Hochschule für Technik Wirtschaft und Kultur Leipzig (FH)
Fachbereich Informatik, Mathematik und Naturwissenschaften
Studiengang Wirtschaftsmathematik
Studienrichtung Operations Research
Prof. Dr. rer. nat. Wolfram Koepf

Diplomarbeit

Das RSA – Verfahren
und die Implementierung in *Mathematica*

Betreuer: Prof. Dr. rer. nat. Wolfram Koepf

Bearbeiter: Inka Tittel
An der Lautsche 13
04205 Leipzig
Matr.-Nr.: 19395
9. Semester

Eingereicht am: 17.01.2000

Kleines Dankeschön

Ich möchte all denen danken, die mich bei der Erstellung dieser Diplomarbeit unterstützt haben, sei es mit Rat und Tat oder durch moralischen Beistand.
Im einzeln hervorheben möchte ich:
Herrn Prof. Dr. rer. nat. Wolfram Koepf, meinen Betreuer, der mir als Lektor zur Seite stand.
Meine Freunde, die nicht müde geworden sind meine Rechtschreibfehler zu korrigieren sowie meinen Eltern, die mir das Studium erst ermöglichten.

Inhaltsverzeichnis II

Abbildungsverzeichnis V

Tabellenverzeichnis VI

Abkürzungsverzeichnis VII

ASCII-Code	engl. American Standard Code of Information Interchange
C	Geheimer Text, Chiffre
D	Entschlüsselungsfunktion, englisch decryption
E	Verschlüsselungsfunktion, englisch encryption
E-Mail	elektronische Post[1]
ggT	größter gemeinsamer Teiler
kgV	kleinstes gemeinsames Vielfaches
LED	engl. light emitting diode, Licht emittierende Diode, Leuchtdiode (Lumineszenzdiode)
M	Originaltext, englisch Message
MIPS	eine Million Anweisungen pro Sekunde
MPQS	Mutiple Polynomial Quadratic Sieve
NFS	Zahlenkörpersieb, englisch Number Field Sieve
Pay – TV	Fernsehprogramm, das meist monatlich zu bezahlen ist, um die codiert ausgestrahlten Sendungen mit Hilfe eines Decoders empfangen zu können.
PGP	Pretty Good Privacy
PIN	Personal Identification Number – zusätzlich zur Ausweiskarte benötigte Geheimnummer, die beim Geldabheben am Bankautomaten eingegeben werden muss. Kann die Nummer nicht innerhalb einer bestimmten Zeit eingetastet werden, wird die Ausweiskarte vom Automaten einbehalten.
PKCS	Public Key Cryptography Standard
QS	Quadratisches Sieb
RSA	Nach Rivest, Adi Shamir und Leonard Adleman benanntes asymmetrisches kryptographisches Verfahren.
SET	Secure Electronic Transactions
SSL	Secure Sockets Layer
TBA	to be announced
TWINKLE	The Weizman Institute Key Locating Engine

[1] Meyersches Lexikon

1. Einleitung

Ziel dieser Arbeit ist es, einen Überblick über das RSA – Verfahren zu geben, den mathematischen Hintergrund aufzuarbeiten und Aspekte der Anwendung zu erläutern. Das RSA - Verfahren ist ein Verschlüsselungsverfahren. Das Kapitel gibt eine kurze Einführung in Verschlüsselungsverfahren und damit in das Gebiet der Kryptographie.

1.1. Kryptographie

Die Kryptographie ist die Lehre vom Ver- und Entschlüsseln von Nachrichten. Das Wort kommt aus dem Griechischen: „krypto-“ und bedeutet: „geheim, verborgen“ und „-graphie“ „beschreiben“. Der Oberbegriff zu Kryptographie ist Kryptologie („logie“ „Wissenschaft“, „Kunde“). Zu ihr gehört des Weiteren noch die Kryptoanalyse. Das ist die Wissenschaft vom unautorisierten Entschlüsseln von Nachrichten.[16]
Eine Nachricht besteht aus einem Originaltext, der mit M bezeichnet wird.[2] Die Verschlüsselung ist das Verfahren, eine Nachricht unverständlich zu machen, um ihren Inhalt zu verbergen. Eine verschlüsselte Nachricht besteht aus dem Geheimtext, der mit C bezeichnet wird.[3] Die Rückumwandlung des Geheimtextes in den Originaltext heißt Entschlüsselung. Dies sind die Hauptbestandteile eines kryptographischen Verfahrens. Weiterhin kommt noch die Bezeichnung „Angriff“ hinzu, darunter versteht man den Versuch des unautorisierten Entschlüsselns.
Das Grundprinzip der Kryptographie sieht wie folgt aus: Es gibt einen Sender, der die Nachricht M an den Empfänger schicken möchte, ohne dass die Nachricht von einer unautorisierten Person gelesen werden kann. Um den Sacherverhalt etwas zu veranschaulichen, nenne ich im Folgenden den Sender Bob und den Empfänger Alice sowie den Kryptoanalytiker Sam.[26]

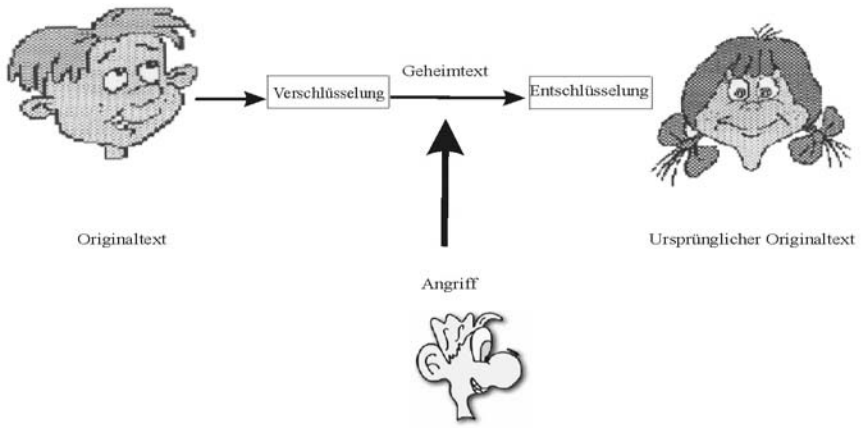


Abbildung 1: Prinzip der Kryptographie
Quelle: Bruce Schneider: Angewandte Kryptographie,

Bob wendet die Verschlüsselungsfunktion E auf die Nachricht M an[4] und erhält damit C. In mathematischer Schreibweise liest sich das wie folgt:

$$E(M) = C.$$

Er sendet die verschlüsselte Nachricht C an Alice. Alice verwendet ihrerseits eine Entschlüsselungsfunktion D auf C an[5] und erhält die Originalnachricht M.

[2] M für englisch Message
[3] C für Chiffretext
[4] E für englisch. encryption

$$D(C) = M$$

Der Sinn und Zweck des Ver- und Entschlüsseln ist, die ursprüngliche Originalnachricht wieder herzustellen. Somit muss folgendes gelten:

$$D(E(M)) = M.$$

Der Kryptoanalytiker Sam spielt bei diesen Verfahren selbst eine untergeordnete Rolle. Er versucht, die Nachricht C abzufangen und sie, ohne D zu kennen, zu entschlüsseln. Die Möglichkeiten des unautorisierten Entschlüsselns sollten daher so gering wie möglich sein, damit der Kryptoanalytiker keine Chance hat, die Nachricht zu entschlüsseln.[4]
Die Ver- und Entschlüsselungsfunktion sind im Allgemeinen mathematische Funktionen, die miteinander „verwandt“ sind (das bedeutet, die eine Funktion ist die Umkehrfunktion der anderen). Sie bilden einen Algorithmus. Heute werden meist Verschlüsselungsverfahren verwendet, bei denen die Algorithmen bekannt sind und die Sicherheit des Verfahrens nicht mehr vom Algorithmus selbst abhängt, sondern von einer Zusatzinformation, dem Schlüssel. Dieser kann aus einem sehr umfangreichen Wertebereich gewählt werden und wird allgemein mit K bezeichnet. [6]
Der Algorithmus kann veröffentlicht werden und viele Anwender können ihn nutzen.

Die Funktionen sehen jetzt folgendermaßen aus:

$$E_K(M) = C$$
$$D_K(C) = M$$

und weiterhin gilt auch hier die Eigenschaft:

$$D_K(E_K(M)) = M.$$

Die beiden Operationen stützen sich auf den gleichen Schlüssel, was durch den tiefgestellten Index angegeben wird. Ist dies nicht der Fall, also wird nicht mit dem gleichen Schlüssel entschlüsselt, mit dem verschlüsselt wurde, bezeichnet man den Verschlüsselungsschlüssel mit K_1 und den Entschlüsselungsschlüssel mit K_2. In diesen Fall gilt:

$$E_{K_1}(M) = C$$
$$D_{K_2}(C) = M$$
$$D_{K_2}(E_{K_1}(M)) = M$$

Die Zusammenfassung vom Algorithmus, von allen möglichen Originaltexten, Geheimtexten und Schlüsseln, nennt man Kryptosystem.[4]

---

[5] D für englisch decryption
[6] K für englisch key

1.2. Die geschichtliche Entwicklung der Kryptologie

Die Kryptologie rückt erst Ende dieses Jahrhunderts durch die Entwicklung der elektronischen Kommunikation mehr und mehr in das öffentliche Interesse, andererseits hat sie schon eine jahrtausendealte Geschichte. Sie wurde auf dem militärischen Sektor sowie auf dem Gebiet der nationalen Sicherheit sehr ernsthaft betrieben.
Bereits die alten Ägypter verschlüsselten einige ihrer Hieroglyphen an ihren Denkmälern.
Die alten Hebräer verschlüsselten bestimmte Wörter in den Heiligenschriften.
Vor 2000 Jahren benutzte Cäsar ein einfaches Verschlüsselungsverfahren, das jetzt als Cäsar – Verschlüsselung[7] bekannt ist.
Im 12. Jahrhundert beschrieb Roger Bacon mehrere Methoden der Kryptographie und Geoffrey Chaucer schloss ebenso mehrere Verfahren in seine Arbeit ein. 1460 entwickelte Leon Alberti ein Verschlüsselungsrad und stellte die Hauptsätze der Frequenzanalyse auf.
Balise de Vigenère publizierte 1585 ein Buch über Kryptologie und beschrieb die polyalphabetische Substitutionsverschlüsselung. Diese wurde in der Diplomatie sowie im Krieg über die Jahrhunderte hinweg angewandt. Bei der polyalphabetischen Substitutionsverschlüsselung werden mehrere einfache Substitutionsverfahren (siehe Cäsar – Verfahren) benutzt. Welches Verfahren auf die einzelnen Zeichen angewendet wird, hängt von der Position des Zeichens im Klartext ab.
Mit dem Jeffersonzylinder hielt die maschinelle Verschlüsselung Einzug. Der 1790 entwickelte Zylinder umfasst 36 Platten, jede mit einem zufälligen Alphabet. Die Anordnung der Platten war der Schlüssel, die Nachricht wurde eingestellt und eine andere Zeile wurde die verschlüsselte Nachricht.
Die Wheatstoneplatte: Das Original wurde 1817 erfunden von Wadsworth. Aber der Öffentlichkeit vorgestellt wurde sie durch Wheatstone erst 1860. Dabei wurden zwei konzentrische Räder benutzt, um eine polyalphabetische Verschlüsselung zu erzeugen.
1843 bekam Edgar Allan Poe für seine Geschichte „Der Goldkäfer“ einen Literaturpreis. In der Geschichte beschreibt er ein kryptoanalytisches Verfahren.
1863 wurde erstmals eine wissenschaftliche Abhandlung zu einem kryptoanalytischen Verfahren veröffentlicht, welches eine Kryptoanalyse erlaubt, die in zwei Schritten an einer periodischen polyalphabetischen Substitutionsverschlüsselung durchführbar ist. Das Verfahren wurde von F.W. Kasiski veröffentlicht. 1918 wurde von William Friedman eine Analysenmethode vorgestellt, die zeigt, dass die Analyse von polyalphabetischen Substitutionsverschlüsselungen auch dann Erfolg haben kann, wenn die Verschlüsselung nicht periodisch ist.[16] Friedman nutzt dabei die Buchstabenhäufigkeit aus, denn in fast allen Sprachen macht schon eine kleine Menge von Buchstaben den größten Teil des Textes aus.[8]

Eine sehr wichtige Verschlüsselungsmaschine wurde von den Deutschen im Zweiten Weltkrieg benutzt: die Enigma[9]. Sie besaß erst drei (später vier) Rotoren, die zwischen zwei feststehenden Scheiben montiert wurden. Eine dieser Scheiben diente zur Ein- und Ausgabe der Zeichen, die andere besaß eine interne Verdrahtung, mit deren Hilfe das ankommende Signal „gespiegelt“ und auf einen anderen Pfad durch die beweglichen Rotoren zurückgeschickt wurde. Dabei ist die Substituierung eineindeutig. Wenn jedes Zeichen x durch y ersetzt wird, dann wird auch jedem y ein x zugeordnet. Weiterhin wird kein Zeichen in sich selbst substituiert, weil der Rotorsatz stets zweifach durchlaufen wird.[16]
Weiterhin wurde im Zweiten Weltkrieg der Geheimschreiber T-52, der von Siemens & Halske entwickelt wurde, eingesetzt.

---

[7] siehe auch Kapitel 1.4. Symmetrische Verschlüsselung
[8] siehe Kapitel 1.4. Symmetrische Verschlüsselung
[9] lat: aenigma, das Rätsel

Mit dem Zweiten Weltkrieg gelangte man zu den Anfängen der Computerentwicklung und damit zur heutigen Kryptographie. Die Engländer entwickelten elektromechanische und elektronische Maschinen, um die deutschen Geheimnachrichten zu entschlüsseln. Die bekannteste Anlage, die Röhrenrechenanlage COLOSSUS, kann als der erste digitale Computer angesehen werden.[2] Mit der COLOSSUS gelang es, eine von Enigma verschlüsselte Nachricht in ca. einer Stunde zu entschlüsseln.[16]

Ein großer Fortschritt in der Kryptologie wurde mit der Einführung von DES Data Encryption Standard erreicht, da zum ersten Mal dieses Verschlüsselungsverfahren in allen Einzelheiten publiziert wurde.[16]
Der DES wurde von dem US-amerikanischen National Bureau of Standards (NBS, das heutige National Institute of Standards and Technology, kurz: NIST) in Form eines Federal Information Processing Standard (FIPS) genormt und in der FIPS Publication 46 von 1977 veröffentlicht.
1978 wurde das asymmetrische kryptographische Konzept von Diffie und Hellmann vorgestellt, worunter auch das RSA - Verfahren[10] zählt.
Der nächste Abschnitt umfasst wichtige Begriffe der Kryptographie und ihre Definitionen.

1.3. Symmetrische Verschlüsselung

Es gibt zwei Konzepte für kryptographische Algorithmen. Zum einen die symmetrischen und zum anderen die asymmetrischen Verschlüsselungsalgorithmen. Die symmetrische Methode ist das klassische und das ältere der beiden kryptographischen Konzepte.
Bei diesem Algorithmus benutzt der Sender den gleichen Schlüssel zum Verschlüsseln wie der Empfänger zum Entschlüsseln oder man kann den einen Schlüssel aus dem anderen berechnen. Das setzt voraus, dass Bob und Alice einen Schlüssel über einen sicheren Kanal vereinbart haben. Dann benutzt Bob diesen Schlüssel um die Nachricht M zu verschlüsseln und sendet sie zu Alice. Sie empfängt die verschlüsselte Nachricht C und entschlüsselt sie mit dem gleichen Schlüssel.

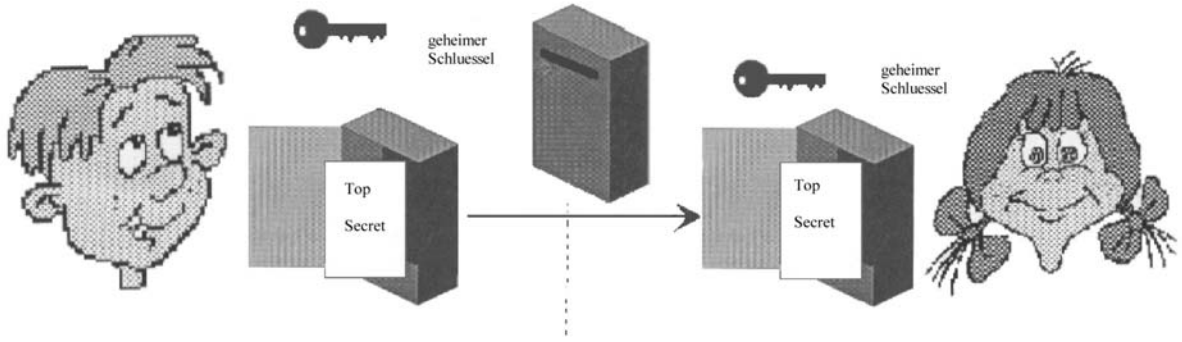


Abbildung 2: Symmetrische Verschlüsselungsmethode
Quelle: Albrecht Beutelspacher, Jörg Schwenk, Klaus Dieter Wolfenstetter: Moderne Verfahren der Kryptographie, Vieweg 1995 [Abgeänderte Version.]

Schon an diesem kleinen Beispiel wird deutlich, wo die Schwächen der symmetrischen Methode liegen. Das größte Problem ist das Schlüsselmanagement. Zu ihm gehören die Erzeugung, die Übertragung sowie die Speicherung des Schlüssels. Das Schlüsselmanagement ist ein Teil jedes Kryptosystems. Bei den symmetrischen Verfahren ist dies besonders schwierig, da der Schlüssel übertragen wird, aber gleichzeitig auch geheim

[10] Siehe Kapitel 2.

bleiben muss, speziell bei einer Umgebung mit vielen Nutzern. Der Sender und der Empfänger müssen sich vorher einigen, welchen Schlüssel sie benutzen. Wenn sie an verschiedenen Orten sind, muss der Schlüsselaustausch über einen sicheren Kanal vonstatten gehen. Somit müssen sich beide einem Kommunikationsmedium anvertrauen. Wenn dieses jedoch abgehört wird, kann jeder, der den Schlüssel empfängt, alle verschlüsselten Nachrichten lesen.
Ein symmetrisches Verschlüsselungsverfahren ist das Cäsar - Verfahren. Es ist ein sehr einfaches Verfahren und beruht darauf, dass jeder Buchstabe des Alphabets um drei Buchstaben nach vorn verschoben wird. Danach fängt man wieder mit dem Alphabet von vorn an. Die Buchstaben werden somit folgendermaßen verschlüsselt. Den Tabellenkopf bildet das normale Alphabet und in der zweiten Zeile steht das dazugehörige verschlüsselte Alphabet.

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
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

Tabelle 1: Cäsar – Verschlüsselungstabelle

Wird das Verfahren auf den Text „VOELKERSCHLACHTDENKMAL“ angewandt, entsteht der verschlüsselte Text „YRHONHUVFKODFKWGHQNPDO“
Betrachtet man das Verfahren unter dem Standpunkt eines Kryptoanalytikers, fällt als erstes auf, dass die Anzahl der möglichen Schlüssel sehr klein ist. Der Kryptoanalytiker kann einfach durch Testen den richtigen Schlüssel finden, denn nur der richtige Schlüssel produziert eine sinnvolle Nachricht.
Ein weiteres symmetrisches Verfahren funktioniert folgendermaßen: Der Schlüssel besteht aus zwei Komponenten, einem Wort und einem Buchstaben. Als erstes wird aus dem Wort eine Buchstabenfolge gebildet, in der jeder Buchstabe nur einmal vorkommt. Dies wird erreicht, indem jeder Buchstabe, der mehrmals vorkommt, bei seinem zweiten Auftreten gestrichen wird.
Beispiel: das Wort heißt Orthographie und daraus ergibt sich: ORTHGAPIE. Danach wird diese Folge unter das Originaltextalphabet geschrieben und es wird bei dem Schlüsselbuchstaben angefangen. Ist der Schlüsselbuchstabe zum Beispiel „I“, erhalten wir folgendes:

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
								O	R	T	H	G	A	P	I	E									

Tabelle 2: Verschlüsselungsschema

Der letzte Schritt besteht darin, die übrigen Buchstaben in alphabetischer Reihenfolge nach dem letzten Buchstaben des Schlüsselwortes aufzuschreiben.

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
Q	**S**	**U**	**V**	**W**	**X**	**Y**	**Z**	O	R	T	H	G	A	P	I	E	**B**	**C**	**D**	**F**	**J**	**K**	**L**	**M**	**N**

Tabelle 3: Verschlüsselungsschema

Das Verfahren wird wieder auf das Beispiel: „VOELKERSCHLACHTDENKMAL“ angewendet, es entsteht der verschlüsselte Text: „JPWHTWBCUZHQUDVWATGQH“.
Um diese Verfahren zu entschlüsseln, könnte man folgende Methode verwenden: es werden die Häufigkeiten der auftretenden Buchstaben ermittelt und diese mit den Häufigkeiten der Buchstaben zum Beispiel in der deutschen Sprache verglichen. Bei dieser Methode ist es von Vorteil, wenn der Text länger ist. Je länger der Text, umso genauer nähern sich die Häufigkeiten einander an.

Buchstaben	Häufigkeit (in %)	Buchstaben	Häufigkeit (in %)
a	6,51	n	9,78
b	1,89	o	2,51
c	3,06	p	0,79
d	5,08	q	0,02
e	17,40	r	7,00
f	1,66	s	7,27
g	3,01	t	6,15
h	4,76	u	4,35
i	7,55	v	0,67
j	0,27	w	1,89
k	1,21	x	0,03
l	3,44	y	0,04
m	2,53	z	1,13

Tabelle 4: Häufigkeitsverteilung der Buchstaben im Deutschen
Quelle: Walter Fumy, Hans Peter Rieß: Kryptographie, Oldenbourg Wien München 1994

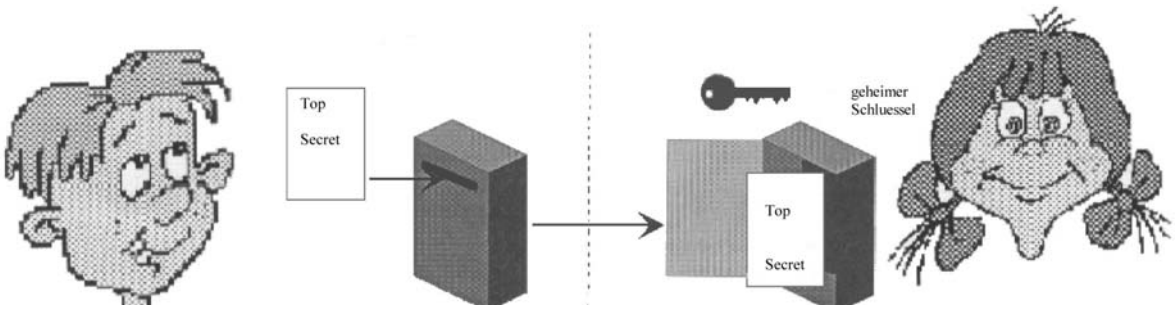
Daraus lassen sich wichtige Schlüsse für den Systemdesigner ziehen. Einmal sollte der Schlüssel nicht zu klein sein, so dass man ihn durch Testen finden kann. Der Algorithmus sollte nicht nur einzelne Buchstaben verschlüsseln, sondern ganze Buchstabenblöcke, so dass der Originaltext vielfältiger wird.

1.4. Asymmetrische Verschlüsselung

Bei den asymmetrischen Verfahren unterscheidet sich der Schlüssel zum Entschlüsseln von dem zum Verschlüsseln. Demnach kann der eine auch nicht aus dem anderen berechnet werden (zumindest nicht in angemessener Zeit). Das Verfahren wird auch „öffentliche Verschlüsselung“ genannt, da der Verschlüsselungsschlüssel öffentlich bekannt gemacht wird. Er wird dann auch öffentlicher Schlüssel genannt, den Entschlüsselungsschlüssel nennt man privaten Schlüssel.

1.4.1. Schlüsselmanagement[17]

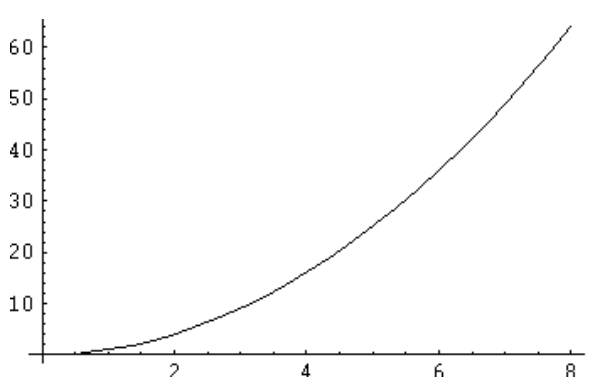
Um das Problem des Schlüsselmanagements der symmetrischen Verschlüsselung zu lösen, entwickelten Whitfield Diffie und Martin Hellman 1976 ein neues Konzept für ein kryptographisches Verfahren. Das Verfahren erlaubt zwei Anwendern, einen geheimen Schlüssel über einen nicht sicheren Kanal auszutauschen, ohne vorher etwas vereinbart zu haben.[26]
Das Konzept hat zwei Systemparameter p und g. Der Parameter p ist eine Primzahl und g ist eine ganze Zahl, kleiner als p. Jedes solche g besitzt folgende Eigenschaft: Für jede Zahl $n \in \mathbb{Z}$ zwischen eins und p-1 gibt es eine Zahl k, so dass $g^k = n \bmod p$. Beide Parameter sind öffentlich und können von jedem benutzt werden. k ist der Schlüssel den Alice und Bob austauschen wollen.
Das Verfahren funktioniert folgendermaßen: Als erstes erzeugt Alice eine zufällige Zahl a und Bob erzeugt eine zufällige Zahl b. a und b sind ganze Zahlen zwischen 1 und p-2 und sind geheim zu halten. Beide benutzen die Parameter p, g und ihre jeweilige geheime Zahl a bzw. b, um folgendes zu berechnen. Alices $\alpha = g^a \bmod p$ und Bobs ist $\beta = g^b \bmod p$. α und β werden ausgetauscht. Jetzt kann Alice den Schlüssel k berechnen
$k = \beta^a \bmod p = (g^b)^a \bmod p$ und Bob berechnet $k = \alpha^b \bmod p = (g^a)^b \bmod p$. Weil $g^{ab} = g^{ba} = k$, haben Alice und Bob jetzt einen gemeinsamen geheimen Schlüssel k.
Das Protokoll basiert auf dem diskreten Logarithmusproblem: Es wird angenommen, dass die Berechnung von a und b aus $\alpha = g^a \bmod p$ und $\beta = g^b \bmod p$ unmöglich ist, wenn die Primzahl p nur genügend groß ist.
Aus diesem Verfahren wurde das asymmetrische Konzept abgeleitet.
Bei diesem Konzept hat jeder Beteiligte zwei Schlüssel, einen öffentlichen und einen privaten. Der Sender und der Empfänger geben beide ihre öffentlichen Schlüssel bekannt. Die privaten Schlüssel bleiben geheim.
Möchte nun Bob eine Nachricht an Alice schicken, sucht er sich ihren öffentlichen Schlüssel und verschlüsselt die geheime Nachricht damit. Danach sendet er die verschlüsselte Nachricht an Alice. Alice entschlüsselt die Nachricht mit ihrem privaten Schlüssel. Sam, der Kryptoanalytiker, hat theoretisch keine Chance die Nachricht zu entschlüsseln, denn Bob und Alice haben keine geheimen Schlüssel ausgetauscht. Voraussetzung für dieses Konzept ist, dass der private Schlüssel sich nicht aus dem öffentlichen Schlüssel berechnen lässt.

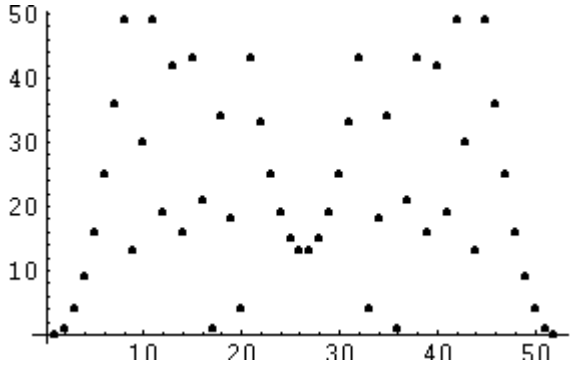


1.4.2. Einwegfunktion und Falltüreinwegfunktion

Die Voraussetzung des asymmetrischen Kryptographiekonzeptes, dass man von dem öffentlichen Schlüssel nicht auf den privaten Schlüssel schließen kann, wird mit Hilfe der Einwegfunktion erfüllt.
Einwegfunktionen sind Funktionen, die leicht berechenbar sind, aber ihre Umkehrung ist äußerst schwierig. Eine Einwegfunktion f bildet die Menge X auf die Menge Y ab. Die Funktionswerte y sind sehr einfach zu berechnen, aber für die Umkehrung $f^{-1}(y)$ ist es praktisch unmöglich die Urbilder x zu finden.
Ein Beispiel für eine Einwegfunktion ist das Telefonbuch. Die Funktion f ist aus dem Telefonbuch von einer bestimmten Person die Telefonnummer zu finden. Dies ist sehr einfach, denn das Telefonbuch ist alphabetisch geordnet. Die Invertierung, zu einer bestimmten Telefonnummer einen Namen zu finden, ist weitaus schwieriger. Ein weiteres Beispiel für Einwegfunktionen ist die Multiplikation zweier großer Primzahlen. Die Zahlen sind einfach zu multiplizieren, aber es ist sehr schwer, das Produkt wieder in die beiden Primzahlen zu zerlegen.[1]

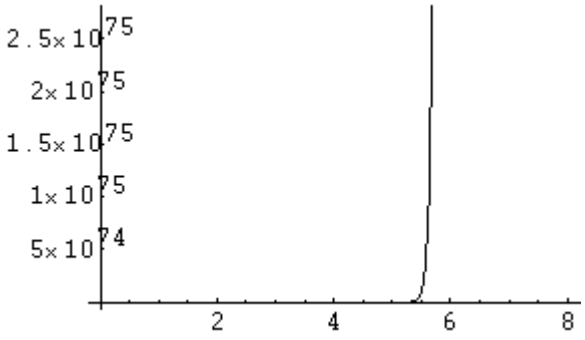
Bemerkung: Es ist noch nicht bewiesen, dass es Einwegfunktionen überhaupt gibt. Es ist sehr schwer zu zeigen, egal welcher Algorithmus verwendet wird, dass eine bestimmte Berechnung unmöglich ist. Bei den Beispielen, welche ich hier verwenden werde, geht man im Allgemeinen davon aus, dass es Einwegfunktionen sind.
Den Begriff Einwegfunktion möchte ich an vier Funktionen grafisch noch veranschaulichen. Die Funktion $f(x) = x^2$ lässt sich leicht berechnen sowie umkehren. Das bedeutet, es ist leicht, für jedes x das dazugehörige Element y zu berechnen. Das zeigt die erste Grafik. Und es ist einfach, für jedes Element y das dazugehörige Element x zu berechnen. Das kann man auch in der Grafik erkennen. Anders ist das jedoch bei der Funktion
$y = f(x) = x^2 \bmod 51$. Es ist immer noch leicht, jedes y aus dem dazugehörigen x zu berechnen. Aber es ist schwierig, x aus dem dazugehörigen y zu berechnen, da die Umkehrfunktion von $x^2 \bmod 51$ nicht eindeutig ist.





