Trave-Gymnasium Lübeck, Jahrgang 1997/98
Facharbeit in dem Fach Mathematik

Thema:	Der RSA- Algorithmus als Beispiel der Public-Key-Kryptographie unter besonderer Berücksichtigung der Generierung geeigneter, großer Schlüssel.
Verfasser:	Lars Böckers
Leistungskurs:	Mathematik
Kursleiter:	Herr Pribnow
Abgabetermin:	30.1.1998

Inhalt

1 EINFÜHRUNG IN DAS THEMA DER ARBEIT

1.1 Kryptographie, warum?

Firmen, Banken, Behörden, Versicherungen usw. haben viel Umgang mit sensiblen Daten, die der Öffentlichkeit nicht zugänglich sein dürfen und daher verschlüsselt werden. Durch die immer weiter fortschreitende Vernetzung von Kommunikationssystemen, insbesondere Computern, die über das Internet verbunden werden, wird es aber auch für Privatleute immer interessanter, Daten verschlüsseln zu können. So hat der Ausbau des Internets die Verbreitung von Codierungsverfahren zur Folge. Dies geht so weit, daß der Landesbeauftragte für den Datenschutz, Dr. Helmut Bäumler, „in der Zeit, in der immer mehr Bürgerinnen und Bürger weltweit in offenen Netzen kommunizieren, (...) wirksame Verschlüsselungsverfahren (als) ein Geschenk des Himmels“[1] bezeichnet.
Die verwendeten Codierungsverfahren sollten vor allem einen hohen Sicherheitsstandard gewährleisten und einfach in der Anwendbarkeit sein.
Ein weit verbreiteter[2] Algorithmus, der zur Codierung von Daten, in erster Linie e-Mails benutzt wird, ist der RSA-Algorithmus. Dieses Verfahren liefert Schlüssel, mit denen nicht nur eine sehr hohe Sicherheit gewährleistet wird, sondern die auch in der Anwendbarkeit enorme Vorteile den meisten anderen Verfahren gegenüber haben. Das RSA-Verfahren ist ein sogenanntes Public-Key-Verfahren, was bedeutet, daß kein Austausch geheimer Schlüssel stattfinden muß. Dieses Verfahren ist Gegenstand meiner Arbeit.

1.2 Was soll diese Facharbeit leisten?

Mit meiner Arbeit zu dem Thema RSA-Algorithmus möchte ich vor allem eine Beschreibung mathematischer Aspekte der Public-Key-Verschlüsselung mit dem RSA-Verfahren liefern.
Ich werde in erster Linie beschreiben, wie und wieso sich die Zahlen, die durch den Algorithmus generiert werden, so zueinander verhalten, daß sie sich für die Verschlüsselung mit öffentlichem Schlüssel eignen. Die in dieser Arbeit dargelegte Mathematik wird dabei nicht nur praktische Aspekte der Anwendbarkeit beleuchten, also das „Wie“ beschreiben, sondern besonders auch das „Wieso“ berücksichtigen. Ein Schwerpunkt wird demnach auf dem Beweisen der in der Praxis angewandten Berechnungen liegen.
Trotz dieser besonderen Berücksichtigung der Generierung der Schlüssel und der dazugehörigen Mathematik darf die nicht-mathematische Seite des Public-Key-Verfahrens keinesfalls fehlen. Dazu gehört mit Sicherheit das Vorstellen der an der Entwicklung beteiligten Personen, ein Einblick in verschiedene Anwendungsbereiche, eine Abschätzung der Sicherheit des Codes usw.

1.3 Übersicht über die Facharbeit

Aus dem, was die Arbeit leisten soll, ergibt sich die folgende Themenfolge:

- *RSA allgemein*: Hier beschreibe ich die allgemeinen Aspekte des Verfahrens. Außerdem vergleiche ich das Public-Key-Verfahren mit den Verschlüsselungen ohne Public-Key.
- *Mathematische Voraussetzungen*: Um die Beweise, die wie in 1.2 beschrieben einen Schwerpunkt der Arbeit bilden, führen zu können, muß ich zunächst einmal einige mathematische Zusammenhänge herleiten.

[1] [37] S.12
[2] [12] S.1

- *Das RSA- Verfahren:* Damit eine Verschlüsselung mit öffentlichem Schüssel möglich ist, müssen die benutzten Schlüssel in einer bestimmten Beziehung zueinander stehen. In diesem Abschnitt der Arbeit zeige ich, wie die Schlüssel generiert werden müssen, damit sie in dieser Beziehung stehen.
- *Effiziente Implementation des Algorithmus:* Hier wird eine Möglichkeit beschrieben, die Rechnungen auf einem Computer durchzuführen, wobei insbesondere der Umgang mit sehr großen Zahlen ein wichtiger Bestandteil dieses Abschnittes ist.
- *Die Jagd auf große Primzahlen:* Für die sichere Verschlüsselung werden Primzahlen benötigt. Wie diese hergestellt werden können, ist Gegenstand dieser Kapitel, wobei mehrere Methoden vorgestellt und verglichen werden.
- *Anwendungen des RSA- Algorithmus:* Mit PGP hat Phil Zimmerman ein Programm geschaffen, mit dem es für jedermann möglich ist, sich u.a. den RSA-Algorithmus zu nutze zu machen, um Texte zu codieren. PGP ist nur eines der Anwendungsbeispiele des RSA-Algorithmus.
- *Auswirkungen:* Zum Schluß möchte ich die Auswirkungen, die das RSA-Codierungssystem hat, darstellen. Hierzu gehören z.B. die Probleme, die sich für die Geheimdienste ergeben, da auch für sie der Code zur Zeit nicht zu knacken ist.

2 RSA ALLGEMEIN

2.1 Verfahren mit nicht öffentlichem Schlüssel

Eine Nachricht zu verschlüsseln bedeutet, sie durch bestimmte Algorithmen so umzuformen, daß sie, wenn sie über einen öffentlichen Kanal transportiert und dabei von einem Lauscher abgehört wird, von diesem nicht gedeutet werden kann. Nur ein ausgewählter Personenkreis kann mit Hilfe des Algorithmus und des dazugehörigen Schlüssels die ursprüngliche Nachricht wieder rekonstruieren.
Alle Verschlüsselungsverfahren mit nur einem geheimen Schlüssel, sogenannte symmetrische Verfahren, haben den entschiedenen Nachteil, daß der Schlüssel selbst von dem Sender zum Empfänger transportiert werden muß, und daher ein geheimer Kanal von Sender zu Empfänger vorhanden sein muß. Er dient, dem Empfänger der Nachricht den Code (Schlüssel) zukommen zu lassen, da dieser ansonsten nicht in der Lage ist, die Nachricht zu entschlüsseln.
Abbildung 1 verdeutlicht dies. Der Sender der Nachricht ist mit einem S gekennzeichnet. Der Empfänger entsprechend mit E. Die Botschaft, die ich in codierter Form als Nachricht bezeichnen werde, ist mit einem N abgekürzt. Der Code, also der Schlüssel, wird durch das C symbolisiert.

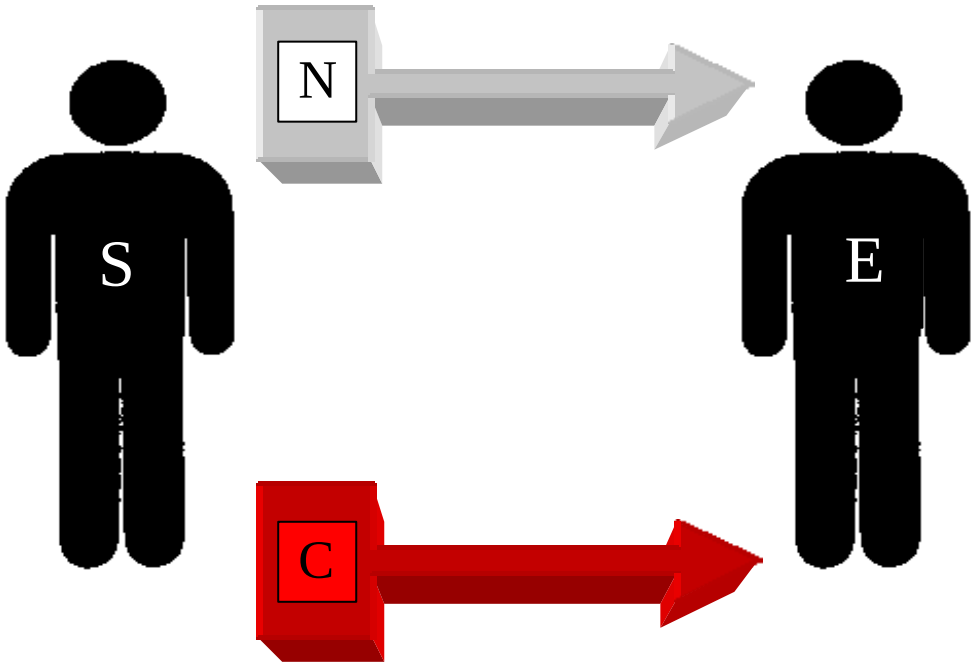

Abb. 1

Der rot gezeichnete Kanal ist ein geheimer Kanal, der in keinem Fall abgehört werden darf, da ein Lauscher so an den Schlüssel und folglich auch an die Nachricht gelangt.
Ein entscheidender zweiter Nachteil kann sein, daß der Schlüssel bereits das Codierungsverfahren beschreibt. Die Kopplung des Codierungsverfahrens an den Schlüssel ist sehr ungünstig, weil dadurch auch das Verfahren selbst geheim bleiben muß. Das bedeutet, daß auch eine Abschätzung der durch das System gewährleisteten Sicherheit nur von Personen, die den Schlüssel und somit das Verfahren besitzen, durchgeführt werden kann. Außerdem sind alle Personen, die an der Entwicklung des Codes beteiligt waren, bereits als „Insider“ zu betrachten.

Ich betone noch einmal, daß dies nicht bei allen Verfahren der Fall sein muß. Meistens ist das Verfahren bekannt und die Sicherheit des Codierungsverfahrens geht allein von dem Schlüssel aus. Das von mir geschilderte Problem gilt demnach nur für Systeme, bei denen der Schlüssel bereits das Verfahren beschreibt, wie es im folgenden Beispiel der Fall ist.
Ein berühmt gewordener symmetrischer Code ist der sogenannte „Cäsar- Code“. Der Name entstand, da dieser Code zu Zeiten Cäsars benutzt wurde um Daten auszutauschen. Er funktioniert, indem jeder Buchstabe des Alphabets um eine bestimmte Anzahl von Buchstaben verschoben wird. Dies ist aus heutiger Sicht sicher ein primitives Verfahren, aber der Code muß natürlich aus der damaligen Situation beurteilt werden, in der das Austauschen von Botschaften in codierter Form nicht üblich war.
In dem Zusammenhang möchte ich noch auf einen Punkt hinweisen: Wenn ein geheimer Kanal vorhanden ist, und das ist er ja, da sonst kein decodieren von Seiten des Empfängers möglich wäre, warum wird dann die Botschaft nicht direkt über den geheimen Kanal ausgetauscht? (s. Abb. 2)
Zumindest bei relativ kurzen Nachrichten besteht so die Möglichkeit, sie schnell und ohne Codierung austauschen zu können; bei langen Nachrichten allerdings ist dies oftmals nicht sinnvoll.
Natürlich gilt dies oftmals auch nicht, wenn der geheime Kanal nur kurzfristig besteht.

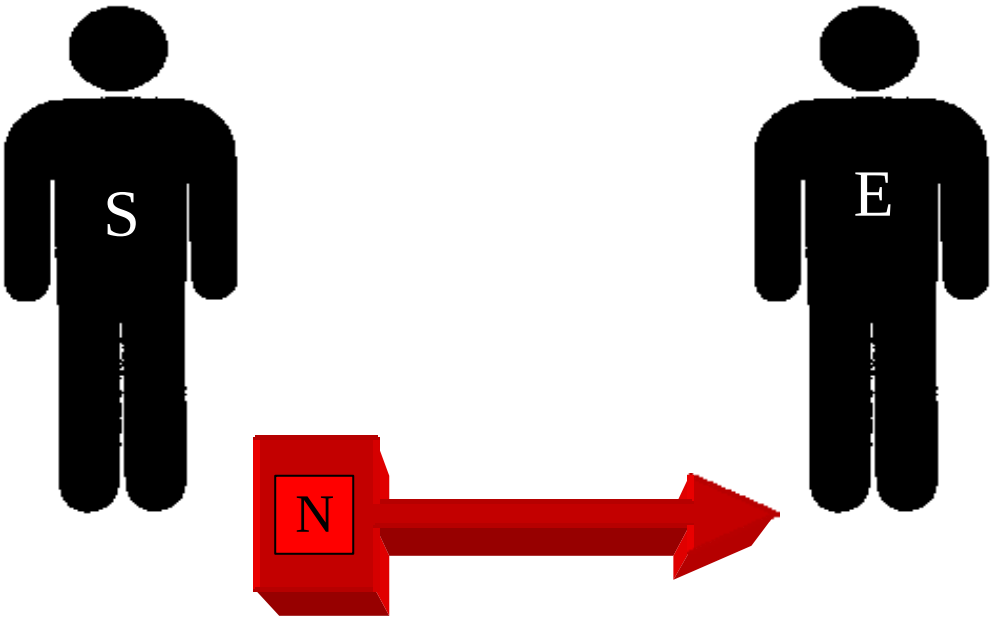

Abb. 2

Neben den von mir aufgeführten Nachteilen hat ein solches Verfahren natürlich auch einige Vorteile.
Diese Art des Codierens ist unter anderem gut dafür geeignet, Daten zu verschlüsseln, die nicht ausgetauscht werden sollen. Das ist zum Beispiel der Fall, wenn mehrere Personen Zugang zu einem Datennetz haben, aber nicht auf alle Daten zugreifen dürfen. Sollen die Daten nicht ausgetauscht werden, ist es oftmals von Vorteil, ein symmetrisches Verfahren anzuwenden, da diese sowohl vom Programmieraufwand, als auch von der Handhabung her meistens einfacher, vor allem aber wesentlich schneller sind (um einen groben Faktor von 1000).

All die Sicherheitsbedenken und Nachteile gelten also nur für Daten, die über öffentliche Kanäle ausgetauscht werden sollen, weil es dadurch zu den oben beschriebenen Problemen des Schlüsselaustausches kommt.
Diese Probleme treten bei der Verschlüsselung mit dem RSA- Verfahren nicht auf, da dieses Verfahren mit öffentlichem Schüssel funktioniert.

