

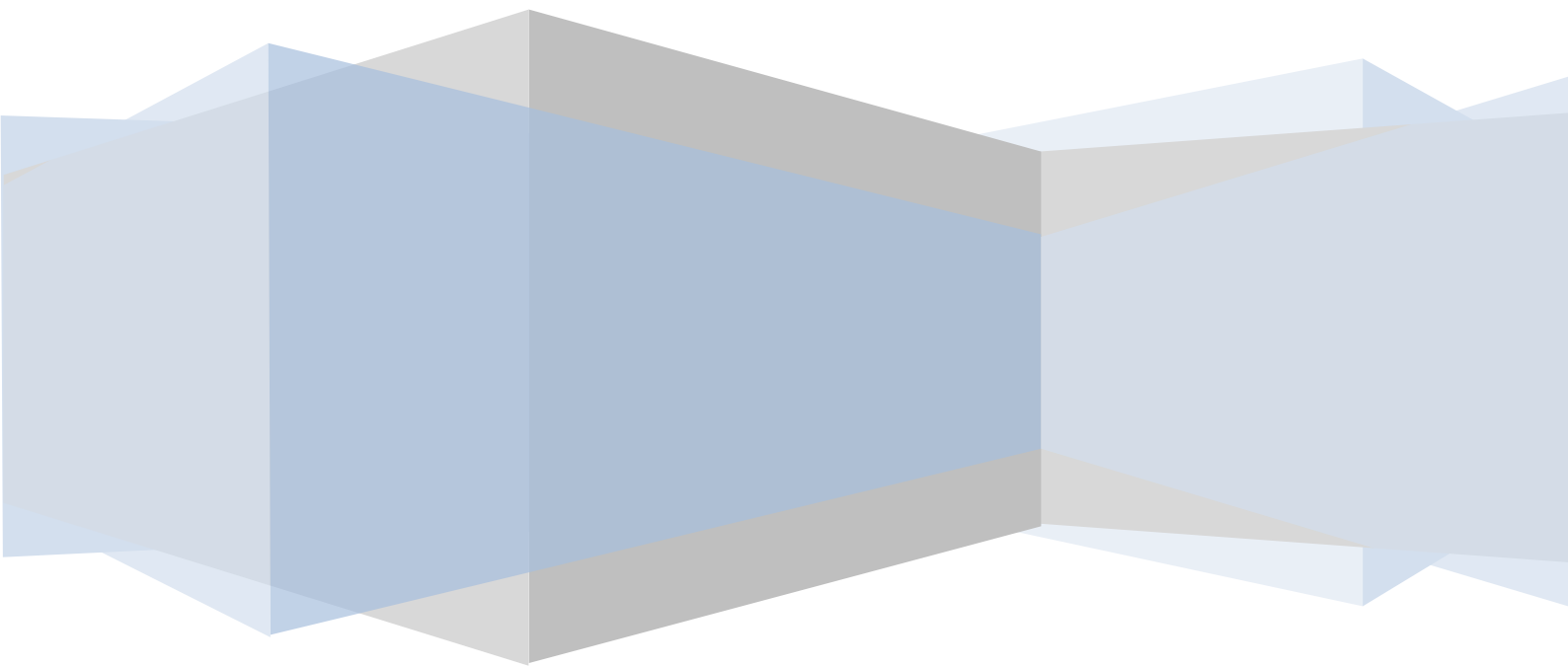
Seminararbeit zum Thema Kryptologie

# PGP

## Vorläufer und mögliche Nachfolger des RSA-Algorithmus

Von Felix Bruckner und Fabian Horsch

Mit einer objektorientierten, an PGP angelehnten RSA-  
Implementierung in Java



## INHALTSVERZEICHNIS

<b>Einleitung.....</b>	<b>5</b>
Problemstellung.....	5
Zielsetzung der Arbeit .....	6
Aufbau der Arbeit .....	6
<b>Die Vorläufer von PGP und RSA .....</b>	<b>9</b>
Historischer Überblick .....	9
Vorläufer .....	10
DES.....	10
Diffie-Hellman- Schlüsselaustausch .....	12
<b>Hauptthema: RSA und PGP.....</b>	<b>15</b>
Einleitung.....	15
Public-Key-Kryptographie .....	16
Historisches zur Public-Key-KRyptographie .....	16
Das Verfahren.....	18
Vor- und Nachteile .....	19
RSA .....	21
Geschichtliches .....	21
Die Mathematik hinter RSA .....	23
PGP .....	27
Was ist PGP?.....	27
Geschichtliches .....	27
Die Idee hinter PGP .....	28
Die Funktionsweise von PGP .....	28

Die Vor- Und Nachteile Von PGP .....	33
<b>Die Nachfolger von PGP und RSA .....</b>	<b>35</b>
El-Gamal .....	35
Zero-Knowledge.....	36
Das IDEA-Verfahren .....	38
Der Rijndael Algorithmus .....	39
<b>Programm: Public-Key-Kryptographie mithilfe von RSA in Java ....</b>	<b>41</b>
Übersicht .....	41
Programmbeschreibung & Schwerpunkt .....	41
PMP – Pretty Much Privacy .....	41
Notation.....	42
Hinweise zum Prototypen .....	43
Bildschirmfotos des Prototypen .....	44
Hinweise zur Finalen Version .....	45
Programmablauf .....	46
Klassendiagramme.....	50
Gesamtübersicht mit Assoziationen .....	51
PMPMain .....	52
RSA.....	52
Converter .....	52
Prime.....	53
GUI.....	53
Hilfsklassen (Splash, ImagePanel).....	53
Sequenzdiagramme .....	54
Verschlüsseln Eines Textes.....	54

Verschlüsseln einer Datei .....	54
<b>Quelltext des Programms.....</b>	<b>55</b>
PMPMain.java .....	55
RSA.java .....	60
Converter.java .....	62
Prime.java .....	64
GUI.java .....	65
ImagePanel.java.....	73
Splash.java.....	73
<b>Glossar .....</b>	<b>75</b>
<b>Abbildungsverzeichnis .....</b>	<b>82</b>
<b>Quellenangaben .....</b>	<b>83</b>
Literaturverzeichnis .....	83
Websites.....	83
<b>Ehrenwörtliche Erklärung.....</b>	<b>86</b>

## EINLEITUNG

### PROBLEMSTELLUNG

Mit dem Begriff „Kryptologie“ verbinden viele Menschen gewisse Vorstellungen. Wie z.B. höchst geheime Briefe, die für den Unwissenden aus unleserlichen Zeichen bestehen, mystische drehbare Kreise, die in bestimmten Kombinationen einen Text verschlüsseln bzw. entschlüsseln, oder sie denken vielleicht an eine versteckte Nachricht zwischen den Zeilen eines Textes.

Das ist durchaus nicht falsch. Es existieren viele unterschiedliche Möglichkeiten zur Verschlüsselung unserer Nachrichten und Daten. Gab es Anfangs simpelste Möglichkeiten (wie die oben genannten) von →Substitution über →Transposition bis hin zu →Steganographie. Später kamen komplexere Dinge wie Codebücher und die Enigma hinzu.<sup>1</sup> All diese Techniken haben eine Gemeinsamkeit, sie sind unsicher und wurden bereits vor langer Zeit geknackt. Sie sind heutzutage irrelevant und unbrauchbar für Verschlüsselungen, die mehr als geheime Briefchen im Unterricht oder Ähnliches nicht übersteigen.

Man mag argumentieren: Wozu auch Dinge verschlüsseln, heutzutage benutzen sowieso nur Geheimdienste und das Militär die Kryptographie. In unserer heutigen Gesellschaft spielt Kryptologie eine große Rolle, wir merken es nur nicht. Sei es beim Geld abheben am Automaten, dem Telefonieren mit dem Handy, beim Online-Shopping, im Netzwerk mit Wireless LAN, beim Pay-TV schauen oder der guten alten, fast vergessenen Telefonkarte. Bei all diesen alltäglichen Dingen ist Kryptographie im Spiel.<sup>2</sup>

Die Techniken, die für solche Verschlüsselungen eingesetzt werden, heißen →DES, →3DES oder →Public-Key-Kryptographie inklusive →RSA, →IDEA oder →Zero-Knowledge<sup>3</sup>.

Mit den Anfangs genannten Methoden haben diese Techniken nichts mehr gemeinsam: Die „moderne Kryptologie“ basiert im Gegensatz zur „klassischen Kryptologie“ auf Mathematik, Algorithmen und komplizierter Zahlentheorie.<sup>4</sup>

---

<sup>1</sup> Vgl. hierzu <http://www.it.fht-esslingen.de/~schmidt/vorlesungen/kryptologie/seminar/historie/History.html>

<sup>2</sup> Vgl. [BEU2], S. 114ff

<sup>3</sup> Vgl. [http://www.wiwi.uni-bielefeld.de/StatCompSci/lehre/material\\_spezifisch/statalg00/rsa/node1.html](http://www.wiwi.uni-bielefeld.de/StatCompSci/lehre/material_spezifisch/statalg00/rsa/node1.html)

<sup>4</sup> Definition nach [BEU2], S. 7ff

Die Kryptologen streben danach, den „unknackbaren“ Algorithmus zu finden – bis jetzt jedoch nur mit Teilerfolgen. Mit den beiden wichtigsten Teilerfolgen in der Geschichte der Kryptologie wollen wir uns in dieser Arbeit beschäftigen: RSA und PGP.

Bei RSA handelt es sich um einen 1977 entdeckten und bis heute - unter bestimmten Voraussetzungen - nicht knackbaren Algorithmus.

PGP – „Pretty good Privacy“ – ist ein Computer-Programm zum Verschlüsseln von E-Mails und anderen Daten, welches von Phillip Zimmermann im Alleingang entwickelt wurde und zu dem Krypto-Programm schlechthin geworden ist. Es machte sichere Verschlüsselung persönlicher Daten und E-Mails massentauglich, zum Verdross vieler Geheimdienste.

Im digitalen Zeitalter gewinnt die Kryptologie immer mehr an Wichtigkeit. Persönliche E-Mails können theoretisch von Jedem gelesen werden, der etwas davon versteht an solche Datenpakete aus dem Internet heranzukommen. Geheimdienste praktizieren dies in ungeahntem Ausmaß unter dem Banner der Terrorismusabwehr.

Der Wunsch der Menschen nach mehr Privatsphäre und Datenschutz wächst dadurch immer stärker proportional dazu an. Die Besorgnis der Regierungen und Geheimdienste der Welt ist: kriminelle Energien können aus Entwicklungen wie PGP ebenfalls einen Nutzen ziehen und ohne Sorgen darüber, dass ihre Nachrichten abgefangen werden könnten, Pläne schmieden.

Die Geschichte der „modernen Kryptologie“ rund um RSA inklusive Vorläufern und Nachfolgern sowie die komplizierte politische Situation derselben und der Rolle von PGP darin bildet die Grundlage unserer Seminararbeit.

## ZIELSETZUNG DER ARBEIT

Wir haben es uns zum Ziel gemacht, den Verschlüsselungsalgorithmus RSA (benannt nach dessen Erfindern) und das Programm namens PGP anschaulich darzustellen und zu erklären.

Das wollen wir mit Hilfe von Erläuterungen, Darstellungen und einem selbst programmierten PGP Programm, das den RSA - Algorithmus beinhaltet, erreichen.

Zusätzlich wollen wir die zeitlichen Vorläufer von PGP und RSA vorstellen.

Außerdem haben wir es uns zum Ziel gemacht, die Nachfolger von RSA und PGP aufzuzeigen und einen Ausblick über die Zukunft der Verschlüsselung von Nachrichten zu geben.

## AUFBAU DER ARBEIT

Die vorliegende Seminarkursarbeit ist in zehn Kapitel gegliedert, die jeweils in Unterkapitel aufgeteilt sind.

Das erste Kapitel beschäftigt sich im Allgemeinen mit der Heranführung zum Thema und der Zielsetzung unseres Seminarkurses.

Im zweiten Kapitel dieser Seminararbeit werden die Vorläufer von RSA und PGP aufgezeigt. In diesem Kapitel ist unter anderem ein geschichtlicher Überblick über die Vorläufer und Nachfolger von RSA und PGP zu sehen.

Das Hauptthema des Seminarkurses - RSA und PGP - wird im dritten Kapitel behandelt und beginnt mit einer Einführung in die Public- Key- Kryptographie.

Deshalb wird als erstes die Verschlüsselungsmethode namens RSA näher beschrieben, danach das Programm das mithilfe des RSA Algorithmus arbeitet, PGP.

Im Teil, RSA, werden zunächst geschichtliche Details und anschließend die mathematischen Hintergründe des RSA Algorithmus dargestellt.

Im Teil, PGP, wird wiederum auf die Geschichte, die Funktionsweise, anschließend auf die Idee, die hinter diesem Programm steckt, und die Problematik, die durch dieses Programm entstand und immer noch besteht, eingegangen.

Die möglichen Nachfolger von RSA und PGP werden im vierten Kapitel dieser Arbeit näher beschrieben und erklärt.

Die Beispielanwendung zu PGP und RSA in JAVA wird im fünften Kapitel näher beleuchtet. Das geschieht anhand von Screenshots und der Erklärung der einzelnen Klassen, die in dem Programm verwendet wurden.

Zudem wird durch zwei Sequenzdiagramme erklärt wie das Programm Nachrichten und Dateien verschlüsselt.

Der Quelltext der PGP- Anwendung ist im sechsten Kapitel abgedruckt.

Das siebte, achte, neunte und zehnte Kapitel setzen sich aus dem Glossar, dem Abbildungsverzeichnis, der Quellenangabe und der Ehrenwörtlichen Erklärung zusammen.

Im Glossar werden Fremdwörter, die in der Seminarkursarbeit verwendet wurden, aufgelistet und erklärt.

Das Abbildungsverzeichnis besteht aus der Auflistung der Abbildungen, in dieser Arbeit.

In der Quellenangabe werden die Bücher und die Webseiten, die für die Erstellung des Kurses nötig waren, gezeigt.

**KAPITEL 1: EINLEITUNG****KAPITEL 2: DIE VORLÄUFER VON RSA UND PGP**

Historischer Überblick

Diffie-Hellman-Schlüsselaustausch

DES

**KAPITEL 3: HAUPTTHEMA: RSA UND PGP**

**Public Key Kryptographie:** Das GCHQ, Ellis', Cocks' und Williamson's Ideen

**RSA:** Geschichtliches & Mathematischer Hintergrund

**PGP:** Geschichtliches, Idee hinter PGP , Funktionsweise, PGP Problematik

**KAPITEL 4: DIE NACHFOLGER VON RSA UND PGP**

Zero-Knowledge

RC4

Idea-Verfahren

Elgamal

Rijndael

**KAPITEL 5: PROGRAMM: PUBLIC-KEY-KRYPTOGRAPHIE MIT RSA IN JAVA**

Übersicht & Schwerpunkt

Screenshots

Ablaufübersichten

Klassendiagramme

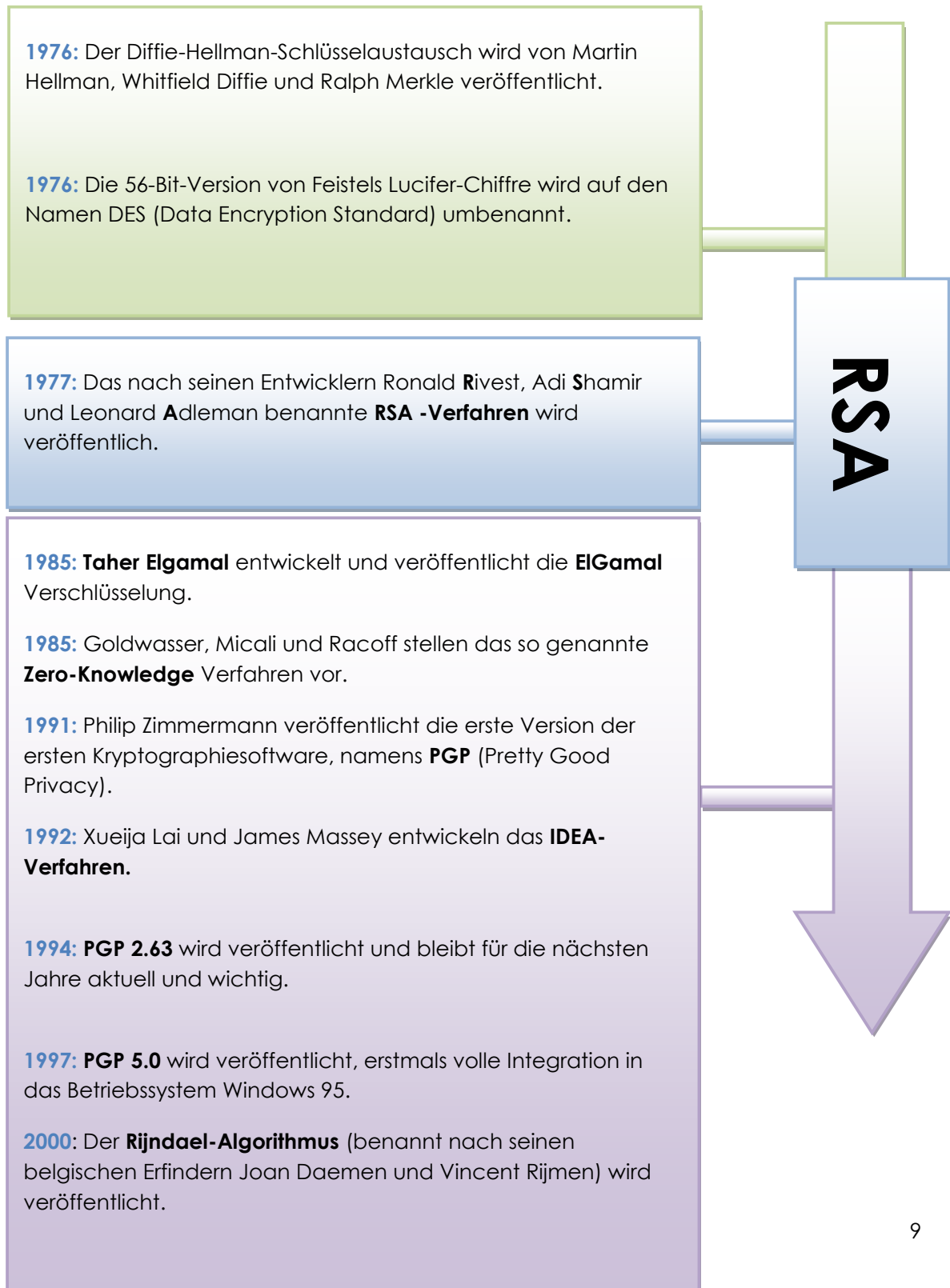
Sequenzdiagramme u.a. zum Verschlüsseln und Entschlüsseln

**KAPITEL 6: JAVA QUELLTEXT DER ANWENDUNG****KAPITEL 7,8,9,10: GLOSSAR,ABBILDUNGSVERZEICHNIS,QUELLENANGABE UND EHRENWÖRTLICHE ERKLÄRUNG**

## DIE VORLÄUFER VON PGP UND RSA

### HISTORISCHER ÜBERBLICK

Zur besseren Übersicht über die in den nächsten Kapiteln behandelten Themen haben wir eine Zeitleiste der wichtigsten historischen Ereignisse angefertigt



## VORLÄUFER

In diesem Kapitel werden die chronologisch geordneten Vorläufer von RSA und PGP beschrieben und vorgestellt. Beide Verfahren die hier vorgestellt werden hatten nicht direkt mit der Entwicklung von RSA oder PGP zu tun. Jedoch wurde das Verfahren namens DES, unter anderem auch in PGP als →symmetrisches Verschlüsselungsverfahren verwendet.

