MixColumns modifiziert jede Spalte eines States indem sie diese mit einer bestimmten Matrix multipliziert. Es wird also eine Matrix-Vektor-Multiplikation durchgeführt, deren Ergebnis die veränderte Spalte ist. Es handelt sich hierbei jedoch nicht um eine gewöhnliche Multiplikation, sondern um eine Multiplikation in Rijndaels Galois-Körper GF(2^8). In diesem Körper sind alle Elemente Bytes, wie auch die Elemente der States bzw. der Matrix [**AES01**].

Hier die Rechnung für die 1. Spalte:

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \circ \begin{pmatrix} d4 \\ bf \\ 5d \\ 30 \end{pmatrix} = \begin{pmatrix} 04 \\ 66 \\ 81 \\ e5 \end{pmatrix}$$

So sieht der veränderte State aus, nachdem die Multiplikation mit allen vier Spalten durchgeführt wurde:

$$\begin{matrix} d4 & e0 & b8 & 1e \\ bf & b4 & 41 & 27 \\ 5d & 52 & 11 & 98 \\ 30 & ae & f1 & e5 \end{matrix} \longrightarrow \begin{matrix} 04 & e0 & 48 & 28 \\ 66 & cb & f8 & 06 \\ 81 & 19 & d3 & 26 \\ e5 & 9a & 7a & 4c \end{matrix}$$

AddRoundKey

In AddRoundKey wird auf einen State ein 128 Bits langer **Rundenschlüssel** (engl. round key) addiert. Die Rundenschlüssel werden zu Beginn des Verschlüsselungsverfahren mit einem eigenen Algorithmus aus dem geheimen 128 Bits AES Schlüssel erzeugt. Für jede Runde wird ein eigener Rundenschlüssel benötigt, es müssen also insgesamt 10 Rundenschlüssel generiert werden. Der Rundenschlüssel, der zu Beginn der Verschlüsselung beim ersten Aufruf von AddRoundKey verwendet wird, ist der unveränderte geheime AES Schlüssel.
Bei der Addition handelt sich um eine XOR-Verknüpfung, wie sie in Kapitel 3.3 beschrieben ist. Dabei wird ein Rundenschlüssel ebenfalls in Form einer 4×4 - Matrix geschrieben und anschließend werden die Elemente beider Matrizen komponentenweise XOR-verknüpft [**AES01**].

Sei der hexadezimale Rundenschlüssel = **a0 fa fe 17 88 54 2c b1 23 a3 39 39 2a 6c 76 05**, dann wird der State aus dem Beispiel folgendermaßen verändert:

$$\begin{matrix} 04 & e0 & 48 & 28 \\ 66 & cb & f8 & 06 \\ 81 & 19 & d3 & 26 \\ e5 & 9a & 7a & 4c \end{matrix} \oplus \begin{matrix} a0 & 88 & 23 & 2a \\ fa & 54 & a3 & 6c \\ fe & 2c & 39 & 76 \\ 17 & b1 & 39 & 05 \end{matrix} = \begin{matrix} a4 & 68 & 6b & 02 \\ 9c & 9f & 5b & 6a \\ 7f & 35 & ea & 50 \\ f2 & 2b & 43 & 49 \end{matrix}$$

3.4.3 Erzeugung der Rundenschlüssel

Da es sich beim AES um ein symmetrisches Verfahren handelt, muss es einen geheimen Schlüssel für das Ver- und Entschlüsseln geben. Sei der zufällig generierte geheime 128 Bits Schlüssel = **2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c**.
Die Erzeugung der Rundenschlüssel aus dem geheimen Schlüssel wird im Englischen „Key Expansion“ genannt. Die Rundenschlüssel werden zu Beginn der Verschlüsselung in einem eigenen Algorithmus generiert und werden für die Funktion AddRoundKey benötigt.