2.2 Das RSA- Verfahren als Beispiel für Verschlüsselung mit öffentlichem Schlüssel

Bei dem RSA-Kryptosystem handelt es sich um ein Codierungssystem, in dem ein Austausch von geheimen Schlüsseln nicht notwendig ist.
Diese Art der Verschlüsselung funktioniert im Prinzip folgendermaßen: Der Empfänger stellt zwei verschiedene Schlüssel her. (Wie ich später zeigen werde, sind es eigentlich zwei Zahlenpaare, von denen je eine Zahl gleich ist.) Diese Schlüssel stehen in einer bestimmten Beziehung zueinander, so daß mit dem einen Schlüssel codiert und mit dem anderen decodiert werden kann. Ein Codierungsverfahren, bei dem ein anderer Schlüssel zum Codieren als zum Decodieren benutzt wird, nennt sich asymmetrisch.
Sollen nun Daten ausgetauscht werden, so gibt der Sender dem Empfänger Bescheid, daß er ihm Daten zukommen lassen will. Der Empfänger gibt darauf hin dem Sender der Nachricht den Schlüssel, der zum Codieren dient.
Dieser Schlüssel ist nicht geheim und wird daher als Public-Key (PK) bezeichnet.
Ein Rückschluß von dem Schlüssel, der zum Codieren dient, auf den, der zum Decodieren gebraucht wird, ist prinzipiell nicht möglich. Daher kann der Transport dieses Schlüssels öffentlich erfolgen!
Derjenige, der die Nachricht senden will, nutzt diesen Schlüssel zum Codieren der Botschaft.
Erhält der Empfänger nun die Nachricht, so kann er aus ihr die Botschaft gewinnen, indem er sie mit seinem zweiten Schlüssel decodiert. Da dieser Schlüssel geheim bleiben muß, heißt er Secret-Key (SK).
Die 3. Abbildung verdeutlicht dies. Es ist deutlich zu erkennen, daß kein geheimer Datenaustausch nötig ist.

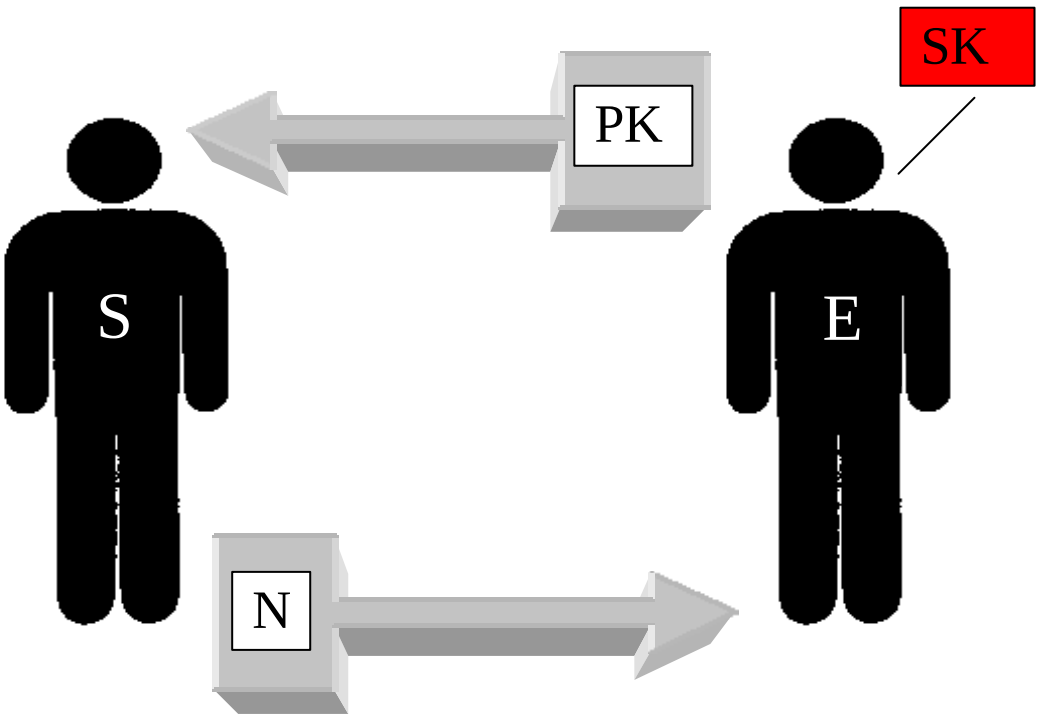

Abb. 3

Der Vorteil dieses Verfahrens für den Austausch sensibler Daten ist offensichtlich: Es muß kein geheimer Kanal vorhanden sein, wodurch es wesentlich einfacher ist, den Geheimschlüssel von Lauschern fernzuhalten.
Außerdem ist in diesem Fall der Grad der Sicherheit bekannt, da das Verfahren selbst nicht geheim ist. Der Sicherheitsstandard, den dieses Verfahren bietet, hängt nur von der Länge der eingesetzten Schlüssel ab. Auf Einzelheiten zur Sicherheit werde ich später eingehen (s. Kapitel 2.4 und 4.6).
Ein weiterer Vorteil der Codierung mit dem RSA-Algorithmus ist die Möglichkeit, eine Nachricht zu signieren. Dies spielt eine große Rolle, um festzustellen, von wem eine Nachricht stammt. Dabei wird im Prinzip genau der umgekehrte Weg gegangen, als bei der Codierung einer Nachricht:
Eine Person A will eine Nachricht an B senden, und B muß sich sicher sein, daß die Nachricht von A stammt. Um das zu erreichen, verschlüsselt A die Botschaft mit seinem Secret-Key. Dadurch bleibt sie für jeden lesbar, der den passenden Public-Key besitzt. Erhält nun B die Nachricht von A, so kann B sich sicher sein, daß die Nachricht von A stammt, wenn er sie ohne Probleme mit seinem Public-Key decodieren kann.
Das ganze ist natürlich auch mit codierten Botschaften möglich, so daß auch verschlüsselte Nachrichten signiert werden können. Damit ist die Möglichkeit gegeben, geheime Nachrichten auszutauschen und gleichzeitig eine Sicherheit über den Absender zu schaffen. (s. dazu Kapitel 4.5)
Diese Vorteile sprechen dafür, ein Verfahren mit öffentlichem Schlüssel zu verwenden wie das RSA-Codierungssystem, um sensible Daten auszutauschen. In der Praxis ist jedoch die Geschwindigkeit mit der Verschlüsselt wird ein entscheidender Faktor, der dazu geführt hat, symmetrische und asymmetrische Verfahren so zu kombinieren, daß die Vorteile beider Verfahren genutzt werden können. Dabei wird die Nachricht zwar mittels eines symmetrischen Verfahrens verschlüsselt, der geheime Schlüssel wiederum wird mit Hilfe der Public- Key-Verschlüsselung codiert und transportiert, so daß das Public- Key- Verfahren als „geheimer Kanal“ dient.

2.3 Die Entstehung des RSA- Algorithmus

Die Public- Key- Kryptographie wurde zwar schon 1974 von Diffie und Hellmann erfunden; jedoch wurden damals noch andere Algorithmen verwendet, die sich nicht durchsetzen konnten.
RSA wurde allerdings erst im Jahre 1977 entwickelt. An der Entwicklung waren im wesentlichen drei Personen beteiligt: Ron **R**ivest, Adi **S**hamir und Leonard **A**dleman. Die Anfangsbuchstaben führten zu der Bezeichnung **RSA**-Algorithmus.
Trotz der staatlich unterstützten Entwicklung im Jahre 1977, wurde erst am 29. September 1983 ein Patent, das U.S. Patent 4.405.829, auf das Verfahren vergeben. Die RSA Data Security Inc. of Redwood City, California erhielt dieses Patent, das 17 Jahre, nachdem es vergeben wurde, im Jahre 2000, erlischt.
In den USA werden von der RSA Data Security Inc. Lizenzen für „make, use or sell“, also für die Programmierung von Programmen, die auf RSA beruhen, sie zu benutzen, oder sie zu verkaufen, vergeben. Für nicht-kommerzielle Zwecke allerdings werden in den meisten Fällen keine Lizenzen benötigt.
Heute ist RSA Teil vieler Offizieller Standards[3]. So gehört RSA zum ITU-T X.509 security Standard, zum Society for Worldwide Interbank Financial Telecommunications (SWIFT)

[3] Offizielle Standards: Ein weltweit verbindlicher Standard wird durch die ISO (International Organisation for Standardization) und der IEC (International Electronical Comission) festgelegt. Dabei prüft ein gemeinsames Komitee, das ISO/IEC JTC1 (JTC: Joint Technical Committee), technische Normen und legt den nationalen Standardisierungorganisationen Entwürfe für neue Standards vor, um darüber abzustimmen.

Standard, zum Frech financial industry´s ETEBAC 5 Standart, sowie zu einigen Standards, die für die Softwareindustrie gelten und einigen Internet-Standards[4].

2.4 Beschreibung der Sicherheit

Um die Sicherheit des Codierungsverfahren beschreiben zu können, muß ich erst einmal das Funktionsprinzip geklärt haben. Daher werde ich nach der Beschreibung der mathematischen Aspekte noch einmal auf die Sicherheit in Kapitel 4.6 eingehen. In dem Kapitel werde ich dann auch erklären, auf welche Weise der Code zumindest in der Theorie zu knacken ist.
Um jedoch vorab einen Einblick in die Wirksamkeit dieses Verfahrens zu geben, möchte ich zunächst nur die erforderliche Rechenzeit auflisten, die zum Knacken des Codes nötig ist.

Schlüssellänge	**Benötigte Rechenzeit für 100 Millionen 8 MB Pentium 100 Rechner, um den Code zu knacken**
429 Bit	14,5 Sekunden
512 Bit	22 Minuten
700 Bit	153 Tage
1024 Bit	280 000 Jahre

Tab.1[5]

Aus dieser Tabelle ist zu erkennen, daß die Rechenzeit exponentiell ansteigt. Es ist also kein Problem, die Schlüssellänge zu verlängern, falls die Computer jemals so leistungsstark werden sollten, daß solche Berechnungen durchzuführen sind und sich der Aufwand lohnt.
Zur Zeit jedoch wären die finanziellen Ausgaben um einen Code zu knacken so groß, daß sie den Gewinn bei weitem überschreiten würden. Selbst wenn jemand es zustande bringen würde, so viele Computer zu vernetzen, daß eine ausreichende Rechenleistung zustande käme, was selbst über das Internet heutzutage noch unmöglich ist, so müßte diese unübersehbare Menge von Computern auch noch koordiniert werden. Der finanzielle Aufwand wäre unwahrscheinlich groß.
Wie das Knacken des Codes von der mathematischen Seite anzugehen wäre, werde ich, wie oben schon erwähnt, in 4.6 erläutern.

[4] [6] S.32 Abschnitt 20
[5] [6]

3 MATHEMATISCHE VORAUSSETZUNGEN

3.1 Einführung in die mathematischen Aspekte

Das Kapitel 3 dient, wie oben schon erwähnt, als eine Einführung in die mathematischen Aspekte der Public-Key-Verschlüsselung. Damit ich in den folgenden Kapiteln die nötigen Beweise führen kann, muß ich in diesem Kapitel erst einmal einige mathematische Zusammenhänge darstellen. Sie werden die Voraussetzung für spätere Beweise sein.
Dazu gehören u.a. einige Zusammenhänge der Exponentialrechnung, des Rechnens mit Modulo, und der Eigenschaften der φ- Funktion.

3.2 Exponentialrechnung

Die Rechnung mit Potenzen setze ich in meiner Arbeit als bekannt voraus. Daher sei auf zwei elementare Zusammenhänge, die ich in meiner Arbeit häufiger gebrauche, nur sehr kurz und ohne Beweis hingewiesen:

(1) Beim Multiplizieren von Potenzen mit gleicher Basis wird die Basis mit der Summe der Exponenten potenziert. Es gilt also:
Satz 1: $(a^x)*(a^y) = a^{x+y}$
Beispiel: $(3^4)*(3^5)= 3^9 =19683$

(2) Eine Potenz wird potenziert, indem die Basis mit dem Produkt der Exponenten potenziert wird. Demnach gilt:
Satz 2: $(a^x)^y = a^{xy}$
Beispiel: $(3^4)^5 = 3^{20} = 3486784401$

3.3 Beweise zur Teilbarkeitsrelation

Im Kapitel 3.8 werde ich ein Verfahren zur Ermittlung des größten gemeinsamen Teilers (ggT) beschreiben. (Für den größten gemeinsamen Teiler t von x und y gilt: $t = ggT(x, y) \Rightarrow t| x$ und $t| y$. Dabei gilt für jedes $u > t$ und $u \in N$: $u \not| x$ und $u \not| y$.) Um dieses Verfahren zu erläutern, ist es allerdings von Nöten, auf die Beweise zurückgreifen zu können, die ich in diesem Kapitel führen werde.
Der erste zu beweisende Satz besagt, folgendes: wenn eine Zahl a eine Zahl b teilt und eine Zahl c ein d teilt, dann teilt das Produkt von a und c das von b und d. Es gilt also:
Satz 3a: Für alle $a, b, c, d \in Z$ gilt, aus $a|b$ und $c|d$ folgt: $ac|bd$.

Beweis: Nach Voraussetzung ($a|b$ und $c|d$) gibt es ein q_1 und ein q_2, so daß $q_1 a = b$ und $q_2 c = d$ gilt. Für die Produkte gilt dann: $(q_1 q_2)(ac) = bd$. Da $q_1, q_2 \in Z$, ist wegen der Abgeschlossenheit von Z bezüglich der Multiplikation auch $q_1 q_2 \in Z$. Daher muß gelten: $ac|bd$.
q.e.d.

Weil $d \in Z$, wird d immer durch 1 und durch d geteilt. Hieraus folgen zwei Spezialfälle: (1.) $c = 1$ und (2.) $c = d$. Durch Einsetzen in Satz 3a ergibt sich
Satz 3b: (1.) aus $a|b$ folgt $a|bd$
und (2.) aus $a|b$ folgt $ad|bd$.

Der zweite Satz, den ich in diesem Kapitel beweisen will, ist folgender:
Satz 4: Für alle $a, b, c, r, s \in Z$ gilt: aus $a|b$ und $a|c$ folgt: $a| rb + sc$.

Beweis: Nach Satz 3b (1.) gibt es ein $q_1 \in Z$ und ein $q_2 \in Z$, so daß gilt: $q_1 a = rb$ und $q_2 a = sc$ mit r, s $\in$ Z. Durch Addition erfolgt: $q_1 a + q_2 a = rb + sc$, bzw. $(q_1 + q_2)a = rb + sc$, also gilt: $a | rb + sc$, da $q_1, q_2 \in Z$ und somit auch $(q_1 + q_2) \in Z$.

q.e.d.

Auf diesen Satz 4 werde ich u.a. im Kapitel 3.8 zurückgreifen, wenn ich den Euklidischen Algorithmus beschreibe.

3.4 Rechnen mit Modulo

Den Satz, den ich im folgenden Beweisen werde, werde ich u.a. in Kapitel 3.8 gebrauchen, um die Funktionswiese des Euklidischen Algorithmus zu erläutern. Er sagt aber auch etwas darüber aus, wie sich die Modulofunktion einfach definieren läßt.

Sei a = 19 und b = 8. Dann gilt: 19 = 2*8 + 3. Es läßt sich also feststellen, daß das Zahlenpaar (2, 3) = (q, r) die Gleichung 19 = q*8 + r erfüllt. Es ist des weiteren zu erkennen, daß es weitere Zahlenpaare gibt, für die diese Gleichung gilt, zum Beispiel (1, 11), (-2, 19), usw. Soll für die Zahl r nun aber die Bedingung $0 \leq r < 8$ gelten, dann scheint es zunächst nur ein ganzzahliges Zahlenpaar zu geben, welches diese Gleichung erfüllt.

Der Satz, den ich im folgenden beweisen werde, besagt, daß es, wenn a, b $\in$ N sind, *genau ein* Paar q, r $\in N_0$ gibt, so daß $a = q*b + r$ gilt, mit $0 \leq r < b$. Er bestätigt also die Vermutung, daß es nur ein solches Zahlenpaar gibt.