Noch deutlicher wird dies bei der Abbildung 5. Wiederum ist es einfach, die Umkehrfunktion von $y = f(x) = x^{100}$ zu berechnen. ($f^{-1}(x) = x = \sqrt[100]{y}$). Aber schon in der Grafik lässt sich erkennen, dass es viel schwieriger ist, die Umkehrfunktion von
$f(x) = x^{100} \bmod 97$ zu berechnen. Anhand dieser Grafik lässt sich aber noch etwas anderes deutlich erkennen. Zum Beispiel, dass die Funktion $x^{100} \bmod 97$ periodisch ist und die Periode 96 besitzt. Aber auch diese Tatsache hilft uns nicht viel weiter, die Umkehrfunktion zu finden. Die Eigenschaft der Einwegfunktion hat zur Folge, dass diese Funktion selbst nicht sehr brauchbar für die Verschlüsselung ist. Denn was nutzt es, wenn man eine Nachricht

verschlüsseln, sie aber nicht entschlüsseln kann. Für die Kryptographie mit einem öffentlichen Schlüssel benötigt man ein andere Art von Funktionen mit denselben Eigenschaften.



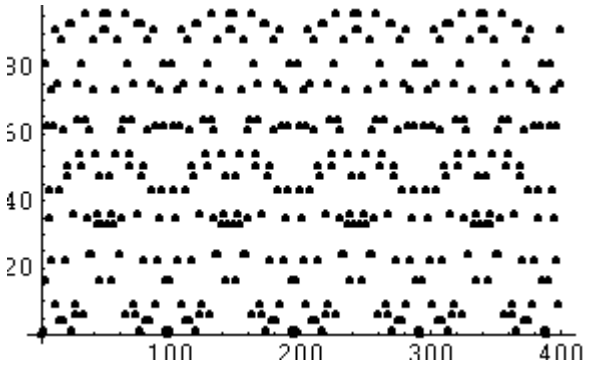


Abbildung 5: Erste Grafik: Funktion x^{100} **Zweite Grafik: Funktion x^{100} mod 97**

Ein Spezialfall der Einwegfunktion ist die Falltürfunktion oder auch Einwegfunktion mit Hintertür (engl.: Trapdoor one-way function). Sie ist wie die Einwegfunktion schwer zu invertieren. Aber unter zu Hilfenahme von zusätzlichen Informationen, der Hintertür, lässt sie sich leicht invertieren. Das heißt, dass f(x) aus x leicht berechenbar ist, x aus f(x) dagegen nur schwer. Es gibt jedoch eine geheime Information y, mit der sich x einfach aus f(x) berechnen lässt. Beispielsweise ist das Quadrieren von $x = a^2 \bmod n$, mit $n = pq$, eine Falltürfunktion. Kennt man die Faktoren q und p von n nicht, ist es praktisch unmöglich, diese Funktion umzukehren. Ist die Faktorisierung von n aber bekannt, kann man a berechnen. Somit ist die Falltürinformation hier die Faktorisierung von n, wie man am unteren Beispiel sieht.
Diese Eigenschaft der Falltürfunktion macht sie zum idealen Kandidaten für das asymmetrische kryptographische Konzept. Allgemein gesagt soll es für jedermann leicht sein, einen Text mit dem Computer zu verschlüsseln, genauso leicht, wie eine geschlossene Falltür zu passieren. Die Entschlüsselung soll aber nur für denjenigen einfach sein, der die Falltür kennt, also für den rechtmäßigen Empfänger.

Beispiel: Die Quadrierung von $x = a^2 \bmod n$. Bekannt ist $n = pq$, p, q, x und gesucht ist $a = \sqrt{x} \bmod n$ (Lösung ohne mathematischen Beweis).
Das Problem ist, dass jedes x entweder keine Wurzel oder vier verschiedene Wurzeln mod n hat und somit Redundanz auftritt. Die Lösung für die Quadratwurzel baut auf folgendem Satz auf:
Ist p eine Primzahl, so gilt für jedes $a \in N$ mit $p \nmid a$ nach dem Kleinen Satz von Fermat[11]:
$a^{(p-1)} \equiv 1 \bmod p \longrightarrow a^{(p-1)/2} \equiv \pm 1 \bmod n$.[12]

[11] Siehe Kapitel 2.2.1.
[12] Beweis siehe [13]

In diesem Fall ist x der quadratische Rest mod n = pq, dabei gilt:

$$a^2 = x \bmod pq$$
$$a^2 = x \bmod p$$
$$a^2 = x \bmod q$$ [13]

Bemerkung: Mit Hilfe des Chinesischen Restsatzes[14] erhält man vier Quadratwurzeln von x mod n mit (p+1)/4 und (q+1)/4 $\in \mathbb{Z}$.

$A_1 = x^{(p+1)/4} \bmod p$
$A_2 = (p-x^{(p+1)/4}) \bmod p$
$A_3 = x^{(q+1)/4} \bmod q$
$A_4 = (q-x^{(q+1)/4}) \bmod q$

mit $c = q(q^{-1} \bmod p)$ und $d = p\,(p^{-1} \bmod q)$ sind die vier möglichen Lösungen:

$a_1 = (cA_1 + dA_3) \bmod n$
$a_2 = (cA_1 + dA_4) \bmod n$
$a_3 = (cA_2 + dA_3) \bmod n$
$a_4 = (cA_2 + dA_4) \bmod n$

Beispiel:

p = 11
q = 3
n = 209
x = 16

$A_1 = 16^{(11+1)/4} \bmod 11 = 4$
$A_2 = (11-16^{(11+1)/4}) \bmod 11 = 7$
$A_3 = 16^{(3+1)/4} \bmod 3 = 1$
$A_4 = (3-16^{(3+1)/4}) \bmod 3 = 2$

$c = 3(3^{-1} \bmod 11) = 12$
$d = 11\,(11^{-1} \bmod 3) = 22$

$a_1 = (12 * 4 + 22 * 1) \bmod n = 4$
$a_2 = (12 * 4 + 22 * 2) \bmod n = 26$
$a_3 = (12 * 7 + 22 * 1) \bmod n = 1$
$a_4 = (12 * 7 + 22 * 2) \bmod n = 7.$

Einer der vier Werte ist das gesuchte a. In unserem Fall ist es a_4.[4]

[13] Beweis: siehe Kapitel 2.2.5.1.
[14] Siehe Kapitel 2.2.1.

1.4.3. Unterschiede zwischen dem symmetrischen und dem asymmetrischen Konzept

Das folgende Beispiel zeigt die Unterschiede zwischen den beiden kryptographischen Konzepten. [26]

- Symmetrische Verfahren:
  Alice und Bob benutzen einen Briefkasten, um sich Nachrichten zu senden. Beide haben einen Schlüssel zu dem Briefkasten. Wenn Bob eine Nachricht zu Alice schicken möchte, steckt er sie einfach in den Briefkasten. Alice kann Bob auf demselben Weg auch Nachrichten übermitteln. Das Problem hierbei ist der Austausch des Schlüssels.
- Asymmetrische Methode:
  Bob und Alice haben beide ihren eigenen Briefkasten. Möchte Bob eine Nachricht zu Alice schicken, steckt er sie in ihren Briefkasten. Nur Alice hat den Schlüssel und somit kann auch nur sie die Nachricht lesen. Nicht einmal Bob kann sie jetzt noch lesen. Wenn Alice eine Nachricht an Bob schickt, benutzt sie einfach seinen Briefkasten, zu dem nur Bob den Schlüssel hat (siehe Abbildung 2 und 3).

Die Methode E, wie verschlüsselt wird, ist bekannt, ebenso der Schlüssel e zum Verschlüsseln. Der Schlüssel d zum Entschlüsseln muss geheim bleiben. Es darf nicht möglich sein, d aus e zu berechnen.

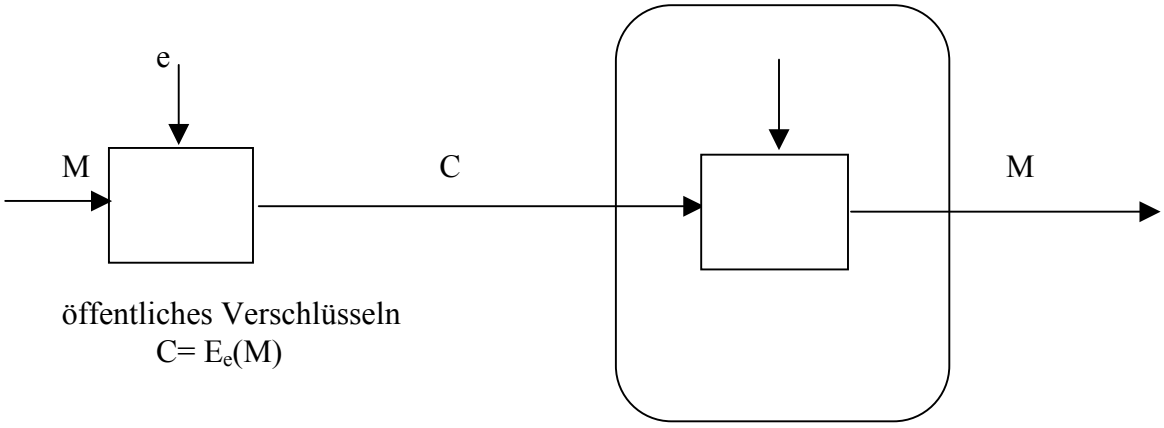


Abbildung 6: Funktionsschema der asymmetrischen Verschlüsselungsmethode.
Quelle: Albrecht Beutelspacher, Jörg Schwenk, Klaus- Dieter Wolfenstetter: Moderne Verfahren der Kryptographie, Vieweg 1995 [Modifiziert]

1.5. Authentifizierung

Die Authentifizierung beinhaltet drei wesentliche Punkte. Erstens - die Nachrichtenintegrität. Darunter versteht man, dass eine Nachricht ohne Veränderung oder Fälschung beim Empfänger ankommt. Zweitens - die Nachrichtenauthentifikation. Diese stellt sicher, dass das Dokument wirklich vom Unterschreibenden stammt. Drittens - die Benutzerauthentifikation oder auch Identifizierung. Hierbei muss der Empfänger sich von der Identität des Senders überzeugen können.
Im normalen Leben ist das sehr leicht, indem man ein Dokument einfach unterschreibt. Die Unterschrift steht auf derselben Seite wie die Erklärung. Sie stellt die Authentifikation sicher. Weiterhin ist es schwierig, eine handschriftliche Unterschrift zu fälschen. Damit kommt die Identitätseigenschaft zum Tragen. Des Weiteren kann der Empfänger die Nachricht verifizieren, indem er die Unterschriften vergleicht und somit wird die Nachrichtenauthentifikation gewährleistet.

Diese Eigenschaften der Unterschrift in elektronischer Form zu gewährleisten, ist etwas komplizierter. Dazu benötigt man noch ein weiteres Hilfsmittel - die Hashfunktion. Eine Hashfunktion ist eine mathematische oder anderweitig definierte Funktion. Der Eingabewert ist ein String oder anderes Zeichen variabler Länge. Der Ausgabewert der Funktion ist ein String oder eine Zeichenkette fester Länge. Der Zweck der Hashfunktion besteht darin, einen Fingerabdruck eines Dokumentes zu erstellen, damit man es mit einem übermittelten Dokument vergleichen kann. Um die Vorgehensweise beim Signieren etwas zu verdeutlichen, wird es am Beispiel von Alice und Bob gezeigt.
Bob erzeugt seine digitale Signatur mit Hilfe einer Hashfunktion und seines privaten Schlüssels. Wenn Bob eine Nachricht an Alice senden und unterschreiben möchte, muss seine elektronische Unterschrift die oben genannten Merkmale besitzen.
Um die Eigenschaft der Nachrichtenintegrität sicherzustellen, verwendet er eine Hashfunktion. Diese berechnet einen Wert, der spezifisch für die Nachricht ist.
Für die Nachrichtenauthentifikation benutzt Bob seinen privaten Schlüssel. Er wendet die Verschlüsselungsfunktion auf den Hashwert und zum Beispiel auf seinen Namen an. Damit wird auch die Identifizierung gewährleistet.
Alice, die Empfängerin der Nachricht, verwendet Bobs öffentlichen Schlüssel, um seine Unterschrift zu entschlüsseln. Sie erhält seinen Namen und den Wert der Hashfunktion.
Sie wendet ebenfalls die Hashfunktion auf die Nachricht an und vergleicht ihren Wert mit Bobs Wert der Hashfunktion. Stimmen die Werte überein, kann Alice sicher sein, dass es die Originalnachricht ist. Weiterhin weiß sie auch, dass die Nachricht nur von Bob stammen kann, denn nur er kennt seinen privaten Schlüssel. Damit wird auch die Nachrichtenauthentifikation gewährleistet.[26]
Eine zweite Möglichkeit, eine Nachricht zu unterschreiben, besteht darin, dass Bob zum Beispiel seine Unterschrift mit seinem privaten Schlüssel verschlüsselt. Alice kann die Unterschrift dann mit seinem öffentlichen Schlüssel entschlüsseln.
Mit diesem etwas einfacheren Verfahren wird eine der wichtigen Eigenschaften einer Unterschrift gewährleistet, die Nachrichtenauthentifikation. Alice kann sicher sein, dass die Nachricht von Bob stammt. Denn nur er kann sie mit seinem privaten Schlüssel unterschrieben haben. Die Nachricht selber kann aber verändert worden sein. Dies ist der Grund warum dieses Verfahren in der Praxis nicht angewendet wird.
Bemerkung: Theoretisch ist dies korrekt, aber Kryptoanalytiker haben eine Möglichkeit

Abbildung 7: Gläserner Tresor als Modell für digitale Signatur
Quelle: Albrecht Beutelspacher, Jörg Schwenk, Klaus- Dieter Wolfenstetter: Moderne Verfahren der Kryptographie, Vieweg 1995

gefunden, mit Hilfe des zweiten Verfahrens Bobs Unterschrift zu fälschen.[15]

Der „Gläserne Tresor“ verdeutlicht die Funktionsweise einer digitalen Unterschrift. Nur der Eigentümer kann das Dokument in den Tresor gelegt haben, denn nur er hat den Schlüssel. Das Dokument wird durch das Einschließen in den Tresor signiert und kann dann von jedem anderen Teilnehmer verifiziert werden.[1]

1.6. Anwendung

Mit Hilfe der Kryptographie können Nachrichten sicher übermittelt werden. Dies macht deutlich, warum gerade heute sich mehr Menschen mit dieser Disziplin beschäftigen. Durch die Entwicklung von Information und Infrastruktur trat die Frage der Sicherheit bei der Datenübermittlung in den Vordergrund. Das hatte zur Folge, dass gerade in den letzten Jahrzehnten immer mehr sichere kryptographische Methoden entwickelt wurden. Zum Beispiel das asymmetrische Verschlüsselungsverfahren, zu dem das RSA – Verfahren[16] zählt. Wie bereits erwähnt, hat die Kryptographie ihr Anwendungsgebiet in der sicheren Übermittlung von Daten. Es ist leicht ersichtlich, dass dieser Bereich mit der Entwicklung des Internets vielseitige Möglichkeiten bietet. Zum Beispiel durch den Handel im Internet, auch Electronic Commerce genannt oder kurz e-commerce, der nicht nur Online Shopping umfasst, sondern auch Online Banking, Aktienhandel sowie das Buchen von Flügen und Hotels. Dabei werden persönliche sensible Daten per World Wide Web übertragen. Um Betrug und Missbrauch dieser Daten auszuschließen, werden sie verschlüsselt. Eine Möglichkeit ist es, die Daten zu verschlüsseln, wenn sie online eingegeben werden. Eine andere Lösung ist, dass die ganze Sitzung vom Computer verschlüsselt wird und dann über das Internet an den Server (Shoppingcenter) geschickt wird. Der Server entschlüsselt die Sitzung wieder. Somit ist es für Dritte unmöglich, die Daten zu lesen. Je mehr Handel über das World Wide Web betrieben wird, um so größer ist die Nachfrage nach immer mehr Sicherheit gegen Diebstahl, Korruption und Betrug. Wie sicher das Internet sein kann, verdeutlicht diese Aussage: „Es ist viel einfacher, die Quittung der Kreditkarte vom Tisch im Restaurant zu stehlen als eine Kreditkartennummer aus dem Netz zu entschlüsseln“.[26]

Des Weiteren benutzen viele Menschen E-Mail zur persönlichen sowie geschäftlichen Kommunikation. Die Nachricht wird dabei durch das Netz übertragen. Durch die Übertragung entstehen viele Kopien der Nachricht, die durch unautorisierte Personen leicht abgefangen werden können. Die Kryptographie beschützt die Nachricht, so dass nur berechtigte Personen sie lesen können.
Dies sind aber nicht die einzigen Anwendungsmöglichkeiten der Kryptographie. Sie wird weiterhin zur Erzeugung elektronischer Unterschriften (Digitale Signatur), der Authentifizierung sowie der Identifizierung verwendet. Das ist notwendig, um zum Beispiel Zugriff auf Daten zu erhalten. Auch für die digitale Unterschrift gilt, dass sie sicherer ist als eine handschriftliche Unterschrift. Unterschreibt man ein Dokument, kann man sich nie sicher sein, ob nicht jemand den Inhalt nachträglich ändert. Mit der elektronischen Unterschrift ist diese Art von Fälschung viel schwieriger, denn sie ist mit dem Inhalt des Dokumentes verbunden.
Kryptographie wird aber nicht nur auf dem Gebiet der Computerkommunikation angewendet, sondern auch in anderen Bereichen der Telekommunikation. Ein Beispiel dafür ist das Handy, das durch einen PIN vor Missbrauch geschützt wird. Die Kryptographie sorgt auch dafür, dass kein anderer auf meine Kosten telefonieren kann.

---

[15] Siehe Kapitel 2.6.4.
[16] Asymmetrisches Verschlüsselungsverfahren von Ron **R**ivest, Adi **S**hamir und Leonard **A**dleman, siehe auch Kapitel 2.

Ein weiteres Anwendungsgebiet ist Pay - TV, das über Satellit oder Kabel übertragen wird. Beim Kabelfernsehen hat das Pay - TV Unternehmen eine direkte Verbindung zu jedem Haushalt. Der Kunde bekommt nur die Programme zu sehen, für die er bezahlt hat.
Es gibt auch die Möglichkeit, dass der Kunde sich wie in einer Videothek einen Film mietet, nur bleibt ihm dabei der Weg zur Videothek erspart. Er ruft einfach bei der Pay - TV Company an und diese sendet den Film über einen bestimmten Kanal verschlüsselt an den Kunden. Außerdem sendet das Unternehmen noch ein Extrasignal an die Kabelbox des Kunden, damit die Box den Film auch entschlüsselt.
Beim Satellitenfernsehen funktioniert es anders, da das Unternehmen keine direkte Verbindung zu dem Kunden hat. Jeder kann das Signal des Satelliten empfangen, aber nur über einen bestimmten Empfänger wird das Signal auch entschlüsselt. Das Mieten von Filmen funktioniert dann wieder genauso wie beim Kabelfernsehen.
Somit hat die Kryptographie ihren festen Platz in unserem täglichen Leben erhalten. Sie findet nicht nur Anwendung bei der Datenübertragung im Internet oder beim Telefonieren, sondern schon in einfachen Haushaltsgeräten wie beim Radio oder TV.[17]

2. Das RSA – Verfahren

In diesem Kapitel wird die mathematische Seite des RSA - Verfahrens beleuchtet, Hintergrundinformationen werden dargelegt und Angriffsmöglichkeiten auf das RSA - Verfahren erläutert.

2.1. Geschichte des RSA – Verfahrens

Mit der Idee des asymmetrischen Kryptographiekonzeptes von Whitfield Diffie und Martin Hellmann wurde auch gleichzeitig die Voraussetzung für die Entwicklung des RSA – Verfahrens geschaffen. Dies wird auch durch den zeitlichen Ablauf widergespiegelt, denn Ron **R**ivest, Adi **S**hamir und Leonard **A**dleman stellten ihr Verfahren nur ein Jahr später vor.[30] Das Verfahren wurde nach seinen Entdeckern benannt. Es verwirklicht alle Anforderungen der asymmetrischen kryptographischen Methode. Es ermöglicht nicht nur, Texte zu ver- und entschlüsseln, sondern es können auch digitale Signaturen erstellt werden. [26]

2.2. Theorie des RSA – Verfahrens

2.2.1. Grundlagen der Zahlentheorie

In diesem Kapitel werden einige zahlentheoretische Ideen dargestellt und wichtige Sätze bewiesen, die für das Verfahren relevant sind. Weitere Ausführungen sind in den im Literaturverzeichnis angegebenen Büchern über Zahlentheorie zu finden.

2.2.1.1. Allgemeine Definitionen

Definition: Ein kommutativer Ring ist eine Menge A zusammen mit zwei Verknüpfungen (Rechenarten) „+“ und „*“, so dass folgendes gilt:

(i) $a + (b + c) = (a + b) + c$
(ia) $a^* (b * c) = (a * b) * c$
(ii) $a + b = b + a$
(iia) $a * b = b * a$
(iii) Es gibt ein Element $0 \in A$ mit $a + 0 = a$
(iv) Für ein bezüglich „+“ neutrales Element 0 gibt es zu jedem a ein Element $-a \in A$ mit $a+(-a) = 0$.
(v) $a * (b + c) = (a + b) + (a * c)$

Definition: Eine abelsche Gruppe ist eine Menge G zusammen mit einer Verknüpfung “+”, welche die Axiome der Addition für Ringe (i)-(iv) erfüllt.

Definition: $\mathbb{Z}$ ist die Menge der ganzen Zahlen. $\mathbb{Z} = \{..,-2, -1, 0, 1, 2,..\}$
$\mathbb{N}$ ist die Menge der natürlichen Zahlen $\mathbb{N} = \{0, 1, 2,..\}$.

Definition: Eine Untergruppe einer abelschen Gruppe G ist eine Teilmenge H von G, für die gilt:i) $0 \in H$;

ii) $a, b \in H \longrightarrow$ wenn $a + b \in H$;
iii) $a \in H \longrightarrow$ wenn $-a \in H$.

Folgende Teilmengen von $\mathbb{Z}$ spielen eine wichtige Rolle.

Definition: Seien a, b $\in \mathbb{Z}$. Man sagt, a ist ein Teiler von b oder a teilt b, und schreibt a| b, wenn es ein c $\in \mathbb{Z}$ gibt mit a * c = b.

Definition: Seien a, m $\in \mathbb{Z}$ mit $m\mathbb{Z} := \{mx \mid x \in \mathbb{Z}\}$ und $a + m\mathbb{Z} := \{a + mx \mid x \in \mathbb{Z}\}$.[7]

1. Satz: Sei m $\in \mathbb{Z} - \{0\}$, a, b $\in \mathbb{Z}$: Dann fällt genau 1 Element von $a + m\mathbb{Z}$ in das Intervall [b, b + | m | [(in welchen | m | aufeinanderfolgende ganze Zahlen liegen).[7]

Beweis: Da $a + m\mathbb{Z}$ nicht nach oben beschränkt ist, ist $M = \{x \in a + m\mathbb{Z} \mid x \geq b\} \neq \{\}$.
Sei y = a + mx minimal in dieser Menge. Wäre $y \geq b + |m|$, so wäre auch
$y - |m| = a + m(x \pm 1) \in M$. (das Vorzeichen von 1 ist „–“, wenn m > 0 und „+“, wenn m < 0 ist.) Dies stünde im Widerspruch zur Minimalität von y. Also ist y das gesuchte Element. Wenn ferner y‘ = a+ m x‘ $\in$ [b, b + | m | [gilt, so ist $|m| > y' - y = m(x' - x) \geq 0$, also $|m| > |m| \cdot |x' - x| \geq 0$, mithin $1 > |x' - x| \geq 0$, d.h. x‘ = x, da gilt, das kleinste Element von $\{x \in \mathbb{Z} \mid x > 0\}$ ist 1. Es folgt y‘ = y.

Definition: Die Menge aller positiven Teiler von a $\in \mathbb{N}$, d.h. die Menge
$T(a) = \{x \in \mathbb{N} \mid x \mid a\}$, nennen wir Teilermenge von a.

Definition des gemeinsamen Teilers von a und b:
Seien a, b $\in \mathbb{N}$. Jedes Element von $T(a) \cap T(b) = \{x \in \mathbb{N} \mid x \mid a \text{ und } x \mid b\}$ wird gemeinsamer Teiler von a und b genannt.

Definition des größten gemeinsamen Teilers (ggT):
Seien a, b $\in \mathbb{N}$. Das größte Element von $T(a) \cap T(b)$ heißt größter gemeinsamer Teiler von a und b.

Definition: Kleinstes gemeinsames Vielfaches
Sind a, b $\in \mathbb{Z}$ so heißt die Zahl v $\in \mathbb{Z}$ kleinstes gemeinsames Vielfaches von a und b, wenn gilt : 1) $v \geq 0$, a | v und b | v.
2) Für jedes gemeinsames Vielfache c von a und b gilt: v | c.

2. Satz: Für alle Zahlen a, b $\in \mathbb{Z}$ gilt: ggT(a, b) * kgV(a ,b)=|a * b|. Beweis siehe [13]

Definition: Zwei Zahlen a, b $\in \mathbb{N}$ heißen teilerfremd oder prim zueinander, wenn a und b nur 1 als gemeinsamen Teiler besitzen, falls also $T(a) \cap T(b) = \{1\}$
Bemerkung: 1 ist zu jeder Zahl b $\in \mathbb{N}$ prim; denn $T(1) \cap T(b) = \{1\}$.

Definition: Die Restklasse von a mod m ist die Teilmenge
$a + m\mathbb{Z}$ von $\mathbb{Z}$ mit a, m $\in \mathbb{Z}$. Sie wird auch mit (a mod m) bezeichnet.

Definition: Für m $\in \mathbb{Z}$ bezeichnet man $\mathbb{Z}/ m\mathbb{Z}$ (sprich $\mathbb{Z}$ modulo m) die Menge aller Restklassen mod m.
Bemerkung: Wenn der sogenannte Modul m fixiert ist, schreiben wir häufig $\bar{a}$ = (a mod m).

Wenn m ≠ 0, ist kann man sich die Elemente von ℤ/ mℤ „kreisförmig angeordnet“ vorstellen, z. B: für m = 12:

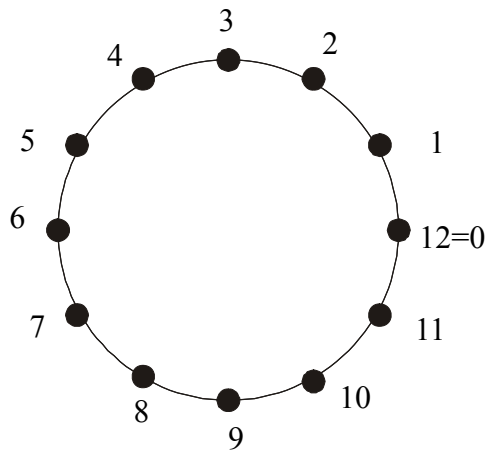


Abbildung 8: Modulo 12

Definition: Die Addition und Multiplikation in ℤ/ mℤ:

$$\overline{a} + \overline{b} = \overline{a + b};\ \overline{a} * \overline{b} = \overline{ab}.$$

Bemerkung: Es muss noch gezeigt werden: dass aus a ≡ a’ mod m und b ≡ b’ mod m folgt a + b ≡ a’ + b’ mod m und ab ≡ a’b’ mod m.

Beweis: Voraussetzung ist a’ = a + sm und b’ = b + tm mit s,t ∈ ℤ.
Daraus folgt für das Produkt:
a’b’ = (a + sm) (b + tm) = ab + (sb + ta + stm) m, mit (sb + ta + sm) ∈ℤ also
a’b’ ≡ ab mod m.
Für die Addition: a’+ b’ = (a + sm) + (b + tm) = a + b + (t + s)m mit (t + s) ∈ ℤ als
a’+ b’ ≡ a + b mod m.