Der geheime Schlüssel wird ebenfalls als Matrix aufgeschrieben und wird in 32 Bits lange sog. „Wörter“ (engl. words) eingeteilt. Ein Wort repräsentiert genau eine Spalte der Matrix, die vier Wörter W_0, W_1, W_2 und W_3 des obigen Schlüssels lauten demnach:

$$\begin{matrix} 2b & 28 & ab & 09 \\ 7e & ae & f7 & cf \\ 15 & d2 & 15 & 4f \\ 16 & a6 & 88 & 3c \end{matrix} \qquad W_0 = \begin{matrix} 2b \\ 7e \\ 15 \\ 16 \end{matrix} \qquad W_1 = \begin{matrix} 28 \\ ae \\ d2 \\ a6 \end{matrix} \qquad W_2 = \begin{matrix} ab \\ f7 \\ 15 \\ 88 \end{matrix} \qquad W_3 = \begin{matrix} 09 \\ cf \\ 4f \\ 3c \end{matrix}$$

Der Key Expansion Algorithmus verwendet die bereits bekannte Funktion **SubBytes** sowie die Funktion **RotWord**.
Analog zur Veränderung eines States, ersetzt die Funktion SubBytes alle Bytes eines 32 Bits Wortes mit den entsprechenden Werten aus der bekannten Tabelle.
Die Funktion RotWord führt eine Rotation innerhalb eines Wortes durch und schreibt dabei das Element der 1. Zeile in die letzte 4. Zeile und verschiebt die restlichen Elemente um eine Stelle nach oben.
Des Weiteren wird pro Runde eine Konstante **Rcon** benötigt. Jede der Konstanten ist ebenfalls ein 32 Bits langes Wort und kann folgender Tabelle entnommen werden:

Runde	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	01	02	04	08	10	20	40	80	1b	36
Rcon	00	00	00	00	00	00	00	00	00	00
	00	00	00	00	00	00	00	00	00	00
	00	00	00	00	00	00	00	00	00	00

Um nun die Rundenschlüssel zu generieren, wird zunächst der geheime Schlüssel in Form eines States in eine Tabelle (siehe unten) geschrieben, anschließend werden rechts daneben neu generierte Wörter eingetragen, von denen stets vier einen weiteren Rundenschlüssel bilden. Die Wörter W_0 bis W_3 stellen den geheimen Schlüssel dar. Die Wörter W_4 bis W_7 bilden den 1. Rundenschlüssel, die Wörter W_8 bis W_{11} den 2. Rundenschlüssel usw.

Index	W_0	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}	...	W_{43}
	2b	28	ab	09	a0	88	23	2a	f2	7a	59	73	...	b6
Wort	7e	ae	f7	cf	fa	54	a3	6c	c2	96	35	59	...	63
	15	d2	15	4f	fe	2c	39	76	95	b9	80	f6	...	0c
	16	a6	88	3c	17	b1	39	05	f2	43	7a	7f	...	a6

Die Wörter W_i mit Indizes, die ein Vielfaches von 4 sind, also 4, 8, 12, 16, ..., 40, werden berechnet, indem auf das Wort mit Index W_{i-1} die Funktionen RotWord und anschließend SubBytes angewandt werden. Das Ergebnis wird dann mit dem Wort an Position W_{i-4} und Rcon der entsprechenden Runde XOR-verknüpft.
Zum Beispiel:

$$W_4 = \textbf{SubBytes (RotWord } (W_3)) \bigoplus W_0 \bigoplus \textbf{Rcon}$$

$$= \textbf{SubBytes (RotWord } \begin{pmatrix} 09 \\ cf \\ 4f \\ 3c \end{pmatrix} \textbf{)} \bigoplus \begin{matrix} 2b \\ 7e \\ 15 \\ 16 \end{matrix} \bigoplus \begin{matrix} 01 \\ 00 \\ 00 \\ 00 \end{matrix} = \begin{matrix} a0 \\ fa \\ fe \\ 17 \end{matrix}$$