Beweis:

1. Zunächst der Beweis, daß es immer mindestens eines der gewünschten Zahlenpaare gibt (Existenzbeweis):
   Behauptung:
   Es gibt zu jedem Paar (a, b) mit a, b $\in$ N *mindestens* ein Zahlenpaar (q, r) mit q, r $\in N_0$ und $0 \leq r < b$, für das $a = qb + r$ gilt.
   Beweis:
   Gilt $a < b$, so ist (0, a) das gesuchte Paar, da $a = 0*b + a$.
   Für $a \geq b$ gilt folgendes:
   Die Menge M der Zahlen, für die gilt, daß $a - x*b$ eine natürliche Zahl ist und $a - x*b \geq 0$ ist, hat mindestens ein Element, nämlich das Element $e = a - 1*b$. Dies ist der Fall, weil (1.) $a - b$ immer ganzzahlig ist und weil (2.) $a - b$ immer größer oder gleich Null ist, da $a \geq b$ ist.
   Zusammenfassend gilt: Die Menge $M = \{ x | a - xb \in N \wedge a - xb \geq 0 \}$ hat mindestens das Element $e = a - 1*b$.
   Nun sei q maximal und $r = a - qb \geq 0$. Dann nimmt r den kleinstmöglichsten Wert an.
   Außerdem ist $r - b = a - qb - b = a - (q+1)b < 0$, da r für x = q den niedrigsten Wert annimmt, bei einer Erhöhung von q um 1 jedoch ist das Element kleiner Null. Aus $r - b < 0$ folgt nun aber $r < b$. Es gilt also insgesamt $0 \leq r < b$.
   Es existiert also immer mindestens ein Zahlenpaar (q, r), für das die Behauptung wahr ist.

2. Nun folgt der Beweis, daß es nur genau ein Zahlenpaar (q, r) mit q, r $\in N_0$ gibt, für das gilt: $a = qb + r$ mit $0 \leq r < b$ (Eindeutigkeitsbeweis):
   Nimmt man zwei gültige Zahlenpaare an, so würde gelten: $a = q_1 b + r_1$ und $a = q_2 b + r_2$, also auch: $q_1 b + r_1 = q_2 b + r_2 \Rightarrow 0 = (q_1 - q_2)b + (r_1 - r_2)$. Dann wäre $-(q_1 - q_2)b = (r_1 - r_2)$. Also muß b den Wert $(r_1 - r_2)$ teilen, da $(q_1 - q_2)$ immer eine ganze Zahl ist. Es gilt: $b | (r_1 - r_2)$. Wegen $0 \leq r_1 < b$ und $0 \leq r_2 < b$ folgt $r_1 = r_2$. (Da $r_1 < b$ und $r_2 < b$ folgt in N_0 das $r_1 - r_2 < b$. Also muß $(r_1 - r_2) = 0$ sein, da es kleiner als b aber durch b teilbar sein soll.) Das bedeutet aber wegen $0 = (q_1 - q_2)b + (r_1 - r_2)$, daß $q_1 = q_2$.

q.e.d.

Satz 5: Es gibt, wenn a, b $\in$ N sind, also *genau ein* Paar q, r $\in N_0$, so daß $a = q*b + r$ gilt, mit $0 \leq r < b$.

Diesen Satz werde ich u.a. in 3.8 wieder aufgreifen.

Das grundlegende Rechnen mit modulo setzte ich als bekannt voraus. Da ich jedoch in der Literatur auf eine Vielzahl von verschiedenen Schreibweisen gestoßen bin, werde ich in diesem Abschnitt erläutern, welche Schreibweisen ich für meine Arbeit gewählt habe und warum ich dies getan habe:
Vorausgesetzt sei, daß die Zahlen a und b bei der Division durch n den selben Rest r haben, also: n|a - b (nach Satz 4). Dann gilt:

$a \equiv b \bmod n$ (a ist kongruent b modulo n).

Da es jedoch in meiner Arbeit häufig nicht darum geht, die Kongruenz zweier Zahlen bezüglich eines Moduls n darzustellen, sondern mit dem kleinsten, positiven Rest r weiter zu rechnen, werde ich eine andere Schreibweise ebenfalls benutzen:

$a \bmod n = r$ sowie $b \bmod n = r$.

Dabei ist r der kleinste positive Rest, es gilt also: $0 \leq r < n$, d.h. r ist stets eindeutig bestimmt.
Beispiel: $5 \bmod 2 = 1$ sowie $7 \bmod 2 = 1$.

Möchte ich jedoch trotzdem einmal den Zusammenhang zwischen a und b darstellen, so werde ich auf die obere Schreibwiese zurückgreifen. Für das Beispiel gilt daher auch: $5 \equiv 7 \bmod 2$.
Die von mir eingeführte Schreibweise (a mod n = r) rechtfertigt sich auch aus der Tatsache, daß eine Standardfunktion in Borland Pascal die Modulofunktion ist, deren Aufruf:

x := a mod n

dem Wert x den Rest r übergibt, mit $0 \leq x < n$.
Da diese Schreibweise im Prinzip meiner entspricht, wird es mir später leichter möglich sein, zu erklären, wie meine Computerprogramme arbeiten.
Ich werde in meiner Arbeit nur diese beiden Schreibweisen ($a \equiv b \bmod n$ und $a \bmod n = r$ bzw. $b \bmod n = r$) benutzen.

Nachdem nun die Darstellungsart von mir festgelegt wurde, mit der ich in der Arbeit das Rechnen mit Modulo beschreiben werde, möchte ich nun einen Satz beweisen, mit dessen Hilfe ich u.a. in Kapitel 3.5 einen weiteren Beweis führen kann. Er sagt etwas über das Kürzen während des Rechnens mit Modulo aus.
Satz 6a: Aus $za \equiv zb \bmod m$ und $ggT(z, m) = d$ folgt $a \equiv b \bmod (m/d)$.

Beweis:

(1.) Sei $ggT(a, b) = t$. Dann folgt $ggT(a/t, b/t) = 1$, denn wenn es ein $q > 1$ mit $q = ggT(a/t, b/t)$ gäbe, dann würde folgendes gelten: $q \mid (a/t)$ und $q \mid (b/t)$. Daraus folgt $qt \mid a$ und $qt \mid b$. Dies ist ein Widerspruch, da $qt > t$, aber $ggT(a, b) = t$.

(2.) Aus $za \equiv zb \bmod m$ folgt $m \mid za$ und $m \mid zb \Rightarrow m \mid za - zb$. Das bedeutet, es gibt ein q, für das $qm = z(a - b)$ gilt.
Bei der Division durch $d = ggT(z, m)$ ergibt sich: $q(m/d) = (z/d)(a - b)$. Da $ggT((m/d),(z/d)) = 1$ (nach (1.)) folgt $(m/d) \mid (a - b)$, also $a \equiv b \bmod (m/d)$.

q.e.d.

Wenn nun auch noch $ggT(z, m) = 1$ gelten soll, so ergibt sich durch einfaches Einsetzen in Satz 6a:
Satz 6b: Aus $za \equiv zb \bmod m$ und $ggT(z, m) = 1$ folgt $a \equiv b \bmod m$.

Zum Beispiel ist: $39 \equiv 54 \bmod 5 \Rightarrow 3*13 \equiv 3*18 \bmod 5 \Rightarrow 13 \equiv 18 \bmod 5$.

3.5 Der kleine Fermatsche[6] Satz

Der kleine Fermatsche Satz wird in meiner Arbeit eine wichtige Rolle sowohl für einen der Primzahltests, als auch für die Verschlüsselung selbst spielen. Er lautet folgendermaßen:

Satz 7a: Wenn r relativ prim zu p ist (ggT(r, p) = 1), dann gilt $r^{p-1} \equiv 1 \bmod p$, unter der Voraussetzung, daß p eine Primzahl ist.

Zum Beispiel sei p=7 und r=3. Dann ist $r^{p-1} = 3^6 = 729$. Daraus folgt für die Division mod p: 729 – 104 * 7 = 1. Also ist $3^6 \equiv 1 \bmod 7$.

Dies ist folgendermaßen zu beweisen:
Man betrachte die zu der Division mod p gehörenden Reste 1, 2, 3,..., p-1. Werden diese Reste mit einem Faktor r multipliziert, so folgt daraus die Menge {r, 2r, 3r, ..., (p – 1)r}. Diese ist kongruent zu der Menge {1, 2, ..., p-1}, wobei die Elemente der ersten Menge in anderer Reihenfolge zu denen der zweiten kongruent sind.
Dies zeige ich, indem ich beweise, daß kein Element der Menge {r, 2r, 3r, ..., (p – 1)r} einem anderen Element dieser Menge kongruent mod p ist.
Seien die Zahlen q_1*r und q_2*r mit $0 \le q_i \le p$ kongruent mod p, dann gilt: $q_1*r \equiv q_2*r \bmod p$. Da r und p nach Voraussetzung teilerfremd sind, gilt nach Satz 6b: $q_1 \equiv q_2 \bmod m$. Da $0 \le q_1, q_2 \le p$ folgt: $q_1 = q_2$. Jedes der Elemente r, 2r, 3r, ..., (p – 1)r ist also genau einer der Zahlen 0, 1, 2, ..., (p – 1) kongruent mod p.
Es gilt demnach für die Produkte sämtlicher Reste:

$$1*2*3*...*(p-1) \equiv (r*2r*3r*...*(p-1)r) \bmod p.$$

Dies läßt sich auch folgendermaßen schreiben:

$$(p-1)! \equiv r^{p-1} (p-1)! \bmod p.$$

Nach dem Kürzen der Fakultäten (p-1)! (Dies ist möglich, weil ggT((p-1)!, p) = 1 ist, nach Satz 6b.) folgt:

$$1 \equiv r^{p-1} \bmod p \quad \text{bzw.} \quad r^{p-1} \equiv 1 \bmod p,$$

was zu beweisen war.
Eine andere Darstellungsweise ergibt sich aus der Multiplikation mit r:

Satz 7b: $r^p \equiv r \bmod p$.

3.6 Definition der Eulerschen[7] j - Funktion

Ist $n \in N$, so wird die Abbildung $\varphi: N \rightarrow N$ mit $n \rightarrow m$ und $m \in N$, wobei m die Anzahl der zu n teilerfremden Zahlen kleiner n ist, als Eulersche φ- Funktion bezeichnet. Zwei Zahlen $a \in N$, $b \in N$ sind dabei teilerfremd, wenn ggT(a, b) = 1 ist.
So ergibt sich zum Beispiel:

φ(6) = 2, da 1 und 5 teilerfremd zu 6 sind.
φ(2) = 1, da nur die 1 teilerfremd zu 2 ist.

Eine Eigenschaft der φ- Funktion hat für das RSA- Verfahren eine besondere Bedeutung:

Satz 8: φ ist multiplikativ, d.h. es gilt für alle $m, n \in N$ $\varphi(m*n) = \varphi(m)*\varphi(n)$, wenn ggT(m, n) = 1.

---

[6] Nach Pièrre de Fermat, 1601-1665, der auch den großen Fermatschen Satz formuliert, aber nicht bewiesen hatte. [30]
[7] nach Leonhard Euler, 1707-1782 [32]

Den Zusammenhang, daß $\varphi(m*n) = \varphi(m)*\varphi(n)$ ist, falls ggT(m, n) = 1 gilt, werde ich im folgenden Beweisen.
Um das Prinzip dieses Beweises zu verdeutlichen, werde ich jedoch zunächst ein Beispiel geben: Sei m = 5 und n = 6. Dann ist zu zeigen: $\varphi(30) = \varphi(5)*\varphi(6)$.
Zum Beweis werden die Zahlen 1 bis 30 folgendermaßen angeordnet:

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30.

Eine Zahl kann nur dann zu 30 teilerfremd sein, wenn sie es auch zu jeden der beiden Faktoren 5 und 6 ist. Nun ist zu erkennen, daß alle zu 6 teilerfremden Zahlen in den Spalten 1 und 5 stehen. Das bedeutet, daß die zu 30 teilerfremden Zahlen eine Teilmenge dieser Zahlen bilden. Da in Spalte 1 genau 4 Zahlen zu 5 teilerfremd sind (1, 7, 13, 19, nicht 25) und in Spalte 5 die Zahlen 11, 17, 23 und 29, also ebenfalls 4, gilt $\varphi(5)*\varphi(6) = 4*2 = 8 = \varphi(30)$.
Um diesen Satz zu verallgemeinern, muß gelten: $\varphi(m*n) = \varphi(m)*\varphi(n)$. Die Tabelle sieht dann folgendermaßen aus:

1	2	3	...	n
n+1	n+2	n+3	...	2n
2n+1	2n+2	2n+3	...	3n
.	.	.		.
.	.	.		.
.	.	.		.
(m-1)n+1	(m-1)n+2	(m-1)n+3	...	m*n

Jede Zahl läßt sich also mit Sicherheit in der Form q*n + r darstellen. Dabei gilt $1 \leq r < n$ und $0 \leq q \leq m-1$. (Bleibt q fest, z.B. q = 3, und nimmt r jeden Wert von 1 bis n an, so ergeben sich alle Elemente einer Zeile, z.B. der dritten. Bleibt hingegen r konstant und q nimmt die Werte von 0 bis m-1 an, so ergeben sich in gleicher Weise alle Elemente einer Spalte.)
Nun sei r_1 teilerfremd zu n. Dann sind auch alle Elemente der Spalte r_1, also alle $q*n + r_1$ zu n teilerfremd, da $q*n + r_1 \equiv r_1 \bmod n$ gilt.
In der ersten Zeile sind $\varphi(n)$ Zahlen zu n teilerfremd, also sind auch alle Zahlen der zugehörigen $\varphi(n)$ Spalten zu n teilerfremd.
Jede dieser $\varphi(n)$ Spalten besteht aus m Elementen, die sich in der Form $q*n + r$ mit $0 \leq q \leq m-1$ darstellen lassen.
Mit festem r ergeben sich die Zahlen

$$r, n + r, 2n + r, ..., (m - 1)n + r.$$

Jede dieser m Elemente ist genau einer der m Zahlen

$$0, 1, 2, ..., (m - 1)$$

kongruent mod m. Das ist zu zeigen, indem man zeige, daß kein Element der Menge {r, n + r, 2n + r, ..., (m – 1)n + r} zu einem anderen Element dieser Menge kongruent mod m ist. Da jede ganze Zahl genau einer ganzen Zahl 0, 1, 2, ..., m – 1 kongruent mod m ist, folgt daraus die Behauptung (vgl. auch Beweis des Satzes 7a).
Seien die Zahlen $q_1*n + r$ und $q_2*n + r$ kongruent mod m, dann gilt $q_1*n + r \equiv q_2*n + r \bmod m$ mit $0 \leq q_1, q_2 \leq m - 1$.Daraus folgt, $q_1*n \equiv q_2*n \bmod m$. Da m und n teilerfremd sind, gilt nach Satz 6b: $q_1 \equiv q_2 \bmod m$. Da $0 \leq q_1, q_2 \leq m - 1$ folgt $q_1 = q_2$.
Jedes der m Elemente r, n + r, 2n + r, ..., (m – 1)n + r ist also genau einer der m Zahlen 0, 1, 2, ..., (m – 1) kongruent mod m.
Unter den Zahlen 0, 1, 2, ..., (m – 1) sind $\varphi(m)$ zu m teilerfremd, daher sind auch unter den Zahlen r, n + r, 2n + r, ..., (m – 1)n + r genau $\varphi(m)$ zu m teilerfremd. Es gibt also in jeder Spalte $\varphi(m)$ zu m teilerfremde Zahlen.
Daraus folgt, daß es unter den Zahlen 1, 2, ..., m*n genau $\varphi(m)*\varphi(n)$ zu m*n teilerfremde Zahlen gibt. Nach der Definition der φ- Funktion gibt es zu m*n jedoch genau $\varphi(m*n)$ teilerfremde Zahlen, daher gilt:

$\varphi(m*n) = \varphi(m)*\varphi(n)$.

q.e.d.