### DES

Unter dem Begriff DES (Data Encryption Standard) versteht man die am 23. November 1976 übernommene 56-Bit-Version der Lucifer-Chiffre von Horst Feistel.<sup>5</sup>

In den 70ern galt die Lucifer-Chiffre als die beste Verschlüsselungsmethode um Nachrichten jeglicher Art zu verschlüsseln.

Da zu dieser Zeit die Computer immer schneller und vor allem billiger wurden, besaßen zunehmend mehr Unternehmen und Institutionen eigene Computer mit denen sie mithilfe der Lucifer-Chiffre schneller und sicherer, Nachrichten verschlüsseln konnten. Die Lucifer-Chiffre war auf dem besten Wege als amerikanischer Standard übernommen zu werden. Jedoch mischte sich die →NSA in Feistels Entwicklungsarbeiten ein. Die NSA hatte Angst, dass der Lucifer Algorithmus ein zu starker Verschlüsselungsstandard werden würde, den sie nicht mehr in der Lage wären zu knacken. Deshalb übte die NSA auf Feistel Druck aus, mit dem Ziel, die Zahl möglicher Schlüssel auf 100 000 000 000 000 000 zu begrenzen. In der Computersprache wird diese Zahl mit →56 Bits dargestellt<sup>6</sup>. Aus diesem Grund erhielt die Version dieser Lucifer-Chiffre den Namen 56-Bit-Version.

### HORST FEISTEL



<sup>7</sup> Horst Feistel wurde am 30. Januar 1915 in Berlin geboren. Im Jahre 1934 emigrierte er nach Amerika, wo er im Zweiten Weltkrieg unter Hausarrest gestellt wurde. Anfang des Jahres 1944 bekam er die amerikanische Staatsbürgerschaft verliehen. In den darauffolgenden Jahren arbeitete er am Lincoln Laboratory, für die Firma MITRE, bei der er oft mit der →NSA im Konflikt stand und seine Entwicklungen einstellen musste. Letztendlich kam er zu IBM. Dort konnte er sich zum ersten Mal ungestört der Kryptologie widmen und entwickelte das Lucifer-Verfahren. Dieses Verfahren

<sup>5</sup> Vgl. [SINGH], S. 304

<sup>6</sup> Vgl. [SINGH], S. 303

<sup>7</sup> Bildquelle: <http://domino.research.ibm.com/comm/pr.nsf/pages/bio.feistel.html>:

wurde jedoch, wie bereits erwähnt, in ihrer Verschlüsselungsstärke, durch die NSA eingeschränkt.<sup>8</sup>

## DIE VERSCHLÜSSELUNG VON NACHRICHTEN MIT DES

Zunächst wird jedem Buchstaben des Alphabets eine Binärzahl mit sieben Stellen zugewiesen (z.B. c = 1 0 1 0 1 0 0). Logischerweise bekommen auch Zahlen und Zeichen jeweils eine eigene Zahlenkombination. Dann verwandelt DES, den zu verschlüsselnden Text, in eine Reihe von Binärzahlen. Dieser „Binärzahlentext“ wird anschließend in Blöcke von 64 Zahlen aufgeteilt.

Jeder Block wird einzeln für sich verschlüsselt. Nachdem der Text in Binärzahlenblocks von jeweils 64 aufgeteilt wurde, wird jeder Block wiederum unterteilt. Somit entstehen zwei 32 Zahlen lange Blöcke Links<sup>0</sup> und Rechts<sup>0</sup>. Nun knöpft man sich die Zahlen von Rechts<sup>0</sup> vor und →substituiert sie, mit einem vorab abgemachten Schlüssel. Das so entstandene Rechts<sup>0</sup> wird nun zu Links<sup>0</sup> addiert und bekommt den neuen Namen Rechts<sup>1</sup>.

Das anfängliche Rechts<sup>0</sup> wird in Links<sup>1</sup> umbenannt. Jetzt besitzt man Rechts<sup>1</sup> und Links<sup>1</sup>. Diese Abfolge von Vorgängen wird als Runde bezeichnet. Ein kompletter Verschlüsselungsvorgang besteht aus 16 Runden. Danach erhält man den Geheimtext, sendet ihn an den Empfänger, der den Verschlüsselungsvorgang, mithilfe des vereinbarten Schlüssels, rückgängig macht und den entzifferten Text vor sich hat.

Die Einzelheiten der Verschlüsselung können durch den vorab abgemachten Schlüssel verändert und umgestellt werden.

---

<sup>8</sup> Vgl. [SINGH], S. 301 f. und [http://en.wikipedia.org/wiki/Horst\\_Feistel](http://en.wikipedia.org/wiki/Horst_Feistel)

## DIFFIE-HELLMAN- SCHLÜSSELAUSTAUSCH

Das nun folgende Verfahren beschäftigt sich als erstes „Verschlüsselungsverfahren“ nicht mit der Verschlüsselung von Nachrichten, sondern mit der Erstellung von geheimen Schlüsseln über → öffentliche Kanäle. Das bedeutet, dass Gesprächspartner, die geheim kommunizieren wollen über eine nicht abhör gesicherte Leitung (z.B. Telefon) Zahlen austauschen und damit geheime Schlüssel erstellen können, ohne dass Dritte diese kennen.

Mit diesem Verfahren ist es nun zum ersten mal möglich über normale öffentliche Kanäle geheime Schlüssel zu erstellen. Es gibt nun nicht mehr das Problem der Schlüsselverteilung über geheime Kanäle oder das Überbringen von geheimen Schlüsseln durch Boten. Wobei wir gleich beim Schwachpunkt der klassischen Kryptologie sind: es gab nie die Sicherheit, dass nicht noch eine dritte Person die „geheimen“ Schlüssel besitzt.

Dadurch, dass dieser Nachteil in der modernen Kryptologie ausgeschaltet ist, wurde die Verschlüsselung von Nachrichten um ein vielfaches sicherer.

### ALLGEMEINES

Der Diffie-Hellman Schlüsselaustausch ist ein → Protokoll, das von Whitfield Diffie und Martin Hellman entwickelt und 1976 veröffentlicht wurde.<sup>9</sup>

Innerhalb des Diffie-Hellman-Schlüsselaustausch tauschen zwei Gesprächspartner Informationen über einen öffentlichen Kanal, z.B. Telefon, aus mit welchen man einen geheimen Schlüssel erzeugen kann. Dieser Schlüssel wird als unknackbar bezeichnet, weil man die Schlüsselerzeugung als Zuschauer/Zuhörer nicht mitverfolgen bzw. rückgängig machen kann.

Der Diffie-Hellman-Schlüssel wird mit Hilfe der → Modul-Arithmetik erstellt bzw. mithilfe einer → Einwegfunktion der Form  $Y^x \pmod{P}$ .

### WHITFIELD DIFFIE



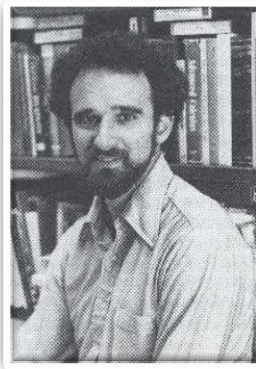
<sup>10</sup> Whitfield Diffie wurde als Sohn eines Spanischlehrers am 5. Juni 1944 in Washington geboren. Er wuchs in New York auf und studierte ab 1969 in Stanford, wo er sich zunächst der künstlichen Intelligenz widmete, später sich jedoch mit der spannenden Kryptologie auseinandersetzte. Mitte der 70er Jahre traf er auf Martin Hellman. Mit ihm entwickelte er das Konzept der Public- Key- Kryptographie bzw. den Diffie- Hellman Schlüsselaustausch. Nach dieser grundlegenden Entwicklung arbeitete er zunächst bei Northern Telecom und wechselte im Jahre 1991 zu Sun Microsystems. Dort ist er heute wie damals Chef

<sup>9</sup> Vgl. [SINGH] S. 316

<sup>10</sup> Bildquelle: <http://research.sun.com/people/diffie/>

Security Officer (Sicherheitsbeauftragter)<sup>11</sup>.

## MARTIN HELLMAN



<sup>12</sup> Martin Hellman wurde am 2. Oktober 1945 in New York geboren. Von klein an interessierte er sich für ausgefallene Dinge, unter anderem für die Kryptologie. Er wurde Professor an der kalifornischen Stanford-Universität. Im September des Jahres 1974 traf er dort Whitfield Diffie, mit dem er, wie bereits erwähnt das Konzept der Public-Key-Kryptographie entwickelte. Da Hellman sich nur mit der Theorie beschäftigte, benötigte er die Hilfe von Diffie um seine theoretischen Überlegungen praktisch umsetzen zu können<sup>13</sup>. Zur Zeit ist Martin Hellman immer noch Professor an der kalifornischen Stanford-Universität.

## SCHLÜSSELERZEUGUNG

Zur besseren Erklärung führen wir 3 Personen ein, → Alice, Bob und Eve.

Zu Beginn der Schlüsselerzeugung einigen Alice und Bob sich auf 2 Zahlen P und Y, wobei P größer als Y und eine Primzahl sein muss. P=11 und Y=7. Diese Zahlenwerte werden nun über einen öffentlichen Kanal ausgetauscht.

Dadurch entsteht die Einwegfunktion  $7^x \pmod{11}$ .

Im Zweiten Schritt wählen Alice und Bob jeweils eine neue Zahl aus z.B.  $A^{\text{Alice}} = 3$  und  $B^{\text{Bob}} = 6$ . Nun berechnet Alice  $7^A \pmod{11}$  und Bob  $7^B \pmod{11}$ .

Alice rechnet:  $7^3 \pmod{11} = 343 \pmod{11} = \text{Ergebnis: } 2$

Bob rechnet:  $7^6 \pmod{11} = 117649 \pmod{11} = \text{Ergebnis: } 4$

Die Ergebnisse werden wiederum ausgetauscht. Nun verwendet Alice das Ergebnis von Bob und berechnet:

$4^3 \pmod{11} = 64 \pmod{11} = 9$

Bob benutzt seinerseits das Ergebnis von Alice und berechnet:

$2^6 \pmod{11} = 64 \pmod{11} = 9$

Wie man sehen kann bekommen Alice und Bob jeweils die gleichen **Zahlen** heraus.

Diese **Zahl** dient als Schlüssel.

<sup>11</sup> Vgl. <http://research.sun.com/people/diffie/> und <http://www.heise.de/newsticker/meldung/47951>

<sup>12</sup> Bildquelle: [http://www.mathe.tu-freiberg.de/~dempe/schuelerpr\\_neu/pics/hellmann.jpg](http://www.mathe.tu-freiberg.de/~dempe/schuelerpr_neu/pics/hellmann.jpg)

<sup>13</sup> Vgl. [http://www.mathe.tu-freiberg.de/~dempe/schuelerpr\\_neu/hellmann.htm](http://www.mathe.tu-freiberg.de/~dempe/schuelerpr_neu/hellmann.htm)

Im Nachhinein könnte man evtl. auf den Gedanken kommen das Eve das Telefonat abgehört und somit auch im Besitz des Schlüssel sein kann. Das lässt sich jedoch schnell widerlegen, da Alice und Bob nur die Zahlen 7,11,2 und 4 weitergeben haben und nicht 3 und 6 mit denen die erste Rechnung durchgeführt wurde<sup>14</sup>.

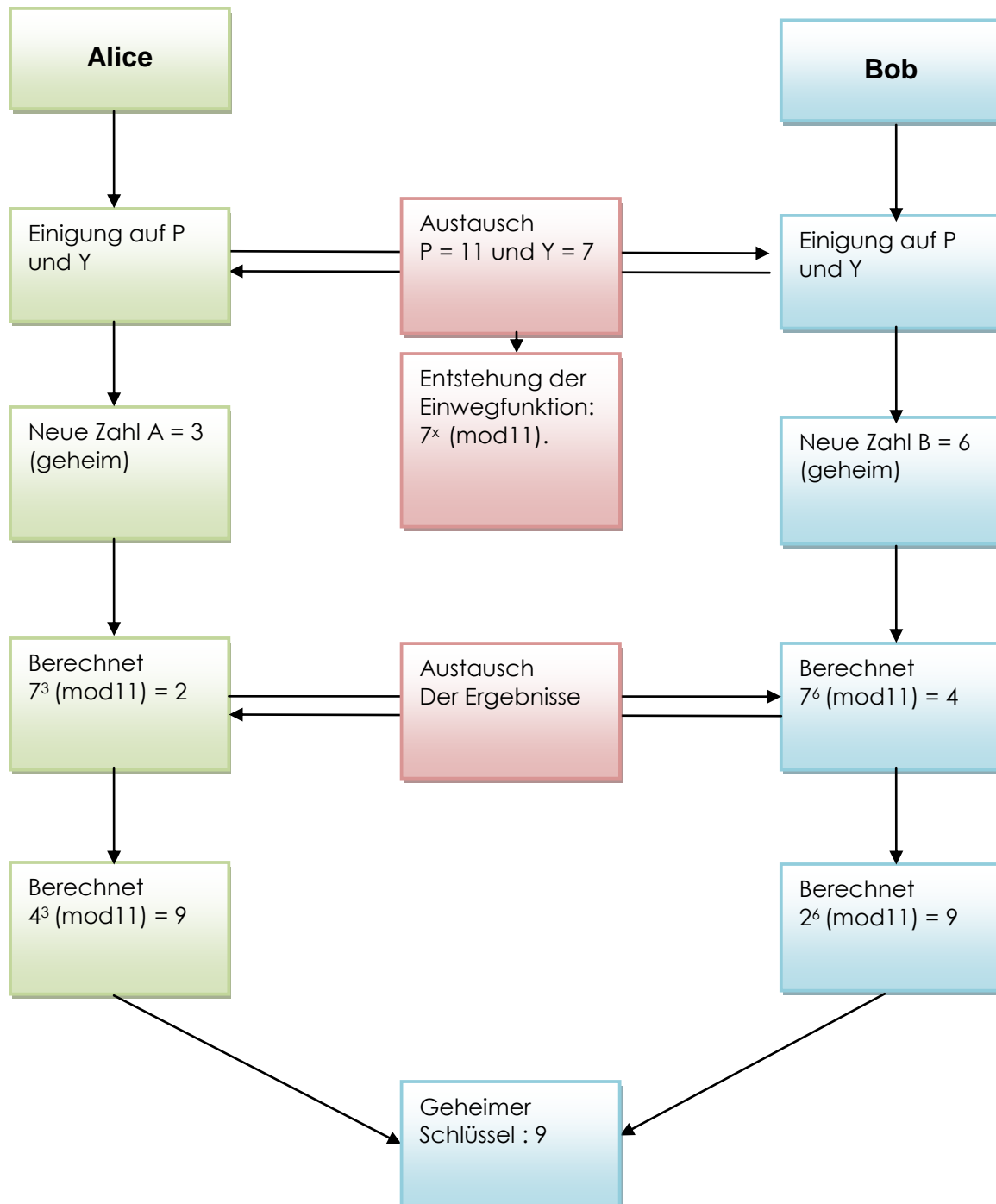


Abb. 1 Beispielhafte Schlüsselerzeugung mit dem Diffie-Hellman-Schlüsselaustausch

<sup>14</sup> Vgl. [SINGH] S. 321

## HAUPTTHEMA: RSA UND PGP

### EINLEITUNG

In diesem Kapitel werden wir uns mit dem Hauptthema unserer Arbeit auseinandersetzen: PGP und RSA.