Die übrigen Wörter W_i werden berechnet, indem die Wörter an den Positionen W_{i-1} und W_{i-4} XOR-verknüpft werden.
Zum Beispiel:

$$W_5 = W_4 \bigoplus W_1 = \begin{matrix} a0 \\ fa \\ fe \\ 17 \end{matrix} \bigoplus \begin{matrix} 28 \\ ae \\ d2 \\ a6 \end{matrix} = \begin{matrix} 88 \\ 54 \\ 2c \\ b1 \end{matrix}$$

[**AES01**]

Entschlüsselung

Um einen verschlüsselten Block bzw. State wieder zu entschlüsseln, muss der Verschlüsselungsalgorithmus in umgekehrter Reihenfolge durchlaufen werden. Außerdem müssen jeweils die zu den Verschlüsselungsfunktionen inversen Funktionen verwendet werden. Die zu ShiftRows inverse Funktion **invShiftRows** rotiert die Zeilen eines States einfach in die entgegengesetzte Richtung. Die zu AddRoundKey inverse Funktion ist ebenfalls **AddRoundKey**, da ja die Umkehrfunktion einer XOR-Verknüpfung ebenfalls die XOR-Verknüpfung ist. Die zu SubBytes und MixColumns inversen Funktionen **invSubBytes** und **invMixColumns** verwenden für die Substitutionstabelle bzw. die Matrix andere Byte-Werte um die Umkehrfunktion zu realisieren [**AES01**].

3.4.4 Betriebsmodi

Da der Algorithmus des AES lediglich einen Block verschlüsselt, aber man in der Praxis in der Regel mehrere Blöcke verschlüsseln muss, gibt es für Blockchiffren verschiedene Betriebsmodi. Der einfachste **Betriebsmodus** ist der **Electronic Code Book Mode**, welcher derzeit auch in **EnDeCrypt** zum Einsatz kommt. Bei diesem Betriebsmodus werden alle Blöcke unabhängig voneinander auf die gleiche Art und Weise verschlüsselt. Das bedeutet u.a., dass gleiche Klartextblöcke auf gleiche Geheimtextblöcke abgebildet werden, was einen Sicherheitsnachteil darstellt. Raffiniertere Betriebsmodi, wie z.B. der **Cipher Block Chaining Mode** lösen dieses Problem, indem sie mehrere Blöcke vor dem Verschlüsseln miteinander verknüpfen [**Sch13**].

Padding

Eine weitere Aufgabe der Betriebsmodi ist, die Klartextblöcke vor dem Verschlüsseln auf eine passende Länge zu bringen. Da jeder Block eine festgelegte Anzahl an Bits (128 Bits beim AES) enthalten muss, aber die zu verschlüsselnden Daten bzw. Nachrichten in der Regel nicht exakt der Blocklänge entsprechen, wird der letzte Block mit Zufallsbits aufgefüllt, bis die geforderte Blocklänge erreicht ist. Dieser Vorgang wird **Padding** gennant [**Sch13**].

Demonstration

Um den Klartext aus Kapitel 3.1 mit dem AES zu verschlüsseln, wurde der Electronic Code Book Mode mit Padding verwendet. Mit dem hexadezimalen, zufällig generierten 128 Bits Schlüssel **e15ad3fe791ad0c31b5483bac96909c3** ergibt sich folgender Geheimtext:

1fd1198d1478c22abc9d50ed9c40bfb1eafe5abc80d07b58a6606f610d304790b4812596e2a5ea90
bd9967eb94b48af3f1bcff31645939324bc101570b8046b68542e120332eb5eb65cc7d935008e78c
a8f6d4ce1d8779b552fb67580ceeaed72cf93afd6c59f8dbf1ebea048b8b53e066f773f84b5d0e16be
30876862b081b93b654ae72139b3c828eae2264461a437cf3e149fd730b35dee85ed3da7214fb24
0dc7a0f462d8b5032756c59ca281c33be00339a9c22e2494197a4d84b61fe085a907ab4a880cdc4
396cf5e0d1430ef361a1435624956874549bfacccd8e46848b80a45c4e83a7067ff5c265bbeb6be90
14528a584efd3ada489c25dac957dd5af8efef3add4c7708650e68c2542abe1a505cf7d4214d10efc
9425523e01f19ae05e88eb643b1f742c63ef0e3120c2cba676ab51f74dd4e57608073cf341c064