Diesen Satz werde ich später benötigen, um zeigen zu können, wie die Schlüssel generiert werden.

Auf eine Besonderheit sei in diesem Zusammenhang noch hingewiesen: Wenn n eine Primzahl ist, so sind alle Zahlen, die kleiner sind als n, relativ prim zu n, außer 1. Dies ergibt sich aus der Definition einer Primzahl, die besagt, daß eine Primzahl nur durch sich selbst und durch 1 teilbar ist. Daher gilt für jede Primzahl p:

$\varphi(p) = p-1$.

3.7 Die Eulersche Verallgemeinerung des kleinen Fermat

Euler erweiterte den Fermatschen Satz, indem er folgende Behauptung aufstellte und bewies:

Für teilerfremde a, $m \in N$ gilt: $a^{\varphi(m)} \equiv 1 \bmod m$.

Beispielsweise sei m=6. Dann ist $\varphi(m) = \varphi(6) = 2$, da 1 und 5 relativ prim zu 6 sind. a sei 7, da ggT(6,7) = 1. Dann ist $a^{\varphi(m)} = 7^2 = 49$. Dividiert durch 6 ergibt sich ein Rest von 1, da $7^2 - 8*6 = 1$ ist.

Den Beweis führte Euler wie folgt:
Sei $m \in N$, so gibt es $\varphi(m)$ natürliche Zahlen, die zu m teilerfremd sind und kleiner als m sind. Sie seien als $r_1, r_2, ..., r_{\varphi(m)}$ bezeichnet.
a sei nun eine Zahl, die teilerfremd ist zu m. Dann sind auch $ar_1, ar_2, ..., ar_{\varphi(m)}$ zu m teilerfremd, da sie Produkte zweier zu m teilerfremder Zahlen sind.
Außerdem sind sie paarweise inkongruent mod m, es gilt also:

$ar_i \not\equiv ar_j \bmod m$ mit $1 \le i, j \le \varphi(m)$ und $i \neq j$ (vgl. auch Beweis des Satzes 7a).

Dies läßt sich beweisen, indem man annimmt:

$ar_i \equiv ar_j \bmod m$ mit $1 \le i, j \le \varphi(m)$.

Da a und m teilerfremd sind, läßt sich durch a dividieren, und es ergibt sich:

$r_i \equiv r_j \bmod m$, also $i = j$.

Daraus folgt, daß $ar_1, ar_2, ..., ar_{\varphi(m)}$ paarweise inkongruent mod m sind.
Dies bedeutet aber, daß jede der Zahlen $ar_1, ar_2, ..., ar_{\varphi(m)}$ genau einer der Zahlen $r_1, r_2, ..., r_{\varphi(m)}$ kongruent ist.
Für die Produkte der Kongruenzen folgt also:

$ar_1*ar_2*...*ar_{\varphi(m)} \equiv (r_1*r_2*...*r_{\varphi(m)}) \bmod m$

bzw. $a^{\varphi(m)} * (r_1*r_2*...*r_{\varphi(m)}) \equiv (r_1*r_2*...*r_{\varphi(m)}) \bmod m$.

$r_1, r_2, ..., r_{\varphi(m)}$ ist nach Voraussetzung zu m teilerfremd, daher kann hier durch $(r_1*r_2*...*r_{\varphi(m)})$ dividiert werden, da auch $(r_1*r_2*...*r_{\varphi(m)})$ als Produkt teilerfremder Faktoren teilerfremd zu m ist (nach Satz 6b).
So ergibt sich:

<u>Satz 9</u>: $a^{\varphi(m)} \equiv 1 \bmod m$

q.e.d.

Daß der Eulersche Satz eine Erweiterung des Fermatschen Satzes ist, zeigt sich, wenn m eine Primzahl ist:
Dann ist $\varphi(m) = m-1$ (s. 3.6) und es gilt:

$a^{m-1} \bmod m = 1$

bzw. $a^m \bmod m = a$,

was genau der Fermatsche Satz ist.

3.8 Der Euklidische[8] Algorithmus

Zu beweisen ist folgender Satz:

Satz 10a: Seien $a,b \in N$ und sei $a = qb + r$ mit $q, r \in N_o$ und $0 \leq r < b$. Dann gilt: $T(a) \cap T(b) = T(b) \cap T(r)$, d.h. daß die Menge der gemeinsamen Teiler von a und b gleich der von b und r ist.

Euklid beweist diesen Satz folgendermaßen:
Im ersten Teil des Beweises zeigt er, daß $T(a) \cap T(b) \subseteq T(b) \cap T(r)$: Sei $t \in T(a) \cap T(b)$, dann gilt: $t|a$ und $t|b$. Aus Satz 4 folgt: $t|a - qb$ (Diese Darstellung ergibt sich, wenn $r = 1$ und $s = -q$ gewählt wird). Daraus folgt aber $t|r$ und somit gilt: $t \in T(b) \cap T(r)$, folglich ist $T(a) \cap T(b) \subseteq T(b) \cap T(r)$.
In dem zweiten Teil des Beweises folgert er analog: $T(b) \cap T(r) \subseteq T(a) \cap T(b)$: Sei $t \in T(b) \cap T(r)$, so folgt $t|b$ und $t|r$. Aus Satz 4 folgt: $t|qb + r$ (wenn $r = q$ und $s = 1$ gewählt wird). Ist diese Darstellung möglich, bedeutet dies aber $t|a$. Somit gilt: $t \in T(a) \cap T(b)$ und $T(b) \cap T(r) \subseteq T(a) \cap T(b)$.
Aus $T(a) \cap T(b) \subseteq T(b) \cap T(r)$ und $T(b) \cap T(r) \subseteq T(a) \cap T(b)$ folgt : $T(a) \cap T(b) = T(b) \cap T(r)$
q.e.d.

Da nach Satz 10a a, b und b, r dieselben gemeinsamen Teiler haben, ergibt sich daraus:

Satz 10b: Seien $a,b \in N$, $a > b$ und es sei $a = qb + r$ mit $q, r \in N_0$ und $0 \leq r < b$. Dann gilt: $ggT(a, b) = ggT(r, b) = ggT(a - qb, b)$.

Dies ermöglicht nun die einfache Berechnung des größten gemeinsamen Teilers: Sei zum Beispiel $a = 248$ und $b = 80$. Dann ist $ggT(248,80) = ggT(248 - 3*80,80) = ggT(8,80) = 8$.
Daß dabei $a > b$ sein muß, ist in der Praxis keine Einschränkung, da in dem Fall $a < b$ einfach die Reihenfolge vertauscht wird und Satz 10a, bzw. 10b auf das Zahlenpaar (b, a) angewandt wird.
Mein vorheriges Beispiel war nun allerdings so günstig gewählt, daß nach einmaligem Anwenden des Satzes ohne weitere Schwierigkeiten der ggT abgelesen werden konnte. Das allerdings nicht immer ein einmaliges Anwenden des Satzes zum Erfolg führt, zeigt folgendes Beispiel: $a = 548$ und $b = 84$. Unter Anwendung von Satz 10a ergibt sich: $T(548) \cap T(84) = T(44) \cap T(84)$. Hier läßt sich nicht auf Anhieb sagen, was das größte Element dieser Menge ist. Aber ein mehrmaliges Anwenden des Satzes ergibt: $T(44) \cap T(84) = T(44) \cap T(40) = T(4) \cap T(40) = T(4) \cap T(0)$. T(0) ist jedoch N, so läßt sich der größte gemeinsame Teiler von 548 und 84 einfach ablesen: $ggT(548, 84) = 4$.
So ergibt sich der Euklidische Algorithmus:

Satz 10c: Seien $a, b \in N$, $a > b$, so gibt es ein Zahlenpaar $q, r_i \in N_0$ und einem Index $n \in N$, so daß gilt:

$a = q_1 b + r_1$ $\qquad 0 \leq r_1 < b$

$b = q_2 r_1 + r_2$ $\qquad 0 \leq r_2 < r_1$

$r_1 = q_3 r_2 + r_3$ $\qquad 0 \leq r_3 < r_2$

.
.
.

$r_{n-1} = q_{n+1} r_n + r_{n+1}$ $\qquad 0 \leq r_{n+1} < r_n$

$r_n = q_{n+2} r_{n+1} + 0$

[8] nach Euklid (365 – 300 v.Chr.)

Man erhält: $T(a) \cap T(b) = T(r_{n+1})$ und somit $ggT(a, b) = r_{n+1}$.

Nach Satz 5 gibt es immer genau ein Zahlenpaar (r_i, r_j), für daß $0 \leq r_j < r_i$ gilt. Es ist also immer möglich, geeignete Zahlen zu wählen.
Auch hier hat der Beweis wieder zwei Teile:

(1.) Nach spätestens b Schritten muß ein Rest Null werden, weil $b > r_1 > r_2 > ... \geq 0$ eine streng monoton fallende Folge ganzer, positiver Zahlen bildet. Es ergibt sich die Gleichung $r_n = q_{n+2} r_{n+1} + 0$.

(2.) Das mehrmalige Anwenden von Satz 10a liefert:
$T(a) \cap T(b) = T(b) \cap T(r_1) = T(r_1) \cap T(r_2) = ... = T(r_n) \cap T(r_{n+1}) = T(r_{n+1}) \cap T(0) = T(r_{n+1})$, also: $T(a) \cap T(b) = T(r_{n+1})$, was bedeutet: $ggT(a,b) = r_{n+1}$.

Folgendes Beispiel soll die Funktionsweise des Euklidischen Algorithmus verdeutlichen:

a = 3218, b = 832.

Das bedeutet für den Algorithmus:

3218	= 3*832	+ 722
832	= 1*722	+110
722	= 6*110	+ 62
110	= 1*62	+ 48
62	= 1*48	+ 14
48	= 3*14	+ 6
14	= 2*6	+ 2
6	= 3***2**	+ 0

Jetzt kann der größte gemeinsame Teiler von 3218 und 832 einfach abgelesen werden : ggT(3218, 832) = 2.

4 DAS RSA- VERFAHREN

4.1 Codierung, Decodierung

Für eine Zahl c soll gelten: $c = m^e \bmod n$. Außerdem soll für x gelten: $x = c^d \bmod n$. Dann lassen sich diese beiden Gleichungen zusammenfassen, indem die erste in die zweite Gleichung eingesetzt wird: $x = (m^e)^d \bmod n$. Nach Satz 2 entspricht dies: $x = m^{ed} \bmod n$. Wird nun e und d so gewählt, daß das für sie gilt: $ed = \varphi(n) + 1$, so ergibt sich $x = m^{\varphi(n) + 1} \bmod n$. Dies bedeutet aber auch: $x = m$ (wegen $m^{\varphi(n) + 1} \bmod n = m$ nach Satz 9). Dies ist dann besonders einfach, wenn n eine Primzahl ist, weil dann $\varphi(n) = n - 1$. Es gilt dann:

$c = m^e \bmod n$

und $m = c^d \bmod n$,

mit $ed = n = \varphi(n) + 1$.

Auf den ersten Blick ließen sich e und n als öffentlicher Schlüssel verbreiten, und es ergebe sich die Möglichkeit, die von anderen verschlüsselte Nachricht mit Hilfe von d und n wieder zu entschlüsseln. Da aber n bekannt ist und $ed = \varphi(n) + 1 = n$ gilt, läßt sich ohne weiteres nur mit der Kenntnis von e und n auch der Secret- Key aus $d = n/e$ berechnen. So ist das Verfahren also wertlos.

Dies läßt sich jedoch vermeiden, indem man nicht einfach eine Primzahl n wählt, sondern indem zwei Primzahlen p und q gewählt werden, deren Produkt n bildet, also $n = pq$. Dann gilt nach Satz 8: $\varphi(n) = \varphi(p)*\varphi(q) = (p - 1)(q - 1)$. Dann ist das $\varphi(n)$ nicht mehr einfach aus $n - 1$ zu berechnen, und somit ist auch der Secret- Key nur zu berechnen, wenn die beiden Primzahlen p und q bekannt sind.

Bei recht kleinen Zahlen ist es noch ohne Probleme möglich, die Zahl n in ihre Primfaktoren zu zerlegen. Z.B. ist $77 = 7*11$. Wenn jedoch die Zahlen sehr groß werden, ist diese Rechnung nur mit enormen Rechenaufwand zu lösen, bis die Primfaktoren schließlich nicht mehr in einer angemessenen Zeit ausgerechnet werden können.

Was sind beispielsweise die Primfaktoren von 44565657599599762843173371730520895308367191471594306333823967849169807802020911231448400165941621610927932659122180977141832235435599197599199678806220800? Dies läßt sich nicht mehr ohne weiteres ausrechnen. Hingegen lassen sich große Primzahlen wie 23861937004091334933147998710948942751300717103449595125344972878557193876984982757864290593007243133622620139994020515497049418574332733235 3 oder 186764626827901 ohne Schwierigkeiten herstellen und deren Produkt (die obige Zahl) ohne weiteres ausrechnen.

Das Produkt $n = pq$ kann also leicht berechnet werden, wobei sich n jedoch ohne Kenntnis von p oder q nicht wieder zerlegen läßt. (Aufgrund dieses algorithmischen Unvermögens wird diese Funktion auch als Falltürfunktion oder One-Way-Function bezeichnet.)

So wird es möglich, e und n öffentlich zu verbreiten, also das Zahlenpaar (e, n) als Public- Key zu benutzen. Die Nachricht c wird dann aus $c = m^e \bmod n$ berechnet, wobei m die unverschlüsselte Botschaft ist. Der Empfänger der Nachricht kann dann mittels des Zahlenpaares (d, n) über die Gleichung $m = c^d \bmod n$ die Botschaft gewinnen.

Voraussetzung ist jedoch, daß $ed = \varphi(n) + 1$ gilt. Das ist jedoch äquivalent zu $1 \bmod \varphi(n)$, da $(\varphi(n) + 1) \bmod \varphi(n) = (\varphi(n) + 1) - \varphi(n) = 1 \bmod \varphi(n)$. Es muß demnach ein Zahlenpaar (e, d) gefunden werden, für das gilt: $ed = 1 \bmod \varphi(n)$. Dabei ist $n = pq$, das Produkt zweier großer Primzahlen. d wird auch das Inverse zu e genannt, da e multipliziert mit d das neutrale Element bezüglich der Multiplikation ergibt, nämlich 1.

Das Problem wird also sein, e und d, bzw. das Inverse zu gegebenem e zu bestimmen. Dieses Problem läßt sich durch den in Kapitel 3.8 beschriebenen Euklidischen Algorithmus lösen. Mit einer Erweiterung dieses Algorithmus ist es möglich, e und d zu berechnen. Diesen sogenannten „erweiterten euklidischen Algorithmus“ werde ich in Kapitel 4.3 beschreiben.