Damit gelten für das Rechnen mit mod die Gesetze wie für die Multiplikation oder Addition in ℤ. Außerdem erhält man das gleiche Ergebnis, wenn man alle Zwischenresultate mod n reduziert oder erst die ganze Rechnung durchführt und dann das Endergebnis mod n reduziert:
(a + b) mod n = ((a mod n) + (b mod n)) mod n
(a – b) mod n = ((a mod n) – (b mod n)) mod n
(a * b) mod n = ((a mod n) * (b mod n)) mod n
(a * (b + c)) mod n = (((a * b) mod n) + ((a * c) mod n)) mod n [4]

3. Satz: Für a, b, m ∈ ℤ sind folgende Aussagen äquivalent:

i) (a mod m) = (b mod m)
ii) a ∈(b mod m)
iii) b∈ (a mod m)
iv) (a mod m) ∩ (b mod m) ≠ {}
v) a – b ∈ mℤ, das heißt m | a – b.

Wenn $m \neq 0$ ist, sind diese Aussagen äquivalent zu:

vi) Aus $a = m\,q_1 + r_1$, $b = m\,q_2 + r_2$ mit $0 \leq r_i < m$ folgt $r_1 = r_2$.[7]

Beweis:
Wenn i) dann ii): $a \in (a \bmod m) = (b \bmod m)$.
Wenn i) dann iii): $b \in (b \bmod m) = (a \bmod m)$.
Wenn ii) dann iv): da $a \in (b \bmod m)$ und $a \in (a \bmod n)$, ist $a \in (a \bmod m) \cap (b \bmod m)$. Daraus folgt, dass $(a \bmod m) \cap (b \bmod m) \neq 0$.
Wenn iii) dann iv): da $b \in (a \bmod m)$ und $b \in (b \bmod n)$, ist $b \in (a \bmod m) \cap (b \bmod m)$. Daraus folgt, dass $(a \bmod m) \cap (b \bmod m) \neq 0$.
Wenn iv) dann i): ist $c \in (a \bmod m) \cap (b \bmod m)$ mit $a, b, m \in \mathbb{Z}$,
somit ist $c = a\ m\ x = b + m\ y$ mit gewissen $x, y \in \mathbb{Z}$. Man erhält $a = b + m(y - x)$,
also $a + mz = b + m(y - x + z) \in (b \bmod m)$, mithin $(a \bmod m) \subset (b \bmod m)$.
Aus Symmetriegründen hat man auch die Inklusion $(b \bmod m) \subset (a \bmod m)$,
also $(a \bmod m) = (b \bmod m)$.

Damit ist die Äquivalenz von i) bis iv) gezeigt.
„ii) $\leftrightarrow$ v) “: $a \in (b \bmod m) \leftrightarrow a = b + mz$ für ein $z \leftrightarrow a - b = mz$ für ein $z \leftrightarrow a - b \in m\mathbb{Z}$.
Wenn i) dann vi): $r_1 = a + m(-q_1)$ und $r_2 = a + m(-q_2)$ sind Elemente der Restklasse $a + m\mathbb{Z}$, die ins Intervall $[0,|m| - 1]$ fallen. Wegen Satz 1 ist $r_1 = r_2$.
Wenn vi) dann v): $a - b = m(q_1 - q_2) + r_1 - r_2 = m(q_1 - q_2) \in m\mathbb{Z}$ da $r_1 = r_2$.

q.e.d.

Definition: Man sagt, „a ist kongruent zu b modulo m“, und schreibt $a \equiv b \bmod m$, wenn a, b, m die äquivalenten Aussagen von Satz 3 erfüllen.
Bemerkung: Zwei Zahlen a und $b \in \mathbb{Z}$, die bei der Division durch m den selben Rest ergeben, heißen kongruent modulo m.[17]

Definition: a) Die Relation ~ heißt eine Äquivalenzrelation, wenn gilt:

- $a \sim a$ (für alle $a \in M$) (Reflexivität);
- wenn $a \sim b$, dann $b \sim a$ (Symmetrie);
- wenn $a \sim b$ und $b \sim c$, dann $a \sim c$ (Transitivität).

(Äquivalent zu diesen Forderungen sind:

- $a \sim a$;
- wenn $a \sim b$, $a \sim c$ dann $b \sim c$ (Komparativität).)

b) Eine Äquivalenzklasse bezüglich einer Äquivalenzrelation ~ ist eine nichtleere Teilmenge $C \subset M$ mit folgenden beiden Eigenschaften:
wenn $a, b \in C$ dann $a \sim b$;
wenn $a \in C$, $b \in M$ und $a \sim b$ dann $b \in C$.

Bemerkung: Seien M eine Menge und ~ eine (zweistellige) Relation auf dieser Menge; d.h. für $a, b \in M$ gilt entweder „$a \sim b$“ oder „nicht $a \sim b$“. Wenn für eine (zweistellige) Relation das Kongruenzeichen $\equiv$ eingesetzt wird, heißt die Äquivalenzklasse auch Kongruenzklasse. (Beispiel für Relationen sind $\equiv$, $|$, $\leq$.)

[17] Die Bezeichnung führte Gauss ein.

4. Satz: p, a, a' $\in \mathbb{Z}$
p ist ein Primzahl, aus $a \equiv 0 \bmod p$ oder $a' \equiv 0 \bmod p$ folgt stets $aa' \equiv 0 \bmod p$.

Beweis: Es wird angenommen, dass gilt $a \equiv 0 \bmod p$ und $a' \equiv x \bmod p$ ($x \in \mathbb{Z}$ sonst beliebig) und es wird vorausgesetzt, dass gilt $p \mid a - 0$ und $p \mid a' - x$. Es folgt $p \mid (a - 0)\, a'$ sowie $p \mid 0\,(a' - x)$, also $p \mid aa'$ q.e.d.

5. Satz: Der Chinesische Restsatz:
Seien $m_1, \ldots, m_n \in \mathbb{N}$ paarweise teilerfremd und $m = m_1 * m_2 * \ldots * m_n$, dann ist das System der Kongruenzen

$$x \equiv a_1 \bmod m_1$$
$$x \equiv a_2 \bmod m_2$$
$$\vdots$$
$$x \equiv a_n \bmod m_n$$

in $\mathbb{Z}/ m\mathbb{Z}$ eindeutig lösbar. Beweis siehe [7].

$m'_1 = m/m_1, \ldots, m'_n = m/m_n$ $\quad ggT(m'_1, \ldots, m'_n) = 1$
$m'_1 g_1 + m'_2 g_2 + \ldots + m'_n g_n = 1$
$x = a_1 m'_1 g_1 + a_2 m'_2 g_2 + \ldots + a_n m'_n g_n \bmod m$ [18]

6. Satz: $\mathbb{Z}/ p\mathbb{Z}$ ist eine abelsche Gruppe bezüglich der Addition

Beweis: Dass die Addition und Multiplikation in $\mathbb{Z}/ m\mathbb{Z}$ gilt, siehe Definition der Addition und Multiplikation in $\mathbb{Z}/ m\mathbb{Z}$:
Die Ring – Axiome für die Addition und Multiplikation in $\mathbb{Z}$ vererben sich auf $\mathbb{Z}/ m\mathbb{Z}$, so dass $\mathbb{Z}/ m\mathbb{Z}$ wieder ein kommutativer Ring (mit Einselement) wird und somit eine abelsche Gruppe in Bezug auf die Addition ist.

7. Satz: Die Restklasse a mod m ist ein Ring $\mathbb{Z}/ m\mathbb{Z}$ und genau dann invertierbar, wenn $ggT(a, m) = 1$.

Beweis: Als erstes wird vorausgesetzt, dass $ggT(a, m) = 1$. Dann gibt es zwei ganze Zahlen s und t, so dass $as + mt = 1$ (Die beiden Zahlen s und t können mit dem erweiterten Euklidischen Algorithmus, siehe Kapitel 2.2.4., berechnet werden). Daraus folgt aber $as \equiv 1 \bmod m$, d.h. s mod m ist ein Inverses von a mod m.
Jetzt setze man voraus, dass a mod m in $\mathbb{Z}/ m\mathbb{Z}$ invertierbar ist mit Inversen s mod m, dann ist $as \equiv 1 \bmod m$, d.h. $as = 1 + km$ mit $k \in \mathbb{Z}$. Aus der Gleichung
$as + (-k)m = 1$ folgt $ggT(a, m) = 1$. q. e. d.

Folgerung: Für jede Primzahl p gibt es in $\mathbb{Z}/ p\mathbb{Z}$ ein multiplikatives Inverses a.

Beweis: Da p prim ist, gilt für jede ganze Zahl a mit $a \not\equiv 0 \bmod p$, dass $ggT(a, p) = 1$, d.h., das Element $\bar{a} \in \mathbb{Z}/p\mathbb{Z}$ besitzt ein Inverses. q. e. d.

[18] Den Beweis für kgV, ggT sowie zu dem Chinesischen Restsatz lesen Sie bitte in den im Literaturverzeichnis angegebenen zahlentheoretischen Büchern nach.

Bemerkung: Somit ist bewiesen, dass $\mathbb{Z}/p\mathbb{Z}$ eine abelsche Gruppe bezüglich der Addition ist und das $\mathbb{Z}^*(p) = \mathbb{Z}/p \setminus \{0\}$ eine abelsche Gruppe bezüglich der Multiplikation ist und somit ein Körper.

8. Satz: Wenn $a \equiv b \bmod p$ und $a \equiv b \bmod q$, dann ist $a \equiv b \bmod n$ mit p und q teilerfremd und $n = p * q$.

Beweis von $a \equiv b \bmod p$ und $a \equiv b \bmod q$ „$\leftrightarrow$“ $a \equiv b \bmod n$, wenn p und q teilerfremd.
„$\leftarrow$“ Da $p \mid n$, $q \mid n$ und $n \mid (a - b)$ gilt auch $a \equiv b \bmod p$ und $a \equiv b \bmod q$: Da
$n * c = a-b$ mit $c \in \mathbb{Z}$, ist pq $c = a - b \longrightarrow p*c_1 = a - b$ und $q*c_2 = a-b$.
„$\longrightarrow$“ Vorausgesetzt wird, dass $p \mid (a - b)$ und $q \mid (a - b)$ somit gilt $p * c_1 = a - b$ und
$q *c_2 = a - b \longrightarrow pq * c = a - b = n * c = a - b \longrightarrow n \mid a - b$ q.e.d.

9. Satz: Es sei $n = p^{k_1}{}_1 p^{k_2}{}_2 \ldots p^{k_t}{}_t$,
10. die Primzahlenzerlegung von $n \in \mathbb{Z}$. Dann gilt: $a \equiv b \bmod n \leftrightarrow a \equiv b \bmod p^{k_j}{}_j$.für $j = 1,2,...,t$

“$\longleftarrow$” Beweis durch Induktion
Induktionsanfang: für t=2
Induktionsvoraussetzung: es gilt für $n = p^{k_1}{}_1 p^{k_2}{}_2 \ldots p^{k_t}{}_t$ die Beziehung $a \equiv b \bmod n \longleftrightarrow$
$a \equiv b \bmod p^{k_j}{}_j$. für alle $j = 1,2,..,t$,
z.Z. das $a \equiv b \bmod n_1$ mit $n_1 = p^{k_1}{}_1 p^{k_2}{}_2 \ldots p^{k_t}{}_t, p^{k\ t+1}{}_{t+1}$, somit ist $n_1 = n\, p^{k\ t+1}{}_{t+1}$ da $n\, p^{k\ t+1}{}_{t+1}$ eine Primzahlenzerlegung ist, gilt: “$\longleftarrow$” Wenn $n \mid n_1$, $p^{k\ t+1}{}_{t+1} \mid n_1$ und $n_1 \mid (a-b)$ gilt auch $a \equiv b \bmod n$ und $a \equiv b \bmod p^{k\ t+1}{}_{t+1}$. Es gilt $n_1\, c = a-b$ und somit ist $n\, p^{k\ t+1}{}_{t+1} * c = a - b \longrightarrow n * c_1 = a-b$ und
$p^{k\ t+1}{}_{t+1} * c_2 = a - b$. q.e.d.

2.2.1.2. Beweis des Kleinen Satzes von Fermat

Als nächstes möchte ich den für das RSA – Verfahren wichtigen Kleinen Satz von Fermat beweisen. Dazu benötigt man einige Begriffe, die ich anhand von Definitionen einführe.

Definition: Eine Gruppe G heißt zyklisch, wenn es ein Element $a \in G$ gibt, so dass gilt
$G = \{a^n / n \in \mathbb{Z}\} = \{\ldots, a^{-2}, a^{-1}, e, a, a^2 \ldots\}$. Dann heißt a ein G erzeugendes Element.
Bemerkung: In einer zyklischen Gruppe sind dementsprechend alle Elemente Potenzen eines einzigen Elementes. In solchen Gruppen lässt sich mit Hilfe der bekannten Potenzregeln sehr einfach rechnen.
Beispiel: m = 17. Die multiplikative Gruppe $\mathbb{Z}^*{}_{17} \setminus \{0\}$ besitzt die Elemente
$\{3^n / n = 1, 2, 3,.., 16\}$

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$3^n \bmod 17$	3	9	10	13	5	15	11	16	14	8	7	4	12	2	6	1

Tabelle 5: Zyklische Gruppe

Definition: Die Anzahl der Elemente einer endlichen Gruppe G wird auch ihre Ordnung genannt und mit ord(G) bezeichnet.[9]

10. Satz: Sei G eine endliche Gruppe und $H \subset G$ eine Untergruppe. Dann ist ord(H) ein Teiler von ord(G), also ord(H) | ord(G).

Beweis: Es sei H gegeben. Da G endlich ist, gibt es endlich viele, paarweise teilerfremde Kongruenzklassen: Ha_1, Ha_2,..,Ha_t in G, so dass gilt:
$G = Ha_1 \cup Ha_2 \cup ... \cup Ha_t$. Daraus folgt $ord(G) = ord(Ha_1) + ord(Ha_2) + .. + ord(Ha_t)$.
Nun gilt: $ord(Ha) = ord(H)$ für jedes $a \in G$, denn die Abbildung $H \longrightarrow Ha$, $x \mapsto xa$ ist bijektiv.
Es folgt $ord(G) = t^* ord(H)$, d.h. $ord(H) \mid ord(G)$.

11. Daraus folgt: Sei G eine endliche Gruppe und $x \in G$, dann ist ord(x) ein Teiler von ord(G), denn ord(x) ist gleich der Ordnung der von x erzeugten Untergruppe $U(x) \subset G$.

12. Satz: Sei G eine endliche Gruppe, dann gilt für jedes Element $x \in G$: $x^{ord(G)} = e$.

Beweis: Nach dem 10. Satz gibt es eine ganze Zahl r mit $ord(G) = ord(x) * r$.
Daraus folgt $x^{ord(G)} = (x^{ord(x)\, r}) = e^r = e$.

Mit diesen Definitionen und Sätzen lässt sich der Kleine Satz von Fermat beweisen.

13. Satz (Kleiner Satz von Fermat): Sei p eine Primzahl, dann gilt für jede nicht durch p teilbare ganze Zahl a: $a^{p-1} \equiv 1 \bmod p$.

Beweis: a, genauer a mod p, kann als Element der multiplikativen Gruppe $(\mathbb{Z}/p\mathbb{Z})^* \setminus \{0\} = \mathbb{F}^*_p$ aufgefasst werden, die aus p – 1 Elementen besteht. $a^{p-1} \equiv 1 \bmod p$, $a \nmid p$.
Nach dem 12. Satz gilt bei endlichen Gruppen F^*_p, mit der Ordnung $ord(F^*_p) = p - 1$, für $a \in F^*_p$ und $a^{ord(F^*p)} = a^{p-1} = e = 1$. Damit folgt $a^{p-1} \equiv 1 \bmod p$. q.e.d.

Die Verallgemeinerung des Kleinen Satzes von Fermat ist der Satz von Fermat – Euler.

Definition: $\varphi(n)$ ist die Anzahl der zu n teilerfremden Zahlen $\leq n$ und wird auch Eulersche Funktion genannt, p_i ist eine Primzahl $\leq n$.

14. Satz: $\varphi(n)$ wird folgendermaßen berechnet $\varphi(n) = n \prod_{i=1}^{k}(1 - 1/p_i)$. Beweis siehe [16]

Daraus folgt: Wenn $n = pq$,

ist $\varphi(n) = n \prod_{i=1}^{k}(1 - 1/p_i)$
$= pq\,(1 - 1/p)\,(1 - 1/q)$
$= pq\,(1 - 1/p - 1/q + 1/pq)$
$= (1 - p)\,(1 - q)$.

1. Folgerung aus dem Satz von Fermat – Euler:
   Ist p eine ungerade Primzahl, so gilt für jedes $a \in \mathbb{N}$ mit $p \nmid a$ genau eine der folgenden beiden Kongruenzen: $a^{\frac{1}{2}(p-1)} \equiv 1 \bmod p$ oder $a^{\frac{1}{2}(p-1)} \equiv -1 \bmod p$.

Beweis: Da p ungerade ist, gilt $m = \frac{1}{2}(p - 1) \in \mathbb{N}$ und
$a^{p-1} - 1 = (a^{\frac{1}{2}(p-1)} - 1)(a^{\frac{1}{2}(p-1)} + 1) = (a^m - 1)(a^m + 1)$. Nach dem Kleinen Satz von Fermat gilt: $a^{p-1} \equiv 1 \bmod p$ daraus folgt: $a^{p-1} - 1 \equiv 0 \bmod p$ und damit $(a^m - 1)(a^m + 1) \equiv 0 \bmod p$. Weiterhin gilt nach der Kürzungsregel: $aa' \equiv 0 \bmod p$ so ist $a \equiv 0 \bmod p$ oder $a' \equiv 0 \bmod p$ und damit folgt $a^{\frac{1}{2}(p-1)} \equiv 1 \bmod p$ oder $a^{\frac{1}{2}(p-1)} \equiv -1 \bmod p$.

2.2.2 Grundaufbau des RSA – Verfahrens

Die Verschlüsselung durch RSA geht folgendermaßen vonstatten: Alice möchte eine Nachricht M an Bob schicken. Sie benutzt seinen öffentlichen Schlüssel e, n und verschlüsselt ihre Nachricht durch modulares Potenzieren: $C = M^e \bmod n$. Sie sendet C an Bob. Er entschlüsselt die Nachricht ebenfalls durch modulares Potenzieren mit seinem privaten Schlüssel d: $C^d \bmod n = M$.
Bob hat seinen öffentlichen Schlüssel (n, e) und seinen privaten Schlüssel d folgendermaßen gebildet. Als erstes sucht er zwei große Primzahlen p und q und bildet ihr Produkt n, sowie $\varphi(n) = (p - 1)(q - 1)$. Dann wählt er eine beliebige Zahl d aus, für die gilt:
$ggT(d, \varphi(n)) = 1$ und berechnet mit Hilfe von d und $\varphi(n)$ e mit $ed \equiv 1 \bmod \varphi(n)$. Es gilt $C^d \bmod n = M^{e*d} \bmod n = M^{(p-1)*(q-1)+1} \bmod n = M$. Da nur Bob d kennt, kann auch nur Bob C entschlüsseln. Die letzte Gleichung geht aus der Eulerschen Verallgemeinerung des Kleinen Satzes von Fermat hervor. Der Kleine Satz von Fermat bildet die Grundlage des RSA – Verfahrens.

In dem obigen Beispiel sind fast alle Grundbestandteile des RSA – Verfahrens enthalten, der öffentliche Schlüssel (e, n) und der private Schlüssel (d). Weitere Elemente des Verfahrens sind die beiden Primzahlen p und q, aus denen sich n zusammensetzt mit
$n = pq$.
Es ergeben sich folgende Fragen aus dem Beispiel: Wie hat Bob die Komponenten e, d, n und p, q ausgewählt? Was ist bei der Auswahl der Komponenten zu beachten?
Wie lang darf die Nachricht sein, damit sie noch eindeutig ver- und entschlüsselt wird? Wie groß müssen die Zahlen p, q und d, e sein, damit das Verfahren sicher ist?
Der nächste Abschnitt beschäftigt sich mit dem ersten Teil der Fragen und dem Aufbau des RSA – Verfahrens.

2.2.3. Auswahl p und q

Die zwei Primzahlen p und q werden als erstes ausgewählt. Sie sind zwar nicht für die Ver- und Entschlüsselung wichtig, aber sie werden zur Berechnung von n und $\varphi(n)$ benötigt. Sie sind geheim zu halten. Kennt der Kryptoanalytiker eine der beiden Primzahlen, kann er den Geheimtext C entschlüsseln.
Für p und q sind Zahlen typisch, die mehr als 100 Stellen haben, denn bei kleineren Zahlen ist es schon möglich, durch Faktorisierung von n die beiden Faktoren p und q zu finden.
Eine Möglichkeit ist es, q und p mit Hilfe eines Zufallszahlengenerators auszuwählen. Ein Zufallszahlengenerator funktioniert im Allgemeinen folgendermaßen: Ein Computerprogramm erzeugt eine Sequenz von Zahlen, die soweit wie möglich statistische Eigenschaften von zufälligen Zahlen besitzt. Die meisten Computerprogramme besitzen voreingestellte Funktionen, die Zufallszahlen erzeugen.[3] So auch das Programm *Mathematica* mit der Funktion *Random[]*. Diese Funktion gibt eine zufällige Zahl r zwischen 0 und 1 aus. Mit zwei zusätzlichen Parametern *Random [type,range]* erzeugt die Funktion eine Zufallszahl von einem bestimmten Typ, zum Beispiel ganze Zahlen, in einem bestimmten Bereich (zum Beispiel *Random[Integer,{1,100}]*).
Als nächstes wird getestet, ob die Zufallszahl eine Primzahl ist. Wenn es keine Primzahl ist, wird r = r + 1 getestet und wieder geprüft. Das wird fortgesetzt, bis ein r gefunden wurde, welches eine Primzahl ist. Der Primzahlentest erfolgt in *Mathematica* durch die Funktion *PrimeQ[Variable]*. Die Funktion gibt *True* aus, wenn die Variable eine Primzahl ist bzw. *False,* wenn sie keine Primzahl ist. [19] Aus den beiden Primzahlen wird n und $\varphi(n)$ folgendermaßen berechnet $n = p * q$ und $\varphi(n) = (p - 1)(q - 1)$. Nach der Festlegung der

[19] Wolfram Research *Mathematica* Book 1986-1996

Komponenten p und q wird der private Schlüssel d und der öffentliche Schlüssel e mit Hilfe des Euklidischen Algorithmus ermittelt.

Primzahlentest
Es gibt verschiedene Primzahlentests. Ein einfaches Verfahren wurde von Lehmen entwickelt. [28]
Der Test überprüft, ob p prim ist:
Als erstes wird eine Zufallszahl a kleiner als p ausgewählt. Danach berechnet man $c = a^{(p-1)/2} \bmod p$. Jetzt wird eine Fallunterscheidung durchgeführt.

1. Fall: ist $c \neq 1$ oder $-1 \bmod p$, so ist p definitiv keine Primzahl.
2. Fall: ist $c = 1$ oder -1, so liegt die Wahrscheinlichkeit, dass p keine Primzahl ist, bei höchstens 50 Prozent.

Der Test wird i mal wiederholt. Dabei sinkt die Fehlerwahrscheinlichkeit mit steigenden i. Sie liegt bei 1 zu 2^i. Diese Wahrscheinlichkeit gibt an, das eine zusammengesetzte Zahl p alle i Tests übersteht und wird mit steigenden i immer geringer.
Dies ist nur eine Möglichkeit zu testen, ob p prim ist. Beweis siehe Kapitel 2.2.1.2. 1. Folgerung aus dem Satz vom Fermat – Euler.

2.2.4. Entschlüsselungsexponent d und Verschlüsselungsexponent e

2.2.4.1. Festlegung von d und Berechnung von e

Die Auswahl von d und e erfolgt abhängig voneinander. Der eine Faktor wird aus dem anderen berechnet.
Als erstes wählt man d aus, damit der Schlüssel nicht zu klein wird und man ihn nicht durch Testen finden kann. Das d muss folgenden Bedingungen genügen: d ist ganzzahlig, größer als Eins, und es gilt: der größte gemeinsame Teiler von d und $\varphi(n)$ ($ggT(d,\varphi(n)) = 1$. Danach wird e mit dem Euklidischen Algorithmus abgeleitet. Es muss folgenden Bedingungen genügen: $1 < e < \varphi(n)$ und $ed \equiv 1 \bmod \varphi(n)$.

Mit Hilfe des Euklidischen Algorithmus' lässt sich der ggT(a, b) zweier natürlicher Zahlen a und b nach folgendem Verfahren in endlich vielen Schritten gewinnen:
$a = r_0$ und $b = r_1$; $r_i \in \mathbb{N}$

$r_0 = r_1 * q_1 + r_2$ mit $r_2 < r_1$
$r_1 = r_2 * q_2 + r_3$ mit $r_3 < r_2$
$r_2 = r_3 * q_3 + r_4$ mit $r_4 < r_3$ usw. [7]

Solange $r_i \neq 0$ ist, kann man q_i, r_{i+1} finden mit
$r_{i-1} = r_i q_i + r_{i+1}$ und $r_{i+1} < r_i$
Es gilt aber $r_1 > r_2 > ..> r_i > r_{i+1}$, somit wird für ein n der Rest r_{n+1} verschwinden und das Verfahren bricht an dieser Stelle ab:

$r_{n-2} = r_{n-1}\, q_{n-1} + r_n$ mit $r_n < r_{n-1}$
$r_{n-1} = r_n\, q_n + 0$

Behauptung : $r_n = ggT(a, b)$
Bemerkung: In $\mathbb{Z}$ gelte $a = b\, c + d$. Dann ist jeder gemeinsame Teiler von a und b auch ein solcher von b und d und umgekehrt. Das heißt, für $t \in \mathbb{Z}$ gilt $t \mid a, t \mid b, \leftrightarrow t \mid b, t \mid d$.
Für das Verfahren gilt $t|r_0,r_1$. Damit erhält man die Äquivalenzen
$t \mid r_0, r_1 \leftrightarrow t \mid r_1, r_2 \leftrightarrow t \mid r_2, r_3 \leftrightarrow ...t \mid r_{n-1}, r_n \leftrightarrow t \mid r_n, 0$,

es folgt ggT(a,b) = ggT(r_n,0) = r_n wegen der Definition des ggT.

Mit diesem Algorithmus wird der ggT(d, φ(n)) berechnet. Ist der ggT(d, φ(n)) = 1, dürfen wir d im RSA – Verfahren verwenden. Ist ggT(d, φ(n)) ≠ 1 muss ein neues d ausgewählt werden.
Beispiel:
p = 17; q = 13; n = 221; φ(n) = 192; d = 109;
ggT(109,192)
109 = 192 * 0 + 109
192 = 109 * 1 + 83
109 = 83 * 1 + 26
83 = 26 * 3 + 5
26 = 5 * 5 + 1
5 = 1 * 5 + 0
damit ist der ggT(109,192) = 1.

Weiterhin erhält man aus dem Euklidischen Algorithmus auch eine Darstellung von r_n als Linearkombination r_n = aa‘ + bb‘ bezogen auf das RSA – Verfahren. Somit ist die Linearkombinationsdarstellung für ggT(d, φ(n)) = 1 gleich 1 = e*d + φ(n) * k und e wird aus dem Euklidischen Algorithmus wie folgt berechnet.
Die Reste lassen sich durch a und b rekursiv darstellen: $r_i = (-1)^i (s_i b - t_i a)$ mit
$s_{-1} = 0$ $s_0 = 1,...,$ $s_i = q_{i-1}\ s_{i-1} + s_{i-2}$
$t_{-1} = 1$ $t_0 = 0,...,$ $t_i = q_{i-1}\ t_{i-1} + t_{i-2}$
Beispiel:

i	-1	0	1	2		n	-1	0	1	2	3	4	5
		q_0	q_1	q_2	...	q_n		0	1	1	3	5	5
	s_{-1}	s_0	s_1	s_2	...	s_n	0	1	0	1	1	4	21
	t_{-1}	t_0	t_1	t_2	...	t_n	1	0	1	1	2	7	37

Tabelle 6: Erweiterter Euklidischer Algorithmus

$r_1 = (-1)^1 (0*192 - 1*109)$
$r_2 = (-1)^2 (1*192 - 1*109)$
$r_3 = (-1)^3 (1*192 - 2*109)$
$r_4 = (-1)^4 (4*192 - 7*109)$
$r_5 = (-1)^5 (21*192 - 37*109)$

Für ggT(d,φ(n)) = 1, mit d = 109 und φ(n) = 192 erhält man die Linearkombination
1 = 37 * 109 – 21 * 192, damit ist e = 37.
Diesen Algorithmus bezeichnet man auch als erweiterten Euklidischen Algorithmus.
So sind alle notwendigen Komponenten ausgewählt und berechnet. Die Funktion $E(M) = M^e \bmod n = C$ ist die Verschlüsselungsfunktion und die Funktion $D(C) = C^d \bmod n$ ist die Entschlüsselungsfunktion.
Diese Funktion $M^e \bmod n$ mit n = pq ist eine Einwegfunktion. Wenn die Faktorisierung von n nicht bekannt ist, lässt sich die Funktion nicht invertieren. Den privaten Schlüssel bilden die Zahlen p, q, φ(n) und d. Sie sind die geheime Falltürinformation der Einwegfunktion. Es ist logisch, dass die Werte voneinander abhängig sind. Ist einer der vier Werte bekannt, sind die anderen berechenbar. Die Zahlen e und n bilden den öffentlichen Schlüssel.

2.2.4.2. Berechnung von d, p, q und φ

Bei den nächsten Berechnungen wird stets vorausgesetzt, dass der öffentliche Schlüssel n und e immer bekannt ist.
Wenn p oder q bekannt sind, lässt sich der jeweilige andere Faktor durch n = pq berechnen.
Nun kann man auch φ (n) berechnen mit φ(n) = (p – 1) (q –1).
d wird mit Hilfe von φ (n) und dem Euklidischen Algorithmus wie e berechnet, siehe oben.