#### WIESO RSA?

PGP ist im Grunde nur eine Anwendung verschiedener Algorithmen und der →Public-Key-Kryptographie (auch Asymmetrische Verschlüsselung genannt).

Obwohl PGP RSA nur zur Erzeugung von sog. →Fingerprints (Digitale Signaturen) und zur eigentlichen Verschlüsselung IDEA verwendet, beschäftigen wir uns trotzdem mit RSA.

Ohne RSA wären heutige Verschlüsselungssysteme undenkbar, denn RSA war der Wegbereiter, der →Public-Key-Kryptographie überhaupt erst möglich machte. Mit dieser werden wir uns auch gleich zu Anfang beschäftigen, um danach zum Verschlüsselungsverfahren RSA und zu PGP überzugehen.

#### DIE NACHTEILE SYMMETRISCHER VERSCHLÜSSELUNGEN

Ein entscheidender Nachteil symmetrischer Verschlüsselungen sind die im vorigen Kapitel bereits erwähnten „→öffentlichen“ also „→unsicheren“ Kanäle über die der Geheimtext sowohl als auch der Schlüssel übertragen werden. Das macht die Verschlüsselung extrem leicht angreifbar, denn jeder der den Schlüssel kennt, kann den Geheimtext wieder entschlüsseln.

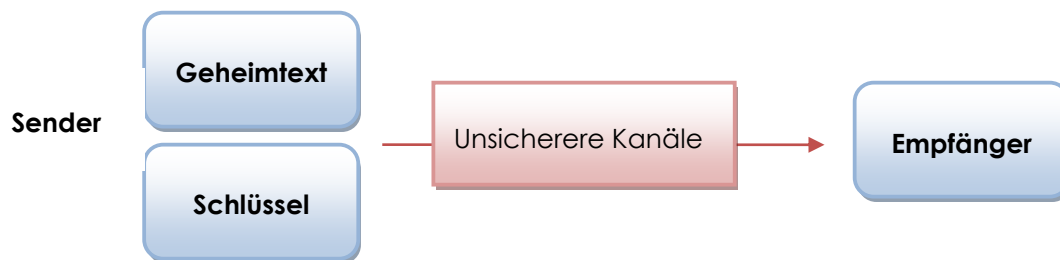


Abb. 2: Typischer Ablauf einer symmetrischen Verschlüsselung

Durch dieses Angriffsproblem müssen beide Teilnehmer den Schlüssel stets geheim halten.

Was bei zwei oder auch 50 Teilnehmern vielleicht noch möglich ist, aber bei sehr vielen Teilnehmern ist das logistisch gesehen kaum realisierbar – denn jedes Empfängerpaar benötigt einen eigenen, individuellen geheimen Schlüssel. Das wären bei 1000 Teilnehmern 499 500 verschiedene Schlüssel. Bei 1.000.000 Teilnehmern 49 999 950 000 unterschiedliche, einzigartige Schlüssel – und alle müssten sicher an die Teilnehmer verteilt werden.

Um dieses Problem zu beseitigen „erfand“ man die Public-Key-Kryptographie, die das Problem elegant umgeht. Wie, wird im folgenden Kapitel erklärt.

## PUBLIC-KEY-KRYPTOGRAPHIE

So abstrakt sich der Begriff „Public-Key-Kryptographie“ anhört, kommen wir damit jeden Tag in Berührung (meist ohne es zu merken). Allem voran im Internet bei fast allen Daten die verschlüsselt übertragen werden. Sei es der Online-Einkauf in einem Onlineshop (per →HTTPS), das sichere einloggen auf einer Website, eben das Verschlüsseln von vertraulichen E-Mails ([Open]PGP, S/MIME) und der Verwendung von →digitalen Signaturen (→SSH).

Um einen Überblick zur Public-Key-Kryptographie und deren Funktionsweise zu erhalten, werden wir zuerst einen kleinen Ausflug in die Geschichte machen.

## HISTORISCHES ZUR PUBLIC-KEY-KRYPTOGRAPHIE

In den 1970er Jahren erlangten Diffie, Hellman und Merkle großen Ruhm durch ihre Ansätze des eingangs beschriebenen Schlüsselaustauschs. Jedoch wurden bereits Jahre zuvor (Ende der 1960er Jahre) ganz ähnliche Entdeckungen von britischen Kryptographen im Staatsdienst gemacht. Jedoch wurden diese Ideen nie patentiert, da die britische Regierung sie geheim hielt und erst fast 30 Jahre später – 1997 – deren Publikation erlaubte.<sup>15</sup>

## DAS GCHQ

GCHQ steht für **G**overnment **C**ommunication **H**eadquarter und ist eine hochgeheime britische Regierungsbehörde, die aus →Bletchley Park nach Ende des Zweiten Weltkrieges hervorging. Sie dient dem Schutz der nationalen Sicherheit.

Eben diese Behörde beauftragte Anfang 1969 ihren fähigsten Kryptographen – James Ellis – damit, eine Lösung zum Schlüsselverteilungsproblem der Regierung ähnlich der eingangs erwähnten Situation zu finden: Experten sagten für die 70er Jahre voraus, dass Funkgeräte immer kleiner und billiger werden würden und somit jedem Soldaten zur Verfügung gestellt werden könnten. Da das Militär auf →symmetrische Verschlüsselung des Funkverkehrs setzte, stand es vor dem Problem der Schlüsselverteilung: Jeder Soldat musste auf sicherem Wege den aktuellen Schlüssel erhalten. Eine damals nicht zu bewältigende Aufgabe.<sup>16</sup>

## JAMES ELLIS



<sup>17</sup> Ellis, ein als Exzentriker beschriebener, brillanter Krypto-Guru im Dienste des GCHQ nahm sich also des Problems an. James H. Ellis, geboren 1924 und eigentlich Australier, studierte in London Physik und trat 1952 dem GCHQ bei. Ellis kam schnell zu dem Schluss, dass geheime Schlüssel ausgetauscht werden müssen.

<sup>15</sup> Vgl.: Vgl.: [SING] S. 338ff

<sup>16</sup> Vgl.: [SING] S. 338ff

<sup>17</sup> Bildquelle: <http://www-ivs.cs.uni-magdeburg.de/bs/lehre/wise0102/progb/vortraege/mschwand/img/image001.jpg>

In einem Bericht der Firma „Bell Telephone“ über abhörsichere Telefone fand er folgende Idee: Der Sender schickt dem Empfänger eine Nachricht die von einem „weißen Rauschen“ – der Schlüssel – überlagert wird. Der Empfänger muss von dem für Drittpersonen sinnfreien Gespräch das Rauschen abziehen und erhält die Nachricht. Damit wurde ein Grundstein für die Public-Key-Kryptographie gelegt: Empfänger und Sender waren gleichsam bei der Verschlüsselung beteiligt. Er nannte diese Idee „non-secret Encryption“, was soviel bedeutet wie „Nichtgeheime Verschlüsselung“.

Ellis stellte seine Idee seinen Vorgesetzten vor und konnte zwar theoretisch die Nützlichkeit eines solchen Verfahrens beweisen, aber es fehlte an einer mathematischen Formel zur praktischen Umsetzung.

Bis Ende 1969 hatte Ellis bereits die Begriffe „öffentlicher Schlüssel“ und „privater Schlüssel“ geprägt, die Suche nach einer Einwegfunktion eines vielköpfigen Teams von Kryptographen des GCHQ war jahrelang nicht erfolgreich.<sup>18</sup>

---

### CLIFFORD COCKS



<sup>19</sup> Clifford Cocks, ein Cambridge-Absolvent und Mathe-Genie (Nahm an der internationalen „Mathematik-Olympiade“ 1968 teil und erreichte den zweiten Platz), kam 1973 zum GCHQ und hörte von Ellis' Idee und der noch nicht gefundenen mathematischen Umsetzung. Das weckte sein Interesse und er machte sich sogleich daran, einen Lösungsweg zur Erstellung einer Einwegfunktion zu skizzieren, die nicht ohne weiteres Umkehrbar war. Als Grundlage wählte er Primzahlen – und formulierte bereits 1973 die Grundzüge des →asymmetrischen RSA-Verfahrens.

Jedoch konnte diese Idee erneut nicht umgesetzt werden, da die Technik Anfang der 70er Jahre noch nicht weit genug entwickelt war.<sup>20</sup>

---

### MALCOLM WILLIAMSON



<sup>21</sup> Malcolm Williamson, ein guter Freund von Clifford Cocks und ebenso Mathematikgenie (Er gewann bei besagter Olympiade Gold) stand dessen Konzept skeptisch gegenüber. Er stellte es auf die Probe und beabsichtigte es zu widerlegen. Er konnte zwar keinerlei Fehler und Ungereimtheiten beweisen, aber entwickelte dabei mehr zufällig ein weiteres Verfahren zum Schlüsselaustausch - das heute als Diffie-Hellman-Schlüsselaustausch bekannt ist - etwa zur gleichen Zeit wie M. Hellman.<sup>22</sup>

---

<sup>18</sup> Vgl.: [SING] S. 342ff

<sup>19</sup> Bildquelle: <http://www-ivs.cs.uni-magdeburg.de/bs/lehre/wise0102/progb/vortraege/mschwand/img/image002.jpg>

<sup>20</sup> Vgl.: [SING] S. 345

<sup>21</sup> Bildquelle: [http://www.livinginternet.com/i/is\\_crypt\\_pkc\\_inv.htm](http://www.livinginternet.com/i/is_crypt_pkc_inv.htm)

Das GCHQ konnte diese Ideen nicht umsetzen, stufte die Dokumente als „Classified“ (geheim halten) ein und hielt diese Errungenschaften jahrzehntelang unter Verschluss. Erst im Jahre 1997 wurden diese Informationen der Öffentlichkeit preisgegeben. All dies erlebte James Ellis nicht mehr, er verstarb im November 1997, nur wenige Wochen vor der Veröffentlichung. Während Cocks noch immer im GCHQ arbeitet, verließ Williamson das GCHQ bereits im Jahr 1982.

Aus diesem Grund galten Diffie und Hellman lange als Erfinder der Public Key-Kryptographie, sowie Rivests, Shamirs und Adlemans RSA-Algorithmus als erste sichere asymmetrische Verschlüsselung.

---

## DAS VERFAHREN

Die Public-Key-Kryptographie befasst sich also mit asymmetrischen Einwegfunktionen die nicht ohne weiteres wieder umkehrbar sind.

Beim Public-Key-Verfahren steht im Gegensatz zu symmetrischen Verfahren der Empfänger und nicht der Sender am Anfang. Es werden zwei Schlüssel benötigt:

Der **Private Key**, welcher geheim (privat) gehalten wird und zum Verschlüsseln/Entschlüsseln von Nachrichten verwendet. Dieser verbleibt beim Empfänger.

Der **Public Key** (öffentlicher Schlüssel) wird nur zur Verschlüsselung verwendet – mit dem Public Key kann man keine Nachricht entschlüsseln.

Der **Public Key** wird dem Sender zugänglich gemacht, z.B. durch Veröffentlichung auf der eigenen Website oder auf sogenannten Key-Servern auf denen viele öffentliche Schlüssel in Listen gespeichert und jedem zugänglich sind.

Der Sender verschlüsselt nun eine Nachricht mithilfe des Public Keys. Sobald die Nachricht verschlüsselt wurde, kann der Sender die Nachricht ohne Kenntnis des privaten Schlüssels nicht mehr entschlüsseln – muss er ja auch nicht.

Die verschlüsselte Nachricht wird an den Empfänger versandt, dabei kann dies problemlos über unsichere Kanäle (siehe Einleitung) geschehen. Wenn jemand den öffentlichen Schlüssel und den Geheimentext abfangen sollte, kann er damit die Nachricht nicht entschlüsseln. Nur der Empfänger kann dies mit seinem privaten Schlüssel tun.

---

<sup>22</sup> Vgl.: <http://www.answers.com/topic/malcolm-j-williamson>

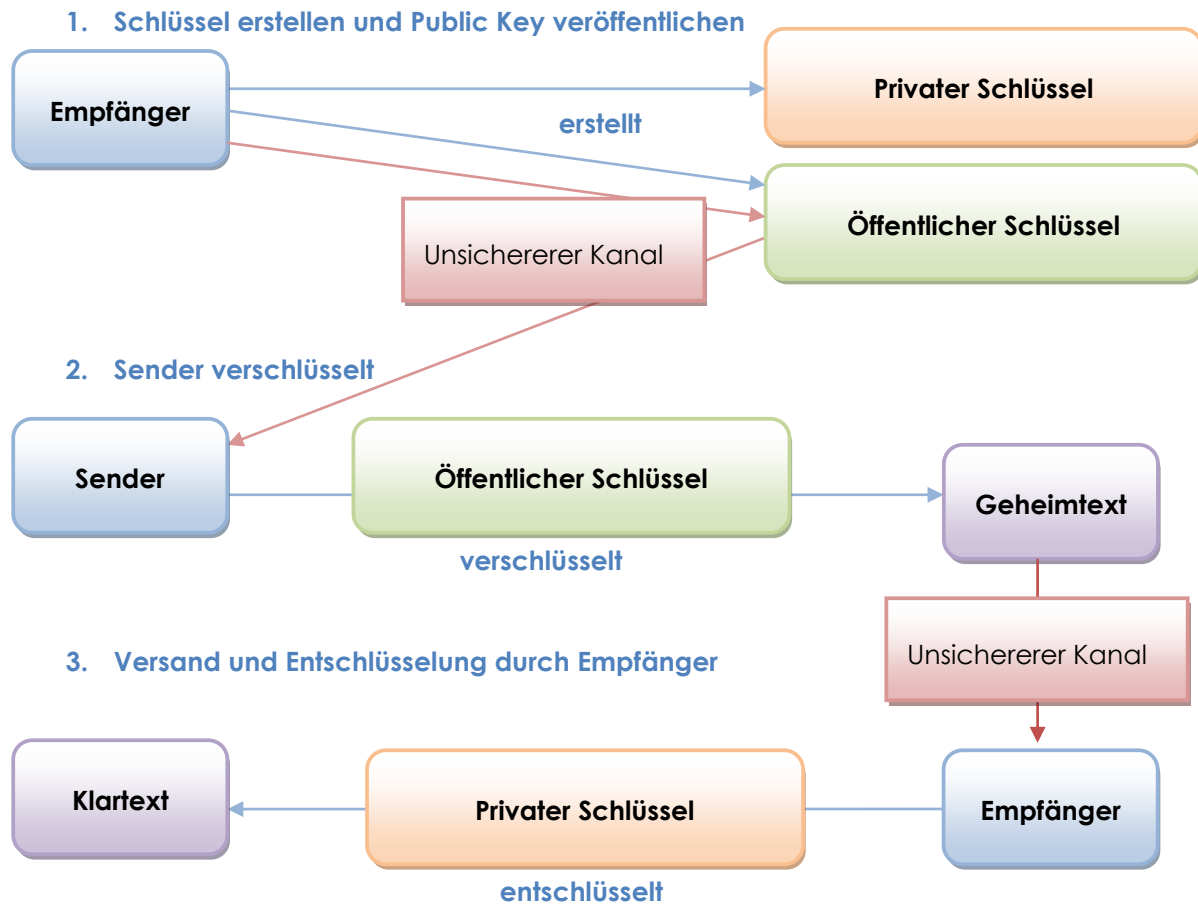


Abb.3: Typischer Ablauf einer asymmetrischen Verschlüsselung

## VOR- UND NACHTEILE

Die wesentlichen Punkte der Public-Key-Kryptographie sollten hiermit klar sein. Bevor wir zu RSA übergehen, werden wir noch einige Vor- und Nachteile der asymmetrischen Verschlüsselung behandeln.

### VORTEILE

- Jeder Teilnehmer besitzt ein individuelles Geheimnis
- Geheime Übertragung von Schlüsseln wird hinfällig
- Große Teilnehmerzahl möglich, da die Anzahl der benötigten Schlüssel linear und nicht quadratisch ansteigt.
- „Spontane“ Kommunikation: Durch digitale Signaturen kein langwieriges Verifizieren

### NACHTEILE

- Die Primzahlen und andere Zahlen müssen korrekt (und groß genug) gewählt sein um das Verfahren sicher zu machen, dadurch teilweise sehr große Schlüssel nötig
- Ausnahmslos alle zurzeit bekannten Verfahren sind extrem langsam: im Schnitt um den Faktor 10.000 langsamer als symmetrische Verfahren

- Die öffentlichen Schlüssel müssen echt sein, eine Verifizierstelle für kritische Übertragungen ist notwendig →PKI (Public Key Infrastructure)<sup>23</sup>

---

<sup>23</sup> Vgl.: [http://de.wikipedia.org/wiki/Asymmetrisches\\_Kryptosystem](http://de.wikipedia.org/wiki/Asymmetrisches_Kryptosystem)

## RSA

In diesem Kapitel beschäftigen wir uns mit RSA, dem wohl bekanntesten asymmetrischen Verschlüsselungsverfahren, welches auch in PGP zum Einsatz kommt.