Der Geheimtext ist - unter Ausnahme der Padding Bits - genauso lang wie der Klartext, da die Anzahl der Bits beim Verschlüsseln nicht verändert wird. Für das Beispiel wurde der Text in 21 Blöcke à 128 Bits eingeteilt.

4 Asymmetrische Verschlüsselung

Im Gegensatz zur symmetrischen Kryptographie verwenden zwei Kommunikationspartner bei der asymmetrischen Kryptographie bzw. den **Public-Key-Verfahren** nicht einen gemeinsamen identischen Schlüssel, sondern zwei verschiedene pro Nachricht. Anders als bei symmetrischen Verschlüsselungsverfahren, spielt hier die Unterscheidung zwischen Sender und Empfänger eine große Rolle. Um das Vorgehen bei den asymmetrischen Verschlüsselungsverfahren zu erklären, werden, wie in der Kryptographie üblich, die beiden Kommunikationspartner fortan als Alice und Bob bezeichnet.
Möchte beispielsweise Bob eine geheime Nachricht an Alice übermitteln, so braucht er deren persönlichen öffentlichen Schlüssel. Diesen öffentlichen Schlüssel hat Alice selbst generieren lassen und stellt ihn jedem zur Verfügung, der an sie schreiben möchte. Der Sender einer Nachricht verschlüsselt dann mit Alice' öffentlichem Schlüssel seine Nachricht und versendet sie anschließend. Damit Alice die Nachricht lesen kann, entschlüsselt sie diese mit ihrem privaten Schlüssel, den sie gleichzeitig mit dem öffentlichen generiert hat. Der private Schlüssel wird zum Entschlüsseln benötigt und muss somit geheim bleiben. Möchte Alice nun eine Antwort an Bob schicken, so braucht sie zum Verschlüsseln Bobs öffentlichen Schlüssel und Bob kann die Nachricht mit seinem privaten Schlüssel entschlüsseln. Bei einer Kommunikation in beide Richtungen sind also insgesamt zwei öffentliche und zwei private Schlüssel vonnöten. **Abbildung 4.1** verdeutlicht das Vorgehen graphisch. Die orange eingefärbten Elemente sind öffentlich, wohingegen die blau eingefärbten unbedingt geheim bleiben müssen. Der große Vorteil der asymmetrischen Verfahren gegenüber den symmetrischen besteht darin, dass keine geheimen Informationen ausgetauscht werden müssen, um die Kommunikation zu ermöglichen. Dadurch kann ein Angreifer, der den Kommunikationskanal abhört, die Nachrichten nicht entschlüsseln, weil ihm der private Schlüsselteil fehlt. Aufgrund dessen eignen sich Public-Key-Verfahren hervorragend für die Kommunikation über das Internet, denn ein unverschlüsselt übertragener symmetrischer Schlüssel könnte leicht abgehört werden. Die asymmetrische Kryptographie bietet damit eine Lösung für das **Schlüsselaustauschproblem** [**Sch13**].