4.2 Zusammenfassung der Codierung und Decodierung

Zunächst werden zwei geeignet große Primzahlen p und q generiert, deren Produkt n bilden: $n = pq$. Anschließend wird $\varphi(n) = \varphi(p)*\varphi(q) = (p - 1)(q - 1)$ (nach Satz 8) ausgerechnet. Dann kann e gewählt und das Inverse d mit dem erweiterten Euklidischen Algorithmus berechnet werden, so daß gilt: $ed = 1 \bmod \varphi(n)$. m wird nun verschlüsselt, indem die Nachricht c nach $c = m^e \bmod n$ berechnet wird. Der Empfänger (der gleichzeitig der Besitzer des Secret- Keys (d, n) ist) gewinnt nun die Botschaft zurück, indem er sie nach $m = c^d \bmod n$ ausrechnet.
Die Rechnung (Codierung und Decodierung) entspricht dann: $(m^e)^d \bmod n = m^{ed} \bmod n = m^{\varphi(n) + 1} \bmod n = m$ (nach Satz 9).

4.3 Die Herstellung der Schlüssel

Mit Hilfe des euklidische Algorithmus ist es, wie in Kapitel 3.8 beschrieben, möglich, den größten gemeinsamen Teiler von zwei Zahlen zu berechnen. Dieser Algorithmus kann je nach Zielsetzung erweitert werden. Eine Erweiterung des Algorithmus ist für die Generierung geeigneter Schlüssel für das RSA- Verfahren von besonderem Interesse, da mit dieser Erweiterung das Berechnen des Inversen von e möglich ist, so daß $ed = 1 \bmod \varphi(n)$ gilt. Im Folgenden werde ich diese Erweiterung des Algorithmus vorstellen.
Der in Kapitel 3.8 beschriebene Euklidische Algorithmus läßt sich in zwei Schritten notieren.
Es soll der größte gemeinsame Teiler von a und b mit $a > b$ berechnet werden.

Schritt 1: Berechne $r = a \bmod b$.
Wenn $r = 0$, ist b der ggT(a, b).
Wenn $r = 1$, dann sind a und b relativ prim (ggT(a, b) = 1).
Wenn $r > 1$, gehe zu Schritt 2.
Schritt 2: Ersetze den Wert von a durch den von b (a := b).
Ersetze den Wert von b durch den von r (a := r).
Gehe zu Schritt 1.

Diese zwei Schritte sollen nun zunächst so erweitert werden, daß während des Ausrechnens des ggT(a, b) gleichzeitig zwei Zahlen a‘ und b‘ generiert werden, so daß $a * a‘ + b * b‘ = ggT(a, b)$.
Bei der Ausführung des ersten Schrittes wird getestet, ob b der größte gemeinsame Teiler ist. Wenn dies der Fall ist, so gilt für a‘ und b‘: $a‘ = 0$ und $b‘ = 1$, da

$$a * 0 + b * 1 = b.$$

Wenn dies nicht der Fall ist, so wird bei der zweiten Ausführung der ggT(b, a mod b) berechnet. Wenn a mod b der ggT(a, b) ist, so ergibt sich für $a‘ = 1$, und b‘ ist der negative ganzzahlige Anteil des Quotienten (a/b), den ich mit int(a/b) abkürzen werde[9]. Es gilt dann:

$$a * 1 - \text{int}(a/b) = a \bmod b.$$

Auf diese Art kann immer ein a‘ und ein b‘ gefunden werden, so daß gilt:

$$a * a‘ + b * b‘ = ggT(a, b).$$

So können, wie im folgenden beschrieben, die zwei Schritte des Euklidischen Algorithmus erweitert werden, damit $a * a‘ + b * b‘$ der eventuelle ggT(a, b) ist. Stellt sich dann heraus, daß der errechnete Wert tatsächlich der ggT(a, b) ist, so sind die gesuchten Zahlen a‘ und b‘ gefunden.
Erreicht wird dies durch folgende Erweiterung:
Errechnet werden soll der ggT(a, b) mit $a > b$ und a‘ bzw. b‘ mit $a * a‘ + b * b‘ = ggT(a, b)$.

---

[9] „int“ steht für das englische „integer“, was soviel wie „ganze Zahl, das Ganze“ bedeutet. Int(a/b) gibt an, mit welchem x die Zahl b maximal multipliziert werden kann, ohne daß das Produkt xb den Wert von a überschreitet. Auf diese Weise läßt sich auch die Modulo-Funktion definieren: $a \bmod b = a - \text{int}(a/b)$.

Initialisierung: $(a_1, a_2, a_3) = (1, 0, a)$
$(b_1, b_2, b_3) = (0, 1, b)$
Schritt 1: Wenn $b_3 = 0$, dann ist $ggT(a, b) = a_3$, $a' = a_1$ und $b' = a_2$.
Wenn $b_3 = 1$, dann ist $ggT(a, b) = 1$, $a' = b_1$ und $b' = b_2$[10].
Andernfalls: Berechne $q = int(a_3/b_3)$.
Berechne $(z_1, z_2, z_3) = (a_1, a_2, a_3) – q(b_1, b_2, b_3)$.
Gehe zu Schritt 2.
Schritt 2: $(a_1, a_2, a_3) = (b_1, b_2, b_3)$
$(b_1, b_2, b_3) = (z_1, z_2, z_3)$
Gehe zu Schritt 1.

Zum Beispiel soll der ggT(27, 12) ausgerechnet werden:

Initialisierung: $(a_1, a_2, a_3) = (1, 0, 27)$
$(b_1, b_2, b_3) = (0, 1 ,12)$
1. Durchführung:
1. Schritt: $b_3 > 1$;
$q = int(27/12) = 2$
$(z_1, z_2, z_3) = (1, 0, 27) - 2(0, 1 ,12) = (1, -2, 3)$
2. Schritt: $(a_1, a_2, a_3) = (0, 1, 12)$
$(b_1, b_2, b_3) = (1, -2 , 3)$
2. Durchführung:
1. Schritt: $b_3 > 1$;
$q = int(12/3) = 1$
$(z_1, z_2, z_3) = (0, 1, 12) - (1, -2 ,3) = (-4, 9, 0)$
2. Schritt: $(a_1, a_2, a_3) = (1, -2, 3)$
$(b_1, b_2, b_3) = (-4, 9, 0)$
3. Durchführung:
1. Schritt: $b_3 = 0 \Rightarrow a' = 1, b' = -2$

Es gilt dann: $a * a' + b * b' = 27 * 1 + 12 * (-2) = 3 = ggT(27, 12)$.
Dabei ist zu erkennen, daß die Umformungen von a_3 und b_3 den Umformungen des einfachen Euklidischen Algorithmus entsprechen. Die selben Umformungen werden auch auf a_1, b_1, a_2 und b_2 angewendet, die jedoch als Startwerte 0 und 1 erhalten.

Da das Endziel jedoch ist, das modulare Inverse zu e zu berechnen, ist das Auftauchen von negativen Zahlen hinderlich, da das modulare Inverse immer eine positive ganze Zahl sein muß. Das Rechnen mit modulo verhindert hier das Auftauchen von negativen Zahlen:
Nachdem a' und b' berechnet wurden, kann nun $A = a' \bmod b$ und $B = b' \bmod a$ berechnet werden. Wenn a' negativ ist, so gilt: $a' \bmod b = (a' + b) \bmod b$. Gleiches gilt für b'.
Für das vorangegangene Beispiel bedeutet dies:
$A = a' \bmod b = 1 \bmod 12 = 1$
bzw. $B = b' \bmod a = -2 \bmod 27 = (-2 + 27) \bmod 27 = 25$.
Es gilt dann: $Aa \bmod b = Bb \bmod a = ggT(a, b)$.
Im Beispiel: $Aa \bmod b = Bb \bmod a = 1 * 27 \bmod 13 = 25 * 12 \bmod 27 = 3 = ggT(a, b)$.

[10] Diese Abfrage ist eigentlich nicht notwendig. Wenn $b_3 = 1$, wird bei der nächsten Durchführung b_3 auf jeden Fall gleich Null sein. Die Werte von b_1 und b_2 werden dann an a_1 und a_2 übergeben, so daß das Ergebnis auch ohne dieser Abfrage das gleiche wäre. Da jedoch, wie ich später noch zeigen werde, der Algorithmus in dem speziellen Fall der RSA-Verschlüsselung immer mit a und b, die relativ prim sind, ausgeführt wird, habe ich diese Abfrage zum Abkürzen eingefügt.

Wenn nun aber zwei Zahlen a und b gewählt werden, die relativ prim sind, dann gilt:

$$Aa \bmod b = Bb \bmod a = 1,$$

was bedeutet, daß A das modulare Inverse zu a bezüglich b und B das Inverse zu b bezüglich a ist.
So kann des Inverse zu e berechnet werden, indem an a der Wert von $\varphi(n)$ übergeben wird und an b der Wert der Wert von e. An dieser Stelle wird deutlich, daß e relativ prim zu $\varphi(n)$ sein muß.

4.4 Beispiel mit kleinen Zahlen

Nachdem ich nun beschrieben habe, wie geeignete Schlüssel hergestellt werden, gebe ich in diesem Kapitel ein Beispiel, bei dem mit sehr kleinen Zahlen gerechnet wird. Mit so kleinen Zahlen ist natürlich keinerlei Sicherheit gewährleistet. Sie eignen sich also nicht zum Verschlüsseln, jedoch genügen sie, um die Rechnung nachvollziehen zu können.
Zunächst müssen zwei Primzahlen gewählt werden. Sei p = 3 und q = 11. Für das Produkt n gilt n = pq = 33, und für $\varphi(n)$ folgt $\varphi(n) = (p-1)(q-1) = 20$.
Nun kann ein e gewählt werden, das jedoch relativ prim zu n sein muß. Dazu wird einfach eine Primzahl gewählt, die $\varphi(n)$ nicht teilt. So ist in diesem Beispiel e = 7 ein mögliche Wahl, da 20 nicht durch 7 teilbar ist und 7 eine Primzahl ist, folglich ist der ggT(20, 7) = 1.
Anschließend muß ein d gefunden werden, so daß $ed = 1 \bmod \varphi(n)$. Dies wird mittels des erweiterten Euklidischen Algorithmus möglich:

Initialisierung: $(a_1, a_2, a_3) = (1, 0, 20)$
$(b_1, b_2, b_3) = (0, 1, 7)$

1. Durchführung:
   1. Schritt: $b_3 > 1$;
   $q = \text{int}(20/7) = 2$
   $(z_1, z_2, z_3) = (1, 0, 20) - 2(0, 1, 7) = (1, -2, 6)$
   2. Schritt: $(a_1, a_2, a_3) = (0, 1, 7)$
   $(b_1, b_2, b_3) = (1, -2, 6)$
2. Durchführung:
   1. Schritt: $b_3 > 1$;
   $q = \text{int}(7/6) = 1$
   $(z_1, z_2, z_3) = (0, 1, 7) - (1, -2, 6) = (-1, 3, 1)$
   2. Schritt: $(a_1, a_2, a_3) = (1, -2, 6)$
   $(b_1, b_2, b_3) = (-1, 3, 1)$
3. Durchführung:

1. Schritt: $b_3 = 1 \Rightarrow a' = -1, b' = 3$

Dann folgt für B: $B = b' \bmod a = 3 \bmod 20 = 3$. Es gilt $Bb \bmod a = 3 * 7 \bmod 20 = 1$.Da e, der Public-Key, in diesem Beispiel also den Wert 7 hat, erhält d, der Secret-Key den Wert 3.
A muß nicht berechnet werden, da A das Inverse zu a, also zu $\varphi(n)$, bezüglich b ist, in diesem Fall also zu e. Da jedoch mit $\varphi(n)$ als Modul gerechnet werden soll, ist dies für die Verschlüsselung vollkommen uninteressant.
Nun will Person A der Person B eine Botschaft schicken. Dazu gibt B seinen Public-Key an A, der nach $c = m^e \bmod n$ die Nachricht, die an B überreicht wird, berechnet. Dazu wird das normale Alphabet in Zahlen umgewandelt. So kann z.B. a den Wert 1 erhalten, b den Wert 2, c den Wert 3, usw., oder aber die Buchstaben werden nach dem ASCII-Code[11] umgewandelt. Soll zum Beispiel der „Text“ m = 8 verschlüsselt werden, so ergibt sich als Nachricht: $c = 8^7 \bmod 33 = 2$. Die Nachricht c = 2 wird nun an B übermittelt. Dieser gewinnt die Botschaft nach $m = c^d \bmod n = 2^3 \bmod 33 = 8 \bmod 33 = 8$ zurück. Als weiteres Beispiel soll die Botschaft 5 verschlüsselt werden:

[11] American Standard Code for Information Interchange

Verschlüsselung: $5^7 \bmod 33 = 78125 \bmod 33 = 78125 - 2367 * 33 = 14.$
Entschlüsselung: $14^3 \bmod 33 = 2744 \bmod 33 = 2744 - 83 * 33 = 5.$

4.5 Signieren

Wie in Kapitel 2.2 schon erwähnt, ist mit dem RSA-Algorithmus auch ein Signieren möglich. Damit ist folgendes gemeint: Der Absender einer Nachricht verschlüsselt die Botschaft mit seinem Secret-Key. Dadurch das $(m^e)^d \bmod n = m^{ed} \bmod n$ dasselbe ist wie $(m^d)^e \bmod n = m^{de} \bmod n$, kann die Nachricht mit dem Public-Key wieder entschlüsselt werden. Da der Public-Key jedoch öffentlich ist, bleibt die Botschaft für jeden lesbar. Wenn also A eine signierte Nachricht an B sendet, so kann nicht nur B die Botschaft mit dem Public-Key entschlüsseln, sondern jeder andere auch. Der Nutzen liegt jedoch darin, daß B sich, sobald ein sinnvolles entschlüsseln möglich ist, sicher sein kann, daß die Nachricht von A stammt. Wenn jedoch die Entschlüsselung fehlschlägt, was heißt, daß keine vernünftigen Sätze oder Worte zustande kommen, so bedeutet dies, daß die Nachricht nicht mit dem zu dem Public-Key gehörendem Secret-Key verschlüsselt wurde oder die Nachricht nachträglich verändert wurde. Die Signierung wirkt so wie eine elektronische Unterschrift und kann im Rahmen eines Kryptosystems benutzt werden, um beispielsweise Kaufverträge oder Bestellungen über das Internet abzuwickeln.
Dabei können natürlich auch verschlüsselte Botschaften oder auch Schlüssel selbst signiert werden, so daß es möglich ist, komplizierte Kryptosysteme aufzubauen, in denen Nachrichten verschlüsselt werden, und gleichzeitig eine Sicherheit über den Absender verschafft wird.