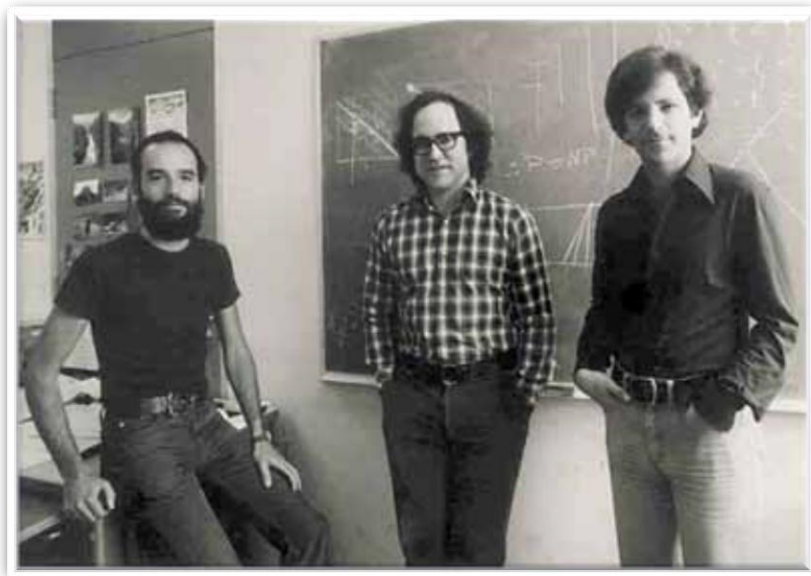
Wichtig sei noch einmal anzumerken, dass RSA in PGP nur für digitale Signaturen verwendet wird und als Verschlüsselung für die eigentliche Nachricht →IDEA verwendet wird. Digitale Signaturen sowie IDEA werden im Kapitel Nachfolger beschrieben. Der Grund dazu ist, dass RSA mit der richtigen Schlüssellänge zwar als unknackbar gilt, jedoch verglichen mit anderen asymmetrischen Verfahren um den Faktor 1000 langsamer ist.

## GESCHICHTLICHES

Diffie und Hellman waren bekanntlich eigentlich erst an zweiter Stelle mit ihrem Verfahren, da Ellis und Co. Ihnen einige Jahre voraus waren und schließlich Cocks und Williamson mit der Primzahl-Mathematik einen wichtigen Ansatz lieferten. Im Jahre 1977 - drei Jahre nachdem Cocks/Williamsons exakt die selbe Idee hatten - veröffentlichten drei Mathematiker einen „unknackbaren Algorithmus“ und revolutionierten damit die Welt der Kryptologie.

## DIE IDEE

Nachdem Withfield Diffie und Martin Hellman ihre Public-Key-Idee in ihrem Buch „New Directions in Cryptography“ 1976 veröffentlicht hatten, wurden die drei Mathematiker Ronald Rivest, Adi Shamir und Leonard Adleman darauf aufmerksam und versuchten das Verfahren zu widerlegen.



*Adi Shamir, Ronald Rivest und Leonard Adleman (v.l.)<sup>24</sup>*

---

<sup>24</sup> Bildquelle: <http://www.at-mix.de/rsa.htm>

Nachdem sie einige Verfahren entdeckt und damit die Unmöglichkeit von Public-Key-Verfahren endlich beweisen wollten, standen sie plötzlich vor einem nahezu unknackbaren Verfahren, welches allen Attacken standhielt.

Hieraus entstand dann RSA (**R**ivest **S**hamir **A**dleman), welches die Drei im Mai 1977 aus einem einfachen Stück Zahlentheorie entwickeln konnten. Dazu jedoch gleich mehr.<sup>25</sup>

---

## DER RECHTSSTREIT

Obwohl RSA eigentlich das zweite Verfahren war, aber die vorherigen nicht veröffentlicht wurden, konnte RSA patentiert werden.

Um den RSA Algorithmus zu verwenden, müssten also eigentlich Patentgebühren entrichtet werden.

Diesen Umstand haben einige Autoren untersucht und sind zu folgendem Ergebnis gekommen:

- Eigentlich könnte man TEILE des RSA-Verfahrens verwenden ohne das Patent zu verletzen
- Das RSA-Patent entspricht (bzw. entsprach) den rechtlichen Anforderungen für Patente in den USA nicht

Die Autoren verweisen darauf, dass dies theoretische Überlegungen sind, die nicht juristisch geprüft seien – die einzelnen Argumente aufzuzählen würden den Rahmen dieser Arbeit sprengen.

RSA wurde also aufgrund seines etwas fadenscheinigen Rechtsumstandes kritisiert, dennoch wurde das Verfahren zum bekanntesten und bis heute unter bestimmten Voraussetzungen unknackbaren Verfahren.

Rechtliche Probleme gab es auch bei PGP: Phillip Zimmermann verwendete RSA ohne Patentgebühren zu entrichten und geriet so in die Kritik. Mehr dazu jedoch im Kapitel zu PGP.

Das Patent lief im September 2000 aus und RSA wurde Allgemeingut und darf frei verwendet werden: Das Verfahren wurde in die „Public Domain“ entlassen (öffentlich zugänglich gemacht).<sup>26</sup>

---

<sup>25</sup> Vgl.: <http://www.hh.schule.de/julius-leber-schule/melatob/historyrsa.html>

<sup>26</sup> Vgl. <http://www.cyberlaw.com/rsa.html>

## DIE MATHEMATIK HINTER RSA

Das RSA-Verfahren ist im Grunde eine Anwendung mehrerer simpler mathematischer Verfahren. Von daher werden wir an dieser Stelle einige mathematische Grundlagen in aller Kürze erläutern, welche im RSA-Algorithmus angewandt werden.

### NATÜRLICHE ZAHLEN

Aus der Oberstufen-Mathematik kennen wir Zahlenmengen, darunter auch die natürlichen Zahlen (**N**) und die ganzen Zahlen (**Z**).

Beispiele:

$$\mathbf{N} = \{0, 1, 2, 3, 4, 5, 6, \dots\}$$

$$\mathbf{Z} = \{-5, -4, -1, 0, 4, 12, \dots\}$$

### PRIMZAHLEN

Primzahlen sind Zahlen, welche nur durch 1 und sich selbst teilbar sind. Sie müssen logischerweise ungleich 1 sein.

Beispiele:

$$2, 3, 5, 7, 11, 13, \dots$$

### DER GRÖßTE GEMEINSAME TEILER

Der größte gemeinsame Teiler (GGT) ist eine natürliche Zahl, welche jeweils den größtmöglichen Teiler von zwei Zahlen darstellt.

Beispiel:

Größter gemeinsamer Teiler von  $a = 6$ ,  $b = 15$  ist 3. Wenn der  $\text{GGT}(a, b) = 1$  ist, spricht man von  $a$  und  $b$  als **teilerfremd**.

### DER EUKLIDISCHE ALGORITHMUS

Wenn wir zwei Zahlen größer Null haben und ihren **größten gemeinsamen Teiler** bestimmen möchten, bedient man sich meist des **euklidischen Algorithmus**.

Der euklidische Algorithmus ist folgendermaßen definiert:

Wenn wir zwei Zahlen ( $a$ ,  $b$  und  $b > 0$ ) haben, gibt es eindeutig bestimmbare ganze Zahlen  $q$  und  $r$ .

$q$  und  $r$  müssen folgende Eigenschaften haben:

„Wenn  $a$  durch  $b$  geteilt wird ergibt dies  $q$  mit dem Rest  $r$  ( $a = b * q + r$ )“ und  $r$  größer gleich 0 und  $b$  größer  $r$ , also:  $\text{GGT}(a, b) = \text{GGT}(b, r)$

Um also den  $\text{GGT}(a, b)$  zu bestimmen, müssen wir solange die obige Operation ausführen – bis sich als Rest  $r = 0$  ergibt.

Dazu verwenden wir den modulo-Operator ( $\%$  oder  $\text{mod}$ ) der den Rest bezeichnet, also bedeutet: „ $r = a \text{ mod } b$ “:  $r$  ist die kleinste natürliche Zahl mit  $a/b-r$ .

Beispiel:

$1024 \text{ mod } 55 = 34$  (denn:  $1024-34 = 990$  und  $990$  ist durch  $55$  teilbar).<sup>27</sup>

---

## DER ERWEITERTE EUKLIDISCHE ALGORITHMUS

Der erweiterte euklidische Algorithmus errechnet zu den bereits vorgestellten Zahlen zusätzlich zwei ganze Zahlen, die folgende Voraussetzung erfüllen:

$$\text{GGT}(a, b) = s * a + t * b$$

Dies bildet eine wichtige Grundlage zum **chinesischen Restesatz**, mit welchem wir verschiedene inverse Elemente und Kongruenzen bestimmen können und der eine wichtige Grundlage zur Berechnung von Primzahlen liefert. Eine detaillierte Beschreibung der Verfahren würde den Rahmen dieser Arbeit sprengen und wir beschränken uns auf die Anwendung in einigen wenigen Fällen.<sup>28</sup>

---

## MODULARE ARITHMETIK

Die Modulare Arithmetik beruht auf den oben vorgestellten Verfahren und findet auch in RSA Anwendung. Man rechnet in der Modularen Arithmetik mit den Resten von Divisionen, wie oben beschrieben und schränkt den Ergebnis-Zahlenraum mithilfe von einem Modulo  $n$  ein.

Das bedeutet, wenn zum Beispiel  $n = 5$  ist, wird eben nur mit  $0$  bis  $4$  gearbeitet. Ein Rechenergebnis das größer als dieser Zahlenraum ausfällt wird dann in eine Restedivision mit modulo  $5$  umgewandelt. Zum Beispiel:  $8 \text{ mod } 5 = 3$  denn der Rest der Division von  $8$  und  $5$  ist  $3$ .

Diese sogenannten „diskreten Logarithmen“ und Quadratwurzeln stellen in sehr, sehr großen Zahlenbereichen schwer berechenbare Probleme für einen Computer da und eignen sich von daher für Verschlüsselungen wie RSA sehr gut.<sup>29</sup>

---

<sup>27</sup> Vgl.: [BEU2] S. 116ff

<sup>28</sup> Vgl.: [http://de.wikipedia.org/wiki/Erweiterter\\_euklidischer\\_Algorithmus](http://de.wikipedia.org/wiki/Erweiterter_euklidischer_Algorithmus)

<sup>29</sup> Vgl.: <http://home.datacomm.ch/th.aes/Daten/Html/Modularitmetik.html>

## DAS RSA-VERFAHREN IM DETAIL

Nachdem die theoretischen Grundlagen überstanden sind, ist der RSA- Algorithmus verhältnismäßig schnell erklärt.

Dem RSA- Algorithmus liegt folgende Tatsache zugrunde:

$$\mathbf{n = p \cdot q}$$

ist das Produkt zweier verschiedener Primzahlen p und q.

Man erstellt eine Gleichung

$$\mathbf{m^{k(p-1)(q-1)} \bmod n = m,}$$

für die m größer/gleich n & k gilt.

Danach wählt man zwei natürliche Zahlen (e und d).

K muss eine natürliche Zahl sein, dann ergeben e und d ein Produkt:

$$\mathbf{e \cdot d = k(p-1)(q-1) + 1}$$

Nun gilt:

$$\mathbf{(m^e)^d \bmod n = m \text{ und } (m^d)^e \bmod n = m}$$

## SCHLÜSSELERZEUGUNG

Um einen Schlüssel zu erzeugen, wählt jeder Teilnehmer zwei unterschiedliche, nach Möglichkeit sehr große Primzahlen (p und q). Das Produkt der beiden Zahlen n wird berechnet und die Zahl  $\mathbf{\varphi(n) = (p-1)(q-1)}$  bestimmt.

Danach wählt man e teilerfremd zu  $\varphi(n)$  und bestimmt mithilfe des erweiterten euklidischen Algorithmus d mit der Formel  $e \cdot d \bmod \varphi(n) = 1$ , mit  $\varphi(n)$  eingesetzt also

$$\mathbf{e \cdot d = 1 + k(p-1)(q-1)}$$

wobei k wie gehabt eine natürliche Zahl ist.

Dann werden die Schlüssel gebildet:

- Öffentlicher Schlüssel: n und e
- Privater Schlüssel: d, die restlichen Zahlen bleiben ebenfalls geheim<sup>30</sup>

---

<sup>30</sup> Vgl. [BEU1], S.19ff

## KRYPTOGRAPHIE MIT RSA

Um mit RSA zu arbeiten müssen wir folgende Formeln anwenden:

Verschlüsseln:  $f_e(m) := m^e \bmod n$

Entschlüsseln:  $f_d(c) := c^d \bmod n$

Solange Korrektheit mithilfe des Eulerschen Satzes gilt:  $f_d(f_e(m)) = (m^e)^d \bmod n = m$

Daraus können wir erkennen: Solange man nur die Zahl  $n$ , also einen Teil des öffentlichen Schlüssels kennt, kann man daraus ohne faktorisieren nicht auf die Primzahlen  $p$ ,  $q$  und somit nicht auf  $e$  oder  $d$  schließen.

Bei entsprechend groß gewählten Primzahlen gewährt dies ein sehr hohes Maß an Sicherheit, da das Faktorisieren von Primzahlen extrem viel Rechenzeit benötigt.

## PGP

In dem nun folgenden Kapitel wird das zweite Hauptthema dieses Seminarkurses behandelt: **PGP**.

## WAS IST PGP?

Unter **PGP** (Pretty Good Privacy) versteht man ein Verschlüsselungsprogramm, das von Phil Zimmermann entworfen, entwickelt und veröffentlicht wurde. Wenn man mit PGP eine Nachricht oder eine Datei verschlüsselt erhält diese einen so hohen Grad an Sicherheit, dass es fast unmöglich ist die verschlüsselte Datei oder Nachricht, ohne den richtigen Schlüssel zu entschlüsseln. Zudem ist es mit PGP möglich seine Dateien und Nachrichten digital zu unterschreiben. Somit ist eine Manipulation von einer dritten Person ausgeschlossen. Aus diesem Grund ist PGP im privaten sowie auch im geschäftlichen Bereich das am weitesten verbreitete Verschlüsselungsprogramm der Welt.

## GESCHICHTLICHES



<sup>31</sup> Phil Zimmermann begann in den 80er Jahren damit, PGP zu entwickeln. Als er nach ca. 10 jähriger Entwicklungszeit das Programm in Umlauf bringen und vermarkten wollte, kam er mit dem Gesetz in Kontakt. Das erste Problem war, dass er keine freie Lizenz des RSA- Verfahrens erhielt, und somit das Programm nicht veröffentlichen durfte. Zudem entließ der Staat ein Gesetz, das besagte, dass alle Anbieter von Kommunikationsgeräten und -diensten bei Verlangen des Klartextes einer Datenübertragung, dieser den staatlichen Behörden vorzulegen ist. Dieses Gesetz ging entschieden gegen die Idee von PGP, das Schützen der Privatsphäre der Nutzer.

Im Jahre 1991 stellte Phil Zimmermann das erste PGP Programm zusammen, das statt DES seinen eigenen Algorithmus Bass-O-Matic als symmetrisches Verfahren verwendete. Dieses PGP Programm, Version 1.0, gab er einem Freund, der es in das Internet stellte und es somit für jedermann zugänglich wurde. Durch diese Tat bekam Phil patentrechtliche Probleme und er hatte damit gegen das Exportrecht der USA verstoßen. Im Jahre 1993 begann Phil bei ViaCrypt zu arbeiten, die eine legale RSA-Lizenz besaßen. Somit gab es keine weiteren Patentrechtlichen Probleme mit PGP und somit konnte die erste legale Version von PGP Version 2.5 im Jahre 1994 verkauft werden. Diese Version beinhaltete bereits statt dem Bass-O-Matic den IDEA- Algorithmus als symmetrisches Verschlüsselungsverfahren.

<sup>31</sup> Bildquelle: <http://www.philzimmermann.com/EN/background/index.html>

Jedoch waren die exportrechtlichen Anschuldigungen der Justiz noch nicht vom Tisch geräumt und ihm wurden mehrerer Ermittlungsverfahren angehängt. Das Verfahren gegen ihn, stellte man aber aufgrund von mangelnden Beweisen am 11. Januar 1996 ein.

Heutzutage ist bereits Version 9.0 in Umlauf, bei der alle bekannten Fehler der Vorgänger behoben und korrigiert wurden und somit eine sehr sichere Verschlüsselung verspricht.

## DIE IDEE HINTER PGP

Die Idee hinter PGP ist, dass Phil Zimmermann ein für alle Menschen zugängliches Verschlüsselungsprogramm entwerfen wollte, mit dem man sowohl Dateien als auch Nachrichten (E-Mails) einfach verschlüsseln kann, sodass nur der Empfänger der e-mails diese lesen kann. Das bedeutet, dass Phil Zimmermann ein Programm entwerfen wollte, dass die Privatsphäre der Menschen schützt und sichert.

„Pretty Good Privacy“ bedeutet auch im übertragenen Sinne – die Privatsphäre soll gegen den Staat geschützt werden<sup>32</sup>.

## DIE FUNKTIONSWEISE VON PGP

### SCHLÜSSELERSTELLUNG

Zu Beginn jeder Schlüsselerstellung wird in PGP, durch das zufällige Bewegen der Maus, eine Zufallszahl erstellt. Nachdem diese Zahl vorhanden ist, wird ein Schlüsselpaar, ein öffentlicher und ein privater Schlüssel, erschaffen. Der öffentliche Schlüssel wird in einer Datei namens „public- key ring“ auf dem lokalen Computer gespeichert und verwaltet. Die Datei, in der der private Schlüssel gespeichert ist, ist Passwort gesichert und wird ebenfalls auf dem lokalen Rechner unter dem Dateinamen „private- key ring“ gesichert<sup>33</sup>.