Beispiel : e = 37; φ(n) = 192
ggT(37,192) = 1, da

192 = 37 * + 7	1 = 7 – 3 * 2
37 = 7* 5 + 2	1 = 7 – 3 (37 – 5 * 7) = 16 * 7 – 3 * 37
7 = 2* 3 + 1	1 = 16 (192 – 37 * 5) – 3 * 37 =
	16 * 192 – 83 * 37

Somit ist $d = \varphi(n) - d^{-1} = 192 - 83 = 109$.
Als nächstes steht die Frage, wie man die Komponenten p und q berechnet, wenn nur φ(n) bekannt ist (Natürlich gilt die oben getroffene Vorrausetzung weiterhin.).

n = pq;	φ(n) = (p – 1) (q – 1);	q = n/ p
φ(n) =	(p – 1) (n/ p – 1)	
φ(n) =	n – p – n/p + 1	*p
φ(n)p =	n p – p^2 – n + p	-φ(n) p
0 =	p^2 – n p + φ (n) p – p + n	
0 =	p^2 – p (n – φ(n) +1) + n	

$$p\frac{1}{2} = +\frac{(n - \varphi(n) + 1}{2} \pm \sqrt{\frac{(n - \varphi(n) + 1)^2}{4} - n}$$

Jetzt ist zu beweisen, dass $\frac{(n - \varphi(n) + 1)^2}{4} - n \geq 0$ ist.

$\frac{(n - \varphi(n) + 1)^2}{4} - n \geq 0$ * 4 und für φ und n wird p und q eingesetzt.

(pq – ((p – 1) (q – 1) + 1)^2 – 4pq	≥ 0	↔
(pq – pq + p + q –1+1)^2 – 4pq	≥ 0	↔
(p +q)^2 – 4pq	≥ 0	↔
p^2 – 2pq + q^2	≥ 0	↔
(p – q)^2	≥ 0	q.e.d

2.2.5. Verschlüsselung und Entschlüsselung

2.2.5.1. Beweis der Eindeutigkeit der Verschlüsselung und Entschlüsselung

Voraussetzung für das Verschlüsseln ist, dass der Originaltext in codierter Form vorliegt. Das bedeutet, dass die Buchstaben in Dezimalzahlen konvertiert sind. Dies kann zum Beispiel mit Hilfe des ASCII – Codes[20] geschehen. Es können aber auch binäre Zahlen verwendet werden. Weiterhin ist es notwendig, dass der Text in Blöcke geeigneter Größe geteilt wird. Eine geeignete Blockgröße ist die ganze Zahl i, für welches gilt $10^i < n < 10^{i+1}$. Damit die Entschlüsselung eindeutig ist, muss die Blocklänge kleiner als n sein.[3]

[20] Abk. für engl. **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange

Der codierte Text M wird dann zur e –ten Potenz mod n erhoben. Daraus ergibt sich folgende Gleichung:

$$E(M) = M^{e} \bmod n = C.$$

Bei der Entschlüsselung wird der Geheimtext C zur d –ten Potenz erhoben und der Originaltext M berechnet. Die Gleichung lautet folgendermaßen:

$$D(C) = C^{d} \bmod n = M.$$

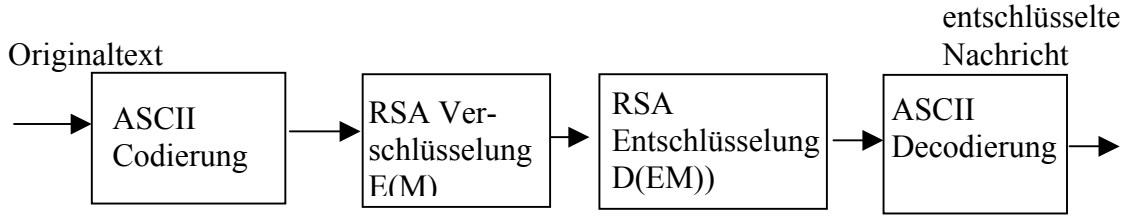


Abbildung 9: Ablaufschema für das Ver- und Entschlüsseln einer Nachricht

Wie das Ablaufschema zeigt, wird die Nachricht als erstes als Zahl codiert und dann mit dem RSA – Verfahren verschlüsselt. Um den Originaltext wieder zu erhalten, muss die Geheimnachricht erst mit dem RSA – Verfahren entschlüsselt werden und mit ASCII Code decodiert werden.

15. Satz: Es gilt $D(E(x)) = x$ mit $E(x)= x^{e} \bmod n$, $D(x) = x^{d} \bmod n$ und $ed \equiv 1 \bmod \varphi(n)$.

Beweis: Die Entschlüsselung und Verschlüsselung mit dem RSA – Verfahren ist eindeutig. Für diesen Beweis benötigen wir den Kleinen Satz von Fermat.[21]

Voraussetzungen: $N_n = \{0,1,..,n-1\}$
$n = pq$ mit $p \neq q$ p und q sind Primzahlen
$e, d \in N_n$ mit $ggT(d, \varphi(n)) = 1$,
$ed \equiv 1 \bmod \varphi(n)$.

Bezeichnung: M Originaltext C Geheimtext $M, C \in N_n$
e, n öffentlicher Schlüssel zum Verschlüsseln d privater Schlüssel zum Entschlüsseln
Für die Funktion E mit E: $N_n \longrightarrow N_n$ gilt $E(M) = M^{e} \bmod n = C$
und für die Funktion D mit D: $N_n \longrightarrow N_n$ gilt $D(C) = C^{d} \bmod n$.

z.Z. $D(E(x)) = x \quad \forall x \in N_n$
$D(E(M) = M$
Es gilt: $(M^{e} \bmod n = C)^{d} \bmod n = D(E(M))$
$ed \equiv 1 \bmod \varphi(n)$
daraus folgt: $1 = e*d + k(p-1)(q-1) \longrightarrow e\,d = k(p-1)(q-1)+1$
Damit gilt: $D(E(M)) = M^{ed} \bmod n$
$= M^{k(p-1)(q-1)+1} \bmod n.$

Nach dem Kleinen Satz von Fermat gilt $a^{(p-1)} \equiv 1 \bmod p$ für $a \nmid p$. Des Weiteren gilt: wenn p und q teilerfremd sind, so ist $a \equiv b \bmod p$ und $a \equiv b \bmod q \leftrightarrow a \equiv b \bmod n$ mit $n = pq$.
Der Kleine Satz von Fermat wurde im Kapitel 2.2.1 bewiesen.

Es gilt $a \equiv b \bmod p$ und $a \equiv b \bmod q \leftrightarrow a \equiv b \bmod n$ mit $n= p\,q$.[22]
Damit gilt: $D(E(M)) \equiv M^{(p-1)(q-1)+1} \bmod q$
$\equiv M^{(p-1)(q-1)+1} \bmod p.$

[21] Der Kleine Satz von Fermat wurde im Kapitel 2.2.1. Grundlagen der Zahlentheorie bewiesen.
[22] Beweis Siehe Kapitel 2.2.1.1. Satz 8.

$$\equiv M^{(p-1)(q-1)+1} \bmod n$$

Nach dem Kleinen Satz von Fermat gilt: $1 \equiv M^{(p-1)} \bmod p$ sowie
$1 \equiv M^{(q-1)} \bmod q$

daraus folgt:
$$\begin{aligned} D(E(M)) &\equiv M^{(p-1)(q-1)+1} \bmod p \\ &\equiv M^{(p-1)(q-1)+1} \bmod q \\ &\equiv M\ 1^{(q-1)} \bmod p \\ &\equiv M\ 1^{(p-1)} \bmod q \\ &\equiv M \bmod p \\ &\equiv M \bmod q \end{aligned}$$

Nach obiger Aussagen gilt: $D(E(M)) \equiv M \bmod n \equiv M$ mit $n = pq$ q. e. d.

Bemerkung: Der obige Beweis gilt unter der Bedingung, dass q und p $\nmid$ M. Wenn jetzt p oder q M teilt, nehmen wir an p teilt M, dann gilt immer noch $M^{\varphi(n)} \equiv 1 \bmod q$, $M^{k\,\varphi(n)} \equiv 1 \bmod q$ und $M^{ed} \equiv M \bmod q$. Die letzte Kongruenz gilt damit mod p. Wenn beide p und q M teilen, gilt trotzdem $M^{ed} \equiv M \bmod n$. Dies folgt aus dem obigen Beweis.

Bemerkung: Bei der Wahl von e muss beachtet werden, dass e – 1 kein Vielfaches von p – 1 oder q – 1 ist, denn sonst gilt für jedes M die Gleichung E(M) = M. Beweis mit Hilfe der 1. Folgerung aus dem Satz von Fermat – Euler.
(e – 1) = k (p – 1) (q – 1), damit ist e = k (p – 1) (q – 1) + 1 = k*φ(n) + 1 mit k $\in \mathbb{N}$. Daraus folgt, wie wir schon oben gesehen haben, dass $M^e \bmod n = M^{(p-1)\,(q-1)\,+\,1} \bmod n = M$ ist. Auf diesem Prinzip beruht die Entschlüsselung.
Auch für e = φ(n)/2 + 1 gilt die Gleichung E(M) = M. Zu beachten ist, dass es im vorgegebenen Bereich von e liegt.

Beweis: Nach dem obigen Beweis gilt:
$$\begin{aligned} E(M) &\equiv M^{(p-1)(q-1)/2+1} \bmod n \\ &\equiv M^{(p-1)(q-1)/2+1} \bmod p,\ q \text{ ist ungerade } q = 2j + 1\ j \in \mathbb{N} \\ &\equiv M^{(p-1)(2j)/2+1} \bmod p,\quad a^{(p-1)} \equiv 1 \bmod p \\ &\equiv M\ 1^{j} \bmod p \\ &\equiv M \bmod p \\ &\equiv M^{(p-1)(q-1)/2+1} \bmod q,\ p \text{ ist ungerade } p = 2k + 1\ k \in \mathbb{N} \\ &\equiv M^{(q-1)(2k)/2+1} \bmod q \\ &\equiv M\ 1^{k} \bmod q \end{aligned}$$

und damit ist auch $E(M) \equiv M^{(q-1)(p-1)/2+1} \bmod n \equiv M$ q.e.d.

Schlussfolgerung: Dieser Beweis gilt auch für $e = \dfrac{\varphi(n)}{ggT(p-1, q-1)} + 1$

Der Beweis für ggT (p-1,q-1) lässt sich schnell nachvollziehen, wenn man anstelle von 2 den ggT (p-1,q-1) einsetzt und p-1 = ggT(p-1,q-1) * j und für q-1 entsprechenden
q = ggT(p-1,q-1) * j einsetzt mit j $\in \mathbb{N}$ für geeignetes j.

Bei der Programmierung wird der Fall e = k*φ(n) + 1 ausgeschlossen, da der Bereich von e zwischen 1 und φ(n) liegt. Demzufolge braucht man nur den Fall e = φ(n)/2 + 1 auszuschließen. Man sollte außerdem darauf achten, dass der ggT(p-1,q-1) = 2 ist und nicht größer ist,

$$e = \frac{\varphi(n)}{ggT\,(p-1, (q-1)} + 1$$

um auszuschließen dass ist. [23]

2.2.5.2. Modulares Potenzieren

Das RSA – Verfahren benötigt sehr viel Rechenzeit wie allgemein alle asymmetrischen Kryptosysteme. Aber es ist möglich, an einigen Stellen Rechenzeit zu sparen, wie zum Beispiel beim modularen Potenzieren. Anstatt die Potenz M^e auszurechnen, kann folgende Rekursionsformel angewandt werden:

$M^e = M\, M^{e-1} \bmod n$ für e ungerade
$M^e = (M^{e/2} \bmod n)^2 \bmod n$ für e gerade.

Wie bereits bemerkt, wird dadurch die Rechenzeit deutlich verringert und der Aufwand kleiner. In *Mathematica* steht hierfür die Funktion *PowerMod* zur Verfügung.
PowerMod[a, b, n] für positive b ergibt dasselbe wie $a^b \bmod n$, aber es ist viel effizienter als Mod[a^b,n]. Für negative b berechnet die Funktion eine ganze Zahl k, so dass
$k\, a^{-b} \equiv 1 \bmod n$ ist, also bestimmt *PowerMod*[a, b, n] das Inverses von $a^{-b} \bmod n$. Wenn solch eine Zahl nicht existiert, gibt die Funktion *PowerMod* eine Fehlermeldung zurück.
Schon an dem nächsten Beispiel wird deutlich, dass die Funktion *PowerMod* deutlich weniger Zeit benötigt als die Funktion *Mod*.[24]

In[1]: = Mod[2222^20000,77777777]//Timing
Out[1] = {3.84 Second,45813196}
In[2]:= PowerMod[2222,20000,77777777]//Timing
Out[2]= {0. Second,45813196}

[23] Beispiel siehe Kapitel 3
[24] Wolfram Research *Mathematica Book*

2.3. Beispiele

Mit Hilfe von Beispielen mit relativ kleinen, für die Praxis ungeeigneten, p und q möchte ich das RSA – Verfahren praktisch darstellen.

2.3.1. Erstes Beispiel

p = 3
q = 11
n = 33
φ(n) = 20
d = 7
e = 3

Die Blockgröße i mit $10^i < n < 10^{i+1}$ ist 1 für n = 33. Die Zahl 22+33j wird verschlüsselt mit j = 0, 1, 2, 3,... .

j	0	1	2	3
Zahl	22	55	88	121
$C = Zahl^e \bmod n$	22	22	22	22
$Zahl = C^d \bmod n$	22	22	22	22

Tabelle 7: Beispiel für Verwendung von einer bestimmten Blocklänge.

Am Beispiel können wir sehen, dass man nur für Null den richtigen Originaltext erhält, deshalb ist es wichtig, dass die Blocklänge kleiner als n ist.
Jetzt unterstellen wir einen Originaltext im Intervall von [1,33] und schließen Zahlen aus, bei denen der ggT(M, n) > 1 ist. Im Allgemeinen gilt: wenn ggT(M, n) > 1 ist, dann können wir n durch die Berechnung des ggT von n und der verschlüsselten Version von M faktorisieren.[3]

M = 24, C = 30
Mit Hilfe des Euklidischen Algorithmus erhält man:
ggT(33,30) = 3 = p.
Im Allgemeinen ist die Wahrscheinlichkeit, dass der Originaltext einen gemeinsamen, nichttrivialen Faktor mit n hat, ungefähr 1/p+1/q. Somit ist die Wahrscheinlichkeit für große p und q's sehr gering.[3]

Originaltext	Geheimtext	Originaltext	Geheimtext	Originaltext	Geheimtext
1	1	13	19	25	16
2	8	14	5	26	20
4	31	16	4	28	7
5	26	17	29	29	2
7	13	19	28	31	25
8	17	20	14	32	32
10	10	23	23		

Tabelle8: Verschlüsselungstabelle

Wenn die Originaltextblöcke sehr klein sind, ist es sehr einfach die Entschlüsselungstabelle aufzustellen, indem man alle in Frage kommende Texte verschlüsselt und sie in einer günstigen alphabetischen Ordnung aufschreibt. Darum ist es bei öffentlichen Verschlüsselungsverfahren notwendig, die Originaltextblöcke sehr groß zu wählen.

Geheimtext	Originaltext	Geheimtext	Originaltext	Geheimtext	Originaltext
1	1	13	7	25	31
2	29	14	20	26	5
4	16	16	25	28	19
5	14	17	8	29	17
7	28	19	13	31	4
8	2	20	26	32	32
10	10	23	23		

Tabelle 9: Entschlüsselungstabelle

Weiterhin ist ersichtlich, dass einige Originaltexte in sich selbst verschlüsselt sind. Dies ist eine Besonderheit, die jedes RSA – Kryptosystem besitzt. Es gibt mindestens vier solcher Textblöcke. Im Beispiel sind das M = 1, 10, 23, 32. Sie genügen den Bedingungen $E(M) = M$ und $ggT(M, n) = 1$.
Das lässt sich mit Hilfe des Chinesischen Restsatzes beweisen.
Unter der Annahme, dass a und b unabhängig voneinander die Werte ±1 annehmen mit $M \bmod p = a$ und $M \bmod q = b$, erhalten wir die vier Zahlen, die der Bedingung $(M^e \bmod n) = M$ genügen. Die vier Zahlen M = 1, 10, 23, 32 korrespondieren in dieser Reihenfolge zu den Paaren (a, b): (1, 1), (1, -1),
(-1, 1), (-1, -1):

$1 \bmod p = 1$ und $1 \bmod q = 1$
$10 \bmod p = 1$ und $10 \bmod q = -1$
$23 \bmod p = -1$ und $23 \bmod q = 1$
$32 \bmod p = -1$ und $32 \bmod q = -1$.

Wird die Bedingung $ggT(M, n) = 1$ ignoriert, gibt es mindestens neun Zahlen, für die $M = E(M)$ ist. Das sind die gleichen Zahlen wie vorher, nur ist jetzt auch Null ein möglicher Wert für a und b. In unserem Beispiel kommen die Zahlen 0, 11, 12, 21, 22 hinzu. Weiterhin ist leicht ersichtlich, dass $(1^e \bmod p) = (1^e \bmod q) = 1$ und
$(p - 1)^e \bmod p = p - 1$, $(q - 1)^e \bmod q = q - 1$. Dabei stellt man fest, dass die letzten beiden Gleichungen eine Konsequenz aus der Tatsache sind, dass e immer ungerade ist.[3]

2.3.2. Zweites Beispiel

Jetzt wird der Text: “San Francisco am Goldenen Tor ist nach der Meinung vieler ihrer Besucher die schoenste Stadt der Welt.“ verschlüsselt. Als erstes wird der Text codiert mit Hilfe des ASCII – Codes, wie in der nachfolgenden Tabelle zu sehen ist.

p = 899370821
q = 701750353
n = 631133791114649813
φ(n) = 631133789513528640
d = 378893791
e = 398152180221563551

Die Blocklänge beträgt i = 17 mit $10^{17} < n < 10^{18}$. Diese Blocklänge muss noch durch drei geteilt werden, da jedes Zeichen durch eine maximal dreistellige Zahl codiert wird. Somit ergibt sich die Blocklänge fünf.

Originaltext	Codierter Text	Geheimtext
San F	83971103270	537802089189950280
ranci	1149711099105	373428272613689705
sco a	115991113297	153836636468353897

m Gol	1093271111108	162203989521754683
dende	100101110100101	390290592656974590
n Tor	1103284111114	12756754969188561
ist	3210511511632	313187211643781929
nach	110979910432	512819467318038584
der M	1001011143277	127683061986508766
einun	101105110117110	228203431461593610
g vie	10332118105101	132854028055400809
ler i	10810111432105	150688908798072136
hrer	10411410111432	4976846796381696
Besuc	6610111511799	596135039970822155
her d	10410111432100	29788324894514185
ie sc	1051013211599	288561774633793409
hoens	104111101110115	401752415614519339
te St	1161013283116	164292328543521579
adt d	9710011632100	390130616568517375
er We	1011143287101	128881162625523704
lt.	10811646	493583342885699785

Tabelle 10: Beispiel Verschlüsselung

2.4. Zusammenfassung

1. Das RSA – Verfahren verwirklicht alle Anforderungen, die an eine Einwegfunktion gestellt werden.
2. Bei gegebenen öffentlichen Schlüsseln e ist die Verschlüsselungsfunktion E(M) leicht berechenbar.
3. Wenn lediglich der Geheimtext C gegeben ist, ist es rechnerisch unmöglich, M zu bestimmen.
4. Ist der Geheimtext C und der private Schlüssel d gegeben, ist es leicht, M zu berechnen.

Die zweite Eigenschaft der Einwegfunktion wird dadurch gewährleistet, dass die Faktorisierung von n sehr schwierig ist, da das Produkt von n aus nur zwei Primzahlen gebildet wird.[3]

2.5. Schwächen des RSA – Verfahrens und Gegenmaßnahmen

2.5.1. Kryptoanalyse

Dieser Abschnitt steht unter dem Gesichtspunkt der Kryptoanalyse und ihrer Methoden. Die Kryptoanalyse wird auch als dunkle Seite der Kryptologie bezeichnet oder als „illegale Welt“. Aber eigentlich ist es nur eine andere Disziplin, die Wissenschaft vom unautorisierten Entschlüsseln, Verletzen von Authentifikationsschemata und im Allgemeinen das Zerstören von kryptographischen Protokollen.
Sie ist genauso wichtig wie die Kryptographie, denn ohne die Kryptoanalyse wäre auch die Kryptographie nicht notwendig. Wenn niemand das Geheimnis wissen möchte, brauche ich es auch nicht zu verschlüsseln.
Ein anderer Aspekt ist, dass die Kryptoanalyse auch dafür sorgt, immer neue kryptographische Methoden zu entwickeln werden und somit sich die Kryptographie weiter verändert.
Des Weiteren sind kryptoanalytische Methoden notwendig, um zu überprüfen, wie sicher ein kryptographisches System ist, und um mögliche Schwächen zu korrigieren.

Die verschiedenen Methoden in der Kryptoanalyse zur Berechnung eines Kryptosystems werden auch als Angriffe bezeichnet. Einige dieser Angriffe sind allgemein, andere wiederum werden nur auf bestimmte kryptographische Verfahren angewendet.
Der Kryptoanalytiker ist bestrebt, entweder einen neuen Geheimtext zu entschlüsseln oder im Idealfall den Schlüssel zu finden. Dafür können ihm unterschiedlich viele Informationen zur Verfügung stehen.

2.5.2. Angriffsmöglichkeiten[4]

1. Ciphertext – Only – Angriff. Nur der verschlüsselte Text mehrerer Nachrichten liegt vor, die mit dem gleichen Schlüssel verschlüsselt wurden. Dies ist der allgemeine Fall. Der Kryptoanalytiker versucht mit Hilfe der Geheimtexte die Originaltexte wiederherzustellen oder besser noch, den oder die zur Entschlüsselung der Nachricht verwendeten Schlüssel abzuleiten, um damit weitere, mit demselben Schlüssel verschlüsselte Nachrichten zu entschlüsseln. Dies ist relativ schwierig und es setzt voraus, dass der Originaltext sehr lang ist oder viele Geheimtexte vorliegen.
2. Known – Plaintext – Angriff. Der verschlüsselte sowie der Originaltext liegen vor. Hier besteht nun die Aufgabe des Angreifers darin, den oder die zu Verschlüsselung verwendeten Schlüssel herauszubekommen oder einen Algorithmus zu finden, mit dem weitere Nachrichten entschlüsselt werden können, die mit denselben Schlüsseln verschlüsselt wurden.
3. Chosen – Plaintext – Angriff. Der Angreifer kann einen Originaltext wählen und erhält den dazugehörigen verschlüsselten Text. Somit verfügt er nicht nur über den Geheimtext und den Originaltext, sondern er kann darüber hinaus den zu verschlüsselnden Originaltext selbst festlegen. Damit hat er bessere Voraussetzungen als bei dem Known – Plaintext – Angriff, da er gezielt spezielle Originaltextblöcke auswählen kann, die zu weiteren Informationen über den Schlüssel führen können.
4. Adaptive – Chosen – Plaintext – Angriff. Dies ist eine andere Variante zum Chosen – Plaintext – Angriff. Dabei kann der Kryptoanalytiker nicht nur den Originaltext auswählen, sondern seine Auswahl auch variieren, indem er die Ergebnisse der vorangehenden Verschlüsselung berücksichtigt. Der Unterschied zum Chosen – Plaintext – Angriff ist, dass er bei dem Chosen – Plaintext – Angriff bestenfalls einen relativ großen Block Originaltext verwenden kann, hier dagegen kann er erst einen kleineren Originaltextblock verschlüsseln. Auf den Ergebnissen aufbauend kann er einen zweiten auswählen und verschlüsseln.
5. Chosen – Ciphertext – Angriff. Eine andere Möglichkeit besteht darin, dass der Kryptoanalytiker verschiedene Geheimtexte zur Entschlüsselung auswählen kann und dass er auch Zugriff auf den entschlüsselten Originaltext hat. Beispielsweise könnte er Zugang zu einem einbruchsicheren Apparat besitzen, der automatisch Entschlüsselungen durchführt. Diese Methode wird meistens bei asymmetrischen kryptographischen Verfahren angewendet. Dabei gibt es noch die Variante, Chosen – Plaintext – Angriff mit Chosen – Ciphertext – Angriff zu kombinieren.
6. Man – In –The – Middle – Attacke. Der Angreifer tritt zwischen beide Kommunikationspartner und spiegelt ihnen die Identität des jeweils anderen vor.
7. Chosen – Key – Angriff. Der Kryptoanalytiker besitzt bestimmte Kenntnisse über die Zusammenhänge zwischen verschiedenen Schlüsseln. Er kann mit diesen Informationen den Schlüssel selbst aber nicht herleiten. Diese Art des Angriffes spielt in der Praxis keine große Rolle.
8. Differentielle Kryptoanalyse. Dieser Angriff ist eine Art Chosen – Plaintext – Attacke, bei der zwischen mehreren Originaltexten Beziehungen bestehen. Der Angreifer versucht,

diese Beziehungen auch in den verschlüsselten Texten zu identifizieren und Rückschlüsse zu ziehen.[26]

Eine Anwendung des Known – Plaintext – Angriffs ist das Verfahren der allumfassenden Schlüsselsuche. Hier wird jeder Schlüssel ausprobiert, bis der Richtige gefunden wurde. Dabei wird folgendermaßen vorgegangen: Der verschlüsselte Text wird mit Hilfe eines beliebigen Schlüssels entschlüsselt und der entstandene Text mit dem Originaltext verglichen. Stimmt der Text mit dem Originaltext überein, ist der richtige Schlüssel bestimmt. Ist das nicht der Fall, wird ein neuer Schlüssel ausgewählt, bis der richtige Originaltext ermittelt wurde. Das Verfahren wird umso effizienter, je mehr man den Bereich der möglichen Schlüssel einschränken kann. Es ist logisch, je schneller die Computer werden und die Technologie sich verbessert, umso schneller wird die Schlüsselsuche.
Eine Möglichkeit, die Suche effizienter zu gestalten, hat sich mit der Entwicklung des Internets ergeben. Es ist nun möglich, Tausende von Computern an der Suche zu beteiligen. Dabei wird der Bereich des Schlüssels aufgeteilt, und jeder Computer durchsucht nur noch seinen Teilbereich. Folgendes Ergebnis konnte bei einem solchen Zusammenschluss von 50 000 Computern erzielt werden: 85 Prozent der möglichen Schlüssel wurden in 39 Tagen durchsucht. Damit zeigt sich, dass die allumfassende Schlüsselsuche ein wirksamer Angriff auf kryptographische Verfahren ist.

Bemerkung: Ein verschlüsselter Quellcode, zum Beispiel von Programmen, ist besonders gefährdet gegen kryptoanalytische Angriffe, da immer wieder dieselben Schlüsselwörter auftauchen wie zum Beispiel: if, else, return.

2.5.3. Kryptoanalytische Angriffe auf das RSA – Verfahren

In diesem Abschnitt gehe ich auf einige spezielle Angriffe auf das RSA – Verfahren ein. Außerdem beschäftige ich mich noch mit Angriffen auf die Implementierung des RSA – Verfahrens.

2.5.3.1. Angriff bei kleinen und gleichen Verschlüsselungsexponenten e

Eine Möglichkeit das RSA – Verfahren „zu brechen“, ist die e –te Wurzel mod n zu berechnen. Weil $C = M^e \bmod n$ ist, ist die e –te Wurzel von C mod n die Originalnachricht M. Dieses Verfahren würde die Entschlüsselung des Geheimtextes erlauben ohne Kenntnis des privaten Schlüssels. Leider ist bisher kein Verfahren gefunden worden, mit dem man die e –te Wurzel bestimmen kann. Ein spezielles Verfahren kann angewendet werden, wenn e sehr klein ist. Dies möchte ich an einem Beispiel verdeutlichen. Drei Personen A, B, C besitzen alle den öffentlichen Schlüssel e = 3 und n_A, n_B, n_C. Weiterhin gilt n_A, n_B, n_C sind zueinander teilerfremd. Es wird somit die Nachricht

$$M^3 \bmod n_i,\ i = A, B, C$$

übermittelt.
Wenn ein Kryptoanalytiker die Nachricht abfängt, kann er die Zahl

$$M_1 = M^3 \bmod n_A\, n_B\, n_C$$

mit Hilfe des Chinesischen Restsatzes[25] folgendermaßen berechnen:

$$M_A = M^3 \bmod n_A$$
$$M_B = M^3 \bmod n_B$$
$$M_C = M^3 \bmod n_C$$

[25] Siehe Kapitel 2.2.1.

$M^3 \equiv M_A \bmod n_A$
$M^3 \equiv M_B \bmod n_B$
$M^3 \equiv M_C \bmod n_C$

$n = n_A * n_B * n_C$ $\quad w_A = n/n_A = n_B * n_C$
$w_B = n_A * n_C$
$wc = n_A * n_B$

$ggT(w_A, w_B, w_C) = 1$ da n_A, n_B, n_C zueinander teilerfremd sind.

$w_A\, g_A + w_B\, g_B + w_C\, g_C = 1$
$M_1 = M_A\, w_A\, g_A + M_B\, w_B\, g_B + M_C\, w_C\, g_C \bmod n$
g_A, g_B, g_C sind die jeweiligen Inversen von w_A, w_B, w_C.

Beispiel: $n_A = 493$, $n_B = 649$, $n_C = 703$ und die drei Nachrichten sind $M_A = 337$, $M_B = 533$, $M_C = 259$. Um die dazugehörigen Inversen $w_i^{-1} \bmod n_i$, i = A, B, C zu berechnen, benötigen wir die Produkte der jeweils anderen Moduli.

$w_c = n_A * n_B = 319957$,
$w_A = n_B * n_C = 456247$
$w_B = n_A * n_c = 346579$

Die dazugehörigen Inversen sind:

$w^{-1}{}_A = 171$,
$w^{-1}{}_B = 50$,
$w^{-1}{}_C = 405$.