4.6 Noch einmal zur Sicherheit

Es gibt kein mathematisches Verfahren, um nachzuweisen, wie sicher ein Codierungsverfahren ist. Deshalb ist die einzige Methode, die Sicherheit eines Verfahrens zu beurteilen, zu sehen, ob irgend jemand einen Weg findet, den Code zu „knacken".
Daher sollen in diesem Kapitel Techniken besprochen werden, die ein Lauscher nutzen könnte, um, auch ohne den Secret-Key zu besitzen, eine Nachricht zu entschlüsseln.
Eine Möglichkeit ist die Zerlegung von n in seine Primfaktoren. n zu faktorisieren, würde es dem Lauscher ermöglichen, $\varphi(n) = (p-1)(q-1)$ zu berechnen und anschließend den Secret-Key d mit Hilfe des erweiterten Euklidischen Algorithmus zu berechnen
Große zusammengesetzte Zahlen zu faktorisieren ist ein bekanntes Problem, für das schon Fermat (1601-1605) und Legendre (1752-1833) algorithmische Lösungsansätze entwickelten. Der schnellste heute bekannte Algorithmus wurde von Richard Schroeppel entwickelt und beruht auf der Arbeit von Legendre. Die Funktionsweise dieses Algorithmus würde den Rahmen dieser Arbeit bei weitem sprengen. Wichtig ist jedoch, die Rechenzeit zu berücksichtigen, mit der es möglich ist, eine Zahl n in ihre Primfaktoren zu zerlegen.
Der Algorithmus nach Schroeppel ermöglicht es, n in ungefähr $S \approx n^{\sqrt{(\ln(\ln(n))/\ln(n))}}$ Schritten zu zerlegen. Tabelle 3 gibt an, wie viele Schritte nötig sind, um eine Zahl der angegebenen Länge in ihre Primfaktoren zu zerlegen.

Zahl der Stellen von n	Anzahl der benötigten Operationen
50	$1,4 * 10^{10}$
100	$2,3 * 10^{15}$
200	$1,2 * 10^{23}$
500	$1,3 * 10^{39}$

Tab.3

In Anbetracht dieser Anzahl der benötigten Rechenoperationen wird nun auch deutlich, warum die in Tabelle 1 aufgeführten Rechenzeiten so enorm groß werden. Ab einer Schlüssellänge von 200 Bit gibt es zur Zeit keinen Algorithmus, der eine Primfaktorzerlegung in einer brauchbaren

Zeit durchführt. Bei der zur Zeit gebräuchlichen Schlüssellänge von 1024- 4096 Bit ist es einem Lauscher oder auch einem Kryptoanalytiker nicht möglich, über die Zerlegung von n den Secret-Key d zu erhalten.
Es stellt sich nun die Frage, ob es einen Weg gibt, an den Secret-Key zu gelangen, ohne n zu faktorisieren.
Im folgenden möchte ich zeigen, daß andere Wege, die es einem Lauscher ermöglichen würden, eine verschlüsselte Nachricht zu entschlüsseln, noch umständlicher wären als der, n zu faktorisieren.
Eine Möglichkeit, an d zu gelangen, wäre es, $\varphi(n)$ zu generieren, ohne n zu faktorisieren. Wäre ein Kryptoanalytiker in der Lage, $\varphi(n)$ zu generieren, so könnte er mit dem erweiterten Euklidischen Algorithmus das Inverse zu dem Public-Key e herstellen und auf diese Weise an d gelangen. Da jedoch mittels $\varphi(n)$ auch n auf einfache Weise in Primfaktoren zerlegt werden kann, wäre mit der Möglichkeit, $\varphi(n)$ zu generieren, auch eine einfachere Möglichkeit gegeben, n zu faktorisieren. Da es, wie am Anfang dieses Kapitels schon festgestellt, zur Zeit keine schnellere Methode gibt, als die oben vorgestellte, kann auch das generieren von $\varphi(n)$ nicht schneller sein, als n zu faktorisieren.
Daher werde ich nun zeigen, daß mit Hilfe der Generierung von $\varphi(n)$ auch eine Zerlegung von n ohne weiteres möglich ist. Dazu muß zunächst nur $(p + q)$ ausgerechnet werden. Dies ist möglich, weil $\varphi(n) = n - (p + q) + 1$ ist. Da n und $\varphi(n)$ bekannt sind, kann auch $(p + q)$ ausgerechnet werden.
Der Beweis hierfür läßt sich wie folgt führen:

$$n - (p + q) + 1 = \varphi(n)$$

$\varphi(n)$ errechnet sich durch $(p-1)(q-1)$, so ergibt sich:

$$n - (p + q) + 1 = (p-1)(q-1)$$
$$n - (p + q) + 1 = pq - p - q + 1$$

Für n gilt: $n = pq$, daher gilt:

$$n - (p + q) + 1 = n - (p + q) + 1$$

q.e.d.

Nachdem nun also $(p + q)$ ausgerechnet wurde, kann auch $(p - q)$ ausgerechnet werden:

$$(p - q) = \sqrt{((p + q)^2 - 4n)}$$

Auch hierzu der Beweis:

$$(p - q) = \sqrt{((p + q)^2 - 4n)}$$
$$(p - q) = \sqrt{(p^2 + 2\,pq + q^2 - 4n)} \qquad \text{(1. Binomische Formel)}$$

Auch hier gilt $n = pq$, daher gilt:

$$(p - q) = \sqrt{(p^2 + 2\,pq + q^2 - 4pq)}$$
$$(p - q) = \sqrt{(p^2 - 2\,pq + q^2)}$$
$$(p - q) = (p - q) \qquad \text{(2. Binomische Formel)}$$

q.e.d.

Nun kann q einfach nach $((p + q) - (p - q))/2$ ausgerechnet werden. Somit ist n in seine Primfaktoren zerlegt.
Auch d zu berechnen, ohne n zu faktorisieren und ohne $\varphi(n)$ auszurechnen, wäre eine Methode, die es auf einfachem Wege ermöglichen würde, n Primfaktoren zu zerlegen. Nach Miller[12] ist es möglich, jede zusammengesetzte Zahl n in ihre Primfaktoren zu zerlegen, wenn ein beliebiges Vielfache von $\varphi(n)$ bekannt ist. Da $ed = 1 \bmod \varphi(n)$ gilt, kann nach $ed - 1$ ein Vielfaches von $\varphi(n)$ ausgerechnet werden. Somit wäre eine Zerlegung von n geleistet. Wenn dies möglich wäre, dann wäre auch in diesem Fall eine

[12] Miller, G.L. Rieman's hypothesis and tests for primality, New Mexico, May 1975

einfache Primfaktorzerlegung möglich. Da diese aber nicht möglich ist, kann auch die Generierung von d nicht einfacher sein als die Zerlegung von n.
Abschließend läßt sich festhalten, daß das Berechnen der „e-ten Wurzel modulo n“, obwohl es ein bekanntes Problem ist, bisher nicht schneller möglich ist als die Zerlegung von n in Primfaktoren, was bei großem n so zeit- und rechenaufwendig ist, daß die Rechnung nicht mehr durchgeführt werden kann.

5 Effiziente Implementation des Algorithmus

5.1 Rechnen mit großen Zahlen

Wie in Kapitel 4.6 deutlich geworden ist, hängt die Sicherheit dieses Verfahrens von der Länge der Schlüssel ab. Im krassen Gegensatz zu der Länge der Schlüssel (mindestens 1024 Bit, normalerweise bis zu 4096 Bit) steht die Länge des längsten verwendbaren Variablenformates mit 32 Bit. Es wird deutlich, daß die standardmäßigen Rechenoperationen nicht durchgeführt werden können. Im folgenden werde ich eine mögliche Methode beschreiben, mit so großen Zahlen zu rechnen.
Eine Zahl im Dezimalsystem dargestellt, ist ein Folge von Ziffern zwischen 0 und 9. Anders ausgedrückt besteht eine 20-stellige Zahl aus einem Feld von 20 Ziffern. Ein solches Feld wird auch als array[13] bezeichnet. Es kann nun jede Ziffer des arrays einzeln betrachtet werden. Dieses Verfahren ist nichts anderes als das „schriftliche“ Addieren, Subtrahieren oder auch Dividieren und Multiplizieren. Das schriftliche Rechnen setze ich als bekannt voraus und werde es daher nicht weiter erläutern.
Jede „Ziffer“ eines arrays kann im Dezimalsystem die Werte 0 bis 9 annehmen. Wenn mit besonders großen Zahlen gerechnet wird, ist dies jedoch nicht optimal. Wird auf dem Computer ein array angelegt, so kann jede Stelle dieses arrays sehr viel größere Werte annehmen. So kann z.B. jede Stelle eines arrays die Werte eines „word“, also die Werte von 0 bis 65535, annehmen. Das Arbeiten mit einer größeren Basis als 10 hat keine Auswirkungen auf das Rechenverfahren. Es kann weiterhin genauso „schriftlich“ gerechnet werden. Lediglich der Überlauf findet zu einem späterem Zeitpunkt statt.
Dieses Verfahren hat folgenden Vorteil: Durch das Rechnen mit einer höheren Basis werden weniger Stellen verwendet, was wiederum bedeutet, daß weniger Rechenschritte durchgeführt werden müssen.
Folgendes Beispiel verdeutlicht die Vorteile des Rechnens mit höherer Basis. Es sollen die Zahlen 88 und 14 Addiert werden. Zunächst geschieht dies im Dezimalsystem, wobei das Ergebnis drei Stellen hat. Dabei werden zwei Additionen durchgeführt und zwei Überträge weitergegeben. Es können also vier Rechenschritte festgehalten werden.

	Basis: 10				Basis: 100	
Stelle	1	2	3		1	2
		8	8			88
+		1	4			14
Überlauf	1	1			1	
Ergebnis	1	0	2		1	2

Wenn jedoch mit einer größeren Basis gerechnet wird, wie im Beispiel mit der Basis 100, so werden nur zwei Rechenschritte durchgeführt: eine Addition und ein Übertrag.
Das Rechnen mit einer hohen Basis hat jedoch einen Nachteil. Wenn die Zahlen ausgegeben werden sollen, müssen die Zahlen in das Dezimalsystem zurückgerechnet werden. Am Beispiel wird dies am Ergebnis deutlich: 12. Die Addition von 88 und 14 hat im Dezimalsystem freilich nicht das Ergebnis 12, sondern 1*100 + 2 * 1 = 102.

[13] englisch für „stattliche Reihe“, „Ordnung“

5.2 Potenzieren mod n mit großen Zahlen

Es gibt jedoch auch andere Möglichkeiten, zu große Zahlen zu vermeiden. Insbesondere beim potenzieren mit dem Secret- bzw. dem Public-Key geraten die Zahlen in Zahlenbereiche, die es nur auf Umwegen zulassen, vom Computer berechnet zu werden. Da jedoch mit modulo gerechnet wird, bietet sich hier die Möglichkeit, die Zahlen niedrig zu halten.
Gerechnet werden soll $m = c^d \bmod n$. d kann mittels der Binärzerlegung in einzelne Summanden zergliedert werden. Es soll gelten: $d = b_1 + b_2 + b_3 + ...$ und somit gilt: $c^d \bmod n = c^{b1 + b2 + b3 + ...} \bmod n$. Nach Satz 1 ist dies dasselbe wie $(c^{b1} * c^{b2} * c^{b3} * ...) \bmod n$. Daher kann wie in dem folgenden Beispiel verfahren werden:
Es soll m berechnet werden, für das gilt: $m = 30^{25} \bmod 13$. Obwohl 30^{25} eine, im Verhältnis zu den Zahlen mit denen tatsächlich verschlüsselt wird, kleine Zahl ist, soll hier das Berechnen so großer Zahlen erspart bleiben. Daher wird zunächst 25 zerlegt: $25 = 2^0 + 2^3 + 2^4$. Dies bedeutet: $30^{25} = 30^1 * 30^8 * 30^{16}$ (nach Satz 1).
$30^1 \bmod 13$ kann leicht berechnet werden: $30^1 \bmod 13 = 4$. Nun kann auch $30^2 \bmod 13$ auf einfachem Wege berechnet werden: $30^2 \bmod 13 = (30^1 * 30^1) \bmod 13 = (4 * 4) \bmod 13 = 3$. Auf diese Art und Weise ergibt sich Tabelle 4.

$30^1 \bmod 13$	-	-	4
$30^2 \bmod 13$	$(30^1 * 30^1) \bmod 13$	$(4 * 4) \bmod 13$	3
$30^4 \bmod 13$	$(30^2 * 30^2) \bmod 13$	$(3 * 3) \bmod 13$	9
$30^8 \bmod 13$	$(30^4 * 30^4) \bmod 13$	$(9 * 9) \bmod 13$	3
$30^{16} \bmod 13$	$(30^8 * 30^8) \bmod 13$	$(3 * 3) \bmod 13$	9

Tab. 4

Ziel war es aber, $30^{25} \bmod 13 = (30^1 * 30^8 * 30^{16}) \bmod 13$ zu berechnen, was nach dem gleichen Schema wie oben geschehen kann. Die Ergebnisse dazu können einfach der Tabelle entnommen werden:
$(30^1 * 30^8) \bmod 13 = (4 * 3) \bmod 13 = 12$. Daraus folgt: $((30^1 * 30^8) * 30^{16}) \bmod 13 = (12 * 9) \bmod 13 = 4$.
Als Ergebnis ergibt sich also $30^{25} \bmod 13 = 4$.
Dies zu berechnen, war mittels dieser Methode möglich, ohne in dem Bereich so großer Zahlen zu geraten, daß ein einfaches Ausrechnen nicht mehr möglich gewesen wäre.

6 DIE JAGD AUF GROßE PRIMZAHLEN

6.1 Wozu große Primzahlen ?

In den Kapiteln 4.1 und 4.6 ist bereits deutlich geworden, daß eine Zerlegung von n in die Primfaktoren nur Schwierig ist, wenn n genügend groß ist. Aus der Tabelle 1 und 3 ist genauer ersichtlich, daß ein guter Schutz erst ab einer Schlüssellänge von 1024 Bit besteht. Um Schlüssel mit dieser Länge (ca. 307 Stellen) herstellen zu können, ist es jedoch, wie auch aus den vorangegangenen Kapiteln deutlich wurde, notwendig, ausreichend große Primzahlen herzustellen. Da sich n nach n = pq berechnet, muß eine der Primzahlen, wenn p und q ca. gleich groß sein sollen, ungefähr die Größe $\sqrt{n}$ haben.
Drei Primzahltests, mit denen das Generieren von Primzahlen möglich ist, möchte ich in diesem Abschnitt der Arbeit vorstellen. Anschließend sollen die verschiedenen Tests kritisiert werden, und ich werde auch begründen, warum ich welche Tests für meine Testprogramme benutzt habe.
Der erste dieser Test, das sogenannte „Sieb des Erathostenes“, ist ein deterministischer Test, was bedeutet, daß die durch ihn generierten Primzahlen nachweislich Primzahlen sind. Dem gegenüber stelle ich zwei probablistische Primzahlentests, die nur zu einer gewissen Wahrscheinlichkeit die Primzahleigenschaften einer Zahl garantieren.