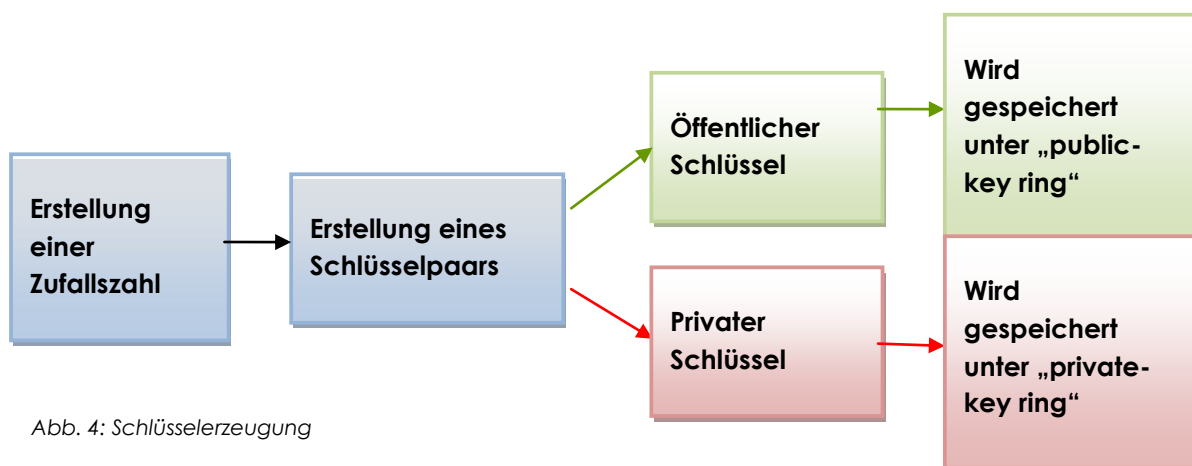


Abb. 4: Schlüsselerzeugung

<sup>32</sup> Vgl. [WOBST] S.308

<sup>33</sup> Vgl. [http://www.informatik.tu-darmstadt.de/BS/Lehre/Sem98\\_99/T11/index.html#tth\\_sEc3](http://www.informatik.tu-darmstadt.de/BS/Lehre/Sem98_99/T11/index.html#tth_sEc3)

## SCHLÜSSELVERWALTUNG

Nach der Erzeugung des Schlüsselpaars bringt der Benutzer, seinen öffentlichen Schlüssel in Umlauf. Das macht er z.B. über einen Key- Server oder durch persönlichen Austausch<sup>34</sup>.

Damit gewährleistet ist, dass der richtige öffentliche Schlüssel des Benutzers in Umlauf ist, wird jeder öffentliche Schlüssel direkt nach seiner Erzeugung signiert, dies wird als „PGP-Fingerprint“ bezeichnet.

Wenn nun z.B. Bob Alice eine sichere Nachricht schicken möchte sucht er auf einem Key-Server nach dem öffentlichen Schlüssel von Alice. Wenn nun aber Eve ihren öffentlichen Schlüssel als den von Alice ausgibt, kann Eve die Verschlüsselte Nachricht lesen und nicht Alice. Somit ist keine sichere Verschlüsselung gewährleistet. Aus diesem Grund wird jeder öffentliche Schlüssel signiert. Wenn Bob den öffentlichen Schlüssel von Alice auf einem Key-Server findet ruft er Alice an und sie vergleichen unter einander den PGP- Fingerprint. Wenn der PGP- Fingerprint von Alice's öffentlichen Schlüssels mit dem auf dem Key- Server übereinstimmt ist bewiesen, dass Bob mit dem richtigen Schlüssel, die Nachricht an Alice verschlüsselt.

Zudem „unterschreibt“ Bob mit dem PGP-Fingerprint seines eigenen Schlüssels den öffentlichen Schlüssel von Alice. Er zeigt dadurch, dass der Schlüssel von Alice der richtige ist.

Wenn Karl nun Alice eine sichere Nachricht senden möchte sucht er wie Bob nach dem öffentlichen Schlüssel von Alice auf einem Key- Server. Er sieht, dass Bob die Richtigkeit des Schlüssels bestätigt hat. Wenn er Bob vertraut kann er Alice ohne vorheriges Überprüfen des richtigen PGP- Fingerprint eine Nachricht senden.

PGP besitzt keine allgemeine Schlüsselverwaltung, sondern jeder Nutzer verwaltet sein Schlüsselpaar alleine. Durch das gegenseitige „unterschreiben“ der öffentlichen Schlüssel der Benutzer entsteht ein „Netz des Vertrauens“ (web of trust).

---

<sup>34</sup> Siehe [http://www.informatik.tu-darmstadt.de/BS/Lehre/Sem98\\_99/T11/index.html#tth\\_sEc3](http://www.informatik.tu-darmstadt.de/BS/Lehre/Sem98_99/T11/index.html#tth_sEc3)

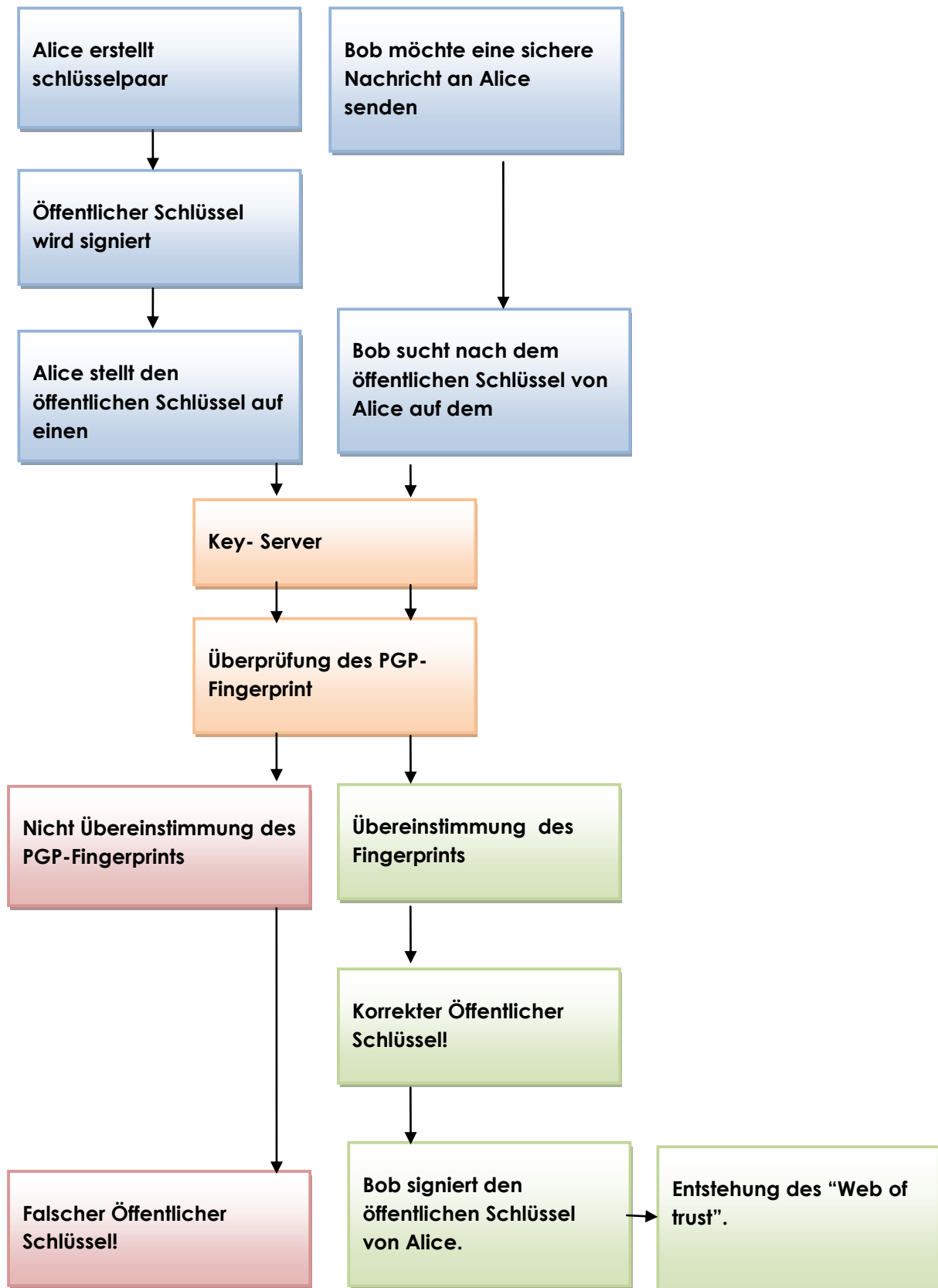


Abb. 5: Die Entstehung des Web of Trust

## VER- UND ENTSCHLÜSSELUNG IM ALLGEMEINEN

Wie bereits erwähnt besitzt jeder PGP Benutzer ein Schlüsselpaar. Einen „öffentlichen“ Schlüssel, den man über e-mails usw. verteilen kann und einen „privaten“ Schlüssel den man auf seinem privaten Computer speichert. Mit dem öffentlichen Schlüssel kann man Nachrichten nur verschlüsseln, jedoch nicht wieder entschlüsseln. Zur Entschlüsselung dieser Nachricht benötigt man den passenden privaten Schlüssel.

Im Übertragenen Sinne kann man sich die Verschlüsselung mit PGP so vorstellen: Sie haben einen Brieffreund und sie geben ihm ein Schloss mit dem er die Post an sie „verschießt“. Wenn die Post bei ihnen ankommt können nur sie sie öffnen, da nur sie den richtigen Schlüssel für das Schloss besitzen.

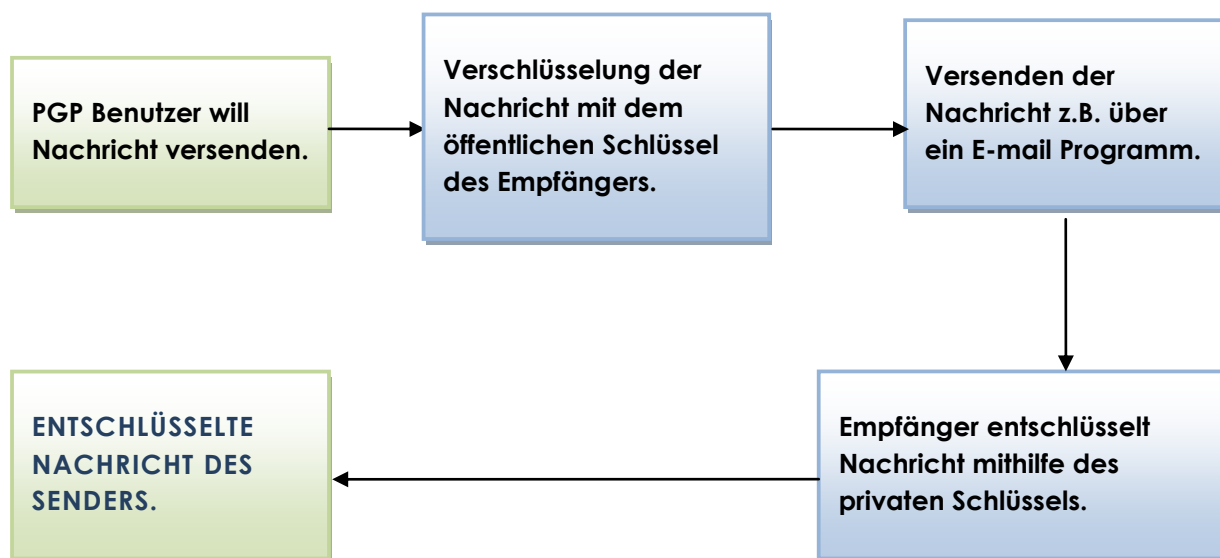


Abb.6: Die Funktionsweise von PGP

## VER- UND ENTSCHLÜSSELUNG IM DETAIL

Zu Beginn der Verschlüsselung erzeugt PGP einen „Einmal-Schlüssel“. Mit diesem „Einmal-Schlüssel“ wird die zu verschlüsselnde Nachricht →symmetrisch verschlüsselt. Danach wird dieser „Einmal-Schlüssel“ mit dem öffentlichen Schlüssel des Empfängers → asymmetrisch verschlüsselt und der symmetrisch verschlüsselten Nachricht hinzugefügt. Nun ist die Nachricht gegen Fremde gesichert und kann getrost und ohne Bedenken versendet werden.

Wenn der Empfänger die Nachricht empfängt, entschlüsselt PGP zuerst mit dem privaten Schlüssel des Empfängers den „Einmal-Schlüssel“. Nun wird mit dem entschlüsselten „Einmal-Schlüssel“ die symmetrisch verschlüsselte Nachricht entschlüsselt. Anschließend kann der

Empfänger die Nachricht lesen, mit der Sicherheit als einzige Person diese Nachricht gelesen zu haben<sup>35</sup>.

PGP verschlüsselt nur den „Einmal-Schlüssel“ asymmetrisch, da es viel zu lange dauern würde um die gesamte Nachricht asymmetrisch zu verschlüsseln. Eine Verschlüsselungsmethode, die ein symmetrisches Verfahren sowohl als auch ein asymmetrisches Verfahren verwendet, wird als hybride Verschlüsselung bezeichnet.

Als symmetrische Verfahren werden in PGP hauptsächlich das IDEA- Verfahren und wahlweise das DES- Verfahren benutzt<sup>36</sup>.

Als asymmetrische Verfahren werden in PGP hauptsächlich der RSA- Algorithmus, neben diesem jedoch auch das Elgamal- Kryptosystem , verwendet<sup>37</sup>.

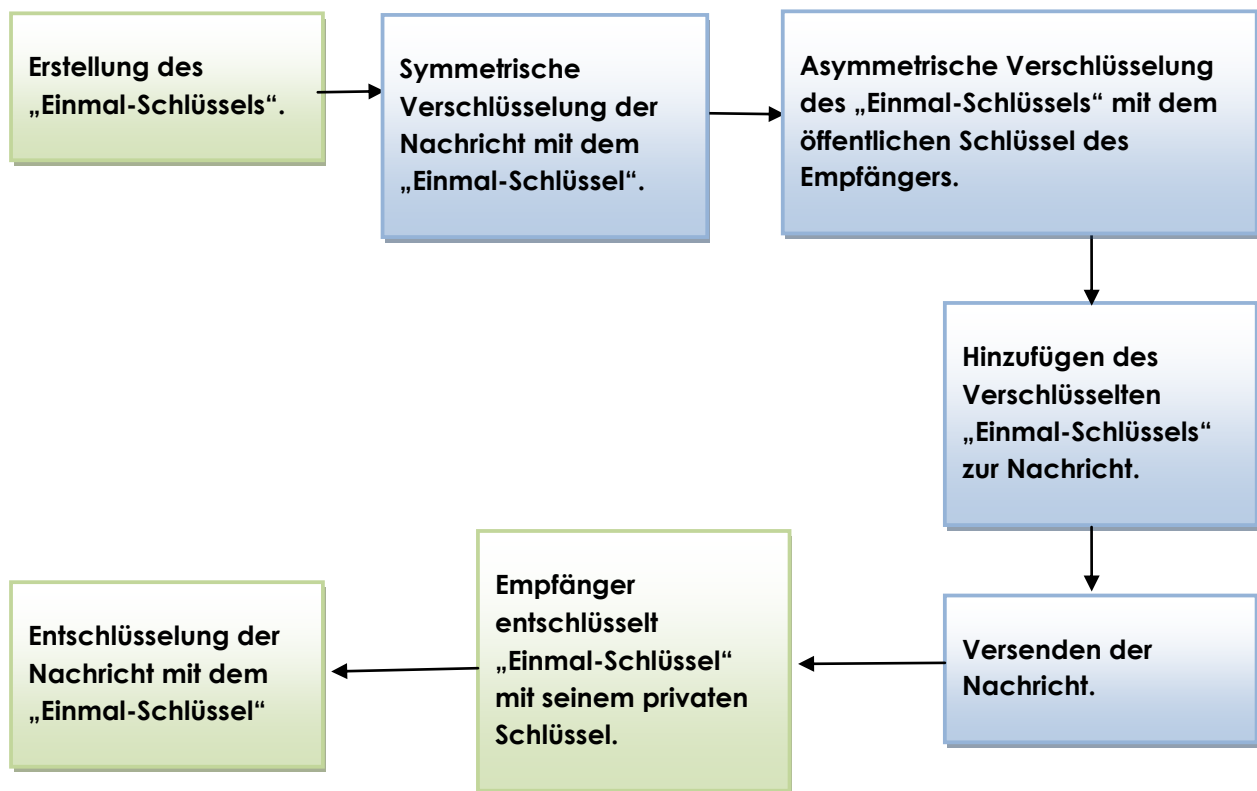


Abb.7: Ver- und Entschlüsselung mit PGP im Detail

<sup>35</sup> Vgl. [http://www.geocities.com/Athens/1802/ger\\_pgpdoc1.html#3.3](http://www.geocities.com/Athens/1802/ger_pgpdoc1.html#3.3)

<sup>36</sup> Siehe [http://www.fz-juelich.de/zam/docs/bhb/bhb\\_html/d0141/keymang1.htm](http://www.fz-juelich.de/zam/docs/bhb/bhb_html/d0141/keymang1.htm)

<sup>37</sup> Siehe <http://de.wikipedia.org/wiki/PGP>

## DIE VOR- UND NACHTEILE VON PGP

### VORTEILE

Es ist ganz klar, dass heutzutage PGP die stärkste und sicherste Methode ist, Nachrichten, Dateien usw. zu verschlüsseln. Nachfolgend sind die Vorteile von PGP zusammengefasst:

- Der Quellcode des Programms steht zur freien Verfügung und kann somit auf Sicherheitslücken, Programmierfehler oder so genannte „Hintertürchen“ überprüft werden.
- Die neuste Version von PGP ist 9.0. Seit der Veröffentlichung von PGP Anfang der 90er wurde das Verschlüsselungssystem immer wieder weiter verbessert und verfeinert. Zudem wurden in der neuesten Version alle bekannten Fehlerquellen beseitigt.
- Das Programm ist plattformunabhängig, das bedeutet, dass es auf Windows, Linux usw. reibungslos eingesetzt und genutzt werden kann.
- Bei richtiger Anwendung des Programms garantiert es eine fast unknackbare Verschlüsselung.
- Das wichtigste ist jedoch, dass es heutzutage eine →Freeware Version von PGP im Internet zum downloaden gibt, sprich jeder Mensch der über das Internet verfügt, kann sich PGP zum Eigenbedarf herunterladen und somit seine Nachrichten gegen jegliches Lesen von Fremden schützen<sup>38 39</sup>.