Damit ist $M_1 \equiv 337\, w_A\, w^{-1}{}_A + 533\, w_B\, w^{-1}{}_B + 259\, w_C\, w^{-1}{}_C \bmod n_A, n_B, n_C = 36926037$. Wird nun die dritte Wurzel aus M^3 gezogen, ergibt es den Originaltext M = 333.

2.5.3.2. p und q als Angriffspunkt

Bei der Wahl von p und q sollte folgendes beachtet werden: Die beiden Primzahlen p und q müssen zufällig ausgewählt werden. Sie dürfen nicht aus einer Tabelle stammen und nicht zu nah aufeinander folgen, sonst lässt sich nämlich n leicht faktorisieren, und zwar folgendermaßen:
Ist p größer q, dann ist $(p - q)/2$ klein und $(p + q)/2$ ist nur etwas größer als die Wurzel aus n. Weiterhin gilt: $(p + q)^2/4 - n = (p - q)^2/4$, damit ist die linke Seite quadratisch. Die Zahl n faktorisiert man nun, indem $x > \sqrt{n}$ getestet wird, bis man $x^2 - n$ als Quadratzahl findet. Dies ist gleich y^2. Somit ist $p = x + y$ und $q = x - y$.
Beispiel: $n = 176399 \quad \sqrt{n} = 419{,}99$
$420^2 - n = 1 \quad x = 420$
$y = 1$ und damit ist $q = 419$ und $p = 421$

Bemerkung: Im Normalfall ist es von Vorteil, wenn sich die Bit – Darstellung von q und p in der Länge um einige Bits unterscheiden.[3]

„Sichere“ Primzahlen
Um einige Angriffsmöglichkeiten zu verhindern, können für p und q „sichere“ Primzahlen verwendet werden. Eine Primzahl p und q ist „sicher“, wenn:

- Der $ggT((p - 1),(q - 1))$ klein ist im Vergleich zu p und q.
- $p - 1$ als auch $q - 1$ große Primfaktoren p_1‘,p_2’ und q_1‘,q_2’ besitzen und somit $(p - 1) = (p_1' - 1)(p_2' - 1)$ und $(q - 1) = (q_1' - 1)(q_2' - 1)$ ist.
- $p_1' - 1$, $p_2' - 1$ als auch $q_1' - 1$, $q_2' - 1$ große Primfaktoren besitzen.
- $(p - 1)/2$ als auch $(q - 1)/2$ prim sind.[4]

„Sichere“ Primzahlen sind viel schwieriger zu erzeugen als „normale“ Primzahlen. Es wurde aber noch nicht bewiesen, dass es unendlich viele „sichere“ Primzahlen gibt.
Beispiele für „sichere“ Primzahlen sind 83, 107 und $10^{100} - 166517$.[3]
Wenn „sichere“ Primzahlen verwendet werden, ist als Angriffsmöglichkeit ausgeschlossen, dass d einfach nur durch Testen gefunden wird. Dies ist möglich, wenn der ggT(p – 1,q – 1) groß und mindestens ein gemeinsames Vielfaches u von p – 1 und q – 1 klein im Vergleich mit φ(n) ist. Jetzt kann jedes Inverses von e mod u als Schlüssel verwendet werden. $1 \equiv e \bmod u$, Beweis siehe Abschnitt „Warum müssen p und q Primzahlen sein?“

Dieser Sachverhalt gilt, weil p – 1 und q – 1 gerade sind und damit φ(n) durch 4 teilbar ist, dies ist stets der Fall. Denn p und q sind ungerade Primzahlen, daraus folgt, dass p-1 und q-1 gerade sind. Eine gerade Zahl kann man auch in der Form
p-1=2n schreiben mit $n \in N$. Somit ist φ(n)= 2n*2m=4 nm und damit durch vier teilbar.

Beispiel: Eine Möglichkeit ist, dass einer der beiden Faktoren p – 1 und q – 1, sagen wir p – 1, den anderen teilt, mit p = 61, q = 181. Jetzt wird nur noch das Inverse von e mod (p – 1) betrachtet. Das n = 11041 und das e = 4013 kann nun jedes Inverse von 4013 mod 180 als Entschlüsselungsexponent benutzen, denn das kleinste gemeinsame Vielfache von p – 1 und q – 1 ist 180, somit ist d = 17.
Ein anderes Problem, was durch „sichere“ Primzahlen gelöst werden kann, tritt auf, wenn φ(n) nur sehr kleine Primfaktoren hat. Nehmen wir an, dass alle Primfaktoren r von φ(n) kleiner sind als eine ganze Zahl k. Somit ist $[\log_r n]$ der Exponent der höchsten Potenz von r. Dies teilt möglicherweise φ(n). Nun können alle Kandidaten v für φ(n) berechnet werden und an den verschlüsselten Texten so getestet werden, dass $C^{(v+1)/e}$ gleich der codierten original Nachricht M ist. Voraussetzung dafür ist, dass der Exponent eine ganze Zahl ist.
Eine weitere Möglichkeit, den Originaltext zu finden ist, wenn bei der Wahl von p und q nicht beachtet wurde, dass p – 1 und q – 1 keine großen Primfaktoren p’ und q’ haben und auch p’ – 1 und q’ – 1 wieder keine großen Primfaktoren besitzen, also keine „sicheren“ Primzahlen sind. Ansonsten kann der Originaltext mit Hilfe von iterativem Entschlüsseln gefunden werden.
Bei dieser Art der Entschlüsselung beginnt man mit dem Geheimtext C_0 und berechnet so lange $C_i = (C_{i-1}^{e} \bmod n)$, i = 1,2,..., bis ein sinnvoller Originaltext $C_i = M$ gefunden wurde. Diese Form des Entschlüsseln ist aber unbedeutsam, wenn Obiges beachtet wird.[3]

Bemerkung: Ob „sichere“ Primzahlen nötig sind, ist noch in der Diskussion. Die genannten Eigenschaften sollen den Einsatz einiger älterer Faktorisierungsalgorithmen verhindern. Die schnellsten Faktorisierungsalgorithmen zerlegen jedoch solche Zahlen, die diese Kriterien erfüllen, genauso gut wie solche, die sie nicht erfüllen.
Dem Erzeugen von „sicheren“ Primzahlen ist abzuraten, denn die Länge der Primzahlen ist viel wichtiger als deren Struktur. Die Struktur könnte sogar Schaden anrichten, da sie nicht zufällig ist.[26]

2.5.3.3. Warum müssen p und q Primzahlen sein?

Im RSA – Verfahren ist festgelegt, dass p und q Primzahlen sein müssen. Dabei stellt sich die Frage, was passiert, wenn dies nicht der Fall ist. Dafür nehmen wir an, dass p zusammengesetzt ist. $p = p_1 p_2$ mit q, p_1 und p_2 als Primzahlen. Jetzt wird angenommen, dass p auch eine Primzahl ist. Man arbeitet deshalb mit einem falschen
φ*(n) = (p – 1) (q – 1). Das richtige φ(n) wird folgendermaßen berechnet
φ(n) = (p_1 – 1) (p_2 – 1) (q – 1). u ist das kleinste gemeinsame Vielfache von $p_1 - 1$, $p_2 - 1$ und q – 1 (u = kgV($p_1 - 1$, $p_2 - 1$,q – 1)). Unter der Annahme, dass ggT (M, n) = 1, gelten die Kongruenzen $M^{p1-1} \equiv 1 \bmod p_1$,

$M^{p2-1} \equiv 1 \bmod p_2$ und $M^{q-1} \equiv 1 \bmod q$. Dies ist bewiesen durch den Satz von Euler, und es folgt: $M^u \equiv 1$ gilt für alle drei Moduli. Damit ist $M^u \equiv 1 \bmod n$. Natürlich teilt u $\varphi(n)$. Jetzt unterscheidet man zwei Fälle.
Erster Fall: u teilt auch $\varphi^*(n)$, dann ist $M^{\varphi^*(n)+1} \equiv M \bmod n$. Das bedeutet, dass die Ver- und Entschlüsselung genau so funktioniert, als wäre p eine Primzahl. Dieser Fall ist der seltenere der beiden Fälle.

16. Satz: Es sei $n \in \mathbb{N}$, $n > 1$; es gelte $n = p_1 p_2 \ldots p_r$ mit p, Primzahl. Setzt man $u = kgV(\varphi(p_1),\varphi(p_2),\ldots,\varphi(p_r)) \in \mathbb{N}$, so gilt $a^u \equiv 1 \bmod p$ für alle $a \in \mathbb{N}$ mit $ggT(a, n) = 1$.

Beweis: Da $ggT(a, p_i) = 1$ für alle i =1,2,...,r, so gilt $a^{\varphi(pi)} \equiv 1 \bmod p_i$ für alle i nach dem Satz von Fermat – Euler. Da u ein Vielfaches von $\varphi(p_i)$ ist, folgt: $a^u \equiv 1 \bmod p_i$ für alle i.
Aus dem 9. Satz ergibt sich somit $a^u \equiv 1 \bmod n$. q.e.d.

1. Fall u | $\varphi^*(n)$. Dann ist $u\,k = \varphi^*(n)$ mit $k \in \mathbb{N}$ und $u = kgV(p_1\text{-}1,p_2\text{-}1,q\text{-}1)$ und $M \equiv M^{\varphi^*(n)+1} \bmod n$

Beweis:

$$M^{\varphi^*(n)+1} \bmod n$$
$$= M\, M^{\varphi^*(n)} \bmod n$$
$$= M\, M^{ku} \bmod n$$
$M^u \equiv 1 \bmod n$ nach den obigem Beweis

und es gilt nach dem Beweis aus Kapitel 2.2.5.1.

$$M\, M^{ku} \bmod n = M.$$ q.e.d.

Beispiel:

p	=	65
p_1	=	5
p_2	-=	13
q	=	31
n	=	2015
$\varphi^*(n)$	=	1920
$\varphi(n)$	=	1440
u	=	$kgV(q-1,p_1-1,p_2-1) = 60$
d	=	19
e	=	1819
M	=	444

60 teilt $\varphi_1(n)$, daraus folgt: wenn $ggT(d,\varphi^*(n)) = 1$ ist auch $ggT(d,\varphi(n)) = 1$. Wenn der Systemdesigner jetzt das falsche $\varphi^*(n)$ benutzt, um e zu berechnen, hat dies keine weiteren Auswirkungen. Es gilt die Gleichung D(E(M)) = M. Es treten auch keine zusätzlichen Sicherheitsrisiken auf.

$$M^e \bmod n = 444^{1819} \bmod n = 1909 = C.$$

Wendet man die Entschlüsselungsfunktion an:

$$C^d \bmod n = 1909^{19} = 444 = M.$$

2. Fall u ⊀ $\varphi^*(n)$, damit gilt $u\,k \neq \varphi^*(n)$ und i. a. $M^{\varphi^*(n)+1} \bmod n \neq M$.
Beweis (anhand eines Beispieles): p = 87, q = 17, p_1 = 29, p_2 = 3, $\varphi(n)$ = 896, $\varphi^*(n)$ = 1376, e = 359, d = 23, u = 112 u teilt $\varphi(n)$ aber nicht $\varphi^*(n)$.
Verschlüsselung: M = 444 E(M) = 876 = C
Entschlüsselung: D(C) = 138 ≠ M.

Die Gleichung $D(E(M)) \neq M$, damit ist die Ver- und Entschlüsselung nicht eindeutig.

q.e.d.

Mit diesen beiden Beweisen ist die am Anfang gestellte Frage, „Warum müssen p und q Primzahlen sein?“, beantwortet.

2.5.3.4. n als Angriffspunkt

Einige Implementierungen von RSA verwenden immer dasselbe n, aber dann natürlich unterschiedliche Exponenten d und e. Das Problem dabei betrifft eine Nachricht, die mit zwei verschiedenen Exponenten verschlüsselt wurde (jeweils mit demselben Modul). Im allgemeinen sind die beiden Exponenten relativ prim zueinander. In diesem Fall kann man den Originaltext bestimmen, ohne die beiden privaten Schlüssel zu kennen.
Die beiden öffentlichen Schlüssel sind e_1 und e_2. Das gemeinsame Modul sei n. Damit lauten die beiden Geheimnachrichten:

$$C_1 = M^{e1} \bmod n$$
$$C_2 = M^{e2} \bmod n.$$

Da e_1 und e_2 relativ prim zueinander sind, findet man mit dem erweiterten Euklidischen Algorithmus r und s mit:

$$r\, e_1 + s\, e_2 = 1$$

Da entweder r oder s negativ sind, legen wir einfach fest, dass r negativ ist. Dann kann man wieder mit Hilfe des erweiterten Euklidischen Algorithmus $C_1^{-1} \bmod n$ berechnen.[4] Dann gilt:

$$(C_1^{-1})^{-r} * C_2^{s}$$
$$= (M^{-e1} \bmod n)^{-r}\, (M^{e2} \bmod n)^{s}$$
$$= M^{e1\, r} * M^{e2\, s} \bmod n$$
$$= M^{e1\, r+e2\, s} \bmod n \text{ (mit } r\, e_1 + s\, e_2 = 1)$$
$$= M.$$

Beispiel:

M = 780970991041141050991041 16 ist „Nachricht“ codiert.

p = 8627670385611922429699807516464917407647534422718134975058673561255291540022072607658918 31

q = 2426914857583008085751817348633975679317108411096921012697746533939789836458629406798 03849

n = 2093862144517049517698325163485543815286158710088205580527168365796271065729431754669182018349105283644157888054304766413823290432151515113665340612631958146072316038989916265257519

φ(n) = 2093862144517049517698325163485543815286158710088205580527168365796271065729431754669182007294520040449227372602679901314930203467508681298609352856211862950990939558197148197618 40

d_1 = 7576801293547127894343788968795986211881164807778303719802336136340 59

e_1 = 1647136848695322425565495007393900722168094430172149433556672449781750952272506988660368405158437722219889787523225581720203041511569115280253892234596568571328276469218723964753 9

d_2 = 6754181160654778039829897921589535838618949188220219405302670755279 509

e_2 = 8700241099969039138866967846746057914356833032665033647641857544351295309983551759193576226967223093570594063464970948123593876258428900394239574246795400178921223730380413850882 9

C_1^{-1} = 6321962374861864508686116554354269255007177257499644090708657765474326409391199342817759424362339616745685846378556210857861577776973 6

C_2 = 594165773257384856946712611996824724417159024
618403365847843902040604610694882836122466102360688188145934707528801614876622124220621094474425422939146277313367606380058565790054212788922202654712838374924649949104233377477187

r = -2055515758730572899140767597223323642562952441150462579724486665941982678831334940015507573008067134511001188870174616418443615857993961264405348467622672544979366415877799968460 8

s = 389151945374375775495919530324904743271374504456391783267038683976865035150816232791174243222760955616715437693474239793661405228446857246174948947821515231416257937569298076779 7

e_1*r+*e_2s = 1

$(C_1^{-1})^{-r}$ *C_2^s= 78097099104114105099104116

Eine andere Möglichkeit besteht darin, dass man annimmt, dass zwei Nutzer Alice und Bob mit dem gleichen n = pq arbeiten. Dann kann Bob die Nachrichten von Alice lesen und umgekehrt. Bob kennt Alice's öffentlichen Schlüssel e_A und natürlich seine öffentlichen Schlüssel e_B und privaten Schlüssel d_B. Damit kann er nun folgendes berechnen:
$(e_B\, d_B - 1)e_A$= kgV $(e_A, e_B d_B - 1)$ ggT$(e_A, e_B d_B - 1)$ durch e_A und ggT$(e_A, e_B d_B - 1)$ geteilt liefert

$$h = \frac{e_B\, d_B - 1}{ggT(e_A, e_B d_B - 1)} = \frac{kgV(e_A, e_B d_B - 1)}{e_A}$$

Die Zahl h ist ein Vielfaches von $\varphi(n) = (p-1)(q-1)$, denn $(e_B\, d_B - 1) = \varphi(n) * k$ somit ist $\varphi(n) = h / k * ggT(e_A, e_B d_B - 1)$. Weiterhin ist h teilerfremd zu e_A, denn

$$h = \frac{kgV\,(e_A, e_B d_B - 1)}{e_A} \; ; \; kgV\,(e_A, e_B d_B - 1) = c * e_A$$

Somit kann man mit Hilfe des erweiterten Euklidischen Algorithmus die Linearkombination c h + d e_A = 1 berechnen. Da c*h mod $\varphi(n) = 0$, weil h ein Vielfaches von $\varphi(n)$ ist, hat Bob die Zahl d gefunden, die d e_A mod $\varphi(n)=1$ erfüllt. Somit kann er alle Nachrichten für Alice entschlüsseln, indem er sie mit d potenziert.[1]

2.5.3.5. Brechen von RSA in PKCS#1 [24]

Zuerst möchte ich klären, was PKCS oder Public Key Cryptography Standards sind. Es ist eine Zusammenfassung von Standards, konstruiert für die Durchführung der Implementierung von Kryptographie. Der Standard deckt ein breites Feld ab, einschließlich RSA – Verschlüsselungen, Smart Cards und Schlüsselverwaltung. Der PKCS wurde etabliert von einer Gruppe von akademischen und wirtschaftlichen Organisationen und wurde herausgegeben von RSA Laboratories.
PKCS #1 beschreibt eine Methode für die RSA – Verschlüsselung. Weiterhin umfasst PKCS #1 Methode und Syntax zum Ausführen von Operationen, die digitale Unterschriften erstellen, sowie eine Syntax für die Beschreibung öffentlicher und privater Schlüssel.[25]

Die Angriffsmöglichkeit auf PKCS#1 beruht auf einem theoretischen kryptographischen Ergebnis, was zu seiner Zeit wenig Aufsehen erregte. Es wurde bewiesen, dass jede Nachricht, die von RSA verschlüsselt wird, so sicher ist wie jedes einzelne Bit der Nachricht.

Dies ist nicht weiter spektakulär, nur lässt sich damit auch beweisen, dass man, wenn man ein Bit der RSA Nachricht entschlüsselt, auch die ganze Nachricht entschlüsseln kann. Auf diesem Ergebnis baut die nächste Angriffsmethode auf.
Sie ist vor allem auf PKCS#1 oder ähnlich arbeitenden Protokollen anwendbar. Es ist kein allgemeiner Angriff auf das RSA – Verfahren, sondern funktioniert nur unter bestimmten Voraussetzungen.

Der Angriff ist sehr einfach zu beschreiben. Der Angreifer möchte den Originaltext eines bestimmten mit dem RSA – Verfahren verschlüsselten Textes herausfinden. Er sendet ca. eine Million ähnliche Nachrichten an das System und beobachtet die Reaktionen. Aus den Reaktionen kann man Schlussfolgerungen ziehen, ob die Nachricht mit einem speziellen Datenformat übereinstimmt, danach ist es möglich mit einigen einfachen mathematischen Analysen die Nachricht zu entschlüsseln. Dabei sind einige Punkte zu beachten:
Erstens: Der Angreifer findet nicht den geheimen Schlüssel, sondern nur den Originaltext zu einer speziellen Nachricht. Das bedeutet, nachdem der Angreifer eine Million Nachrichten gesendet und die Reaktion darauf ausgewertet hat, kann nur eine Nachricht gelesen werden. Wenn eine weitere geheime Nachricht gelesen werden soll, müssen wieder eine Million ähnliche Nachrichten versendet und ausgewertet werden.
Zweitens: Der Angreifer ist abhängig von Informationen, die das angegriffene System zurück gibt. In diesem Fall muss er wissen, wann eine ähnliche Nachricht auf einem bestimmten Weg entschlüsselt wurde. Dies ist eine alte und effektive Idee und sehr leicht zu implementieren. Computersysteme sind gut im automatischen Erzeugen von Eingabewerten und Verarbeiten der Ergebnisse wie zum Beispiel Fehlermeldungen und Statusmeldungen, die das angegriffene System zurückgibt.
Drittens: Der Angreifer muss dem angegriffenen System sehr viele ähnliche Nachrichten zusenden, um eine Nachricht zu entschlüsseln. Für einen „normalen“ Angriff werden eine Billion Nachrichten benötigt. Diese Zahl kann verkleinert werden. Das Experiment gegen SSL[26] benötigt zum Beispiel nur 300 000 bis zu 2 Millionen ähnliche Nachrichten. Aber dies sind immer noch sehr viele Nachrichten. Dennoch: Computer können gut so viele Nachrichten verarbeiten, dass es kaum bemerkt wird.
Um solch einen Angriff zu verhindern, müssen einige Vorkehrungen getroffen werden.
Das angegriffene System sollte dem Angreifer nicht mitteilen, ob eine Nachricht gültig ist oder nicht. Die schnellste Möglichkeit diese Angriffe zu vereiteln ist, die Zahl der ähnlichen Nachrichten zu erhöhen, die benötigt werden, eine Nachricht zu entschlüsseln. Diese Methode macht es sehr viel schwerer, Angriffe gegen ein Onlinesysteme zu starten. Die Menge der Nachrichten blockiert das angegriffene System und somit kann es nicht mehr effektiv arbeiten. Anders sieht es bei Offlinesystemen[27] aus, denn dort hat diese Methode keinen Erfolg.
Eine bessere Möglichkeit ist, das PKCS#1 Protokoll selbst zu ändern, im speziellen den Teil, welcher spezifiziert, wie die Bits von Originaltext gepackt werden, so dass das RSA – Verfahren sie verschlüsseln kann. Das RSA Nachrichten – Verpackung – Schema in SET[28] zum Beispiel ist nicht anfällig für diesen Angriff.

Bemerkung: Dieser Angriff stellt keine Gefährdung des RSA – Verfahrens dar, aber er zeigt auf, welche Bedingungen ein Systemdesigner beachten muss, damit das Verfahren korrekt implementiert wird.

---

[26] Das SSL (Secure Sockets Layer) Protokoll wird benutzt, um den Zugriff auf einen Webserver und andere Systeme zu sichern
[27] Zum Beispiel Smart Card.
[28] Secure Electronic Transactions

2.5.3.6. Chosen – Ciphertext – Angriff gegen das RSA – Verfahren

Der Chosen – Ciphertext – Angriff ist im Kapitel 2.5.2. ausführlich beschrieben. Im wesentlichen umfasst der Angriff folgendes: Es stehen verschieden Geheimtexte zur Verfügung sowie die dazugehörigen entschlüsselten Originaltexte.
Dieser Angriff richtet sich nicht gegen den Algorithmus selbst, sondern gegen eine Implementierung des RSA – Algorithmus. Dabei ist zu erkennen, dass der bloße Einsatz des Verfahrens nicht genügt, sondern es kommt auch auf die Einzelheiten an.
Ich möchte diese Angriffsmöglichkeit anhand von drei Beispielen erläutern:

Beispiel 1:
Sam fängt eine Nachricht ab, die für Alice bestimmt ist. Die Nachricht ist mit Alices öffentlichem Schlüssel verschlüsselt. Sam möchte natürlich die Nachricht lesen. Mathematisch ausgedrückt sucht er $M = C^d \bmod n$.
Zuerst besorgt er sich Alices öffentlichen Schlüssel e und wählt eine Zufallszahl r aus, die kleiner als n ist. Damit berechnet Sam folgendes:

$$x = r^e \bmod n$$
$$y = x\,C \bmod n$$
$$t = r^{-1} \bmod n.$$

Wenn $x = r^e \bmod n$, dann gilt $r = x^d \bmod n$. Siehe Beweis Kapitel 2.2.5.Verschlüsselung und Entschlüsselung. Sam bringt jetzt Alice dazu, die Nachricht y zu unterschreiben, also y zu verschlüsseln. (Dabei muss bedacht werden, dass Alice die Nachricht y nie vorher gesehen hat.) Somit sendet Alice $u = y^d \bmod n$ an Sam, und dieser kann jetzt folgendes berechnen:

$tu \bmod n = r^{-1}y^d \bmod n$ da $u = y^d \bmod n$ und $t = r^{-1} \bmod n$
$= r^{-1} x^d\ C^d \bmod n$ da $y = x\,C \bmod n$
$= r^{-1} r^{ed} C^d \bmod n$ da $x = r^e \bmod n$
$= r^{-1}r\ \ C^d \bmod n$
$= \quad C^d \bmod n = M$. Jetzt kennt Sam die Originalnachricht M.[4]

Jetzt möchte ich diesen Angriff noch anhand eines Zahlenbeispiels verdeutlichen[29].
Die abgefangene Nachricht C lautet:

126194074993427084863773428940781074024456775180774417128437013546106958817
865440717420057935684487537652544754039010303287321869219465303570026834744
2391660666160001199403986602 1
Alices öffentlicher Schlüssel ist:
n = 171269513391354180262576808969972446515146123624332483010678987722 7
804924060023892659779281946876427368931849686974069448421866925493 4
653040870989481787642390988645077284276245257
e = 777199111905751399058093375186685551514585385718581501067496886012 8
709920194204977267591676513488417507128998722158330661295303065127 9
109608559162696140727235698273711452609404 03
r wird zufällig ausgewählt. r liegt zwischen 1 und n:
r = 906551590984586261671454347687200855343992798458600518475651367479 4
182783309716243544001412028459924921173932550336214673950882940408 8
472285529408807347035805026259718017743325 06
Jetzt kann wie oben vorgegeben x, y, t berechnet werden und man erhält:
x = 149305193492627343027282101086986585134678379198486424979794878035 8
035016219379033114153079937306298545099960953361526368247499313872 6
816797485844005183408834739217011427763032118

[29] Siehe Anhang Beispiele Kapitel 2.5.3.6.

y = 160623583654653451834060819260840691064070587134976560611992441879643877885986945106664297756607696990443445298726678531660229130576197932298086258896826806618032266606855338996104

t = 148496155846725527852232634738932608281592169110067561600227503873855084166948382904813154070615936429349825215616993924201756128385562923611760696500718856431756539559698887069 37

y muss jetzt zu Alice geschickt werden, damit sie Die Nachricht verschlüsselt. Dies ist der schwierigste Teil, denn Alice verschlüsselt eine Zahlenkolonne, die ihr völlig unbekannt ist. Sam erhält das verschlüsselte y:

u =

92141151168912882346179122829991463405379654256462028478111218919404211617236983587101647788715981838429958325771335182249898177446944599858591862413588532779223750563363829974 10

Jetzt kann Sam die Originalnachricht berechnen mit

t * u mod n = M = 78097099104114105099104116.

Wenn wir das mit C^d mod n = 78097099104114105099104116 vergleichen, sehen wir, dass die Nachricht M berechnet wurde. Alices privater Schlüssel d zum Vergleich ist:

d = 505346689227534732426764607705612433220072497352070509376254356214034 7

Beispiel 2: Sam möchte, dass Bob eine Nachricht unterschreibt, die er normalerweise nicht unterschreiben würde. Diese Nachricht bezeichne ich mit M‘. Als erstes wählt Sam wieder einen beliebigen Wert x aus und berechnet:

$$y = x^e \bmod n.$$

Dabei ist e der öffentliche Schlüssel von Bob. Danach berechnet er

$$M = y\ M' \bmod n = x^e\ M' \bmod n$$

und sendet M an Bob, damit dieser die Nachricht unterschreibt. Sam erhält die unterschriebene Nachricht M^d mod n zurück. Damit kann er jetzt M‘ unterschreiben:

$$\begin{aligned}&(M^d \bmod n)\ x^{-1} \bmod n\\ &= (x^e\ M' \bmod n)\ x^{-1} \bmod n\\ &-= (x^{ed}\ M'^d \bmod n)\ x^{-1} \bmod n\\ &= M'^d \bmod n\end{aligned}$$

und stellt damit die Unterschrift von M‘ dar.

Sam nutzt dabei, dass beim Potenzieren die multiplikative Struktur erhalten bleibt.

$$(x\ M)^d \bmod n = x^d\ M^d \bmod n[4]$$

Nun auch hierzu ein kleines Zahlenbeispiel: n, e und d sind die gleichen Zahlen wie im ersten Beispiel.

M‘ = „Ich gebe Sam 1 000 000 Dollar“. Das ist die Nachricht, die Sam von Bob unterschrieben haben möchte. Codiert laute sie:

M‘ = 730991040321031010981010320830971090320490320480480320480480480320681111081080971140 46

x = 375375062862997082603607704871652902836067042500293423620061471330108288573840517816782946

y = 112509205500290009238575220124094929826787686182339643202375235361139189712530824764548596049377935445279745353917488387

074256778351766235653084301646250685521216132833798138222 26

$M = y * M' =$ 822432211743496581413440570197873749118267998986847711546052
919396066951609035336424854405814511466739886240056382535339
451845996291092765955720403468845203649903869976229412639495
514731810444469177300006892409492197884624405851342483120132
389852544137483411091586396

Bob verschlüsselt M:

$M^d \bmod n =$ 749040103393609905528078578405603266176965569714504315745406
847123061492340114514172845894414890750534151869009454680170
454054319029702776571779200144112592221485176030780264 2817

Sam braucht jetzt nur noch $(M^d \bmod n)\ x^{-1} \bmod n$ berechnen und erhält damit die Nachricht M' unterschrieben:

$M'^d_1 =$ 169684329314652342877028922453428321451965263717129999316487
507459846928616529360720168908417982992966550339403423385569
976084698767791908651241703008932118360374602200597768497492
974896492092485357956711576830203068089221913701636619436

Leicht können wir dies mit dem verschlüsselten M'^d vergleichen [30]

d = 984137983457318124836997701653649319322970646458043441477339
8271380849

$M'^d_2 =$ 169684329314652342877028922453428321451965263717129999316487
507459846928616529360720168908417982992966550339403423385569
976084698767791908651241703008932118360374602200597768497492
974896492092485357956711576830203068089221913701636619436

$M'^d_1 - M'^d_2$ 0

2.5.4. Weitere kryptoanalytische Angriffe auf das RSA – Verfahren

2.5.4.1. Faktorisierung

2.5.4.1.1. Allgemeine Faktorisierung

Unter der Faktorisierung oder der Primfaktorzerlegung einer Zahl versteht man die Bestimmung ihrer Primfaktoren:
Zum Beispiel:
$15 = 3 * 5$
$45 = 3 * 3 * 5$
$252601 = 41 * 61 * 101$
$2^{113}-1 = 3391 * 23279 * 65993 * 1868569 * 1066818132868207$[4]
Es gilt der Hauptsatz der elementaren Zahlentheorie: Jede natürliche Zahl $a > 1$ besitzt (bis auf die Reihenfolge der Faktoren) genau eine Primfaktorzerlegung $a = p_1*p_2*...*p_s$ mit den Primzahlen p_i $(i = 1,..,s)$.[31]
Die Primfaktorzerlegung ist die effektivste Methode, das RSA – Verfahren „zu brechen", wenn man zu einem öffentlichen Schlüssel den dazugehörigen privaten Schlüssel finden möchte. Der beste Weg dies zu tun ist, den öffentlichen Schlüssel n in die zwei Primzahlen p und q zu zerlegen.
Danach kann sehr einfach der private Schlüssel d mit Hilfe der Exponenten q, p und dem öffentlichen Schlüssel e berechnet werden.[32] Die Schwierigkeit dabei ist, n zu faktorisieren.