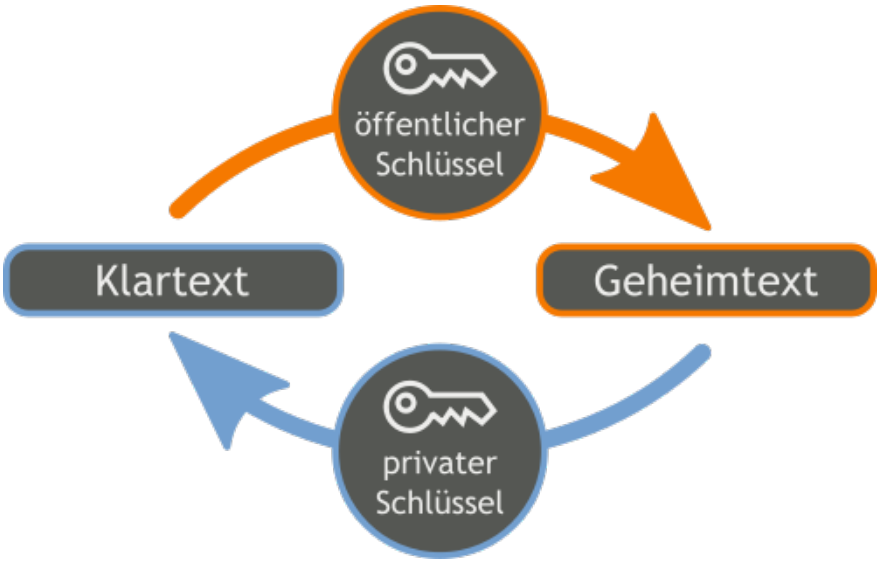

Abbildung 4.1: Schlüssel bei der Public-Key-Kryptographie, Bildquelle: **[Bil18b]**

4.1 RSA

Das RSA Verfahren, bennant nach dessen Erfindern Rivest, Shamir und Adleman, ist ein recht modernes Verschlüsselungsverfahren aus dem Jahr 1977. Es folgt den oben beschriebenen Prinzipien der assymetrischen Verschlüsselung.

Schlüsselerzeugung

Der erste Schritt bei der Anwendung des RSA Verfahrens ist die Schlüsselerzeugung um ein Schlüsselpaar, bestehend aus privatem und öffentlichem Schlüssel, zu erhalten. Der Algorithmus zur Schlüsselerzeugung lautet wie folgt **[Ert12]**:

- Erzeuge zwei zufällige, sehr große Primzahlen p und q (z.B. jeweils 1024 Bit).
- Berechne $n = p \cdot q$
- Wähle eine kleine ungerade natürliche Zahl e, für die gilt $\text{ggT}(e, \varphi(n)) = 1$, wobei $\varphi(n) = (p-1)(q-1)$.

- Berechne mit dem erweiterten euklidischen Algorithmus d als Lösung der Gleichung $e \cdot d \equiv 1 \mod \varphi(n)$
- Das Paar (e, n) stellt den öffentlichen Schlüssel dar.
- Das Paar (d, n) stellt den privaten Schlüssel dar.

Anwendung

Um einen Klartext m in den Geheimtext c zu verschlüsseln, verwendet man die Gleichung

$$c \equiv m^e \mod n$$

und zum Entschlüsseln die Gleichung

$$m \equiv c^d \mod n$$

wobei gilt $m, c \in \mathbb{N}$ und $m < n$ [**Ert12**].

Beispiel

An einem Beispiel mit kleinen Zahlen lässt sich der RSA Algorithmus gut nachvollziehen. Seien $p = 5$ und $q = 17$, dann folgt

$$n = p \cdot q = 5 \cdot 17 = 85$$

$$\varphi(n) = (p-1)(q-1) = 4 \cdot 16 = 64$$

Sei außerdem $e = 3$, so ist der öffentliche Schlüssel das Paar (3, 85). Für die Berechnung von d folgt

$$3 \cdot d \equiv 1 \mod 64 \quad \Rightarrow \quad d = 43$$

Der private Schlüssel lautet somit (43, 85).
Sei nun $m = 2$ und soll mit dem öffentlichen Schlüssel verschlüsselt werden, so rechnet man

$$c \equiv 2^3 \mod 85 \quad \Rightarrow \quad c = 8$$

Der Geheimtext c lautet also 8. Um ihn zu entschlüsseln, berechnet man mit dem privaten Schlüssel

$$m \equiv 8^{43} \mod 85 \quad \Rightarrow \quad m = 2$$

und erhält somit wieder den ursprünglichen Klartext $m = 2$ [**Sch13**].