### NACHTEILE

Der größte Nachteil von PGP kann durch den Benutzer selbst entstehen, denn PGP lässt seinen Benutzern viele Freiheiten offen, somit kann das Programm bei falscher Benutzung genau einen anderen Effekt erzielen als erhofft:

- Nachrichten werden verschlüsselt, können aber leicht von anderen Personen gelesen und entschlüsselt werden.
- Das „Web of trust“ kann durch falsche Handhabung gefährdet werden.

Ein anderes Problem, das durch eine solch starke, für jedermann frei zugänglichen, Verschlüsselung entsteht, ist das nicht nur Privatpersonen das Programm einsetzen, sondern auch Verbrecher, Terroristen usw., die mithilfe von PGP ihre Pläne sicher und schnell über das

---

<sup>38</sup> Vgl. [http://www.informatik.tu-darmstadt.de/BS/Lehre/Sem98\\_99/T11/index.html#tth\\_sEc3.5](http://www.informatik.tu-darmstadt.de/BS/Lehre/Sem98_99/T11/index.html#tth_sEc3.5)

<sup>39</sup> Vgl. [WOBST] S. 307 ff.

Internet austauschen und verbreiten können. Dann ist es selbst für die größten Geheimdienste der Welt extrem schwierig diese Nachrichten zu entschlüsseln.

## FAZIT

Nachdem nun die Vor- und Nachteile von PGP vorgestellt wurden, sieht man sofort, dass die Vorteile den Nachteilen von PGP überlegen. Trotzdem entsteht die Frage, ob man Programme wie PGP weiter entwickeln soll und ob das überhaupt von Nutzen für die Privatsphäre von Menschen ist.

Heutzutage gibt es in jedem Land Geheimdienste und Abhörstationen, die den Nachrichtenfluss von Unternehmen, Vereinigungen und unter anderem Einzelpersonen abhören können. Für diese Geheimdienste ist es eine Leichtigkeit sich in den privaten Nachrichtenfluss (z.B. Telefonleitungen) einzuschalten und diesen abzuhören. Der UKUSA-Verband ist ein Geheimdienstverband, dem die Länder USA (NSA), Großbritannien(GCHQ), Kanada(CSE), Australien(DSD) und Neuseeland (GCSB) angehören. Das Echelon- System, ist ein weltweites Spionagenetz innerhalb dieses Geheimverbundes. Die Aufgabe dieses Systems, ist das Abhören der Satelliten-, Mikrowellen- und Mobilfunk- Kommunikation. Dieses System verfügt über 120 Abhörstationen und diverse Satelliten zur Nachrichtenüberwachung. Diese Abhörinstallationen, die unter anderem auch in Deutschland, Japan und anderen Ländern installiert sind, sollen in der Lage sein ca. 90 % des Internetverkehrs mit verfolgen zu können.<sup>40</sup>

Diese 90% sind regelrecht ausschlaggebend dafür, dass es wichtig ist, seine privaten Nachrichten und Dateien zu verschlüsseln und vor großen Geheimdiensten, oder sei es „nur der Nachbar“, zu schützen und zu sichern.

Damit das möglich ist, müssen Programme wie PGP immer weiter entwickelt werden und den Menschen zur freien Verfügung gestellt werden.

---

<sup>40</sup> Vgl. [WÖBST] S. 345 und <http://de.wikipedia.org/wiki/Echelon>

## DIE NACHFOLGER VON PGP UND RSA

In diesem Kapitel werden die bekannten und weniger bekannten – möglichen - Nachfolger des RSA-Algorithmus behandelt. Die einzelnen Verfahren werden chronologisch abgehandelt, wobei der Fokus nicht auf Bekanntheit, Nützlichkeit oder Vollständigkeit gelegt wurde, sondern darauf, inwiefern diese Algorithmen einzigartig in ihrem Verfahren sind.

### EL-GAMAL

Im Jahre 1985 veröffentlichte Taher El-Gamal, ein amerikanischer Wissenschaftler ursprünglich aus Ägypten stammend, seinen Aufsatz „*A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms*“ (Ein Public-Key-Kryptosystem und ein Signaturschema basierend auf „diskreten Logarithmen“). In diesem beschrieb er das nach ihm benannte El-Gamal-Verfahren.

### EINFÜHRUNG

Das El-Gamal-Kryptosystem ist ein →asymmetrisches Verschlüsselungsverfahren. Der „diskrete Logarithmus“ des Diffie-Hellman-Schlüsselaustausches findet hier Verwendung.

Es kann sowohl zur Signaturerzeugung (→Fingerprint) als auch zur Verschlüsselung eingesetzt werden.

Damit El-Gamal verwendet werden kann, müssen zwei Voraussetzungen erfüllt sein:

- Korrektheit – Das Protokoll muss immer auf Betrugsversuche überprüft werden
- Spezifikation – Die Durchführbarkeit des Protokolls muss gewährleistet sein, indem sich alle Beteiligten an das Protokoll halten <sup>41</sup>

El-Gamal gilt als sicher, es wurde bis jetzt noch kein Angriff auf den Algorithmus erfolgreich durchgeführt, solange die Voraussetzungen (die oben genannten Rahmenbedingungen sowie die mathematischen Grundvoraussetzungen, wie im Abschnitt über das Diffie-Hellman-Verfahren erklärt) erfüllt waren.

### DAS VERFAHREN

Das El-Gamal-Kryptosystem setzt wie alle asymmetrischen Verfahren auf einen öffentlichen und privaten Schlüssel. Das Verfahren ist eine abgeänderte Version des Diffie-Hellman-Schlüsselaustauschs, die Mathematik ist die gleiche.

Taher Elgamal trug wesentlich zu vielen heutigen Krypto-Standards bei, unter anderem zu →SSL und SET, ein Kreditkartensystem.

---

<sup>41</sup> Vgl.: [http://www.wiwi.uni-bielefeld.de/StatCompSci/lehre/material\\_spezifisch/statalg00/rsa/node10.html](http://www.wiwi.uni-bielefeld.de/StatCompSci/lehre/material_spezifisch/statalg00/rsa/node10.html)

## ZERO-KNOWLEDGE

### EINFÜHRUNG

In diesem Kapitel beschäftigen wir uns mit einer höchst interessanten, jedoch weitgehend unbekanntem Variante aus der Welt der Kryptologie: dem so genannten Zero-Knowledge-Verfahren.

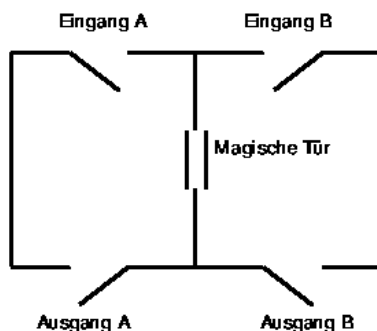
Im Grunde geht es hierbei nur um Authentifizierung – quasi dem Kommunikationspartner versichern, die Person zu sein, die man vorgibt zu sein.

Das möchte man erreichen indem man als „Beweiser“ dem „Verifizierer“ plausibel macht, dass man ein gemeinsames „Geheimnis“ kennt und dabei natürlich so wenig wie möglich über das Geheimnis selbst verrät. Wie kann man so etwas erreichen?

Zum besseren Verständnis unternehmen wir einen kleinen Exkurs in die Geschichte. Im 16. Jahrhundert fand der italienische Mathematiker Niccolo Fontana Tartaglia eine Lösung für Gleichungen dritten Grades. Er wollte seine Lösungsformel nicht publik machen, aus Angst jemand könnte diese als seine ausgeben. Man glaubte ihm zuerst nicht. Tartaglia konnte jedoch die Nullstellen jeder ihm gestellten Gleichung dritten Grades sehr schnell bestimmen – er musste also eine Lösungsformel besitzen.<sup>42</sup>

Zwar verstößt das gegen die eigentliche Idee hinter Zero-Knowledge – nichts preiszugeben – (in diesem Fall wird die Lösung der Formel als Beweis preisgegeben), aber zur Veranschaulichung des Grundproblems sollte dieses Beispiel genügen.<sup>43</sup>

### DAS VERFAHREN IM DETAIL



<sup>44</sup> Man möchte also dahinter kommen, ob jemand ein Geheimnis kennt, ohne etwas von dem Geheimnis preiszugeben. Aus Sicht des Verifizierers sähe das z.B. so aus: Man kennt ein Geheimnis und gibt dem Beweiser Anweisungen, die ohne das Wissen des Geheimnisses nur teilweise lösbar sind. Etwa: Ein Raum mit 2 Türen, die im Raum selbst von einer weiteren, verschlossenen „Magischen Tür“ - deren Schlüssel das Geheimnis ist - getrennt sind.

Man gibt dem Beweiser die Anweisung, durch eine der Türen zu gehen (Eingang A oder Eingang B). Wenn man dabei wegsieht (also nicht weiß durch

<sup>42</sup> Vgl. [BEU2], S. 79ff

<sup>43</sup> Vgl. <http://www.wisdom.weizmann.ac.il/~oded/zk-tut02.html>

<sup>44</sup> Bildquelle: [http://www.it.fht-esslingen.de/~schmidt/vorlesungen/kryptologie/seminar/ws9798/html/chip/Zero\\_Knowledge.gif](http://www.it.fht-esslingen.de/~schmidt/vorlesungen/kryptologie/seminar/ws9798/html/chip/Zero_Knowledge.gif)

welche er geht) und dem Beweiser dann z.B. mitteilt, durch Ausgang A wieder nach draußen zu gehen und er durch Eingang B gegangen ist, kann dieser nur durch die „magische Tür“ zu diesem gelangen indem er das Geheimnis (Schlüssel) kennt um die magische Tür zu öffnen. Wenn man dies ein Mal durchführt, besteht eine Chance von 50% dass der Beweiser zufällig durch Eingang A hinein ist und man ihn an Ausgang A zu sehen wünscht.

Diesen Ablauf kann man mehrmals wiederholen und die Wahrscheinlichkeit, dass die Person richtig liegt nimmt exponentiell ab: Beim ersten Mal: 50% ( $1/2$ ), beim zweiten Mal 25% ( $1/2 * 1/2 = 1/4$ ), beim dritten Mal  $1/8$  und beim zehnten  $1/1024$ . Also besteht in nur 10 Durchläufen eine Wahrscheinlichkeit von ca. 0,09%, dass der Beweiser immer in der richtigen Tür erscheint ohne das Geheimnis zu kennen.<sup>45</sup>

Wie wir sehen, sind viele, wiederholte Durchläufe nötig, um ein angemessenes Maß an Sicherheit zu gewähren. Deshalb sind Zero-Knowledge-Protokolle in der Praxis kaum tauglich, man setzt eher auf digitale Signaturen, wie z.B. →RSA-Fingerprints.

## DIE MATHEMATIK HINTER ZERO-KNOWLEDGE

Heutzutage werden Zero-Knowledge Operationen nach dem Fiat-Shamir-Protokoll durchgeführt, auch wenn es ein Bit preisgibt – also wiederum kein Zero-Knowledge-konformes Verfahren, wenn man es genau nimmt.

Als Verständnisgrundlage wird das Kapitel zu RSA empfohlen, da hier als der RSA-Algorithmus eingesetzt wird und wir die gleichen Buchstaben für die RSA-Algorithmuselemente verwenden.

Dieses Verfahren benötigt eine dritte Partei, die einen →RSA-Fingerprint ( $n=p*q$ ) veröffentlicht wobei p und q geheimgehalten werden.

Danach werden folgende Schritte durchgeführt:

1. Der Beweiser berechnet  $v=s^2 \bmod n$  (Öffentlicher Schlüssel)
2. Dieser Öffentliche Schlüssel wird bei der dritten Partei registriert.
3. Der Beweiser wählt eine Zufallszahl x, er berechnet  $y=x^2 \bmod n$
4. Der Verifizierer wählt zufällig  $e=1$  oder  $e=0$  aus und sendet diese Zahl an den Beweiser
5. Dabei gibt er an, ob er die Wurzel aus x (bei  $e=0$ ) oder die Wurzel aus  $x*v$  (bei  $e=1$ ) wählt.
6. Der Beweiser rechnet  $k=x*s^e \bmod n$  und sendet das Ergebnis an den Verifizierer
7. Der Verifizierer überprüft, ob  $k^2 = x*v^e$  gilt.<sup>46</sup>

Diese Schritte werden solange durchgeführt, bis die gewünschte Genauigkeit bzw. Sicherheit erreicht ist.

---

<sup>45</sup> Vgl. [BEU2], S. 80ff

<sup>46</sup> Vgl. [BEU2], S. 82ff

## DAS IDEA-VERFAHREN

### EINFÜHRUNG

Der IDEA (**I**nternational **D**ata **E**ncryption **A**lgorithm) ist ein kommerzieller Algorithmus. Er wurde 1992 von der Firma ASCOM in der Schweiz entwickelt und patentiert (darf für nicht-kommerzielle Zwecke frei verwendet werden). Der Grund weshalb wir IDEA behandeln ist, dass – wie schon mehrmals erwähnt - dieser Algorithmus das eigentliche, in PGP verwendete Kryptosystem ist. RSA wird beim „echten“ PGP nur für sog. →Fingerprints verwendet. Der Einfachheit halber haben wir unser begleitendes Programm auf RSA aufgebaut.<sup>47</sup>

### DAS VERFAHREN IM DETAIL

IDEA ist ein symmetrisches Verfahren, welches mit Blockchiffrierungen arbeitet. Die Blockgröße ist mit 64 Bit äquivalent zu →DES. Jedoch ist die Schlüssellänge mit 128 Bit um einiges größer und sicherer.

Die verwendeten mathematischen Operationen sind sog. XOR (Exklusiv-Oder) und modulo.

IDEA erzeugt aus seinem 128 Bit Schlüssel 52 16-Bit Einzelschlüssel, indem zuerst acht 16 Bit Schlüssel erzeugt, die Ziffern des großen (128 Bit) um 25 Bit nach links verschoben werden und der Vorgang solange wiederholt wird, bis man 52 einzelne, je 16 Bit lange Schlüssel vorliegen hat. Danach werden die 64 Bit Textblöcke ebenfalls in (vier) 16 Bit Blöcke zerlegt.

Die eigentliche Verschlüsselung verläuft in acht sich wiederholenden Schritten.

Zuerst werden der erste und letzte Einzelschlüssel mit dem ersten und letzten Block multipliziert.

Der zweite und dritte Einzelschlüssel jeweils mit dem zweiten und dritten Block addiert.

Die Ergebnisse werden mit XOR verknüpft

Daraus entstehen vier Blöcke, die im zweiten Durchgang verwendet werden, bis man acht Durchgänge abgeschlossen hat. Die Reihenfolge der Blöcke wird vor den unterschiedlichen Durchgängen geändert.<sup>48</sup>

---

<sup>47</sup> Vgl. [http://www.regenechsen.de/phpwcms/index.php?krypto\\_idea](http://www.regenechsen.de/phpwcms/index.php?krypto_idea)

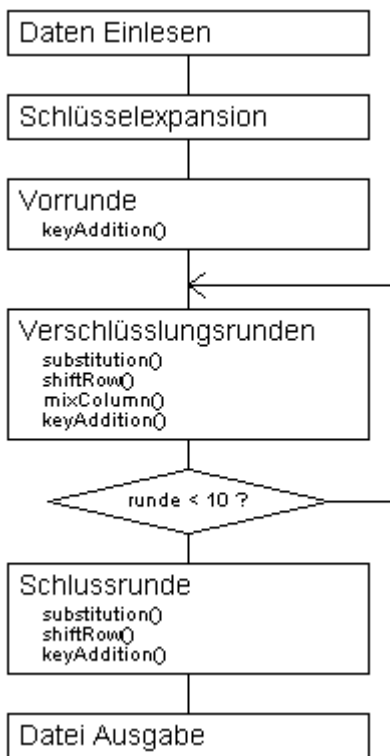
<sup>48</sup> Vgl.: [http://www.regenechsen.de/phpwcms/index.php?krypto\\_idea](http://www.regenechsen.de/phpwcms/index.php?krypto_idea)

DER RIJNDAEL ALGORITHMUS

Der Rijndael-Algorithmus (benannt nach seinen belgischen Erfindern Joan Daemen und Vincent Rijmen) wurde im Oktober 2000 vom NIST (National Institute of Standards and Technology) als Nachfolger zu →DES und →3DES unter dem Namen AES (Advanced Encryption Standard) bekannt gegeben. AES ist ein →symmetrisches Verschlüsselungsverfahren mit einer Blockgröße von 128 Bit, variablen Schlüssellängen (128, 192 oder 256 Bit) und ist kostenlos (also lizenzfrei) nutzbar.