6.2 Sieb des Erathostenes

Zu der Entstehung des Sieb des Erathostenes gibt es folgende Sage:
Erathostenes, der vom dritten Ptolomäer Ptolomaeus Euergetes, dem Herrscher Ägyptens, als Direktor an die Bibliothek von Alexandria um 220 v.Chr. gerufen wurde, schlägt bei einem Spaziergang am Strand folgendes vor:
Ein Bibliotheksdiener, damals wahrscheinlich ein Sklave, soll alle Zahlen von 1 bis 10000 in den Sand schreiben. Nach getaner Arbeit soll er alle Vielfachen von 2 wieder streichen. So werden alle geraden Zahlen ausgesiebt. Als kleinste Zahl bleibt so die 3 stehen. Nun soll der Sklave alle Vielfachen von 3 wegstreichen. Anschließend alle von 5, 7, usw., bis schließlich nur noch Zahlen stehen bleiben, die nicht Vielfaches einer kleineren Zahl, also Primzahl sind.
Zu Ehren Erathostenes und weil diese Methode entfernt an das Sieben von Sand erinnert, heißt diese Methode das Sieb des Erathostenes.
Der Sklave beschwerte sich, er müsse ja alle Zahlen bis 10000 überprüfen, was unglaublich lange dauern würde. Doch Erathostenes beruhigte ihn, er müsse nur die Zahlen bis $\sqrt{10000}$ prüfen[14].
Dies ist folgendermaßen zu zeigen:
Wenn die Vielfachen einer Zahl x weggestrichen werden sollen, dann wird als kleinstes noch nicht weggestrichene Vielfache eine Zahl y = nx weggestrichen, für die gilt $y \leq x^2$. Jede Zahl, die kleiner als x^2 ist und Vielfaches von x ist, ist auch ein Vielfaches von einer Zahl $z < x$ und ist somit schon gestrichen worden. Daher brauchen nur die Zahlen bis $\sqrt{m}$ geprüft werden, wenn alle Primzahlen bis m mit dem Sieb des Erathostenes ermittelt werden sollen.
Das Sieb des Erathostenes ist ein erster Primzahlentest, der den Vorteil hat, daß die getesteten Zahlen nachweislich Primzahlen sind. Eine mögliche Umsetzung in ein Programm liegt auf CD bei.

---

[14] [23] S. 66f

6.3 Der Fermat-Test

Eine andere Möglichkeit, Primzahlen zu Testen, bietet der in Kapitel 3.5 bewiesene Satz 7b. Die Umkehrung dieses Satzes könnte lauten:
Berechne $x = r^p \bmod p$, mit einer beliebigen zu p teilerfremden Basis r. Wenn x = r, dann ist p möglicherweise eine Primzahl.
Wichtig ist, daß die Zahl p nur *möglicherweise* eine Primzahl ist! Wenn nach Satz 7a gilt, das für jede Primzahl p, unter der Voraussetzung, daß ggT(r, p) = 1, $r^p \bmod p = r$ gilt, dann bedeutet dies noch nicht, daß unter gleichen Voraussetzungen nicht auch zusammengesetzte Zahlen diese Bedingungen erfüllen könnten.
Es läßt sich aber sagen, daß eine Zahl keine Primzahl ist, wenn sie diese Bedingung nicht erfüllt, und daß kann für einen Primzahltest nutzbar gemacht werden.
Soll beispielsweise untersucht werden, ob die Zahl 17 eine Primzahl ist, so kann gerechnet werden (mit der Testbasis 2): $2^{17} \bmod 17 = 2$. So kann es sein daß 17 eine Primzahl ist. Um dieses Ergebnis zu bestätigen, kann nun mit einer weiteren Testbasis gerechnet werden: $3^{17} \bmod 17 = 3$. Die Wahrscheinlichkeit, daß eine Zahl prim ist, steigt mit der Anzahl der bestandenen Tests zu verschiedenen Testbasen. Dies läßt sich auch leicht an einem Beispiel zeigen:
Getestet werden soll, ob 341 eine Primzahl ist. Dies läßt sich nach der in Kapitel 5.2 vorgestellten Methode zum Potenzieren mod n berechnen:
$341 = 2^0 + 2^2 + 2^4 + 2^6 + 2^8$. Mit der Testbasis 2 folgt also: $2^{341} = 2^{1 + 4 + 16 + 64 + 256}$. Es ergibt sich folgende Tabelle 5:

$2^1 \bmod 341$	-	-	2
$2^2 \bmod 341$	-	-	4
$2^4 \bmod 341$	-	-	16
$2^8 \bmod 341$	-	-	256
$2^{16} \bmod 341$	-	-	64
$2^{32} \bmod 341$	$(2^{16} * 2^{16}) \bmod 341$	$(64 * 64) \bmod 341$	4
$2^{64} \bmod 341$	$(2^{32} * 2^{32}) \bmod 341$	$(4 * 4) \bmod 341$	16
$2^{128} \bmod 341$	$(2^{64} * 2^{64}) \bmod 341$	$(16 * 16) \bmod 341$	256
$2^{256} \bmod 341$	$(2^{128} * 2^{128}) \bmod 341$	$(256 * 256) \bmod 341$	64

Tab. 5

$2^{341} \bmod 341$ berechnet sich nun folgendermaßen: $(2^1 * 2^4) \bmod 341 = (2 * 16) \bmod 341 = 32$. Weiterhin gilt: $((2^1 * 2^4) * 2^{16}) \bmod 341 = (32 * 64) \bmod 341 = 2$ und $((2^1 * 2^4) * 2^{16}) * 2^{64}) \bmod 341 = (2 * 16) \bmod 341 = 32$. Schließlich folgt: $(((2^1 * 2^4) * 2^{16}) * 2^{64}) * 2^{256}) \bmod 341 = (32 * 64) \bmod 341 = 2$.
341 scheint also eine Primzahl zu sein. Zur Sicherheit wird dieser Test noch einmal zur Basis 3 durchgeführt:

$3^1 \bmod 341$	-	-	3
$3^2 \bmod 341$	-	-	9
$3^4 \bmod 341$	-	-	81
$3^8 \bmod 341$	-	6561 mod 341	82
$3^{16} \bmod 341$	$(3^8 * 3^8) \bmod 341$	(82 * 82) Mod 341	245
$3^{32} \bmod 341$	$(3^{16} * 3^{16}) \bmod 341$	$(245 * 245) \bmod 341$	9
$3^{64} \bmod 341$	$(3^{32} * 3^{32}) \bmod 341$	$(9 * 9) \bmod 341$	81
$3^{128} \bmod 341$	$(3^{64} * 3^{64}) \bmod 341$	$(81 * 81) \bmod 341$	82
$3^{256} \bmod 341$	$(3^{128} * 3^{128}) \bmod 341$	$(82 * 82) \bmod 341$	245

Tab. 6

$3^{341} \bmod 341$ berechnet sich: $(3^1 * 3^4) \bmod 341 = (3 * 81) \bmod 341 = 243$. Außerdem gilt: $((2^1 * 2^4) * 2^{16}) \bmod 341 = (243 * 245) \bmod 341 = 201$ und $(((2^1 * 2^4) * 2^{16}) * 2^{64}) \bmod 341 =$

(201 * 81) mod 341 = 254. Es folgt: ((((2[1] * 2[4]) * 2[16]) * 2[64]) * 2[256]) mod 341 = (254 * 245) mod 341 = 168.
Es stellt sich durch den zweiten Test zur Basis 3 heraus, daß 341 keine Primzahl ist. Tatsächlich gilt 11 * 31 = 341. Trotzdem hat sie den Fermat-Test zur Basis zwei bestanden. 341 wird daher als Pseudoprimzahl zur Basis 2 bezeichnet. Ebenso gibt es Pseudoprimzahlen zur Basis 3 usw. Soll der Fermat-Test also wirksam sein, so muß er mit mehreren Testbasen durchgeführt werden.
Es gibt jedoch auch Zahlen, die den Fermat-Test zu jeder Basis bestehen und trotzdem zusammengesetzt sind. Diese Zahlen werden Carmichael-Zahlen genannt. Tabelle 7 zeigt alle Carmichael-Zahlen die kleiner als 100000 sind.

561	1105	1729	2465
2821	6601	8911	10585
15841	29341	41041	46657
52633	62745	63973	75361

Tab. 7

Da jedoch die Anzahl der Carmichael-Zahlen sehr viel kleiner ist als die der Primzahlen, ist der Fermat-Test eine brauchbare Methode zur Bestimmung von Primzahlen[15].

6.4 Test nach Miller und Rabin

Ein anderer Primzahlentest, der von Miller und Rabin entwickelt wurde, beruht auf folgender Überlegung:
Wenn eine Zahl p den Fermat-Test besteht, dann gilt: $b^{n-1} \bmod n = 1$ (nach Satz 7a). Das bedeutet, daß $b^{n-1} - 1$ von n geteilt wird ($n | b^{n-1} - 1$). Da n jedoch eine ungerade Zahl ist, läßt sie sich in jedem Fall in der Form n = 2m + 1 schreiben. Wenn dies eingesetzt wird, bedeutet dies: $n | b^{2m} - 1$. Für $b^{2m} - 1$ kann auch $(b^m - 1)(b^m + 1)$ geschrieben werden. Daraus folgt: $n | (b^m - 1)(b^m + 1)$, was bedeutet, daß n entweder $(b^m - 1)$ teilt oder $(b^m + 1)$. Würde n beide der Faktoren teilen, dann müßte es auch deren Differenz 2 teilen. n ist jedoch ungerade. Es gilt demnach $b^m \equiv \pm 1 \bmod n$. Wenn jedoch n keine Primzahl ist, dann wäre es denkbar, daß einige Faktoren von n $(b^m - 1)$ teilen und andere $(b^m + 1)$. Dann hätte die Zahl zwar den Fermat-Test bestanden, nicht jedoch den Test nach Miller und Rabin.
Dies läßt sich wie folgt anwenden: Wie in Kapitel 6.3 schon gezeigt, besteht 341 den Fermat-Test zur Basis 2. In einem zweiten Schritt kann nun $2^{170} \bmod 341$ ausgerechnet werden. Dies gestaltet sich recht einfach, da die benötigten Zwischenergebnisse wiederum aus der Tabelle 5 abgelesen werden können:
$170 = 2^1 + 2^3 + 2^5 + 2^7 = 2 + 8 + 32 + 128$. Nach der Tabelle ergibt sich für $(2^2 * 2^8) \bmod 341 = (4 * 256) \bmod 341 = 1$, $((2^2 * 2^8) * 2^{32}) \bmod 341 = (1 * 4) \bmod 341 = 4$ und schließlich $(((2^2 * 2^8) * 2^{32}) * 2^{128}) \bmod 341 = (4 * 256) \bmod 341 = 1$. 341 könnte demnach immer noch eine Primzahl sein. Dann müßte nach dem oben geführten Beweis auch folgendes gelten: $2^{85} \bmod 341 = \pm 1$.
Und wieder können die Zwischenergebnisse der Tabelle 5 entnommen werden:
$85 = 2^0 + 2^2 + 2^4 + 2^6$. Daher gilt: $(2^1 * 2^4 * 2^{16}) \bmod 341 = (2 * 16 * 65536) \bmod 341 = 2$ und folglich: $((2^1 * 2^4 * 2^{16}) * 2^{64}) \bmod 341 = (2 * 16) \bmod 341 = 32$. An dieser Stelle wird deutlich, daß 341 keine Primzahl ist.[16]
Dies läßt sich nun folgendermaßen verallgemeinern:
Da n eine ungerade Zahl ist, ist n – 1 mit Sicherheit gerade. Daher kann n – 1 immer in der Form 2a*t mit $a \geq 1$ und t ungerade dargestellt werden. Es gilt dann:

---

[15] Es gilt tatsächlich: $\lim_{x \to \infty} \frac{Anzahl\_der\_Carmichael-Zahlen < x}{Anzahl\_der\_Primzahlen < x} = 0$ [32]

[16] [23] S.165

$b^{n-1} - 1 = b^{2a*t} - 1$.

Da nun $b^{2a*t} = b^{2*2\,\wedge(a-1)*t}$ gilt, kann die dritte binomische Formel angewandt werden. Es folgt dann:

$b^{2a*t} - 1 = b^{2*2\,\wedge(a-1)*t} - 1 = (b^{2\,\wedge(a-1)*t} - 1)\,(b^{2\,\wedge(a-1)*t} + 1)$.

Wenn dieses Verfahren erneut auf den Ausdruck in der ersten Klammer angewendet wird, so ergibt sich:

$b^{2a*t} - 1 = (b^{2*2\,\wedge(a-2)*t} - 1)\,(b^{2\,\wedge(a-1)*t} + 1)$

$b^{2a*t} - 1 = (b^{2\,\wedge(a-2)*t} - 1)\,(b^{2\,\wedge(a-2)*t} + 1)\,(b^{2\,\wedge(a-1)*t} + 1)$.

So kann nun weiter gerechnet werden, bis sich schließlich für $2^{(a-i)} = 2^0$ ergibt. Dies ist dann der Fall, wenn i = a ist. Dann läßt sich folgende Gleichung aufstellen:

$b^{2a*t} = (b^t - 1)\,(b^t - 1) * \ldots * (b^{2\,\wedge(a-2)*t} + 1)\,(b^{2\,\wedge(a-1)*t} + 1)$.

Es gilt letztendlich:

$b^{n-1} - 1 = (b^t - 1)\,(b^t - 1) * \ldots * (b^{2\,\wedge(a-1)*t} + 1)$.

Diese Gleichung erlaubt nun folgenden Primzahltest:
Wenn p eine Primzahl ist, dann teilt p genau einen der Faktoren auf der rechten Seite der Gleichung $b^{n-1} - 1 = (b^t - 1)\,(b^t - 1) * \ldots * (b^{2\,\wedge(a-1)*t} + 1)$. Das bedeutet, daß $b^t \bmod p = 1$ oder $b^{2\,\wedge i*t} = -1 \bmod p$, für ein i für das gilt $0 \le i < a$.
Zwei der Faktoren kann p nur teilen, wenn es zusammengesetzt ist.
Natürlich gilt auch hier die Umkehrung nicht! Wenn eine Zahl den Test besteht, kann sie trotzdem zusammengesetzt sein. Solche Zahlen nennen sich dann starke Pseudoprimzahlen. So gibt es zum Beispiel in dem Zahlenbereich bis $25*10^9$ genau 21835 Pseudoprimzahlen, jedoch nur 4842 starke Pseudoprimzahlen[17].
Auf diese Art und Weise bietet der Test nach Rabin und Miller die Möglichkeit, Zahlen zu testen, ob sie prim sind oder nicht. Dabei ist diese Methode sicherer als der Fermat-Test.

6.5 Andere Tests

Neben den drei von mir beschriebenen Primzahltests gibt es zahlreiche andere Algorithmen zum testen von Primzahlen (so sei hier verwiesen auf z.B. Monte-Carlo, Lehman, Solovay). Ein besonders für die Kryptographie interessant gewordener Algorithmus ist der 1989 von Ueli M. Maurer vorgeschlagene. Der Maurer-Algorithmus ist eine Methode, prüfbare Primzahlen herzustellen. Dabei zeigt die Diplomarbeit von Robert Müller, daß dieser Algorithmus mit geringfügigen Modifikationen nur unwesentlich langsamer ist, als ein probablistischer Test. Besonders geeignet ist dieses Verfahren zur Herstellung von Primzahlen, die bestimmte Anforderungen, wie z.B. einer vorgegebenen Länge, erfüllen sollen. Auch das macht diesen Algorithmus für die Public-Key-Verschlüsselung sehr reizvoll.
Die Funktionsweise dieses Algorithmus ist jedoch zu umfangreich, als daß eine Beschreibung in dieser Arbeit noch Platz gefunden hätte.