Solch kleine Zahlen werden in der Praxis nicht verwendet. Um einen Eindruck zu vermitteln, wie groß praxistaugliche Zahlen sind, finden sich im **Anhang A** realistische Public und Private Keys mit einer Schlüssellänge von 2048 Bits, welche mit **EnDeCrypt** erzeugt wurden. Die Angabe der Schlüssellänge bezieht sich in der Regel auf die Länge von n.

Demonstration

Abbilung 4.2 zeigt einen Ausschnitt von der **EnDeCrypt** Benutzeroberfläche. Dort können für den RSA Algorithmus die notwendigen Public und Private Keys eingegeben werden. Mit dem Button „generate public and private keys“ können diese auch automatisch erzeugt werden. Der voreingestellte Wert für $e = 65537$ kann gegebenenfalls geändert werden, wird aber aufgrund seiner Recheneffizienz häufig in der Praxis verwendet [**Sch13**].

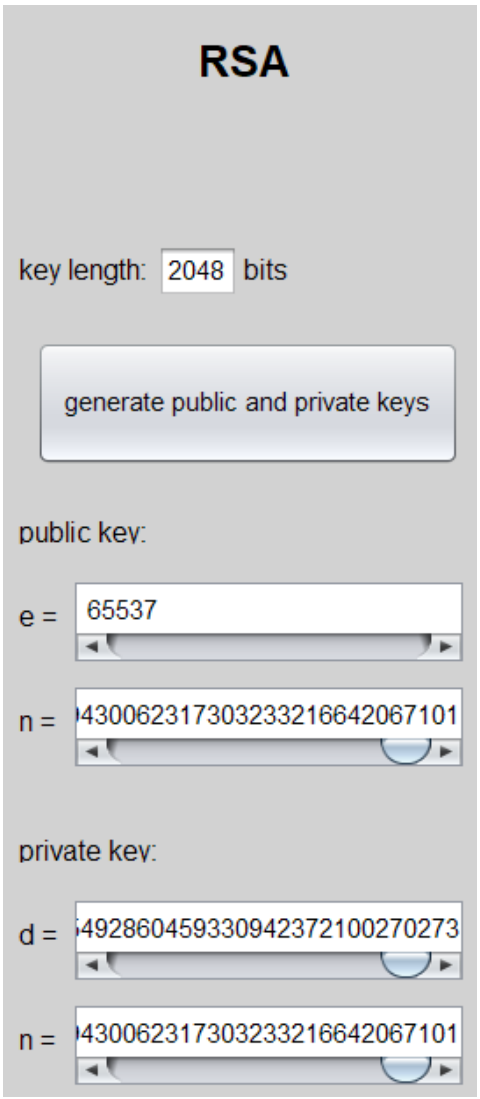

Abbildung 4.2: RSA Public Key und Private Key

Versucht man nun wie schon bei den vorherigen Verfahren den Klartext aus Kapitel 3.1 mit einer Schlüssellänge von 2048 Bits zu verschlüsseln, wird **EnDeCrypt** dem Benutzer eine Fehlermeldung ausgeben. Denn mit dem RSA Algorithmus können nur Nachrichten bis zu einer Länge von $Schlüssellänge/8$ verschlüsselt werden (für ein Zeichen werden 8 Bit verwendet). Sollen längere Texte verschlüsselt werden, so muss die Schlüssellänge erhöht werden, was aber den Nachteil mit sich bringt, dass das Verschlüsseln und besonders das Entschlüsseln dann erheblich längere Rechenzeiten benötigen. Der Beispieltext lässt sich mit einer Schlüssellänge von 4096 Bits verschlüsseln und lautet in dezimaler Zahlendarstellung:

108623909202336815225300715469367125127421117705257082587929724497367749092646743643861926037497045392240667832981204638751346039323739095896596166077500983376094330006546202620293299386030205349226748894001756837236763325157833778017692310992499767799087853639552369184261383335436050821338792081881685547913026585173370615399828406418911960211093381290484802802833422245520465015481839029066421873441162268698743555314537859648638257094014628009998473654080121707350386740367888532964148809859572462911806520076779870358662188337271062033975187597071475765810351845505849259924201632951975957241622453434792730895665168317675044186407671130497367317296839250422277771910882363108797451242347530859463135976039341109479968464701925349138009677788031159863692196324319623246174342814494557094445340881972215424837329488144827091689788175235447043097990720632026205261813949530436644650516420730326053216538172514268364823928713370674000527739242807507797593841125191431661823021343220717843998096629916253945237662108364119995977744245159545191521374778846042466144511810587734049316273180006767703086062957303945839180946815523052686099159296901266132797916396650942305797996866137111927400110095868284978189975434306244432356809660

Theoretisch könnten auch längere Nachrichten mit einer zu kleinen Schlüssellänge verschlüsselt werden, indem man die lange Nachricht in mehrere kürzere Abschnitte teilt, die dann nicht mehr Zeichen enthalten als $Schlüssellänge/8$. Für lange Nachrichten ist es in der Praxis aber besser die Nachricht mithilfe eines effizienteren symmetrischen Verfahrens, wie z.B. AES zu verschlüsseln und anschließend den AES Schlüssel mit dem RSA Algorithmus zu verschlüsseln. Damit würde das RSA Verfahren in diesem Fall nicht zur Verschlüsselung der Daten, sondern zur Verschlüsselung des symmetrischen Schlüssels eingesetzt und löst somit das Schlüsselaustauschproblem des AES. Gleichzeitig kann man die Vorteile des effizienteren AES für große Datenmengen nutzen [**Sch13**].

Das RSA Verfahren gilt als sehr sicher und findet aktuell in der Praxis viel Verwendung. Seine Sicherheit basiert auf der Tatsache, dass die Faktorisierung großer natürlicher Zahlen in ihre Primfaktoren nach aktuellem Forschungsstand nicht effizient lösbar ist. Wenn ein Angreifer die Primfaktorzerlegung von n kennt, also die Primzahlen p und q, so kann er auch leicht $\varphi(n)$ berechnen und damit d des geheimen privaten Schlüssels. Mit wachsender Schlüssellänge steigt die Rechenzeit für einen solchen **Faktorisierungsangriff** aber ins Unermessliche. Schon ab einer Schlüssellänge von 1024 Bits bietet RSA hohe Sicherheit, in **EnDeCrypt** sind standardmäßig 2048 Bits eingestellt. Eine Erhöhung der Schlüssellänge bringt mehr Sicherheit, aber auch längere Rechenzeiten für das Ver- und Entschlüsseln mit sich [**Sch13**].

5 Fazit und Ausblick

Die intensive Beschäftigung mit den implementierten Verschlüsselungsverfahren war eine spannende und lehrreiche Erfahrung. Die Programmierung der Algorithmen setzt voraus, sich eingehend mit diesen auseinandergesetzt und sie vollständig verstanden zu haben. Der Lerneffekt ist dabei sicherlich höher, als würde man sich nur theoretisch mit den Verfahren, z.B. durch Lesen eines Buches, befassen.
Dem Verschlüsselungstool **EnDeCrypt** können in der Zukunft noch weitere Verfahren hinzugefügt werden. Denkbar wäre auch, das Programm so zu erweitern, dass neben Texten auch Dateien (Textdateien sowie Binärdateien) verschlüsselt werden können. Interessant wäre außerdem, eine Chat-Funktion zu ergänzen, die verschlüsselte Nachrichten direkt über ein Netzwerk versenden kann.