Rijndael ist ein sehr sicherer, sowohl als auch überdurchschnittlich schneller →Blockchiffre.<sup>49</sup>

FUNKTIONSWEISE



<sup>50</sup> Rijndael wendet (im Gegensatz zu DES) einzelne Teile des Schlüssels nacheinander auf den zu verschlüsselnden Text an und durchläuft mehrere Runden. Ein beispielhafter Ablauf ist links skizziert.<sup>51</sup>

Die einzelnen Schritte sind:

**1. Schlüsselexpansion**

In diesem Schritt wird der Schlüssel in n+1 (n=Anzahl der Runden) Teile unterteilt. Die Größe der Teile entspricht der vorher festgelegten Schlüssellänge (z.b. 128 Bit). Diese Teile werden in eine Tabelle (Blöcke) „geschrieben“, die Tabelle ist 4x4 groß.

**2. KeyAddition**

Nun werden die einzelnen Blöcke mit Exklusiv-Oder (XOR) verknüpft.

**3. Verschlüsselungsrunden**

**a. Substitution**

Jedes Byte wird monoalphabetisch verschlüsselt.

**b. ShiftRow**

Nun werden einzelne Zeilen/Spalten in jedem Block verschoben. Überlaufende Zellen werden von rechts fortgesetzt. Die Anzahl der Verschiebungen ist zeilen- und blocklängenabhängig.

**c. mixColumn**

<sup>49</sup> Vgl.: <http://de.wikipedia.org/wiki/Rijndael>

<sup>50</sup> Bildquelle: <http://home.datacomm.ch/th.aes/Daten/Html/Visualisierung.html>

<sup>51</sup> <http://csrc.nist.gov/CryptoToolkit/aes/rijndael/>

Nun werden die Spalten vermischt indem die Zeilen mit einer Konstante multipliziert und mit XOR verknüpft werden.

Es gilt für die Konstanten: Zeile 1: 2, Zeile 2: 3, Zeile 3: 1, Zeile 4: 1

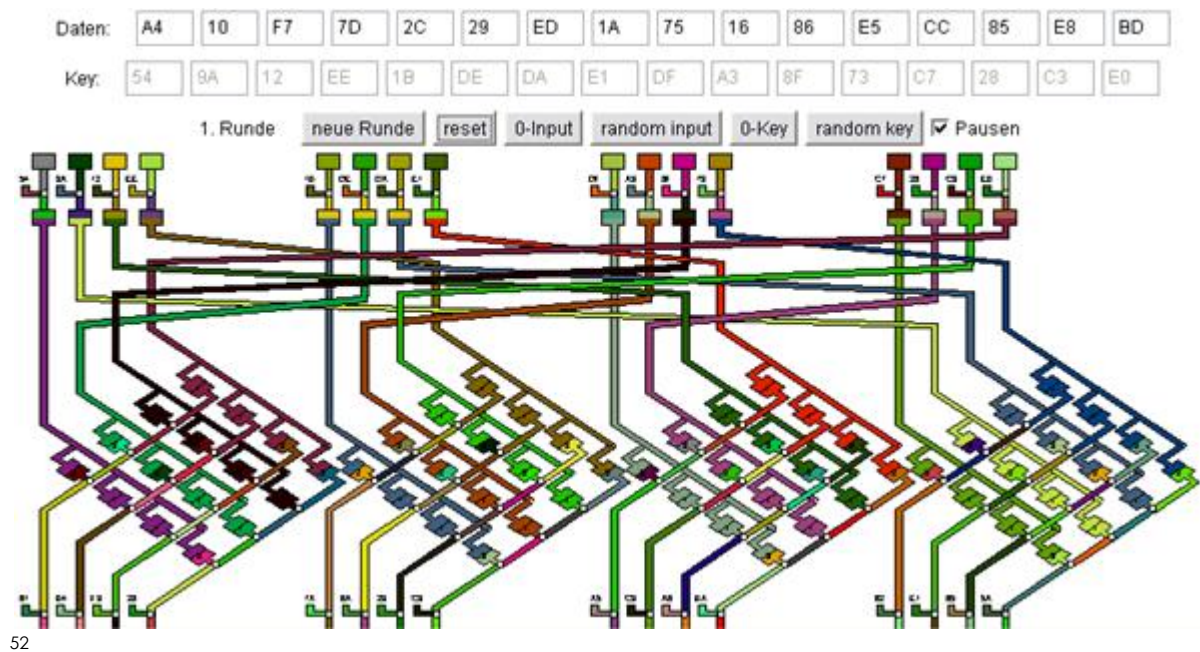
#### d. KeyAddition

Ist die maximale Anzahl von Runden erreicht:

#### 4. Schlussrunde

Wiederholung der oben genannten Schritte exklusive mixColumn

Die Folgende Flussvisualisierung demonstriert den komplizierten Ablauf einer Verschlüsselung mit Rijndael, wobei hier nur die erste Runde gezeigt wird:



Hinter all diesen Vorgängen stecken komplizierte mathematische Vorgänge, auf die wir nicht näher eingehen werden.

<sup>52</sup> Bildquelle: <http://www-math.uni-paderborn.de/~aggathen/rijndael/2001/flussvisualisierung/>

## PROGRAMM: PUBLIC-KEY-KRYPTOGRAPHIE MITHILFE VON RSA IN JAVA

### ÜBERSICHT

#### PROGRAMMBESCHREIBUNG & SCHWERPUNKT

In diesem Teil beschäftigen wir uns mit dem informationstechnischen Teil der Seminararbeit. Wir haben uns entschieden, ein nahe an PGP angelehntes Programm mit RSA als →asymmetrisches Verschlüsselungsverfahren zu programmieren. Das Programm soll ähnlich wie PGP aufgebaut sein, also folgende Funktionalitäten bieten<sup>53</sup>:

- Die Möglichkeit aus einem Text, den der Benutzer eingibt, einen privaten und öffentlichen Schlüssel zu erstellen.
- Mit eben diesen Schlüsseln
  - o Texte (z.B. E-Mails) an andere zu verschlüsseln
  - o Dateien sicher zu ver- und entschlüsseln
  - o Texte anderer Anwender zu entschlüsseln.
- Die eigenen Schlüssel abzuspeichern und
- Den eigenen öffentlichen Schlüssel abzurufen und im öffentlichen Schlüsselverzeichnis zur Verfügung zu stellen , um geheime Nachrichten mit anderen Anwendern auszutauschen.
- den privaten Schlüssel geheim zu halten

Dabei soll das Programm dem Benutzer eine gewisse Transparenz bieten. Es soll keine Simulation darstellen und alle Vorgänge akribisch genau beschreiben, sondern dem Benutzer zumindest die Möglichkeit geben, die erzeugten Schlüssel und andere Daten einzusehen. Das Ergebnis – und auch der Weg dorthin – werden wir auf den folgenden Seiten ausführlich beschreiben. Den Schwerpunkt der Dokumentation unseres Programms haben wir hierbei auf die Klassendiagramme, Ablauf-Übersichten und Sequenzdiagramme gelegt. Der Quelltext ist im folgenden Kapitel einsehbar und durchgehend kommentiert.

#### PMP – PRETTY MUCH PRIVACY

Als Namen für unser Programm haben wir PMP, eine Abkürzung für „Pretty Much Privacy“ gewählt, in Anlehnung an PGP – Pretty Good Privacy. Der von uns gewählte Name für unser Programm beschreibt unserer Meinung nach das Programm passend: „Ziemlich viel“, aber doch nicht alles von PGP – wir erheben in keinsten Weise Anspruch darauf, dieses Programm auf Niveau des großen Vorbildes anzusiedeln.

---

<sup>53</sup> Vgl. [ZIMM], S. 7 ff

## NOTATION

Entgegen der ursprünglichen Absicht, für Attribute und Operationen die ungarische Notation<sup>54</sup> zu verwenden, welche zu kompliziert für ein solch eher kleines Projekt wäre, haben wir uns für ein eigenes System entschieden:

**Namensgebung der Attribute/Operationen:** Alle Operationen und Attribute wurden in Englisch mit entsprechendem Präfix benannt.

Präfix (Attribut)	Bedeutung
p	Parameter
GROSSBUCHSTABEN	Konstante

Präfix (Operation)	Bedeutung
set	Attribut ändern
get	Attribut auslesen
convert	Umwandlung in anderen Datentyp
add	Hinzufügen

Präfix (Objekt)	Bedeutung
btn	Jbutton - Knopf
lbl	Jlabel – Text
ta	JTextArea – Großes Textfeld
ep	JeditorPanel – Textfeld, mit HTML formatierbar

<sup>54</sup> Vgl. <http://www.uni-koblenz.de/~daniel/Namenskonventionen.html> , S. 1

## HINWEISE ZUM PROTOTYPEN

Zu dem vorliegenden Prototypen zur Halbjahresabgabe sind noch folgende Punkte anzumerken:

**Wichtig:** Die erzeugten Schlüssel werden noch nicht abgespeichert und sind somit bei jedem Programmstart neu zu erzeugen.

Verschlüsseln von Dateien wird noch nicht unterstützt.

Die vorliegende Oberfläche ist nur zu Testzwecken ausgelegt und wird noch komplett umgestaltet.

Bekannte Probleme, die bis zur Abgabe noch nicht behoben wurden/werden konnten:

Textfelder bzw. Textareas sind Copy & Paste untauglich.

Sehr selten ist es nicht möglich einen Text zu entschlüsseln, da die Klasse Converter eine `StringIndexOutOfBoundsException` erzeugt, was rein codetechnisch eigentlich ausgeschlossen wurde. Die genauen Umstände unter denen dieses Problem auftritt sind nicht bekannt.

Das Programm wurde unter folgenden Betriebssystemen erfolgreich getestet:

Linux 2.6.18-1.2869.fc6 #1 SMP i686 GNU/Linux

Windows XP Service Pack 2

Diese Punkte werden bis zur finalen Abgabe-Version des Programms bestmöglich ausgebessert bzw. behoben.

## BILDSCHIRMFOTOS DES PROTOTYPEN

Hier sind einige Bildschirmfotos unseres Programmprototypen abgebildet. Sie sollen dazu dienen, einen visuellen Eindruck des Programms bzw. des Prototypen zu bekommen.



Prototyp bei der Schlüsselerzeugung.



Prototyp nach der Verschlüsselung eines Textes mithilfe eines Öffentlichen Schlüssels.

## HINWEISE ZUR FINALEN VERSION

Die im Prototyp bekannten Probleme wurden weitgehend ausgebessert.

### ANMERKUNGEN:

Der Benutzer wurde aus geschwindigkeitstechnischen Gründen in seiner Freiheit eingeschränkt: Die maximale Länge eines Textes wurde auf 256 Zeichen beschränkt um das Programm nicht unnötig zu verlangsamen.

In dieser Version ist ein Schlüsselaustausch auf minimalistischer Ebene möglich: Jede Kopie des Programms greift auf den Order /shared/ eine Ebene über dem Programmordner zu. Von daher ist es wichtig dass sich alle Kopien des Programmordners im selben Verzeichnis wie der Ordner /shared/ befinden.

Das Logfenster scrollt nicht automatisch mit, eine Lösung für das Problem ergab sich nicht, von daher wurde es so belassen.

### WICHTIG:

Da das Programm einige neue Features der **Java Standard Edition Version 6** verwendet, ist es unbedingt zu empfehlen, das Programm via der mitgelieferten \*.exe-Datei zu starten, welche eine Kopie des JDK 6.1 verwendet um das Programm zu starten.

### STARTEN DES PROGRAMMS:

Entweder PMP.exe (integrierte JRE) oder PMP\_externe\_JRE.exe (externe, bereits installierte JRE) ausführen.

Zur Struktur der Ordner:

/java/            Kopie der JRE v6, es ist also kein installiertes Java nötig um das Programm zu verwenden

/shared/        Verzeichnis, in welchem die Schlüssel abgespeichert werden

/PMP/            Das Programm, zu starten via PMP.exe oder javaw pmp.PMPMain

/PMP\_2/        Eine Kopie des Programms um den Schlüsseltausch zu testen.

Zusätzlich die Datei jdk-6u1-windows-i586-p.exe, mit dieser Version wurde das Programm entwickelt.

### KOMPATIBILITÄT:

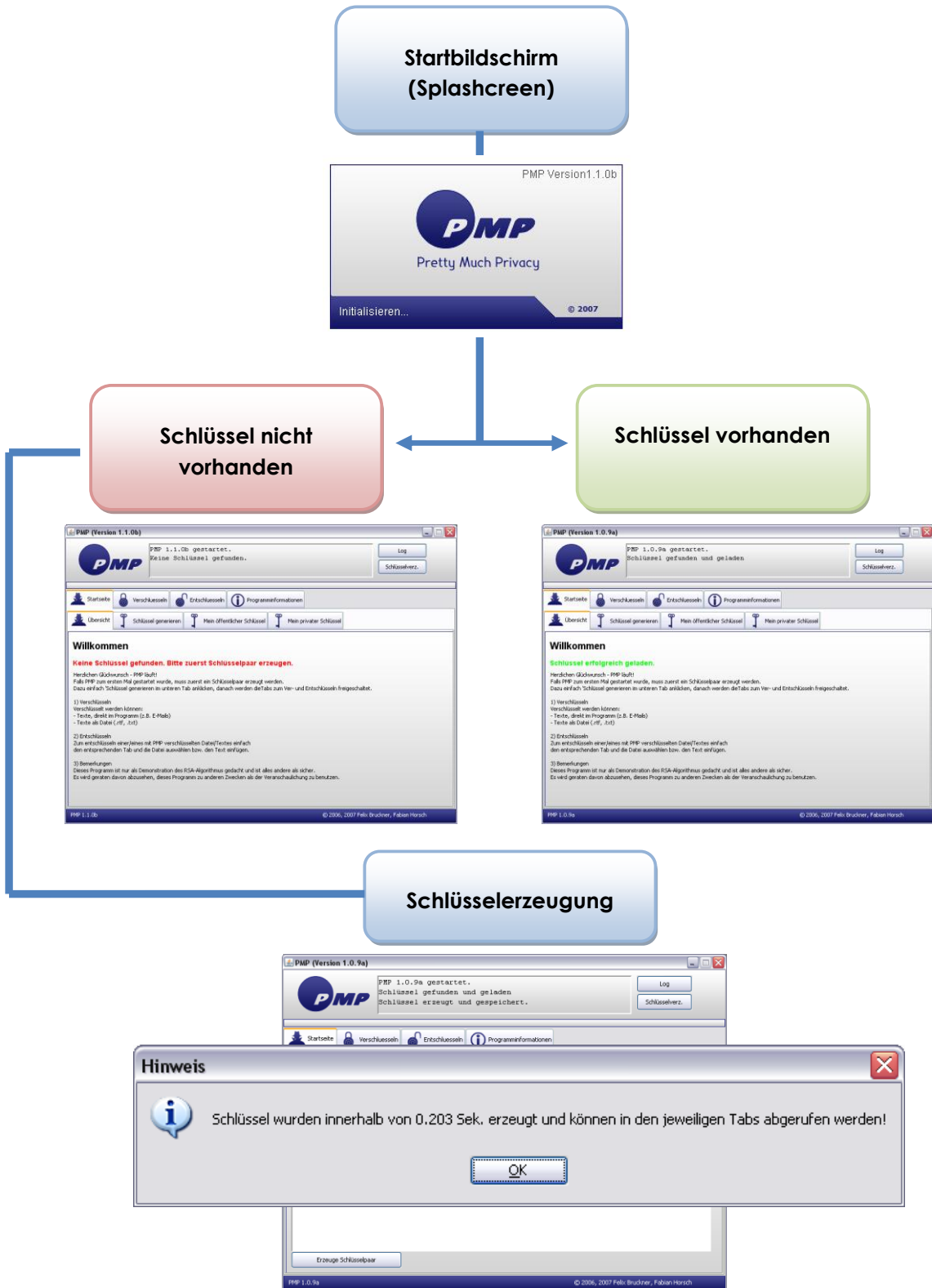
Das Programm wurde unter folgenden Betriebssystemen erfolgreich getestet:

- Linux 2.6.18-1.2869.fc6 #1 SMP i686 GNU/Linux
- Windows XP Service Pack 2

PROGRAMMABLAUF

Zur Veranschaulichung der Funktionsweise unseres Programms sind einige Punkte im Programmablauf auf den folgenden Seiten exemplarisch und bildreich skizziert.