6.6 Vergleich und Kritik der Methoden

Wenn auch das Sieb des Erathostenes eine sehr einfache Möglichkeit bietet, prüfbare Primzahlen herzustellen, so ist dieses Verfahren zur Herstellung von Primzahlen für die Public-Key-Verschlüsselung denkbar ungünstig. Mittels dieser Methode läßt sich nämlich nicht entscheiden, ob eine sehr große Zahl prim ist oder nicht. Vor dem Entscheiden ob eine Zahl prim ist, steht das Ausrechnen aller vorhergehender Primzahlen. Auch wenn beim Prüfen einer Zahl m „nur“ bis $\sqrt{m}$ geprüft werden muß (s. 6.2), so ist dies bei so großen Zahlen, wie sie für die Verschlüsselung benötigt werden, viel zu zeit- und rechenaufwendig.
Der Fermat-Test verschafft diesem Problem Abhilfe. Mit ihm ist es möglich, auch sehr große Zahlen in angemessenem Zeitrahmen daraufhin zu überprüfen, ob sie prim sind oder nicht. Jedoch sind die von dem Fermat-Test geprüften Zahlen nur zu einer gewissen

[17] [23] S.166

Wahrscheinlichkeit prim. Die Wahrscheinlichkeit, daß eine zusammengesetzte Nicht-Carmichaelzahl n den Test mit einer zufällig gewählten Basis b (ggT(n, b) = 1) nicht besteht, ist mindestens ½. So ist die Wahrscheinlichkeit bei k zu den Basen b_1, b_2, ..., b_k bestandenen Tests dafür, das n zusammengesetzt ist: $1/2^k$. Das bedeutet, daß n mit einer Wahrscheinlichkeit von $1 – 1/2^k$ eine Primzahl oder eine Carmichaelzahl ist.
Bei der Geschwindigkeit, mit der ein Computer diese Rechnung durchführen kann, ist dieses Verfahren durchaus ein geeignetes, wenn auch nicht optimales Verfahren zur Generierung von Primzahlen.
Eine Optimierung dieses Verfahren stellt der Test nach Miller und Rabin dar. Bei ihm nehmen die Wahrscheinlichkeiten, daß eine Zahl prim ist noch wesentlich schneller ab. Es gilt, daß n mit einer Wahrscheinlichkeit von $1- 1/4^k$ eine Primzahl ist[18]. Im Vergleich zum Fermat-Test wird also entweder Rechenzeit eingespart, oder die Wahrscheinlichkeit, daß die getestete Zahl prim ist, nimmt zu. Dadurch ist dies von den hier vorgestellten Verfahren das zur Generierung der Primzahlen am besten geeignetste.
Noch wesentlich besser ist natürlich die Benutzung eines Verfahrens, wie es Maurer vorschlägt und Robert Müller in seiner Diplomarbeit optimiert. Dann wäre es möglich, mit prüfbaren Primzahlen zu rechnen, während die Laufzeiten in einem annehmbaren Rahmen liegen. Ein solches Verfahren vorzustellen, ist mir jedoch leider im Rahmen dieser Arbeit unmöglich.

---

[18] [23] S.169

7 ANWENDUNGEN DES RSA-ALGORITHMUS

7.1 PGP

Pretty **G**ood **P**rivacy Inc. ist eine von Philip Zimmermann gegründete Firma, die es sich zur Aufgabe gemacht hat, Datenverschlüsselungssysteme zu entwickeln und zu verbreiten. Insbesondere das Programm „PGP“ sorgt immer wieder für Schlagzeilen. PGP benutzt den RSA-Algorithmus, um E-mail und Daten zu schützen. PGP bietet dabei nicht nur die Möglichkeit, Schlüssel zu generieren und damit zu verschlüsseln, sondern auch, die Schlüssel zu verwalten. Es werden sogenannte Schlüsselringe angelegt, in denen die Schlüssel aufgehoben werden. Um eine Verschlüsselte Nachricht zu entschlüsseln, wird aus dem Schlüsselbund automatisch der passende Secret-Key ausgewählt. Auch besteht natürlich die Möglichkeit, Nachrichten zu signieren.
Die zur Zeit (6.1.1998) aktuellste Version auf dem deutschen Markt ist PGP 5.0i, wobei das i für international version steht. (dazu mehr in Kapitel 9). Diese Version allerdings verschlüsselt nicht mehr mit dem RSA-Schlüssel, sondern mit dem Diffie-Hellmann-Schlüssel, ebenfalls eine Methode der Public-Key-Verschlüsselung. Weil ich denke, daß das Prinzip der Verschlüsselung mit öffentlichem Schlüssel in der Praxis angewandt sehr viel eindrucksvoller ist, als in bloßer Theorie, habe ich dieser Arbeit auf CD die aktuelle PGP-Version beigelegt. PGP ist Freeware, und das Kopieren und Verbreiten dieser Software ist ausdrücklich erlaubt!

7.2 Andere Anwendungen

Wie in Kapitel 2.3 schon angedeutet ist RSA Teil einiger internationaler Standards. An Standards wie der „Society for Worldwide Interbank Financiel Telecommunications standard“, „French financial industry’s ETEBAC 5 standart“ oder dem „ANSI X9.31 draft standard for the U.S. banking industry“, von denen RSA ein Teil ist, ist zu erkennen, daß RSA vor allem dann eingesetzt wird, wenn es darum geht, Geldgeschäfte über Datennetze abzuwickeln. Aber auch viele Internet-Standards, wie z.B. dem PEM (Privacy Enhanced Mail) Standard wären ohne RSA nicht denkbar (verwiesen sei auch auf S/MINE, PEM-MINE, S-HTTP und SSL).
Viele andere Standards werden zur Zeit entwickelt, und einige von denen sollen auch das RSA-Verfahren beinhalten. Inwieweit dieser in den nächsten Jahren von anderen Verfahren (z.B. Diffie-Hellmann) abgelöst werden, bleibt abzuwarten.

8 AUSWIRKUNGEN

Das Schaffen eines Codierungssystem, daß so sicher ist, wie das RSA-System, hat nicht nur Auswirkungen auf den Datenverkehr und die Möglichkeiten im Internet, sondern auch auf die Politik und Wirtschaft eines Landes. So nutzen selbst höchste Regierungsstellen die Möglichkeit, Daten sicher zu Verschlüsseln, sehen es jedoch nicht so gerne, wenn jeder die Möglichkeit hat, Daten so zu verschlüsseln, daß nicht einmal die Polizei, Regierung oder diverse Nachrichtendienste die Botschaft entschlüsseln können.
In Amerika begann daher ein energischer Kampf darum, Verschlüsselungssoftware so weit wie möglich einzuschränken, bzw. entsprechende Software mit Hintertür, also der Möglichkeit auch ohne geheimen Schlüssel an die verschlüsselten Daten zu kommen, zu verbreiten.
Dieser Kampf begann auch in der Öffentlichkeit diskutiert zu werden, als gegen PGP-Gründer Phil Zimmermann Klage erhoben wurde. Die US-Regierung klagte Zimmermann an, illegal eine Verschlüsselungstechnologie exportiert zu haben. Ihm drohte somit eine Geldstrafe von bis zu einer Millionen Dollar oder mindestens 41 Monate Gefängnis. Es bildete sich eine Lobby heraus, die das Recht auf Verschlüsselung für jedermann forderte. Daraus folgte die Freigabe PGPs als Freeware. Seitdem ist das kompilieren und verbreiten von PGP für private Zwecke ausdrücklich erlaubt. Die Klage wurde schließlich fallen gelassen. Trotzdem ist und bleibt der Export von Verschlüsselungstechnologien verboten, mit einer Ausnahme:
Auf Grund dieses Verbotes darf PGP nicht exportiert werden. Jedoch kann, vollkommen legal, in langen Listen und Büchern der Quelltext ins Ausland gebracht werden. So wird (meistens in Norwegen) von freiwilligen Helfern der gesamte Quellcode neu eingescannt und erneut kompiliert. Diese neu kompilierte Version erhält den Zusatz i für „international“ zu dem Versionskürzel. Daher ist PGP auch außerhalb den USA als Freeware erhältlich.
Dieses Verfahren ist jedoch sehr zeitaufwendig. So kam erst ein halbes Jahr nachdem in der USA PGP 5.0 erschien die PGP 5.0i-Version auf den Markt. Außerdem darf durch die Bestimmungen niemand eigenständig „Bugfixes“, also Fehlerbehebungen bzw. Verbesserungen, vornehmen. Dadurch ist die i-Version nicht auf dem aktuellen Stand zu halten. Gleichzeitig mit der PGP 5.0i-Version erschien in den USA die Version 5.53, an der erhebliche Verbesserungen gegenüber den Vorversionen vorgenommen wurden.
Das Interesse an „starker“ Verschlüsselungstechnologie jedoch mit Hintertür von Seiten des Staates ist auch an folgendem zu erkennen.
Firmen wie z.B. McAfee erhalten für den Beitritt in die „Key Recovery Alliance“ Zuschüsse und Wohlwollen des Staaten. Die Key Recovery Alliance ist ein Firmenbund, der sich der Entwicklung „starker“ Verschlüsselungstechnologien verschrieben hat. Diese jedoch sollen eingebaute Hintertüren haben, so daß sie dem Wunsch der Regierung auf Abhörmöglichkeit entspricht.
So kam es, daß McAfee sich mit Netzwerksoftwarenhaus Network General zusammenschloß und so die Network Associates bildete. Network Associates kaufte um den 22.12.1997 PGP Inc. auf. Wenige Tage nach der Übernahme von PGP zog sich allerdings Network General aus der Key Recovery Alliance zurück. Was bleibt ist die Skepsis bei den Anwendern.
In einigen anderen Ländern, vornehmlich Diktaturen, ist die Verschlüsselung mit dem RSA-Algorithmus gänzlich verboten. Doch auch in Deutschland ist die Diskussion um die Public-Key-Verschlüsselung entfacht. Innenminister Kanther forderte eine strenge Regulierung der angewandten Kryptologie im Internet. Nach dem Vorbild Amerikanischer „Trust Center“ sollte jeder bei unabhängigen Instanzen seinen Secret-Key hinterlegen. Die Begründung lautete, daß „Staatsfeinde und Kriminelle“ ihre Verbrechen mittels dieser Technik verschleierten.
Eine Begründung, die ich für nicht schlüssig halte, denn kein Verbrecher würde davor zurückschrecken, einen Secret-Key zu benutzen, ohne ihn hinterlegt zu haben. Bei dem derzeitigen Zustand des Internets ist eine sinnvolle Kontrolle meiner Meinung nach nicht einmal im Ansatz möglich.
Jedenfalls sprach sich das Justiz- und das Wirtschaftsministerium gegen eine solche Verfahrensweise aus. Auch Wirtschaftsverbände konnten eine Hinterlegung der Schlüssel auf keinen Fall erdulden. Sie befürchteten ein Schlupfloch für Wirtschaftsspionage. So ist die

Diskussion um die Hinterlegung des Secret-Keys wieder Abgeklungen, ohne daß eine solche Kontrollinstanz eingeführt wurde.
Auch international ist die Nutzung von PGP noch weitestgehendes möglich. So kommt es zustande, daß der Untergrund in Birma, die Exilregierung von Tibet, die Zapatista-Guerillas in Mexiko, aber auch amnesty international PGP ohne Einschränkungen nutzen können.

Dieses Kapitel habe ich meiner Arbeit vor allem beigefügt, um die aktuelle Brisanz des Themas zu umreißen. Aufgrund der aktuellen Information (6. Jan. 98) dieses Abschnittes ist es jedoch möglich, daß einige Informationen mittlerweile schon wieder veraltet sind.

9 STICHWORTVERZEICHNIS

R

S

T

U

V

W

Z

10 QUELLEN

10.1 Quellen aus dem Internet

Die nun folgenden Quellen aus dem Internet waren für die Erstellung dieser Arbeit sehr wichtig und informativ. Jedoch kann ich keine Garantie dafür übernehmen, daß die hier angegebenen Internetseiten noch abrufbar sind oder in der Zwischenzeit geändert wurden! Die entnommenen Informationen beziehen sich (soweit nicht anders vermerkt) auf den Stand vom 24.2.1997.

1. ftp://ftp.cert.dfn.de/pub/doc/german
2. ftp://ftp.cert.dfn.de/pub/tools/crypt/pgp
3. ftp://ftp.cert.dfn.de/pub/tools/crypt/pgp
4. ftp://ftp.cert.dfn.de/pub/tools/crypt/pgp/langauge
5. ftp://ftp.de.pgp.net/pub/pgp
6. ftp://ftp.uni-paderborn.de/FAQ/alt.security.pgp
7. http://pgpEurope.com
8. http://tucson.com/2001/pgpjumps.html
9. http://www.faui81.informatik.uni-erlangen.de/SchemeTeach/pages/ch2/sec7_18.html
10. http://www.faui81.informatik.uni-erlangen.de/SchemeTeach/pages/ch2/sec7_20.html
11. http://www.faui81.informatik.uni-erlangen.de/SchemeTeach/pages/ch2/sec7_28.html
12. http://www.kit-bremen.com/kodierung/interview.html (vom 3.2.1997)
13. http://www.kit-bremen.com/kodierung/versionen.html (vom 3.2.1997)
14. http://www.utm.edu/research/primes/prove2.html
15. http://www.utm.edu/research/primes/prove3.html
16. http://www.utm.edu/research/primes/prove4.html
17. http://www.utm.edu/research/primes/proving.html
18. http://www.voicenet.com/~jank/astec/entschließung-Krypt-datenschutz.html
19. http://www.voicenet.com/~jank/astec/kryptver.html
20. http://www.voicenet.com/~jank/astec/offenkry.html
21. http://www.voicenet.com/~jank/astec/pgpman.html

10.2 Sonstige Quellen

22. Aigner, Alexander. Zahlentheorie. Berlin 1974, Göschen'sche Verlagsbuchhandlung

23. Andreas Bartholomé, Josef Rung, Hans Kern. Zahlentheorie für Einsteiger. Braunschweig 1995, Vieweg & Sohn Verlagsgesellschaft mbH

24. Bauer. Kryptologie. Berlin 1993, Springer-Verlag

25. Beth, Heß, Wirl. Kryptographie. Stuttgart 1983, B.G. Teubner Verlag

26. Franke, Herbert W. Die geheime Nachricht: Methoden und Technik der Kryptologie. Frankfurt am Main 1982, Umschau-Verlag

27. Ihringer, Thomas. Diskrete Mathematik. Stuttgart 1994, B.G. Teubner-Verlag

28. Jungnickel, Dieter. Codierungstheorie. Heidelberg 1995, Spektrum Akademischer Verlag GmbH

29. Leutbecher, Armin. Zahlentheorie, eine Einführung in die Algebra. Berlin 1996, Springer-Verlag

30. Padberg, Friedhelm. Elementare Zahlentheorie. Freiburg im Breisgau 1972, Verlag Herder KG

31. Tietzke, Heinrich. Mathematische Probleme. München 1959, C.H. Beck'sche Verlagsbuchhandlung

32. Well, André. Zahlentheorie, ein Gang durch die Geschichte von Hammurapi bis Legendre. Berlin 1992, Birkhäuser Verlag

33. Franz Grieser, Helmut Weiss. „Der Schlüssel zur Sicherheit" PC Professionell. Ausgabe 6/97. S.207ff

34. Lösch, Jürgen. „Krypto-Gesetz wird Realität" Chip, das Computer Magazin. Ausgabe 5/97. S. 9

35. Luckhardt, Norbert. „Verwirrung um PGP" c't Magazin für Computer Technik. Ausgabe 16 (Heft 22.12.97 – 4.1.98). S.20

36. Manfred Meyer, Rüdiger Weis, Thomas Albinus, Michaela Haass, Jürgen Pötzsch. „Special: Kryptologie – Schutz vor Hackern" PC Magazin DOS. Ausgabe 4/97. S.227 ff

37. Bäumler, Dr. Helmut. Bericht des Landesbeauftragten für den Datenschutz bei dem Präsidenten des schleswig-holsteinischen Landtages. Neunzehnter Tätigkeitsbericht