Das Feld der Kryptologie bringt für die kommenden Jahre große Herausforderungen mit sich. Mit der Entwicklung der Quantencomputer werden neue kryptographische Verfahren benötigt, da die momentan eingesetzten aufgrund der enormen Rechenleistung der Quantencomputer nicht mehr als sicher gelten werden. Einerseits muss es auch in der Zukunft möglich sein, sicher und privat kommunizieren zu können und andererseits müssen aktuell noch sicher verschlüsselte Daten vor Angriffen in der Zukunft geschützt werden. Dieses noch recht neue Forschungsgebiet bezeichnet man als Quantenkryptographie. Schon seit einigen Jahren wird intensiv daran geforscht, neue sichere Verfahren zu entwickeln. Kryptologie-Begeisterte dürfen sich also auf eine spannende Zukunft freuen.

Literaturverzeichnis

[AES01] Advanced encryption standard (AES). Technical report, National Institute of Standards and Technology, nov 2001.

[Bil18a] Bildquelle. Advanced encryption standard ablauf, [Online; Stand 11. April 2018]. http://www.formaestudio.com/rijndaelinspector/archivos/Rijndael_Animation_v4_eng.swf.

[Bil18b] Bildquelle. Asymmetrisches kryptosystem, [Online; Stand 17. März 2018]. https://de.wikipedia.org/w/index.php?title=Asymmetrisches_Kryptosystem&oldid=173556943.

[Bil18c] Bildquelle. American standard code for information interchange, [Online; Stand 4. März 2018]. https://de.wikipedia.org/w/index.php?title=American_Standard_Code_for_Information_Interchange&oldid=174244939.

[Bil18d] Bildquelle. Symmetrisches kryptosystem, [Online; Stand 5. März 2018]. https://de.wikipedia.org/w/index.php?title=Symmetrisches_Kryptosystem&oldid=174122795.

[Ert12] Wolfgang Ertel. *Angewandte Kryptographie*. Hanser, 2012.

[Gem65] Wilhelm Gemoll. *Griechisch-Deutsches Schul- und Handwörterbuch*. G. Freytag Verlag/Hölder-Pichler-Tempsky, 1965.

[Sch13] Klaus Schmeh. *Kryptografie - Verfahren, Protokolle, Infrastrukturen*. dpunkt.verlag, 2013.

Abbildungsverzeichnis

A Anhang

In diesem Anhang finden sich beispielhaft ein Schlüsselpaar aus Public Key und Private Key für das RSA Verfahren. Diese wurden mit **EnDeCrypt** erzeugt.

Public Key

e =

65537

n (2048 Bits) =

165454828540530998395503060363705335579748204170516440392578346425089607851467209115627528025111695116282119469154535235482999545070441191327486037074381010916853543415928109911540642761882220310013654095935396190891862807437744188001629086635332566347150183887636505085672569874112810754025314110819070459808304510806230241267579639537281062599428753737879219535059583132552155680214061835867522918616839973245417803952020885799850249777679178707159627178570563578987631010828788665059850467794987940800412612973273681245164610561868168102964048456745426158863087272920288254867286361611282693992174217475383636409 7

Private Key

d =

14051681190377061567809489047077948751137838619294100364390372131275558056372409813108124049369011678666577122470310248664380162607241052576116780648721023648720745742555677353184680506505272212679104929021112992140142239040416467607608324050999584251692331367777280045861642975208392958749758908417241961444832938042328866063455616413052017804756570411877295423094482515492928896473018220099433164470187405908921222179705745508978010343920340487680294934408271018214592392249478338298409050562313825728280948705286964995379434725309254261746102865226666249076389970941358144598698941461837384945164006685076954272045

n (2048 Bits) =

16545482854053099839550306036370533557974820417051644039257834642508960785146672091156275280251116951162821194691545352354829995450704411913274860370743810109168535434159281099115406427618822203100136540959353961908918628074377441880016290866353325663471501838876365050856725698741128107540253141108190704598083045108062302412675796395372810625994287537378792195350595831325521556802140630586752291861683997324541780395202088579985024977767917870715962717857056357898763101082878866505985046779498794080041261297327368124516461056186816810296404845674542615886308727292028825486728636161128269399217421747538363640 97