---

[30] Siehe Anhang Beispiele Kapitel 2.5.3.6.
[31] Beweis siehe [6]
[32] Siehe Kapitel 2.2.4.2.

Das Problem der Primfaktorzerlegung ist eines der Ältesten der Zahlentheorie. Die Faktorisierung einer Zahl ist im Prinzip einfach, aber sehr zeitaufwendig. Es gab zwar in dieser Beziehung schon einige Fortschritte, aber dieser Grundsatz gilt heute noch.

Bemerkung: Ich möchte in meinen Ausführungen nur die Faktorisierungsalgorithmen in Bezug auf ihre Geschwindigkeit und Bedeutung vergleichen. Weitere Ausführungen zu den Algorithmen findet man in den jeweils dazu angegebenen Büchern (siehe auch Literaturverzeichnis).

Es werden im Allgemeinen zwei Faktorisierungsalgorithmen verwendet: einmal das spezielle Zahlenkörpersieb[10] (NFS)[33] und zum anderen das mehrfache polynomielle quadratische Sieb (MPQS)[34].
NFS ist der effizientere von beiden im Faktorisieren von großen Zahlen. MPQS ist besser in der Faktorisierung von kleineren Zahlen. Zum Beispiel wurden mit MPQS Schlüssel der Länge 129 Stellen faktorisiert in 2000 MIPS, heute ist es mit NFS in einem Viertel der Zeit möglich.

2.5.4.1.2. Faktorisierungserfolge verschiedener Algorithmen

Ganzzahlige Faktorisierung:

Bessere Erfolge wurden mit Algorithmen erzielt, die spezielle Zahlen faktorisieren. Zum Beispiel mit dem speziellen Zahlenkörpersieb, welches auf die Cunningham – Zahlen angewendet wird. So wurde am 02.02.1999 eine Zahl mit 211 Stellen faktorisiert. Cunningham – Zahlen sind Zahlen der Form b^n+1 für b kleiner als 13 und $n \in \mathbb{N}$. Der Rekord für das allgemeine Zahlenkörpersieb liegt für Cunningham Zahlen bei 114 Stellen. Er wurde am 31.12.1998 von Adrjen Lenstra aufgestellt. Weiterhin wurden 108 – stellige Cunningham – Zahlen durch das zusammengesetzte allgemeine Zahlenkörpersieb faktorisiert und 117 – stellige durch das Quadratische Sieb. Es laufen viele verschiedene Projekte, die sich mit der Faktorisierung von großen Zahlen beschäftigen. Ein Grund dafür ist, dass die Firma RSA einen Wettbewerb ausgeschrieben hat, bei dem Preisgelder an diejenigen vergeben werden, die als erste eine noch nicht faktorisierte Zahl faktorisieren. Das hat auch dazu geführt, dass alle Zahlen, die im Wettbewerb ausgeschrieben waren, bis 127 Stellen faktorisiert wurden. Nur zwei Zahlen mit 128 Stellen und nur 16 Zahlen von 128 bis 137 Stellen wurden noch nicht faktorisiert. Die Zahlen die zur Faktorisierung ausgeschrieben wurden, besitzen eine bestimmte Form. Sie sind jeweils das Produkt zweier Primzahlen.

Zahl	Datum	MIPS-Jahre[35]	Algorithmus
RSA – 100[36]	April 1991	7	Quadratisches Sieb[5]
RSA – 110	April 1992	75	Quadratisches Sieb
RSA – 120	Juni 1993	830	Quadratisches Sieb
RSA – 129	April 1994	5000	Quadratisches Sieb
RSA – 130	April 1996	500	Allgemeines Zahlenkörpersieb[37]
RSA – 140	Februar 1999	2000	Allgemeines

[33] special number field sieve (NFS)
[34] Dies ist eine schnellere Version des Quadratischen Siebes
[35] Eine MIPS ist eine Million Anweisungen pro Sekunde.
[36] RSA - 100 ist eine 100-stellige Zahl, die das Produkt zweier Primzahlen ist.
[37] general number field sieve, englisch für Allgemeines Zahlenkörpersieb[11]

			Zahlenkörpersieb
RSA – 155	August 1999		TBA[38]

Tabelle 11 RSA Challenge List Status.[27]

Die RSA Challenge List zeigt die Erfolge, die bei der Faktorisierung von großen Zahlen erzielt wurden. Erst jetzt im August 1999 wurde eine 155-stellige Zahl faktorisiert. Diese Daten machen deutlich, dass immer größere Zahlen faktorisiert werden können. Diese Entwicklung geht einher mit der technischen Entwicklung von Computern, die immer leistungsfähiger werden.
Die Tabelle zeigt, wie weit die Faktorisierungsalgorithmen entwickelt sind und dass die Schlüssel immer größer werden müssen, damit der Verschlüsselungsalgorithmus sicher ist. Weitere Gründe, warum die Faktorisierung immer einfacher wird, sind, dass Computer immer schneller werden und sie zu Netzwerken zusammengeschlossen werden können. Aber auch die Faktorisierungsalgorithmen selbst werden immer effizienter und der grundlegende Fortschritt in der Mathematik gibt uns immer bessere Faktorisierungsalgorithmen. Mit der heutigen Mathematik und Technologie ist es unmöglich, die Faktorisierung von 1024 – Bit Schlüsseln in Betracht zu ziehen, aber morgen kann es schon ganz anders sein.
Dabei entspricht ein 1024 – Bit Schlüssel einer 308-stelligen Zahl.

2.5.4.2. TWINKLE [21]

2.5.4.2.1. Das Verfahren

TWINKLE (The Weizman INstitute Key Locating Engine) wurde dieses Jahr auf der Eurocrypt‘99 Konferenz von Adi Shamir vorgestellt. Es handelt sich dabei um ein opto-elektronisches Gerät, das den ersten Teil der Faktorisierung beschleunigt. Es ist also kein neuer Faktorisierungsalgorithmus, sondern es beschleunigt den Siebvorgang. Dabei kann TWINKLE die Arbeit von 100 bis zu 1000 Computern bei diesem Vorgang übernehmen.
Der optische Prozessor ist in einem undurchsichtigen schwarzen Zylinder untergebracht, welcher einen Durchmesser von 15,24 cm (6 Inches) und eine Höhe von 25,4 cm (10 Inches) hat. Der Boden des Zylinders enthält viele LED’s (Licht Emittierende Dioden), welche auf unterschiedlichen Frequenzen Licht aussenden. Auf der Oberseite des Zylinders befindet sich ein Fotodetektor. Der Fotodetektor misst ständig die gesamte Menge des emittierten Lichtes. Überschreitet die Lichtmenge eine bestimmte Schranke, ist das gesuchte Ereignis eingetreten. Der Fotodetektor meldet dies einem Computer, und die Ausgabe von TWINKLE ist der Zeitpunkt, an dem ein bestimmtes Ereignis eingetreten ist. Das Ereignis hängt mit der Ermittlung einer „glatten“ Zahl zusammen. Dieses Ereignis ist sehr selten, und somit hat der angeschlossene Computer genügend Zeit, bei jedem gemeldeten Ereignis zu überprüfen, ob es sich wirklich um eine „glatte“ Zahl handelt. Dazu wird ein Divisionsverfahren verwendet, und mit Hilfe der konventionellen Implementierung von QS oder NFS Algorithmen wird dann n faktorisiert.
Der optische Prozessor arbeitet dabei wie folgt: TWINKLE addiert hunderttausende Werte, indem jeder mögliche Summand durch eine Leuchtdiode (LED) von spezifischer Helligkeit erzeugt wird. Die jeweilige Gesamtlichtausbeute entspricht mit hinreichender Genauigkeit der Summe der „aktiven“ Werte. Damit die LED’s synchron angesteuert werden können, wird ein optischer Taktgeber verwendet. Schon mit handelsüblichen Gallium – Arsenid – LED’s und Fotodetektoren, wie sie für das Glasfasernetz benutzt werden, erreicht man Taktraten von bis zu 10 Gigahertz.[22]

2.5.4.2.2. Unterschiede zwischen TWINKLE und der Siebtechnik

[38] TBA to be announced

Bei der Siebtechnik werden modularen Quadraten Feldelemente zugewiesen, und es werden Schleifen über die Primzahlen ausgeführt. Bei TWINKLE weist man LED's Primzahlen zu und führt Schleifen über modulare Quadrate aus, was genau das Gegenteil der Siebmethode ist.
Jede LED wird mit einer Periode p_j und einer Verschiebung d_j verbunden und seine einzige Aufgabe ist es aufzuleuchten, um einen Zeitkreis zu beschreiben von dem arithmetischen Prozess $p_j * r + d_j$ mit $r \geq 0$. Zur Nachahmung des quadratischen Siebvorganges benutzt man LED's mit unterschiedlicher Lichtintensität. Im speziellen wird die LED der Primzahl p_j zugeordnet und die Lichtintensität der LED ist proportional zu $\log_2(p_j)$. Daraus folgt, dass die gesamte Lichtintensität der binären Länge des „glatten" Divisors von f(i) zur Zeit i entspricht. Das können wir erreichen durch ein Feld von LED's mit unterschiedlicher Größe oder mit unterschiedlichen Widerständen.
Die Methode ist aber etwas umständlich. Einfacher ist es, einheitliche Felder von identischen LED's zu benutzen und das Feld mit transparenten Filtern von Graustufen zu überlagern. Die Graustufen können einmal von unten nach oben angeordnet sein (die dunkelste ist oben und die transparenteste Stufe ist unten) oder spiralförmig (dabei ist die dunkelste in der Mitte). Um die Sensitivität des Fotodetektors zu erhöhen, wird eine Linse vor ihm angeordnet. Die Linse konzentriert das einfallende Licht auf eine schmale Fläche. Somit wird die Messung genauer und es geht kein Licht verloren.
Die Messung der Lichtintensität kann durch viele verschiedene Fehlerquellen beeinflusst werden. Einmal können die Graustufen nur approximativ gleich dem Logarithmus sein, andererseits können die einheitlichen LED's tatsächlich um 20 % oder mehr variieren. Solche Messfehler gefährden aber nicht die Berechnung des quadratischen Siebs oder auch der Zählfeldersieb Algorithmen, denn diese verwenden nur eine grobe Approximation ganzer Zahlen für die Logarithmusfunktion, um die Geschwindigkeit der Berechnung zu erhöhen. Es gibt aber noch andere Fehler, die bei der Messung auftreten können. Die typischsten sind zum einen, dass ein Ereignis verpasst wurde und zum anderen, dass ein Fehlalarm ausgelöst wurde. Um den ersten Fehler zu minimieren, kann man die Schranke herabsetzen, bei der eine Meldung an den Computer erfolgt. Um den Fehlalarm herauszufiltern, werden alle berichteten Ereignisse noch per Computer getestet.
Das gesuchte Ergebnis tritt durch klar isolierte Spitzen (zehnmal größer als die anderen Ergebnisse) hervor.
Das optische Sieb ist aus folgenden Gründen besser als das konventionelle Zählfeldersieb:
Es ist schneller als das konventionelle Zählfeldsieb. Ein normaler PC mit einem Silicon RAM Chip arbeitet mit 100 Megahertz. Eine LED mit Gallium Arsenide (GaAs) Technologie arbeitet mit 10 Gigahertz und mehr, sie werden auch in Glasfaserkabeln benutzt, um 10 Gigabyte pro Sekunde zu übertragen.
Man kann also die Leistung um das Hundertfache erhöhen, wenn dabei Einbußen in der Genauigkeit hingenommen werden.
Weiterhin benötigt die optische Technik nicht so viel Speicherplatz wie die herkömmliche Technik, und somit sinken auch die Kosten. Des Weiteren muss das Feld nicht am Anfang durchsucht werden und auch nicht am Ende, um den höchstens Eintrag zu finden. Somit wird Zeit gespart da die Operation schon während des Siebvorganges ausgeführt wird.
Dies wird auch deutlich durch einen Vergleich von Robert D. Silvermann[23]. Er beschreibt, dass sieben solcher Einheiten die benötigten Gleichungen für die Faktorisierung von 465 – Bit – Schlüsseln in sechs Tagen aufstellen können. Anfang des Jahres wurden dazu noch 200 Computer benötigt die vier Wochen lang parallel arbeiten mussten. Weiterhin könnte man auch 512 – Bit – Zahlen in angemessener Rechenzeit faktorisieren, das entspricht der Verschlüsselungsstärke der Export – Version von US – Software.

Andererseits dauert es auch mit 45 000 TWINKLE noch eine halbe Million Jahre, die Gleichungen für einen 1024 – Bit – Schlüssel aufzustellen. Somit ist TWINKLE keine große Bedrohung für das RSA – Verfahren oder andere skalierbare Verfahren, die auf der Faktorisierung aufbauen.

3. Implementierung des RSA - Verfahrens in Mathematica

Meine Implementierung des RSA - Verfahrens besteht aus dem Package sowie einem Notebook mit Anwendungsbeispielen. Das Package enthält Hilfsfunktionen für das Verfahren und Funktionen, die den Text so umwandeln, dass das Verfahren aus ihm angewendet werden kann, sowie die Verschlüsselungs-, Entschlüsselungsfunktion und das Modul zum Erzeugen der Schlüssel.
Das RSA Notebook enthält die Anwendung der Funktion an zwei Beispielen. Einmal wird es auf eine kurze Nachricht angewandt und das andere Mal auf eine Datei.

3.1. Das Package RSAMethoden.m

Dieses Package enthält alle Hilfsfunktionen, die für die Implementierung des RSA – Verfahrens benötigt werden, sowie Funktionen, die den Text umformen, so dass dieses Verfahren auf den Text angewendet werden kann.

Ich möchte nun auf die einzelnen Funktionen eingehen und sie gegebenenfalls erklären. Dabei werde ich auf allgemeine Mathematicabefehle, wie zum Beispiel *BeginPackage* nicht eingehen, sondern mich nur auf die von mir definierten Funktionen beschränken.

BeginPackage["RSAMethode`"]

Im ersten Teil des Packages wird jeder Funktion ein Hilfstext zugewiesen, der die Anwendung der Funktion erleichtern soll und dann im Notebook durch die „*?*“ Funktion abgerufen werden kann.

Combine::usage =
"Combine[liste_] fügt eine Liste von dreistelligen Zahlen zu einer Zahl zusammen."

MakeList::usage =
"MakeList[zahl_] zerlegt eine Zahl in eine Liste von dreistelligen Zahlen."

NextPrime::usage =
"NextPrime[n_] testet, ob n Primzahl ist, wenn nicht wird n um eins erhöht und wieder getestet bis n Primzahl ist."

ExpMod::usage =
"ExpMod[a_,b_,c_] wird zum Verschlüsseln und Entschlüsseln verwendet."

DeCodierung::usage =
"DeCodierung[zahl_] wandelt mit Hilfe von FromCharacterCode dreistellige Zahlen in Buchstaben um."

BlockLength::usage =
"BlockLength[n_] berechnet die Blocklänge."

ETest1::usage =
"ETest1[p_,q_,φ_,e_] prüft ob e =φ(n)/2 +1 ist und ob der ggT nicht größer als 2 ist, wenn das der Fall ist setzt der Test j auf 1."
ETest2::usage =
“ETest2[φ_] prüft ob e =PowerMod[d,-1,φ] ein Ergebnis hat,

wenn das der Fall ist setzt der Test i auf eins."

ETest::usage =
"ETest[φ_] überprüft die Ergebnisse von ETest1 und ETest2 und ruft gegebenenfalls den KeyGenerator noch einmal auf."

Off[General::spell1]
DGCD::usage =
"DGCD[φ_] wählt eine Primzahl im Bereich 10^{60} bis 10^{70} aus und testet ob sie einen ggT >1 mit φ hat."
On[General::spell1]

LDatei::usage =
"LDatei[datei_,n_] liest eine Datei in eine Liste ein und teilt sie gleichzeitig in Blöcke auf mit vorgegebener Blocklänge. Dateinamen werden in Anführungszeichen angeben."

ListDatei::usage=
"ListDatei[datei_] liest eine Datei ein die eine Liste enthält."

loeschen::nde=
"Die Globalen Variablen d, n ,e werden neu definiert bitte löschen Sie diese und rufen Sie danach den KeyGenerator noch einmal auf."

Off[General::spell1]
SDatei::usage = "SDatei [datei_,list_] speichert Daten in einer Datei ab."
On[General::spell1]

RSA::usage = "RSA[Nachricht_,e_,n_,v_] verschlüsselt und entschlüsselt Nachrichten. Der Parameter v wird dabei zur Fallunterscheidung verwendet. Die Nachricht wird verschlüsselt, wenn der Parameter v =1 ist und entschlüsselt, wenn v eine andere Zahl ist. e ist der private bzw. öffentliche Schlüssel und n ist der dazugehörige öffentliche Schlüssel."

KeyGenerator::usage = "KeyGenerator erzeugt den privaten sowie öffentlichen Schlüssel und gibt sie aus. Die Schlüssel werden global den jeweiligen Variablen zugewiesen."

CryDatei::usage = "CryDatei[ldatei_,sdatei_,e_,n_] liest Daten aus der Datei ldatei aus und verschlüsselt diese. Der verschlüsselte Text wird ausgegeben und in die Datei sdatei geschrieben."

DeCryDatei::usage = "DeCryDatei[ldatei_,sdatei,d_,n_] Der verschlüsselte Text wird aus der Datei ldatei ausgelesen entschlüsselt und in die Datei sdatei geschrieben."

ABCCode::usage = "ABCCode[zahl_] Die verschlüsselte Nachricht zahl wird in Buchstaben umgewandelt."

ABCDeCode::usage = "ABCDeCode[Nachricht_] ist die Umkehrfunktion zu ABCCode die Buchstabenfolge wird in eine Zahl umgewandelt."

ABCCodeDatei::usage = "ABCCodeDatei[ldatei_,sdatei_] wendet ABCCode auf ldatei an und speichert das Ergebnis in sdatei ab."

ABCDeCodeDatei::usage = "ABCCodeDatei[ldatei_,sdatei_] wendet ABCDeCode auf ldatei an und speichert das Ergebnis in sdatei ab."

Nun beginnt das eigentliche Package, der sogenannte private Teil. Nur die oben erklärten Funktionen werden exportiert.

Begin ["`Private`"]

```
Combine[liste_]:=First[liste]/; Length[liste]==1
Combine[liste_]:=1000*Combine[Reverse[Rest[Reverse
                    [liste]]]]+Last[liste]
```

Die erste Funktion *Combine* fügt eine Liste von dreistelligen Zahlen zu einer Zahl zusammen, wie wir dem Hilfstext entnehmen. [29]

```
MakeList[zahl_]:={zahl}/; zahl<1000
MakeList[zahl_]:=Append[MakeList[(zahl-Mod[zahl,1000])
                    /1000],Mod[zahl,1000]]
```

Diese Funktion ist die Umkehrfunktion zu *Combine*. Sie teilt die bereits wieder entschlüsselte Zahl in Listenelemente ein, so dass die Funktion *FromCharacterCode* angewendet werden kann. *FromCharacterCode* arbeitet mit dem ASCII Standard Code und kann demzufolge nur Zahlen im Bereich von 0 bis 255 umwandeln. [29]

```
NextPrime[i_]:=i/; PrimeQ[i]
NextPrime[i_]:=NextPrime[i+1]
```

Laut Hilfstext testet *NextPrime*, ob n eine Primzahl ist, wenn dies nicht der Fall ist, wird n um eins erhöht und wieder getestet, bis n Primzahl ist. Dabei wird die Mathematicafunktion *PrimeQ* verwendet. [29]

ExpMod[a_,b_,c_]:=PowerMod[a,b,c]

Die *Mathematica* interne Funktion *PowerMod* wurde im Kapitel 2.2.5.2. näher erklärt. Sie leistet einen wesentlichen Beitrag, damit das RSA – Verfahren relativ effizient implementiert werden kann. *PowerMod[M,e,n]* gibt das Ergebnis von M^e mod n an.[29]

DeCodierung[zahl_]:=FromCharacterCode[MakeList[zahl]]

Diese Funktion wandelt mit Hilfe der Mathematicafunktion *FromCharacterCode* eine Liste von Zahlen, die durch *MakeList* auf der Grundlage des ASCII–Standard Codes erzeugt wurde, in Buchstaben um.

BlockLength[n_]:=IntegerPart[Log[10, n]/3]-1

Die Zahl *BlockLength* bestimmt, wie lang ein Block sein kann. Wie wir bereits wissen, ist die Blocklänge i wie folgt definiert von $10^i < n < 10^{i+1}$. Es ist notwendig, die Blocklänge noch

einmal durch drei zu teilen, da ich sie im Folgenden direkt auf Buchstaben beziehe und ein Buchstabe einer Zahl mit genau drei Stellen entspricht. Wird die Zahl nicht durch drei geteilt, ist die Entschlüsselung nicht eindeutig, da die Blöcke zu groß sind.

ETest1[p_,q_,φ_,e_]:=Module[{j=0},If[e==φ/2+1,j=1 ||GCD[p-1,q-1]> 2,j=1];j]

Wie im Hilfetext schon angegeben testet *ETest1,* ob e mit φ/2 +1 übereinstimmt oder nicht. Wenn die Aussage wahr ist, wird j = 1 gesetzt und im *ETest* weiter verarbeitet. Dieser Test ist notwendig, denn wenn e = φ/2+1 ist, würde bei der Verschlüsselung der Originaltext entstehen.[39]
Weiterhin wird getestet, ob der ggT(p-1,q-1) größer als 2 ist.

Ist dies der Fall gilt auch für $e = \frac{\varphi(n)}{ggT\ (p-1, q-1)} + 1$ Geheimtext ist gleich Originaltext.

Bemerkung: Es ist nicht notwendig zu testen, ob e noch mit einem anderen Wert der Form a*φ+1 übereinstimmt, da diese Werte für ganze a außerhalb des Bereiches von e liegen. Der Bereich für e ist 1≤e≤φ. Wenn dies nicht der Fall ist und e die Form von a*φ+1 besitzt, so würde auch hier bei der Verschlüsselung der Originaltext entstehen.

```
ETest2[φ_]:=
    Module[{i=0},Off[PowerMod::ninv];
        Global`e=PowerMod[Global`d,-1,φ];
        On[PowerMod::ninv];
        If [Head[Global`e]==PowerMod,i=1];
    i]
```

ETest2 erzeugt e mit Hilfe der *PowerMod* Funktion. Damit keine Fehlermeldung bei der *PowerMod* Funktion entsteht, wird diese Fehlermeldung mit Off ausgeschalten. Danach wird getestet, ob ein Fehler bei der Berechnung des Inversen e mit $e\ d^{-1} \equiv 1 \bmod \varphi(n)$ aufgetreten ist oder nicht.

```
ETest[φ_]:=
    Module[{i,j},
        i=ETest2[φ];
        While[i==1,
            KeyGenerator;
            i=ETest2[φ]];
        j=ETest1[p,q,φ,Global`e];
        While[j ==1 ,
            KeyGenerator;
            j=ETest1[p,q,φ,Global`e]];
    Global`e]
```

ETest erzeugt als erstes e mit Hilfe von *ETest2*. Das Ergebnis von *ETest2* wird geprüft und gegebenenfalls wird der *KeyGenerator* aufgerufen und somit neue Schlüssel erzeugt. Die Prüfung erfolgt solange, bis der Test erfolgreich war. Wenn *ETest2* positiv war, wird *ETest1* geprüft und genauso wie vorher verfahren.

---

[39] Siehe Kapitel Verschlüsselung mit e = φ/2 +1.

```
Off[General::spell1]
DGCD[φ_]:=
        Module[{j},j=0;i=0;
            While[j≠1 && IntegerQ[i],
                i=NextPrime[Random[Integer,{10^60,10^70}]];
            j=GCD[φ,i]];
        i]
On[General::spell1]
```

Das ist eigentlich ein Test, und es wird geprüft, ob der größte gemeinsame Teiler von d und φ(n) gleich eins ist. Als erstes wird j = 0 gesetzt, damit wird die Schleife auf jeden Fall durchlaufen. Danach wird i der Kandidat für d zugewiesen und dann getestet. Geht der Test positiv aus, wird j eins zugewiesen, was dem ggT entspricht. Im negativen Fall wird j auch ein ggT zugewiesen, der aber nicht mehr eins ist, und somit wird die Schleife automatisch noch einmal durchlaufen. Als letztes wird i aufgerufen, damit es als Funktionswert zurückgegeben wird.

```
Off[General::spell1,General::spell]
LDatei[datei_,n_]:=
    Module[{ch,string,RDatei},string="";slist={};
        RDatei=OpenRead[datei];
        While[Not[SameQ[ch,EndOfFile]],ch=Read[RDatei,Character];
            If[Not[SameQ[ch,EndOfFile]],
            string=string<>ch,string=string<>""];
            If[StringLength[string]==BlockLength[n],
            AppendTo[slist,string];string=""]
        ];
        AppendTo[slist,string];Close[datei];
    slist]
On[General::spell1,General::spell]
```

LDatei steht für Lesen der Datei. Der Prozess liest eine Datei in eine Liste ein. Dazu wird der Dateiname und n benötigt. Das n benötigen wir, um die Blocklänge der Listenelemente zu bestimmen, damit die Eindeutigkeit gewährleistet wird.
Die Datei wird mit *OpenRead* geöffnet. Danach wird eine Schleife so lange durchlaufen bis das Ende der Datei erreicht ist. Der Inhalt der Datei wird in die Variable *string* geschrieben. Wenn die Variable *string* so groß ist wie die Blocklänge, wird sie in *slist* als Element eingefügt und danach wieder gelöscht und der Prozess beginnt wieder von vorn. Wenn das Ende der Datei erreicht ist, wird nur noch *string* der *sliste* als letztes Element hinzugefügt und danach die Datei geschlossen. Als Ergebnis erhält man eine Liste, die in *sdatei* (*sdatei* steht für Speichern der Datei) geschrieben wurde, wobei jedes Element der Liste die vorgegebene Blocklänge besitzt mit Ausnahme des letzten Listenelementes.

```
Off[General::spell]
ListDatei[datei_]:=
    Module[{RDatei,ch,list},
        RDatei=OpenRead[datei];
        list=ReadList[RDatei];
        Close[datei];list=Flatten[list];
    list]
On[General::spell]
```

ListDatei hat die gleiche Aufgabe wie *LDatei,* nämlich aus einer Datei Daten einzulesen. Der einzige Unterschied zu *LDatei* ist, dass es sich in diesem Fall bei den Daten nicht um einen String handelt, sondern um eine Liste von Zahlen, also eine verschlüsselte Datei.

```
Off[General::spell1]
SDatei[datei_,list_]:=
        Module[{SDatei},Sdatei=OpenWrite[datei];
                Write[Sdatei,list];
                Close[Sdatei]
        ]
On[General::spell1]
```

SDatei steht für Speichern der Datei. Dieser Prozess legt die Daten in eine Datei ab, dabei braucht es sich nicht um eine Liste zu handeln, sondern mit dieser Funktion werden jede Art von Daten abgespeichert.

Jetzt erfolgt die eigentliche Verschlüsselung mit Hilfe der oben beschriebenen Funktionen.

```
RSA[Nachricht_,e_,n_,v_]:=
        Module[{i},
                If [v==1 ,
                        ExpMod[Combine[ToCharacterCode[Nachricht]],e,n],
                        DeCodierung[ExpMod[Nachricht,e,n]]
                ]
        ]
```

Das RSA – Modul hat vier Eingabeparameter. Erstens die Nachricht, die verschlüsselt oder entschlüsselt werden soll. Zweitens *e* der öffentliche Schlüssel oder *d* der private Schlüssel. Weiterhin der öffentliche Schlüssel *n*, und der vierte Parameter ist *v*. Mit diesem Wert wird festgelegt, ob verschlüsselt oder entschlüsselt wird. Wenn *v* den Wert eins hat, wird verschlüsselt. Bei jedem anderen beliebigen Wert wird entschlüsselt, vorzugsweise ist der Wert zwei. Das v wird als erstes getestet und dementsprechend wird im Modul verzweigt. Bei der Verschlüsselung wird die Nachricht mit *ToCharacterCode* in eine Liste von Zahlen nach den ASCII- Code umgewandelt. So erhält man eine Liste mit Zahlen im Bereich 0 bis 255. Danach wird die Liste mit Hilfe von *Combine* zu einer Zahl zusammengefasst, und auf diese Zahl wird dann PowerMod Zahl^{e} mod n berechnet. Somit ist die Nachricht verschlüsselt. Die Entschlüsselung funktioniert umgekehrt. Als erstes wird die Zahl^{d} mod n berechnet, danach wird die Liste von Zahlen wieder erzeugt, und dies wird durch *FromCharacterCode* wieder in Buchstaben umgewandelt. Somit wird der Originaltext hergestellt.

```
KeyGenerator:=
        Module[{i,p,q,φ},
                If [SameQ[Head[Global`d],Symbol] &&
                SameQ[Head[Global`n],Symbol] && SameQ[Head[Global`e],Symbol],
                        p=NextPrime[Random[Integer,{10^80,10^90}]];
                        q=NextPrime[Random[Integer,{10^80,10^90}]];
                        φ=(p-1)*(q-1);
                        Global`n=q*p;
                        Global`d=DGCD[φ];
                        ETest[φ];
```

```
                Print["geheimer Schlüssel d:" ]; Print[Global`d];
                Print["öffentlicher Schlüssel n:"]; Print[Global`n];
                Print["öffentlicher Schlüssel e:"]; Print[Global`e],
            Message[loeschen::nde]]
    ]
```

Das Modul *KeyGenerator* erzeugt die benötigten Schlüssel sowie die Zahlen *p* und *q*. Diese beiden Zahlen werden für die Verschlüsselung und Entschlüsselung nicht benötigt und daher werden sie auch nicht ausgegeben. *e, d* und *n* werden ausgegeben und den entsprechenden globalen Variablen zugewiesen. Da die drei Variablen global zugewiesen werden, wird mit Hilfe einer *If* Abfrage sicher gestellt, dass sie nicht belegt sind. Sind die Variablen belegt wird abgebrochen und der Nutzer muss die Variablen löschen und den *KeyGenerator* noch einmal ausführen. Die Funktionen, mit denen die Variablen erzeugt wurden, wurden bereits erläutert.

Damit sind alle Voraussetzungen geschaffen, um Nachrichten zu verschlüsseln und zu entschlüsseln. Andererseits fehlt jetzt noch, wie man längere Texte oder ganze Dateien verschlüsselt oder entschlüsselt. Dazu müssen noch einige Dinge beachtet werden.

```
Off[General::spell1]
CryDatei[ldatei_,sdatei_,e_,n_]:=
            Module[{i,RSAlist,strlist,f},
                strlist:=LDatei[ldatei,n];
                RSAlist[w_]:=RSA[w,e,n,1];
                f=Map[RSAlist,strlist];
                SDatei[sdatei,f];
            f]
On[General::spell1]
```

Mit Hilfe von *CryDatei* kann man Daten in *ldatei* einlesen und verschlüsseln. Diese werden dann in *sdatei* abgespeichert. Dabei wird die verschlüsselte Datei auch ausgegeben. Wir wissen, dass die Prozedur *LDatei* eine Datei einliest und die Daten in eine Liste ablegt, wobei jedes Listenelement maximal die Größe der vorgegebenen Blocklänge hat. Nun braucht man nur noch die RSA Funktion auf jedes Listenelement anwenden und die Datei ist verschlüsselt. Dieser Vorgang wird durch die *Mathematica* interne *Map* – Funktion sehr vereinfacht. Da es nicht notwendig ist, eine Schleife zu schreiben, wird RSA auf jedes Listenelement angewandt mit Hilfe der Funktion *Map angewandt*.

```
Off[General::spell1,General::spell]
DeCryDatei[ldatei_,sdatei_,d_,n_]:=
    Module[{c,zahl,RSAdlist},
        zahl=ListDatei[ldatei];
        RSAdlist[w_]:=RSA[w,d,n,2];
        c=Map[RSAdlist,zahl];
        c=StringJoin[c];SDatei[sdatei,c];
    c]
On[General::spell1,General::spell]
```

DeCryDatei ist die Umkehrfunktion zu *CryDatei*. Als erstes wird die verschlüsselte Datei *ldatei* eingelesen, dann wird jedes Listenelement entschlüsselt, und danach werden sie wieder zu einem zusammenhängenden Text vereint. Dieser wird ausgegeben und in die *sdatei* abgespeichert.

Damit ist der wesentliche Teil des RSA - Verfahrens abgeschlossen. Jetzt möchte ich noch zwei Module vorstellen: *ABCCode* und *ABCDecode*. Die beiden Module gehören zusammen und dienen zur Umwandlung der Zahlen in Buchstaben. Der verschlüsselte Text wird mit Hilfe von *ABCCode* wieder in Buchstaben umgewandelt und *ABCDeCode* ist die dazugehörige Umkehrfunktion.

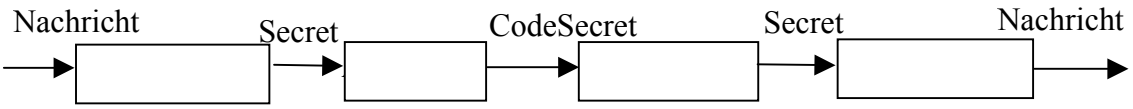


Abbildung 10: Ablaufschema für das Ver- und Entschlüsseln einer kurzen Nachricht

Somit entsteht folgender Ablauf: Die Nachricht wird mit *RSA* verschlüsselt und mit *ABCCode* wird die verschlüsselte Nachricht in Buchstaben umgewandelt. Um die Originalnachricht wieder zu erhalten, wird die Funktion *ABCDeCode* auf die verschlüsselte und codierte Nachricht angewandt und mit *RSA* wird die Nachricht entschlüsselt. Diese erfolgt wie im obigen Ablaufschema dargestellt. *ABCCode* und *ABCDeCode* dienen dazu, die verschlüsselte Nachricht mit Buchstaben darzustellen.

```
ABCCode[zahl_]:=
    Module[{i,m,d,list,element},
        list=MakeList[zahl];
        For[i=1,i<=Length[list],i++,
            If[list[[i]]>96,
                m=Mod[list[[i]],96];
                d=IntegerPart[list[[i]]/96];
                list=ReplacePart[list,{m,d},i];
            list=Insert[list,Random[Integer,{97,110}],i+1];i++]];
        list=Flatten[list];
    For[i=1,i<=Length[list],
        i++,
        If[ list[[i]]<=48 ,
            element=list[[i]]+48;
            list=ReplacePart[list,element,i];
            list=Insert[list,Random[Integer,{111,122}],i+1];i++]];
        FromCharacterCode[list]
    ]
```

Das Problem, den verschlüsselten Text zu codieren, besteht darin, dass der Zahlenbereich von 000 bis 999 in einen relativ kleinen Zahlenbereich von 048 bis 122 codiert werden muss. Denn dieser Zahlenbereich von 048 bis 122 wird nach ASCII Code als Buchstaben und ein paar Sonderzeichen codiert. Dabei teilt sich der Zahlenbereich von 000 bis 999 in drei Zahlenbereiche ein, Zahlen, die kleiner sind als der ASCII Zahlenbereich für Buchstaben. Zahlen, die in diesen Bereich fallen, und Zahlen, die größer sind. Am einfachsten sind die Zahlen zu handhaben, die in den zweiten Bereich fallen. Sie können direkt codiert werden. Die Zahlen, die kleiner als der Bereich sind, habe ich einfach mit 48 addiert, so dass sie dann in den Bereich fallen.
Bei den Zahlen größer als der Bereich, gestaltete sich das schon etwas schwieriger. Es erwies sich als günstig, sich dabei zwei Zahlen zu merken. Einmal den ganzzahligen Teil von der Division durch 96 sowie den dazugehörigen Rest. Somit erhält man Zahlen im Bereich von Null bis 96. Dabei wendete ich als erstes diese Funktion auf die Liste an und danach die

Funktion für die Zahlen die kleiner als 48 sind. Als Ergebnis erhält man eine Liste von Zahlen, die als Buchstaben decodiert werden können mit der Funktion *FromCharacterCode.*
Für die Decodierung benötigte ich nun einen Anhaltspunkt, ob ich eine Zahl addiert hatte oder ob ich für eine Zahl zwei eingefügt hatte. Als erstes probierte ich es mit einem speziellen Zeichen, was sich als sehr uneffektiv herausstellte, da dieses Zeichen damit sehr gehäuft auftrat, um nicht zu sagen, der ganze Text bestand aus ihm.
Somit versuchte ich es mit einem Zahlenbereich, was sich als sehr effektiv erwies. Ich teilte den Buchstabenbereich der ASCII Zahlen in drei Bereiche auf. In dem Bereich von 48 bis 96 werden die Originalzahlen abgebildet. Zweiter Bereich von 97-110, aus diesem Bereich wird eine zufällige Zahl in die Liste eingefügt, wenn die Originalzahl größer als 96 ist. Und der letzte Bereich von 111-122, aus diesem Bereich wird eine Zahl in die Liste eingefügt, wenn die Originalzahl kleiner als 48 ist.
Somit ergibt sich folgende Methode. Ist die Originalzahl größer als 96, wird sie durch 96 geteilt und der Rest gebildet. Diese beiden Zahlen werden in die Originalliste eingefügt sowie dahinter eine Zahl aus dem Bereich 97-110. Jetzt wird die Liste neu durchlaufen. Trifft man auf eine Originalzahl, die kleiner ist als 48, wird 48 dazu addiert und anstelle der Originalzahl in die Liste eingefügt und zusätzlich noch eine Zahl aus dem Bereich 111-122. Das Ergebnis spricht für sich, wie man im nächsten Beispiel vergleichen kann.
Um diesen Vorgang etwas zu verdeutlichen, stelle ich ihn kurz an dem Beispiel der verschlüsselten Liste g = {385} dar. Als erstes wird das Listenelement durch 96 geteilt und wir erhalten 4 und den Rest 1. Diese beiden Zahlen werden anstelle von 385 in die Liste g eingefügt g = {4, 1}, und zusätzlich wird noch ein Element aus dem Bereich von 97 bis 110 eingefügt. Somit entsteht eine neue Liste g = {4, 1, 98}. Als nächstes wird die Liste noch einmal durchlaufen und zu den Zahlen, die kleiner als 48 sind, wird 48 addiert, und es wird ein Element aus dem Bereich 111 bis 122 eingefügt. Jetzt sieht unsere Liste g folgendermaßen aus g = {52, 112, 49, 119, 98}. Diese Zahlen lassen sich durch Buchstaben und Sonderzeichen ersetzen, und man erhält: 4p1wb.

```
ABCDeCode[Nachricht_]:=
        Module [{i,list},
                list=ToCharacterCode[Nachricht];
                For[i=1,i<=Length[list],
                        If[ list[[i]]>=111&&list[[i]]<=122,
                                list=ReplacePart[list,list[[i-1]]-48,i-1];
                                list=Delete[list,i];i--];
                i++];
                For[i=1 ,i<=Length[list],
                        If[ list[[i]]>=97&&list[[i]]<=110,
                                list=ReplacePart[list,list[[i-2]]+list[[i-1]]*96,i-2];
                                list=Delete[list,i];list=Delete[list,i-1];i=i-2];
                i++];
                list=Combine[list]
        ]
```

Die *ABCDecodierung* bildet das Gegenstück zur *ABCCodierung*. Es werden hier die entsprechenden Bereiche abgesucht und die entsprechende Zahl aus der Liste wieder entfernt sowie die Originalzahl wieder zusammengefügt.

```
ABCCodeDatei[ldatei_,sdatei_]:=
                Module[{c,zahl},
                        zahl=ListDatei[ldatei];
```

```
            c=Map[ABCCode,zahl];
            SDatei[sdatei,c];
        c]

ABCDeCodeDatei[ldatei_,sdatei_]:=
        Module[{c,zahl},
            zahl=ListDatei[ldatei];
            c=Map[ABCDeCode,zahl];
            SDatei[sdatei,c];
        c]
```

ABCCodeDatei und *ABCDeCodeDatei* bilden wieder eine Einheit, das bedeutet, die eine Funktion ist die Umkehrfunktion der anderen. *ABCCodeDatei* wandelt eine verschlüsselte Datei um, indem es mit Hilfe von *ABCCode* jeder Zahl Buchstaben zuordnet. Danach werden die Buchstabenfolgen in einer Datei abgespeichert. Es ist eine Erweiterung von *ABCCode,* so dass diese Funktion jetzt auf die ganze Datei angewandt werden kann.
ABCDeCodeDatei kehrt nun diese Umwandlung der Zahlen in Buchstaben um, und speichert das Ergebnis ebenfalls in einer Datei ab. Dabei gilt wie vorher: *ldatei* steht für die Datei, die eingelesen wird, und *sdatei* für die Datei, in die geschrieben wird.

```
End[]
EndPackage[]
```

Damit sind die wichtigsten Bestandteile das Package *RSAMethoden* erläutert. Als nächstes möchte ich einige Anwendungsbeispiele aufzeigen.

3.2. Anwendungsbeispiele der Implementierung

3.2.1. Einfache Beispiele der Verschlüsselung in der Datei *RSAnotebook*

In diesem Kapitel dokumentiere ich die Ergebnisse jeder Funktion. Die Funktionen werden wie folgt durchlaufen: Als erstes wird das *Package RSAMehoden.m* geladen, danach der String „Nachricht“, der verschlüsselt werden soll, der Variablen *News* zugeordnet. Als nächstes wird der *KeyGenerator* aufgerufen. Er erzeugt die Schlüssel *d*, *e* und *n*, die für die nächsten Module benötigt werden. Die Variable *News* wird verschlüsselt.
Als nächstes wird die verschlüsselte Variable *News* wieder in Buchstaben zurück codiert, und um das Ergebnis überprüfen zu können wird dieser Vorgang rückgängig gemacht. Dazu verwende ich *ABCCode* und *ABCDeCode*. Danach kann man das Ergebnis mit der verschlüsselten Version von *“Nachricht“* vergleichen, die verschlüsselte Version von *„Nachricht“* wird entschlüsselt mit RSA.

Die Datei „*Reise“*, welche einen längeren Text enthält, wird verschlüsselt mit *CryDatei*. Wir wissen, dass der Geheimtext viel länger ist als der Originaltext. Die Hauptursache liegt darin, dass jedes Zeichen durch eine zwei- oder dreistellige Zahl dargestellt wird. Bei der Umcodierung der Zahlen in Buchstaben werden zusätzlich ein bzw. zwei Zahlen eingefügt. Das hat zur Folge, dass der Text wiederum verlängert wird. Wir können davon ausgehen, dass der verschlüsselte Text drei bis fünf mal so lang wie der Originaltext wird.
Die Originaldatei ist *„Reise“*, der Text wird in die Datei *„EnText“* verschlüsselt. In der Datei „*ABCText“* steht die codierte Version der verschlüsselten Datei *„Reise“*, und in

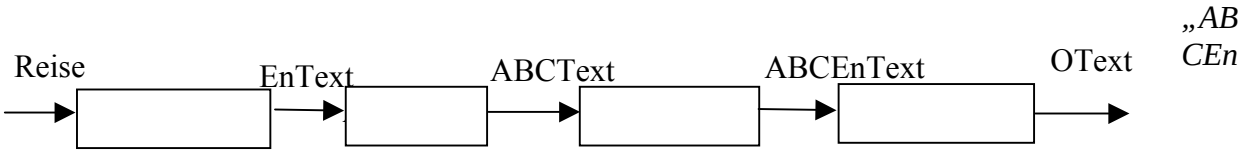


Abbildung 11: Ablaufschema für das Ver- und Entschlüsseln einer Datei

Text“ steht das Ergebnis von *ABCDeCode*. Den ursprünglichen Originaltext findet man in der Datei *„OText“*, wohin *„ABCEnText“* entschlüsselt wurde.

In[1]: = <<d:/Diplom/Notebook/RsaMethoden.m

In[2]: = News="Nachricht"
Out[2] = "Nachricht"

In[3]: = KeyGenerator

"geheimer Schlüssel d:"
923430719533662603646315923998299834853921132216953776498872725920538 1
"öffentlicher Schlüssel n:"
627753393175319596853455577935359057703136514135526386363053241838425700954039764889950387530662745671515176432455591813947656790941707398188828163929623431281677913174434124872771
"öffentlicher Schlüssel e:"
339768640312901496388068203243324031312810234871064530743080507912550688169191605484234898184631293523274540836252119093597832766766064346436253346144933385252892579277658330575021

In[4]: = Secret=RSA[News,e,n,1]
Out[4] =
394211175797444995400145160847029190399967233615425681713680627563856937944452506991808329203675040800935743713218248167516835984636546035040934307718562869459026922988638277998587

In[5]: = CodeSecret=ABCCode[Secret]
Out[5] =
:u4znCy2tfO1umMt8zl<4xbSv:sk@u4ti11xn@1wgO8wkMt^1xk?p4zk7o:thYv2vbWq6zgYu4sd9p7tbYz7ue8w7vb36vbS5rcX8qcI9snP9qcD4wiJy5rbOx:siXr8rjYw3pg;v2ob3p7umXxPp8wjG9ygG7pjYo7unJq2oh82yeG1vnTs5smC8znHs:oc<6xlB5zbStXpF9oaCy3yl^v7xfR5sf5z9ykK4riJx:9sdLs:yl>6qdU2ymVu:xl;t6um

In[6]: = ABCDeCode[CodeSecret]
Out[6] =
394211175797444995400145160847029190399967233615425681713680627563856937944452506991808329203675040800935743713218248167516835984636546035040934307718562869459026922988638277998587

In[7]: = Secret
Out[7] =
394211175797444995400145160847029190399967233615425681713680627563856937944452506991808329203675040800935743713218248167516835984636546035040934307718562869459026922988638277998587

In[8]: = RSA[Secret,d,n,2]
Out[8] = "Nachricht"

In[9]: = SetDirectory["D:\Diplom\Notebook"]
Out[9] = "D:\\Diplom\\Notebook"

In[10]: = c=CryDatei["Reise","EnText",e,n]
Out[10] =
{49926590650358999111273230772118243862849669842406683049980070235240721575623982542984901847260051199470434351646715480653545891470262207873864641895393988950412877708702071072222 8,
60433387380347481898561378470049336733489706933214694432400387223888947890912887043206174611259039336048198179116857723748187735114213357296450942591429088843897508512576017410735 7,
14549494191975270875666082190044965647993085263267386142846657647714266133255220895512694620174144687496042456478980611729545762765961993435061960348750128040802558627119399269955 4,
34650594308838684256997038748998568937957141585426594384302550703510012101666337048901658879361601666162787209493845041921764063052883052803428502417865227673611300965415601990061 0,
21775843982423989475094317031553502473970421335129716431572763623600240957914782673730437704655686471594138214439024393761508774530546481266977028854581469654827268133487676376250 9,
12681112236996203234372617830094789970926628269288426204612172467917617048875161304417345319507939938982626168462880459877688658906291034850584560905407574679776751925014834300607 3,
21019155034314257765879568256666880078521070009542578938436090867643769738199265927615482105324049144095004518009309529771641624427574343078092886713390216123077646666620404915121 8,
88014651802801390258316317537107160507711627950276253222381651079675096814324101847558696465554704309917361333194921182272703448737324737787684742691243781797959672603427267066080 ,
53558026240314838197678424808815002980254281893419328380502693594239206198746357693557277321482899458223497818127859926368148827304715329049379492833473617794894224091722490411194 5,
36198695107040877364730003544827377606822357560428343764310495183591670962392569976597297486605778103663374776133151140920666988066156594412385140876079317141048382350669079677136 6,
35895401941599335360875951487871867218183234425564466907077326434264325741323514044035812069941987177487534813483300663369422686084558267799884709526126917381205192487680690517707 6,
15688019166155011551267233703892535340850091626699033999975215606310216496921604944977582963512171871293928228449331770929594884783941866330213148396825365781838160889347741284331 ,
18513649941141820336161789447140572046191480763821246163718377585612814904520201347994879446989107827325680755490378554394515727550309703064853185636200195045181387884966691216221 0,
11656041115178536702438846061943597204385395368527052677874881546559939879822117704469482769560863173886578442637443777556813187555202130858643314439169816934243133145242962549699 6,
38230395614959075227052901618534365457274549305040145671811612569886959748340582999064878470522776845095227423447378451374336270136699704249321688620430706558493984696876710751059 6,
37264494592107264793255747161505874755485658147286891126900190551584106146060433919373943380914469631588393586575391665644675619331140098026040662984918307280085264516645908819385 2,

3146273433802138574471489537473141888320416090081090446540749002428792521415897224158298062916477127043307054140512966991801833814904229755246584941066193167150494222533540705782 20,
1282673906153083676093854369403447543924766685062052080399225522578263605805534320511154638534313845734791851720579639013100150566971567353019759847243415021175568692052673293218 14,
2323030235532148672322824891336806129118989966551291509725845237497131342664537757105905489422069515672925349653943521147964702979131820314433736770786450471636651746199048966944 25,
1410365056189079186431426434410702117548795763784708083865213880933850655965119435746973615987306723931953519140510137543525341907416711239177785980823932042211513846733843384678 08,
8697981218719581317183872049662534123099030244828590066640217641775449854955245203762177464797126222789169819080845542253077880718965772540205517539918783933476402730417954730592,
4408439560081317941955022732963752448651580973127760234808796781425770648531050515397882573860944203466937647730300686328832367900971519239255169156414315857290427352708227864853 38,
3124969928607648946371109874365345160344212144055524459424410185589154989886453461971433538027481940533140896999194484291431364158434980441288626433783531241521601475468180807466 81,
4536745019860639720186768846538902802638655474618670821537032043452907322639593799267821024183563516445976081904016222797924274043765845349561524452331501623260713364151358187350 87,
3666309706328662834994418318664590477285025391964348470066944991044026550326728968254857568968736307247462332879539763206335509631282052944150111379917740835064491707985037520494 00,
5930561569329184875439725806123711270419016768874695648578552290798597780431766290300703124420140945933218016622461924730660063500518716281884625545010317453275917669236675373978 21,
5976402562651013468560794151273330387517344195376351927094635034970843589950967232797028041333084558556806412914509378739569250368241429817465613532249213905034367662025470855119 23,
2300870898643955123327988385182158788264907356016676170889586995876698413161111921570255060910805260240581795701446198488745922118853633220348153165592269840572132764501828093287 0,
2218565745553937824235401912129572112298029981976093184848228301840898596301052535229415103505993079392924271225840885023526024678374085482070472651959013570460607968092723594123 08,
3040711348571264035543919818291291058916121882829710924496139199061039477523712583689953962044724703355630690062450304798880295039905444163708079439230997813438251081289005380450 24,
4685759289444703240170658558362794687461215234968682590669448775926104080636532459647917770763637270006143528080057278394448977416646111355273720182537591221291988635750534445245 29}

In[11]: = g = ABCCodeDatei["EnText","ABCText"]
Out[11] =
{Cv5xbI2tmZu9ojGy5pj=s6weOr:sd@q1pe<7pfCp3oe17ocV1xg64vd46yd@v5umJo7yhXu4yjB>8uiCw5yfPw8wfNx7wa@3rbGo4zlGq2pgT7ti_q2qh98pb]o4yfQ8sdBuX4piHw6wcOr5zaR

s:ogPy7vn73okTx5tjS4yj:1ueVy8td75ucJ4sa29tcNw7rd^u6zeNB7rmF6piRu4rhY9piK9qmIo9 xlHq5ybPu1ti9z8wlWDsVz7qk27phTv2vd,
Lr6rh]t3xj9y9ocSx8qiZ4vi28oaIp:reUq6rb@v8seLu7va=u5piO3qm^y3zjQz9weE\o3ph21weP 9seTw3of3w8u9qn^y2zhIs9qc^4pg]u9rcPq1qc6w9vl`o4qe=J7yn@q1rl>p6rl9w4veH3tf1q5xb Es:plGx8uhH1ph1r6uk]o2oa1r5yb=y9qm?3vk^z1oaUx1ue\\5sg4u:rmMp5qhYw4pd29oj2r3w bHv9tk64ti?u:odUMq1zgX7uiN1uh;r1pkE3te,
11wm>u5rjM9yd79ugP7xfTr7unT7ykT6ua58tdTz9pcA4xbP6om_4veB9zeT8wh86wn1z7qf]8u l\\y4qdR4vj0u6ra]4uk^t1sbU6wd\\t3rgH5qb@o2xl[9tmNy1siR9yh9r2tlE7yj>4on:q9wn0t:zkX y4rcT5sgEs8wgVo8riEz1pd7u3zeI4yk36skS6og[t6tjF9tl>3un[x6wiKo6vc7o5snEz5zdX2rhHz 4sdIo:s6obO2va1t2sjPu:sbKw7tjJ5qn,
:3rhIv5riO9qeX2t4rdJ8riY5om:y:pm3p4tc9u5qnIp:vbAv7ue[3qi[5taOq4ucV8qbI2zbO9xbK8y dIwKs5tiSs4w1uiIw1ra@vW6shR3qi9q5qi@t<r6raIx8tdXx6tj@tU6tg36um8v9qk^J9viB4vmS q4taIv2wf@6xg66sa`y5un>8rc`p5vgRy]2onHqR1zdL6slT2sl@7ueAs1sf9xN6uf<1qkCxTt9yn Ru6xb,
Iv2zgV7qc74tj88sj_x2rgNz9ubN7sjO9tcJ1olKs3sm75vaHuC7ykPv7smEs2vm?3tn9v3zmD1wb Ky3yh77ta<6oi\\z2xe2uIp4sg3w6xl31sh:8zjA7sn@z3teY3uj^pL5qc0v9vf[w7ueM9ra^3yk`r1te 6p4ta32zeI9ohWp6oaWI7sdAx3qcP4wj\\o8zf]6ub2u8wc0r3vkA5sd^q8vjHv7wkD5unP2sj9r7o c^z3zi<s9ua[7sdZ7rbMq5pl,
Np1pe[u8xbJt1ugQ3xb2t:ukPu73qb67tbR1zf<t3sgS9rcSz9tjUy7yaJ2whZ2wbDq7vnDq9rnF2t g^yIr1vi47xn7t7olP1whJ1wm8u5ocO7onUy6vg\\wM1plE4re3x2sdO?p4qf5r4ui:8qdE2qm<z7 qi46ofTo8xaFv6vj8q8wbFz9ye=t6zj>^z9re<3waIo5qcM8tgQq6sf6KJ7wmMq8vi_7pbWt5pg: 2oc41sn73ym6yI,
Bo2tf_1pkF5ya73pc^q1vh1p6uaR6qbKq8ya:r7tfV5pk\\6vbPs8uhAy8xdBx2reLw7zi_Yu4rkEu 8of0p4zeH3sh\\r9zi4o7yk54piIp7wn]3udPt:wlS6taT2zl:1se58sm5`u2wi;r5rd84yhV9sf]uT1yl] _9v3vc\\y7tcPo4qm42xdS2rhG7vg^s4xh<r8un@9zf3v9xmUp1qcVy9xgA1plVp2vg8y8sjR4ubZ 6xf<y2vm171rjJs2wi,
X>pK6wdRu8qfQz8ve6v4whB2tkLx3yeMq3vk95qh;t1pk@1zdKx5ufWq7tf36paV9xaT2rg=2ye Nr2ua]3ofK6qlO3w7qc`^q8rjTo3sk5r1unO8xaN5ybHr7tkQ4uhJ5yePx7plEo3rm59zaI3xg]u3 wk2o2vm99veV1ugP2saOp7zc@4sgA7thTq3pbA7sdCu8pk<u7oeF7pfCp7tc32ye=t8skMz8ri_ 9xe0z7ynKo6sh[y4okK2ygBP,
75sc4p6ojF2paCz4xj41va]3ql@w:ve@w8tn82oeX61xdMtRy8yc>5ra28smF9sf1x2vn[2xhUr8 raJsG9sfN9rl8y4tm=Kz:rgO4sa0r6viG9yf\\5rd5t8rlFq2qh<8tnRw:ze6z6ycZp2vkBt:vgU1um V2vgGq6uhG2sf9x7rf8o5rjQ2pg_t91ue2v3uj=r5rhJt8sd@9yn^o3vb@7tnQ1oaT9xkN9ve`p2p a59zcPo2ynXx9rh?v1thQ9rd,
I3qlJw:ynW9ubFHq4sn5y8wmG6pf<v3zhSw@4qmQ2ug8w8zhDOx2uc_5tkLs6yk[2ue54ylC6 tn8p1qiW9odC8te49unUv7xb_y6rg=9shKp7ve]7ql<q:oe>v:qm2y9rh9=z8zbTz96qkK7shY7w g[x3veOr5qiIy4qc>v2tb]6rf@o9tnU6zkU5xkP9wjKs1veS8olHs4rhX7zcIy8zcK1ogJz4ul3y5tg 78plJp5vnBs7zjLt8qd3o8qaN3wd,
F3okZ9riCqOt4pdQz:peA3ojPr6rdW7vfRt5vk>w9xm^y7pm0u7plU1yc@8xe83oe?2wgD6sa] 6ueF5y8tgH2pk63yfC6paA2paMz4vk[q2yb\\r1re84ubF3ohHx1ygKz7wjSv4pk7u9qj6w8pk;z9 ye<3zmVp1waA8vb6w96rkFw7vbRo2xe\\8seM8uf6y6ve5s7waVq:xlO8pa_E2uhM2vdM1pf\\z 8rm3<9va<x9zgVw8zeYs9obQ1oeL,
?v@x7xkCoF1zj;1od;tG5rnK2zmYw2obOp7pmLv9rd75yl43tcR8uh[26ojKp7qhQrWw:ql?x:u cGo2qeNr6ymFx3omHu2xm@u5yn99yiLv6uiP9odAv:pi6y6zi3t:xfPv5weK1og7u9ob5p3te@9 rbTq2sfA4wm[q3xl2t8tdA9rgBo6zfDy9wg?v8rlM9oi2s9ucZs3shEt2pn41vj<p4ti98tgM3ze=u8 vhF8xn@1pnIx9rh;3omE7tb\\2yj[q3zi,
Y1pfXx1xlCs5xaKy4pfRt4vd;o2xgI3ucYz6ulNt9uhW4ukEp4wf`p7uiM4rm29zhWq8qi>6yjDo2 raM4xj=6ocW1wk7w8yiX8piPv1oa51ue]w:y2qi=s_4ujT9odJu8snU4zfKz9qnNQ2wa@2xfWr 8peJ5pfWs9xeAz8sc?5ucQ9xl=1tmS2onGx5uj1y1qbNuH6vn35yfX8wmJ3xj1pV9wkC4tl]w8vj >y9wnQ8qaZ6od`s9peB1phBq2pf,
*Dz1qmP5yjKs4rj71obAp8vcO3ufHp4o4qaL4ri[o6rm34wi<r:tl[wU8skY9zd=o7qjN2vd^*

z5tb:w8tcL7zl_y8sfQ4vhGo6sd>o4yaNs8wgMo2piQ1sm\\oFs7oa;8ykGp7vcPu6ua76peB7we 1u9yg@r8rkZt4ohV3oh54vf7x8qmX5qkSy1xh;p9qhH5zcErDz3sa:r6ue14od`q1rk7q4phJp7tiI 1yc63qe_s4wj[r3okD4ud]q4zl16zm@p5ybTx:qh,
^3rn?u3zi\\9re51sk>t6rdP7ykN2wh15uh@xY1yc73zfN6ya\\5znI7rd=r5zj2Az4xgH4sk^q7wcD t1omMw1wiJz7xj5p9vkEu6xf3r5xnEr4oa=8pgNt:piH6yf@o8rfQz7ofSy2zc0w8zhB4onX9vfR2 uaZy2uaY4re@x8rdQv5yjG7yaJ3ocMu7omN3okUp:zlZw=p5xbHv2ugFv9sj<y2vcCz3xiA8t6 uiK9xeN8of8t:ue_7un;o1rdNw5xjDy6og,
T3ynD6xcQ9wl99teHG6zhD9ulM5okW4zbWx6rh:K7olJ5rhX8yf5s6pgX4oe4u9vm_y9ylM2wn 1uYy9qjSs5sgI8wm=L4ogLy6zm33xi1w2snC7zc14thYy8vk`r1xhHr7vhKv3plCv9tnG9on1u9w cQ7se49onP6ta>4zhT7xd1t2rhGq3ql@o4vnDt:oiD2vhFq4zc56odQ8skW1ujHPt8zfT8rmE6q gF1zbK4vfX1u2zgT8yg,
Ju3pb36pk73ua\\3snEx2yfY8ud?4va41qiY9sgK7snJv3vg\\1qj@8ulYvQt6pg8t=p1sc\\vN6wcJ Ty9rd22rf?y9sm<2ri]q1qd=y6on27pkOs4tj=8ygVy8oh3r3rdG6ydXv7rdPw7wiZt3xeQq7wnN q4uh38u3oiKo7ydT1olW1wb]3tj:o5wiVp4on?t:pe\\x5tgR6rc>r5qf:z1ol[t6vhLs3xn[p7tb1Vs4 oa=2reB3rkF2u6obLt2sj,
Pw1vnK2xe6v4weWp6vfDv3ulO3xbQx6pk1y4oi44xjL9xc83veR7xf8s4xi\\4ti\\6pjJs5sh=q2zf@ r2qnWp:9yjH5wbA2oi:8zhH3zd4t6znI5sn`s4og3Cy1vlO4rhU8pl_u4zl0q4rk]5pk_4rkY1ykL1s a93w:okUx9wnFv3yj?v8It7wb<1yb?7pn=t3xi?v:ufHw:ra47oi53ubFw5sfEs1sfL5td5p9xn=t2 onK2qeYz3wkQu3pa^x8xj,
Xz2qj?z3raGzI5rfFz2vc3p9viXo2rjZ2pg9t5yhUu1ti8v7rdTx6ye_p9wlRw9qlTz:paO6qiQx1oc6 1zi<r:vh8z6qc[q5rhM7zgYr7ukVu1zlJ2tfE4xb7o8olVq7rm>o6ukD5viN9ze>t2odW9pcW5za4 x3pm65sl5q:qc:z4qc@3ogBx1ziLp8rlV4qi9o3zi19pfV1pnOs;4unU3re5r7wkNE6vc_tC1qkY6o gN1sm[u6ufXs9rfPq9olFu7plYx4zj,
]z1ucTyIx5tjZv6tb[w9re69vnC6tm^u1odC6oj94pgFCs2ybR7oi?q9yd0u6riZ3tjV4ugXp8zl2y4z mYo5wb4s4oe]1v4zdADp6zkOu5ymO9pd^5vmIt7qgI3pjFs6wb:7tg0z7xk9y4sn3t2rn?3yk29yg 3=qR7rg@3yb65yb^1tbE7tj_6skKo1wl59uc:o8wbFq6tjR9t4qk<z2qjMv2wb71vn0s4rc1x7vj0 u4ra23yfS4zkXo8wl,
5p9tlNz8riIt1vb7t9qh^9odSs1rg^q7yg3u4uk<q2sg6o:ri=2xnLs4riEr3qnWy9pcHo2s5si[8xh6o X6qkEv\\7uhQ1wg@5ydIr:om?y5ya52rn;t2qnZ7vcQ1qkP4xnMr8unNr1vmNq2tjEx8wcI1va3 8vkPM8sj>5se=2zbM@v9yi^o7uf5t:xk4o8vf<5ub=q2teUq5om;5pa69th?s8wlE9pn\\4onBr4o j:7uaQs4pmZ9tb:7ri@z6xe,
84xdK8rf\\9pj8wSv1ofJp8ud3q2rmFt5phQ2ti8o3oeW3wd42ya1t9sb>1xd1w1xiHp3xe8v8ofGv 0w5vl?t9zm6u7uh^r1xd1u6tn@U8sl9o1zd3;5pnDv8vaA2yg2z4qm^To4pe:3qlEw7yf\\7xg5s8vj NtD86rcCx9zh\\s2yeFq8sn1t1wg71yd;9qh=9rgTr5rf39ofA6og_o4re9u6sh97omZy?7zlN2wj6 8rdBv8qe5q5sg23on,
Hx3zc@q5yfPv:sf\\8zh\\7vgNv9rl=6pc>x1qnKy:yj44zg65ofTs5piRxUy4yhFo2uaEp4vkH5tb= 4qdN9pf94waBxN5rk39ocBr5raLw:xgE6se:3sm5s2oj_v1qbA3qmRt8wcL7zm2o2xf5Ju3zdYK w7sg79ph@4yb]y4re_w1obXz1xkOy4zdK8xfBv5xc\\yPu1yd^8qgC6oaZ3tcA3zeLu1sc81ug@1 sb31xiB5pa28qmPJ7tb9w7tl,
E4xf2p7zmEv5rgJu:sg?<s:ydBz4u7yiDv9pnM6tnJo9yiX2qjG2zd1w9wbC5tfM4tf3r9vhR91on Ou7yi<s2sc93se2q3xk<7thG2yc_9xg[3zg>9wj>v8ue6x1shRv4oeD3wf?3ybD6uaEt6tcPv6sh^ 1waAt4ui^v6odW2unHv8of[x4rnDz4raX3on8u6wm65xe\\9tc81ua=4tdYo2yk61tiB1seVr3sdG` v3thOx4pkWv1tl28ra?7okW,
N3wg66rl:t:ul86sf2v9pm[2onCu5qi94pi?8vm2s9wjK4sc_s87sfFs5uf;5ua4s2qa24snO8pm6zF z7rgCo5zm8t1riBz4scO6zdPu0q7tjPz9oh98xj5q5pkT7ogPt9rn9r9si66ym47qfJ7rlYy2ym_2piY 9uk@o:qgPs3rj96ybF5ue3x:qcPz1vb=t2wl6o3rmOr4uc;wYp1rhOu:xd6o8vhSJt5pkA4ujJ1rm Nz8xbGw5vaP7sg1@v4se,
Ar6rm8<1ylD9qj69zf7y5sh?5uj<o:ug4w6wdTz6ocS3pmOt1rhYwUq9sg4v7okGw9sfU4zdT5o eY8onW8wkUw2pnO[8rh:u8rj[rP1za56vfNwFHu3pi:4pl>z^Ay6tiQs3rcQx8zcV6pc62zn0x2pl Y4vaB6y>3pb37s9zh46yi\\1pbN4ucJ5yhEz5vmOpI7pkWv3oi?o6td^7qa;9ye[6vm95ya=r4oj58 td,

Ey6pg@6xa@2zcI2yl5y1pk:3olX8pkOOo4yeOp1sd]p3znVuO7oi>7waSs4vb95ua;6xa0o2wjU
x7ufO4tiGx5znAo5rkTF3wfSy:qm`37tcW2yfNx7siTp8pmUu1zaDx3zgG4xmW8yl8w7qmA6qc
3z3xgB4ulI9xc9s9qf\\9yd=9qkTt88tk^r1zjEw:ogJ7reQ5uaA3qfPy2zh99yk6p4xdGt5qn44on^7
sb:s2rgC5slUOx5ri;9xe,
Gu8xTr7plJz:rd74tlG5qnYt2rkW2weCs9wnS8xh58oi;r6reBy9snI6xmIP5rlF1ohY7xaTr7qlOu
9sg5s9sm^9zk^7rgHq:ohSw1ogSq6xhGu1udGq2rfNo7wmF5vlQy6uf<y1yl4Jo6ojEp4pm18ua
]9vn>rM4wfHq:wkGr9vfK4ynMw2zm\\1sm85ve\\p3th;t2vb1x5re35rlO6qd:9xnJs7pjEo4yf17y
bWv3waE6sbBpX2vjD9wi6z9pe,
My2okX8qf^5ykK5ze9o4qd>q8zjWp4on<5rn_1xaDz2zn]9sjCq2phUp2qjRt8ocVo:vd5p2zeQu
6taNr3sj4s5og68ub>8wbX1tnY[8ub66oj9x1va=2vjZo5vlM9sbNv5om>3ogGo6zcCt3vfK9yi4
w3ph[s4xkJy1wd8w6xhXFv5ya@3tcJt6phS4rhE8wjHz4tgD5sk?s2wb_wI2ol3y2wcUo9veE3tj
^x<Lq8yhYq8zaP2udG3vcLv4tgDo3rb,
@x3tbGVp1qdY8phNt1weCz4umJ5qn7p4ynEx:ti=8qjQq1oi9v1snKp9tgTz6ob\\1siZ2wk;r:zk
A4tiUo6zf79zcZp9ok7v1ogS9znP7ybS3phB2uiP3ymSr:rn<x4un<w2raX4waV4tc_q3zgS5wgE
6w52tkNw_4rdHx9wkMrGp5zgNy:xe@5zhPp4rkR3ydWx8vkO9uk;9oa3u1vc=v8ua73rf98ph
<x1umPx1xfTx9pe:5tf]qHo,
T4zn_5uh@9qfP9zhV4plTp3rgAuAW8wnD8skW2rfT4xcJ7ohIq1ol[w5vc@t5qd4w9smC2omB
P9od=y9rj@y6okRz6raHx4wc?M6pn52yb4w:tkGx8za9v8sjLK3tf77ub0sVz6uc@3ymXs8um5
u77zaG8zk<4zcQt9phE7zjX6wmSt6zjWu1ql_q5vnT3wnBt=2xnW7wjJu1xeQt1pe6s2qc_8zf_5
pb5<4xg\\p5yi15yk}

In[12]: = ABCDeCodeDatei["ABCText","ABCEnText"]
Out[12] =
{49926590650358999111273230772118243862849669842406683049980070235240721575
62398254298490184726005119947043435164671548065354589147026220787386464189 5
39398895041287770870207107222 28,
60433387380347481898561378470049336733489706933214694432400387223888947890 9
12887043206174611259039336048198179116857723748187735114213357296450942591 4
29088843897508512576017410735 7,
14549494191975270875666082190044965647993085263267386142846657647714266133 2
55220895512694620174144687496042456478980611729545762765961993435061960348 7
50128040802558627119399269955 4,
34650594308838684256997038748998568937957141585426594384302550703510012101 6
66337048901658879361601666162787209493845041921764063052883052803428502417 8
65227673611300965415601990061 0,
21775843982423989475094317031553502473970421335129716431572763623600240957 9
14782673730437704655686471594138214439024393761508774530546481266977028854 5
81469654827268133487676376250 9,
12681112236996203234372617830094789970926628269288426204612172467917617048 8
75161304417345319507939938982626168462880459877688658906291034850584560905 4
07574679776751925014834300607 3,
21019155034314257765879568256666880078521070009542578938436090867643769738 1
99265927615482105324049144095004518009309529771641624427574343078092886713 3
90216123077646666620404915121 8,
88014651802801390258316317537107160507711627950276253222381651079675096814 3
24101847558696465554704309917361333194921182272703448737324737787684742691 2
43781797959672603427267066080,
53558026240314838197678424808815002980254281893419328380502693594239206198 7
46357693557277321482899458223497818127859926368148827304715329049379492833 4
73617794894224091722490411194 5,

36198695107040877364730003544827377606822357560428343764310495183591670962392569976597297486605778103663374776133151140920666988066156594412385140876079317141048382350669079677136 6,
35895401941599335360875951487871867218183234425564466907077326434264325741323514044035812069941987177487534813483300663369422686084558267799884709526126917381205192487680690517707 6,
15688019166155011551267233703892535340850091626699033999975215606310216496921604944977582963512171871293928228449331770929594884783941866330213148396825365781838160889347741284331,
18513649941141820336161789447140572046191480763821246163718377585612814904520201347994879446989107827325680755490378554394515727550309703064853185636200195045181387884966691216221 0,
11656041115178536702438846061943597204385395368527052677874881546559939879822117704469482769560863173886578442637443777556813187555202130858643314439169816934243133145242962549699 6,
38230395614959075227052901618534365457274549305040145671811612569886959748340582999064878470522776845095227423447378451374336270136699704249321688620430706558493984696876710751059 6,
37264494592107264793255747161505874755485658147286891126900190551584106146060433919373943380914469631588393586575391665644675619331140098026040662984918307280085264516645905808819385 2,
31462734338021385744714895374731418883204160900810904465407490024287925214158972241582980629164771270433070541405129669918018338149042297552465849410661931671504942225335407057822 0,
12826739061530836760938543694034475439247666850620520803992255225782636058055343205111546385343138457347918517205796390131001505669715673530197598472434150211755686920526732932181 4,
23230302355321486723228248913368061291189899665512915097258452374971313426645377571059054894220695156729253496539435211479647029791318203144337367707864504716366517461990489669442 5,
14103650561890791864314264344107021175487957637847080838652138809338506559651194357469736159873067239319535191405101375435253419074167112391777859808239320422115138467338433846780 8,
86979812187195813171838720496625341230990302448285900666402176417754498549524520376217746479712622278916981908084554225307788071896577254020551753991878393347640273041795473059 2,
44084395600813179419550227329637524486515809731277602348087967814257706485310505153978825738609442034669376477303006863288323679009715192392551691564143158572904273527082278648533 8,
31249699286076489463711098743653451603442121440555244594244101855891549898864534619714335380274819405331408969991944842914313641584349804412886264337835312415216014754681808074668 1,
45367450198606397201867688465389028026386554746186708215370320434529073226395937992678210241835635164459760819040162227979242740437658453495615244523315016232607133641513581873508 7,
36663097063286628349944183186645904772850253919643484700669449910440265503267289682548575689687363072474623328795397632063355096312820529441501113799177408350644917079850375204940 0,
59305615693291848754397258061237112704190167688746956485785522907985977804317662903007031244201409459332180166224619247306600635005187162818846255450103174532759176692366753739782 1,

59764025626510134685607941512733303875173441953763519270946350349708435899509672327970280413330845585568064129145093787395692503682414298174656135322492139050343676620254708551192 3,
23008708986439551233279883851821587882649073560166761708895869958766984131611119215702550609108052602405817957014461984887459221188536332203481531655922698405721327645018280932870,
22185657455539378242354019121295721122980299819760931848482283018408985963010525352294151035059930793929242712258408850235260246783740854820704726519590135704606079680927235941230 8,
30407113485712640355439198182912910589161218828297109244961391990610394775237125836899539620447247033556306900624503047988802950399054441637080794392309978134382510812890053804502 4,
46857592894447032401706585583627946874612152349686825906694487759261040806365324596479177707636372700061435280800572783944489774166461113552737201825375912212919886357505344452 4529}

In[13]: = DeCryDatei["ABCEnText","OText",d,n]
Out[13] ="Kalifornien (Golden State)
Kalifornien ist nicht nur Hollywood und San Francisco sondern auch an der West Seite der Majove Wüste bei Barstow an der Interstate I-15 die Ghost Town of Calico. In dieser gut besuchten Geisterstadt kann man schon einen ganzen Tag verbringen. Die Besichtigung eines der dortigen Stollen, in dem mit unglaublicher Mühe nach Gold gesucht wurde, hinterläßt einen kaum nachvollziehbaren Eindruck unter welchen Bedingungen die Menschen in der Mine gearbeitet haben. Den abendlichen faszinierenden Sonnenuntergang kann der Besucher in Lils Saloon beim Bud Beer mit den vielen anderen Eindrücken des Tages noch einmal Revue passieren lassen. Es ist ein Camping-platz vorhanden, der für 16 Dollar incl. Strom, Wasser und Abwasser-anschluß zu haben ist. Weiter auf den Spuren der Historic ROUTE 66 kommen wir nach Durchquerung der Majove Desert in Needles an. Schon Meilen zuvor auf der Interstate I-40 kommt ein Schild mit dem Text, es ist tatsächlich eine Oase, die hier vor einem liegt, und nicht etwa eine Fata Morgana. Needles ist laut Statistik die wärmste Stadt in den USA, ohne Klimaanlage nicht auszuhalten. Vorbei geht es dann an den Sacramento Mts. und nur wenige Meilen hinter Needles überqueren wir den Colorado River, der dort Kalifornien von Arizona trennt.
Wem das noch nicht reicht, der kann ja mal im Death Valley National Park vorbei fahren. Dort werden es im Juli schon mal 55°C im Schatten. Die höchste je gemessene Temperatur betrug 57°C. Man sollte nicht gerade zu dieser Jahres-zeit durch die Wüste fahren, wenn es nicht sein muß. Einen Tip hätte ich da noch, wenn Sie mal durch die Wüste fahren, schalten Sie das Licht ein, der Gegenverkehr (kommt ab und zu vor) könnte einen sonst glatt übersehen."

In dem nächsten Beispiel möchte ich auf einen Sonderfall des RSA – Verfahrens eingehen.

3.2.2. Verschlüsselung mit e = φ/2 +1

Das *RSA* Modul wird nicht verändert, nur das *KeyGenerator* Modul. *e* wird auf φ/2+1 gesetzt und d wird aus e berechnet.

```
In[1]:= <<d:/Diplom/Notebook/RsaMethoden.m
Out[1]:= "RSAMethode`Private`"

In[2]:= KeyEGenerator :=
            Module[{i, p, q, φ},
            p = NextPrime[Random[Integer, {10^80, 10^90}]];
            q = NextPrime[Random[Integer, {10^80, 10^90}]];
            φ = ((p - 1))*((q - 1));
            n = q*p;
            e = φ/2 + 1;
            d = PowerMod[e, (-1),φ];
            Print[„geheimer Schlüssel d:“];      Print[d];
            Print[„öffentlicher Schlüssel n:“];  Print[n];
            Print[„öffentlicher Schlüssel e:“];  Print[e];
In[3]: = News = „Nachricht“

Out[3] = „Nachricht“

In[4]: = KeyGenerator

„geheimer Schlüssel d:“
279238669472420731063851332112826335342742080979442868492698235745137913434
109390402571734157465345748874527396546207594978471963560447858537854544829
309750353719289289422810596225

„öffentlicher Schlüssel n:“
558477338944841462127702664225652670685484161958885736985396471490275826868
218780805143469867609146085659432469954927832448364383398140117593091275411
600294166422677628976667602253

„öffentlicher Schlüssel e:“
279238669472420731063851332112826335342742080979442868492698235745137913434
109390402571734157465345748874527396546207594978471963560447858537854544829
309750353719289289422810596225

In[5]: = Secret=RSA[News,e,n,1]

Out[5] = 78097099104114105099104116

In[6]: = DeCodierung[Secret]

Out[6] = „Nachricht“
```

Das erwartete Ergebnis tritt ein, der Originaltext wird in sich selbst verschlüsselt, $M^e \bmod n = M$. e kann einfach φ/2+1 gesetzt werden, es braucht auch nicht getestet zu werden, ob es eine natürliche Zahl ist, denn φ=(p-1)(q-1). Daraus folgt: da p und q Primzahlen sind, sind diese ungerade und (p-1)(q-1)=2n*2n=4n gerade, und somit ist φ durch zwei teilbar.

3.2.3. Verschlüsselung mit $e = \frac{\varphi(n)}{ggT(p-1,q-1)} + 1$

```
In[1]: = <<d:/Diplom/Notebook/RsaMethoden.m
Out[1]: = "RSAMethode`Private`"

In[2]: = p = NextPrime[Random[Integer, {10^80, 10^90}]]
Out[2]: = 72747091950971350249462932657396878541329096523330310344761791099421496410600868679984343
In[3]: = q = NextPrime[Random[Integer, {10^80, 10^90}]]
Out[3]: = 37838764799632606847438075367338645549516325588188747544376892626531676200694421706498562 3

In[4]: = While[GCD[p-1,q-1]<3,p = NextPrime[Random[Integer, {10^80, 10^90}]];
          q = NextPrime[Random[Integer, {10^80, 10^90}]]]

In[5]: = p
Out[5]: = 45111335082226538011075618407177215291000630824866893401859380893281625959656657956926869

In[6]: = q
Out[6]: = 78900088437276266641946556129841011491628273621123482385575961178536533787970582465469820 1

In[7]: = φ = ((p - 1))*((q - 1))
Out[7]: = 35592883275112772729991009730255194436133683885290316760912227878381895579718926323739816895965200195563724102516510583111277439863694186875066580377773021522920950409493001123760 0

In[8]: = n = p*q
Out[8]: = 35592883275112772729991009730255194436133683885290316760912227878381895579718926323739817019976623715066528755538685120129504222492598632865442367813115093341080684345741262286266 9

In[9]: = e = φ/GCD[p-1,q-1]+1
Out[9]: = 88982208187781931824977524325637986090334209713225791902280569695954738949297315809349542239913000488909310256291276457778193599659235467187666450944432553807302376023732502809401

In[10]: = d = PowerMod[e, (-1),φ]
Out[10]: = 26694662456334579547493257297691395827100262913967737570684170908786421684789194742804862671973900146672793076887382937333458079897770640156299935283329766142190712807119750842820 1

In[11]: = News = "Nachricht"
Out[11]: = "Nachricht"

In[12]: = Secret=RSA[News,e,n,1]
Out[12]: = 78097099104114105099104116

In[13]: = DeCodierung[Secret]
Out[13]: = "Nachricht"

In[14]: = RSA[Secret,d,n,2]
```

Out[14]: = "Nachricht"

Bemerkung: Wie bei dem vorigen Beispiel tritt das erwartete Ergebnis ein, der Originaltext wird in sich selbst verschlüsselt, M^e mod n= M.

4. Zusammenfassung und Ausblick

Das RSA - Verfahren ist ein typisches Beispiel für das ganze Gebiet der Kryptographie. Es wurde erst 1978 entwickelt, aber es baut auf mathematischen Sätzen auf, die bereits im 16. Jahrhundert bewiesen wurden. So ähnlich verhält es sich auch mit der Kryptographie. Sie wurde schon Jahrtausende betrieben, wurde aber erst jetzt als Wissenschaft anerkannt. Ein Grund dafür ist der technische Fortschritt und das stärkere Verlangen nach mehr Sicherheit von Computersystemen. Dieses Interesse führte auch zur Entwicklung des asymmetrischen kryptographischen Konzeptes, das dem RSA – Verfahren zugrunde liegt.
Es wird sicher nicht so schnell ein neues kryptographisches Konzept geben wie das asymmetrische. Die existierenden Algorithmen werden sich verändern und neuen Gegebenheiten anpassen. Zur Zeit werden Verfahren verwendet, in denen beide Konzepte zum Tragen kommen. Dabei wird der Originaltext mit einem symmetrischen Verfahren verschlüsselt und der Schlüssel des symmetrischen Verfahrens mit einem asymmetrischen Verfahren verschlüsselt. Dies resultiert daraus, dass symmetrische Verfahren auch heute noch viel schneller sind als asymmetrische. Zum Beispiel ist RSA etwa tausendmal langsamer als DES.[4]
Die Computertechnologie verbessert sich ständig, daraus ergibt sich, dass die Faktorisierungsalgorithmen immer schneller werden und auch die Schlüssel immer länger. Ein weiterer Punkt ist die Frage der Sicherheit des Kryptosystems. Im Zweiten Weltkrieg zum Beispiel brachen die Alliierten die Verschlüsselung der Enigma in erster Linie durch sorgfältige Auswertung von Fehlern der Operatoren. Das zeigt, selbst wenn der Angreifer nur Zugriff auf den Geheimtext hat, kann er durch scheinbar unbedeutende Lücken im System genügend Informationen sammeln, um das System zu brechen. Es liegt nicht nur an den Systemdesignern ein sicheres Kryptosystem zu schaffen, sondern auch an den Programmierern, die es implementieren.

Das RSA – Verfahren wird vielfältig angewendet, dies ermöglichte RSA SECURITY Inc.. Diese Firma und ihre Tochtergesellschaften, wie RSA Data Security, RSA BSAFE und RSA Keon, helfen anderen Organisationen und Firmen mehr Sicherheit in den Handel im Internet zu bringen. Dabei arbeitet RSA SECURITY Inc. mit über 500 führenden Unternehmen der Computerbranche zusammen zum Beispiel mit 3COM, AOL/Netscape, Apple Computer, Ascend, AT&T, Nortel Network, CISCO Systems, Compaq, IBM, Oracle, Microsoft, Novell und Intel. Diese Unternehmen integrieren RSA Security Technologien in ihre Produkte.[30] Durch diese Zusammenarbeit begegnet uns das RSA – Verfahren im täglichen Leben so dass wir es meist gar nicht bemerken. Zum Beispiel der Netscape Browser verschlüsselt mit RSA sowie Krypto – Software Pretty Good Privacy (PGP).

Literaturverzeichnis I

[1] Albrecht Beutelspacher, Jörg Schwenk, Klaus- Dieter Wolfenstetter: Moderne Verfahren der Kryptographie, Wiesbaden, Vieweg, 1995

[2] Albrecht Beutelspacher: Kryptologie, Braunschweig/Wiesbaden, Vieweg, 1994

[3] Arto Salomaa: Public – Key Cryptography, Berlin/Heidelberg, Springer –Verlag,1990

[4] Bruce Schneider: Angewandte Kryptographie, München, Addison – Wesley Publishing Company, 1996

[5] C. Pomerance, J.W. Smith, R. Tuler: „A Pipe-Line Architecture for Factoring Large Integers with the Quadratic Sieve Algorithm“, SIAM Journal on Computing, April 1988, S.387-403

[6] Friedhelm Padberg: Elementare Zahlentheorie, Mannheim/Wien/Zürich, Bi-Wissenschaftsverlag, 1991

[7] Friedrich Ischebeck: Einladung zur Zahlentheorie, Mannheim/Leipzig/Wien/Zürich, Bi-Wissenschaftsverlag, 1992

[8] Harald Scheid: Zahlentheorie, Mannheim/Wien/Zürich, Bi-Wissenschaftsverlag, 1991

[9] Otto Forster: Algorithmische Zahlentheorie, Wiesbaden, Vieweg, 1996

[10] A. K. Lenstra etal..: „The Number Field Sieve“, Lect Notes Mathe 1554, Berlin, Springer-Verlag, 1993, S. 11- 42

[11] J. Buchmann, J Loho, J. Zayer: „An Implementation of the General Number Field Sieve“ Advances in Cryptology- CRYPTO’93, Springer Verlag, 1994

[12] Ralph –Hardo Schulz: Codierungstheorie, Braunschweig/Wiesbaden, Vieweg, 1991

[13] Reinhold Remmert, Peter Ullrich: Elementare Zahlentheorie, Bassel Birkhäuser, 1995

[14] R.D. Silverman: „The Multiple Polynomial Sieve Quadratic Sieve“, Mathematics of Computation, January 1987

[15] Stephan Kaufmann: Mathematica als Werkzeug, Bassel, Birkhäuser, 1992

[16] Walter Fumy, Hans Peter Rieß: Kryptographie, Wien/ München, Oldenbourg, 1994

[17] RSA Laboratories‘ Frequently Asked Questions About Today’s Cryptography, v4.0

[18] http://www.sonic.net/bear/rsa.htm,1999 (existiert nicht mehr)

[19] http:// www.rsa.com/, 1999

II

[21] C’t 11/99 Seite 21: Datenschutz und Sicherheit http://www.heise.de/ct/99/11/021, 1999

[22] Adi Shamir Factoring Large Numbers with TWINKLE Device The Weizman Institute of Science http://jya.com/twinkle.htm, 1999

[23] Robert D. Silverman RSA Laboratories : An Analysis of Shamir's Factoring Device http://www.rsasecurity.com/rsalabs/bulletins/twinkle.html, 1999

[24] Crypto Gram 15.7.98 Bruce Schneider http://www.counterpane.com/crypto-gram-9807.html, 1999

[25] RSA Laboratories - Bleichenbacher Discovery Q&A http://www.rsasecurity.com/rsalabs/faq/, 1999

[26] http://www.iks-jena.de/mitarb/lutz/security/cryptfaq/ ,1999

[27] http://www.rsasecurity.com/rsalabs/challenges/factoring/status.html,1999

[30] http://www.rsasecurity.com/news/corporate.html, 1999

[28] D.J. Lehmann: On Primality Tests, SIAM Journal on Computing 11, 1982, S. 374-375

[29] Wolfram Koepf: Vortrag über Kryptographie 1999

[30] R.L. Rivest, A. Shamir, and L.M. Adleman, „A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Communications of the ACM 21, 1978, S. 120 -126

2. Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich meine Diplomarbeit mit dem Thema:

Das RSA – Verfahren
und die Implementierung in Mathematica

selbständig verfasst und keine anderen als die angegeben Quellen und Hilfsmittel benutzt habe.

_____	_____	_____
Ort	Datum	Unterschrift