PROGRAMMSTART

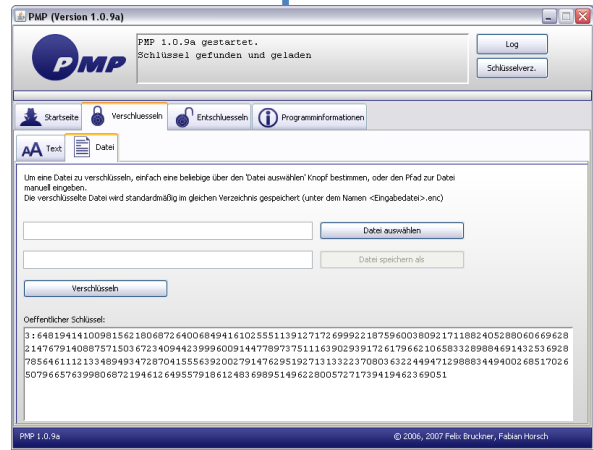
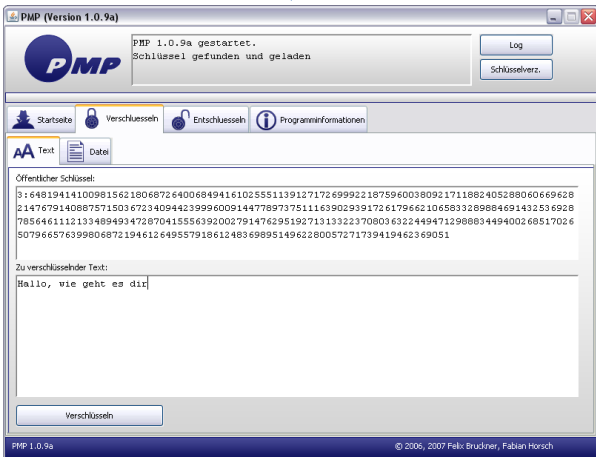


VERSCHLÜSSELN

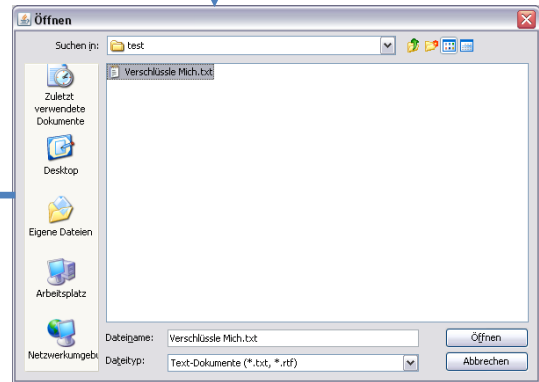
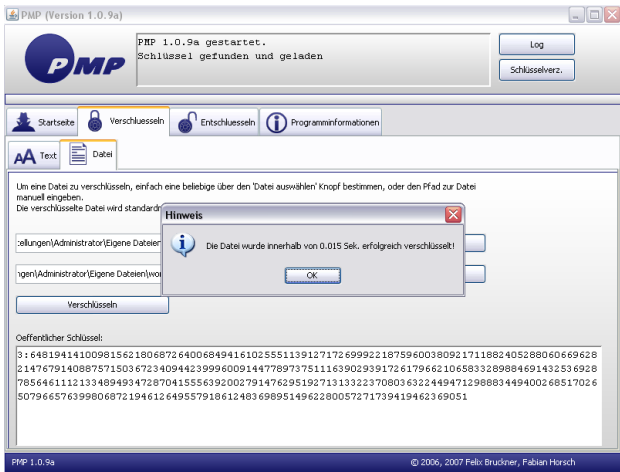
Verschlüsseln

Text

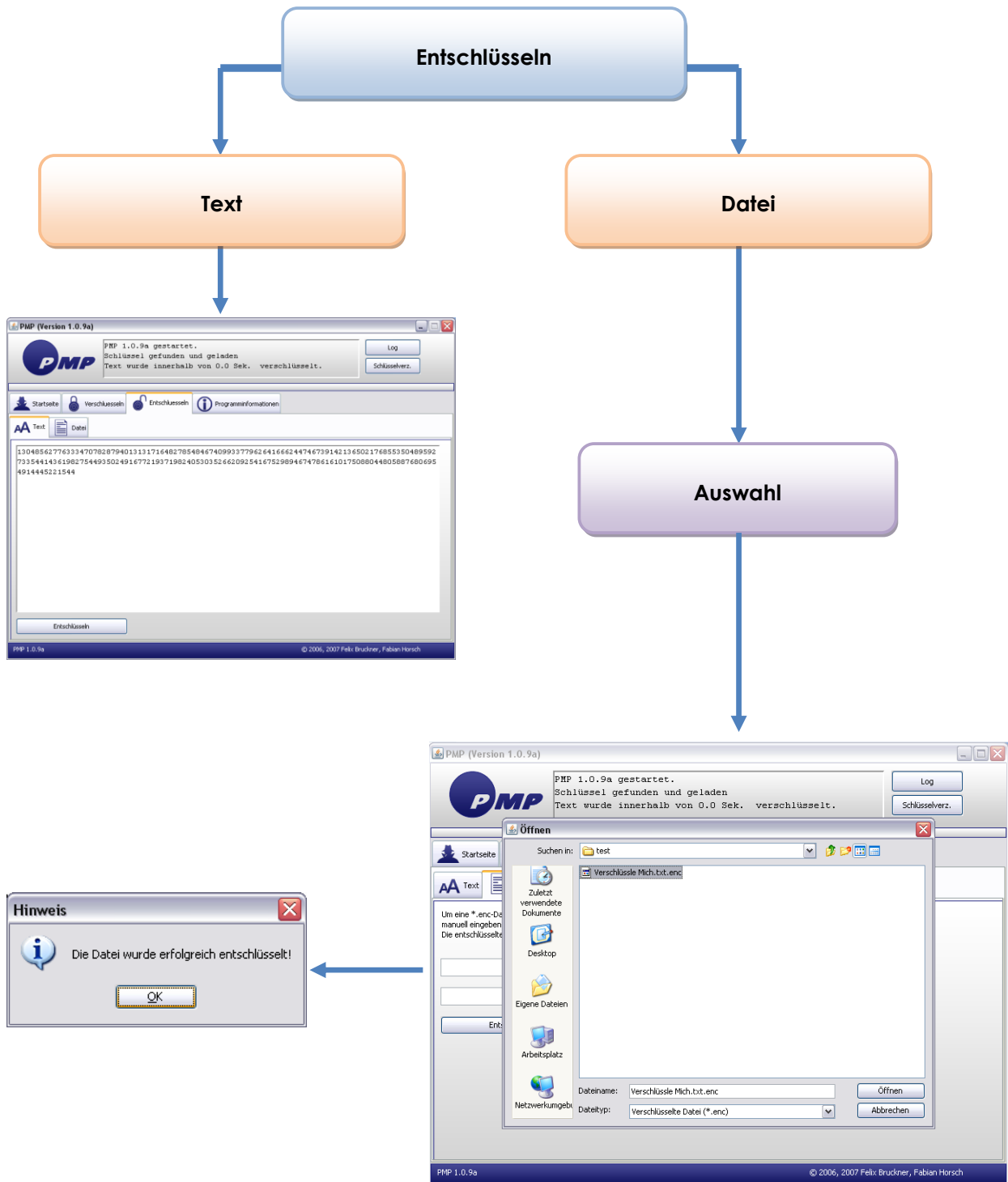
Datei



Auswahl



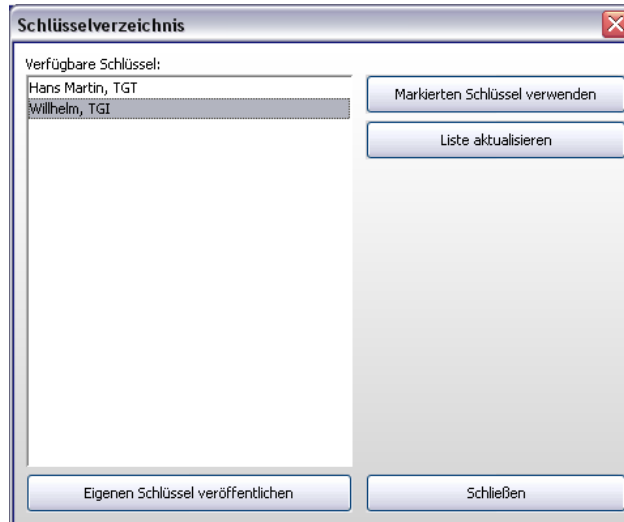
ENTSCHLÜSSELN



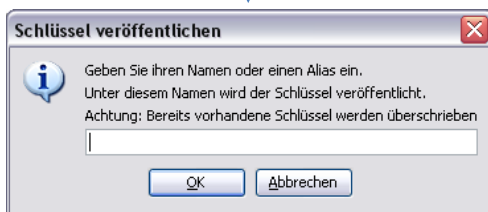
## SCHLÜSSELVERZEICHNIS

Eigenen Schlüssel veröffentlichen

Auswahl des Schlüssels



Verwendung als Public Key



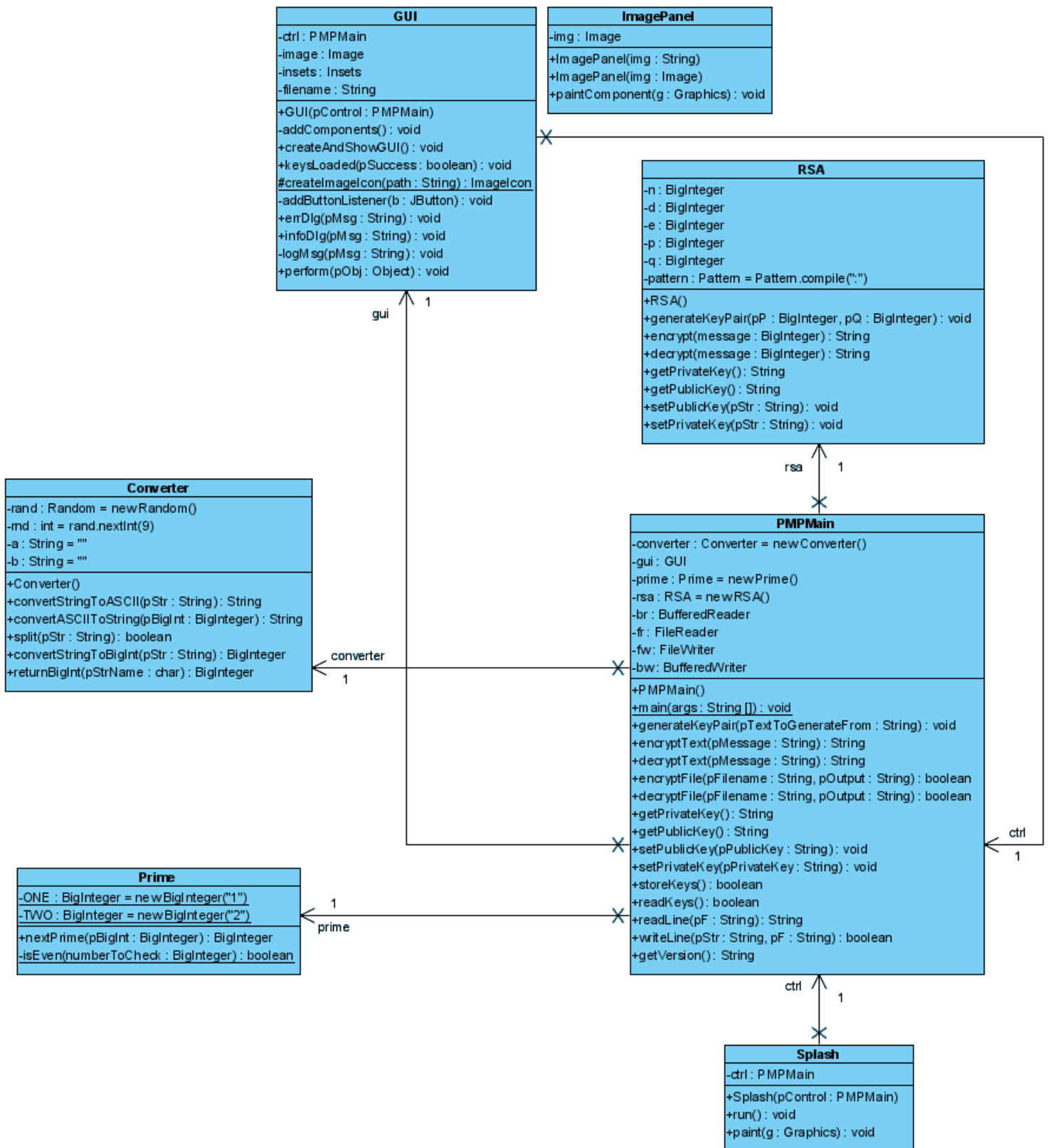
## KLASSENDIAGRAMME

Nachfolgend werden die verwendeten Klassen in unserem Programm in aller Kürze beschrieben, sowie als Klassendiagramme dargestellt.

Zu den vorliegenden Klassendiagrammen bleibt zu sagen, dass in dieser Entwicklungsphase (Prototyp) in den Klassendiagrammen noch keine Operationen zum Verschlüsseln von Dateien existieren. Diese sollen später als zusätzliche Operationen von der Steuerung PMPMain implementiert werden.

Zur Generierung der Klassendiagramme wurde das Casetool **Visual Paradigm** verwendet. Dieses Case-Tool verwendet den **UML 2.1 – Standard** und wendet diesen auch in Klassendiagrammen an, weshalb einige neue, ungewohnte Elemente auftauchen können.

## GESAMTÜBERSICHT MIT ASSOZIATIONEN



## PMPMAIN

<b>PMPMain</b>
<pre>-converter : Converter = newConverter() -gui : GUI -prime : Prime = newPrime() -rsa : RSA = newRSA() -br : BufferedReader -fr : FileReader -fw : FileWriter -bw : BufferedWriter</pre>
<pre>+PMPMain() +main(args : String []): void +generateKeyPair(pText ToGenerateFrom : String) : void +encryptText(pMessage : String) : String +decryptText(pMessage : String) : String +encryptFile(pFilename : String, pOutput : String) : boolean +decryptFile(pFilename : String, pOutput : String) : boolean +getPrivateKey() : String +getPublicKey() : String +setPublicKey(pPublicKey : String) : void +setPrivateKey(pPrivateKey : String) : void +storeKeys() : boolean +readKeys() : boolean +readLine(pF : String) : String +writeLine(pStr : String, pF : String) : boolean +getVersion() : String</pre>

Die Klasse PMPMain bildet die Steuerungsklasse des gesamten Programms. Sie besitzt Assoziationen zu allen „unterstützenden“ Klassen und vermittelt zwischen der Oberfläche und den restlichen Klassen.

## RSA

<b>RSA</b>
<pre>-n : BigInteger -d : BigInteger -e : BigInteger -p : BigInteger -q : BigInteger -pattern : Pattern = Pattern.compile(":")</pre>
<pre>+RSA() +generateKeyPair(pP : BigInteger, pQ : BigInteger) : void +encrypt(message : BigInteger) : String +decrypt(message : BigInteger) : String +getPrivateKey() : String +getPublicKey() : String +setPublicKey(pStr : String) : void +setPrivateKey(pStr : String) : void</pre>

Diese Klasse implementiert den RSA-Algorithmus, wie er im RSA-Teil dieser Arbeit beschrieben ist, in Java-Programmcode.

## CONVERTER

<b>Converter</b>
<pre>-rand : Random = newRandom() -md : int = rand.nextInt(9) -a : String = "" -b : String = ""</pre>
<pre>+Converter() +convertStringToASCII(pStr : String) : String +convertASCIIToString(pBigInt : BigInteger) : String +split(pStr : String) : boolean +convertStringToBigInt(pStr : String) : BigInteger +returnBigInt(pStrName : char) : BigInteger</pre>

Die Klasse Converter ist, wie der Name schon sagt, für das Konvertieren zuständig. Die Hauptaufgabe besteht darin, ASCII in Strings umzuwandeln sowie

zurück, 2 Strings zu teilen und führende Nullen zu verhindern.

## PRIME

<b>Prime</b>
-ONE : BigInteger = new BigInteger("1")
-TWO : BigInteger = new BigInteger("2")
+nextPrime(pBigInt : BigInteger) : BigInteger
-isEven(numberToCheck : BigInteger) : boolean

Mit der Klasse Prime wird die jeweils nächstgelegene Primzahl einer Zahl bestimmt. Diese Funktionalität wird zur Erzeugung eines Schlüsselpaares benötigt.

## GUI

<b>GUI</b>
-ctrl : PMPMain
-image : Image
-insets : Insets
-filename : String
+GUI(pControl : PMPMain)
-addComponents() : void
+createAndShowGUI() : void
+keysLoaded(pSuccess : boolean) : void
#createImageIcon(path : String) : ImageIcon
-addButtonListener(b : JButton) : void
+errDlg(pMsg : String) : void
+infoDlg(pMsg : String) : void
-logMsg(pMsg : String) : void
+perform(pObj : Object) : void

Die Klasse GUI verwaltet alle Elemente der Oberfläche und besitzt eine Assoziation zur Steuerungsklasse (PMPMain), welche den eigentlichen Programmablauf steuert.

Bei diesem Klassendiagramm wurden die als „private“ deklarierten GUI-Komponenten außen vorgelassen, um das Diagramm nicht künstlich zu vergrößern.

## HILFSKLASSEN (SPLASH, IMAGEPANEL)

<b>ImagePanel</b>
-img : Image
+ImagePanel(img : String)
+ImagePanel(img : Image)
+paintComponent(g : Graphics) : void

Diese Klasse erbt von JPanel und erweitert es um die Funktion, ein Hintergrundbild einzubinden.

<b>Splash</b>
-ctrl : PMPMain
+Splash(pControl : PMPMain)
+run() : void
+paint(g : Graphics) : void

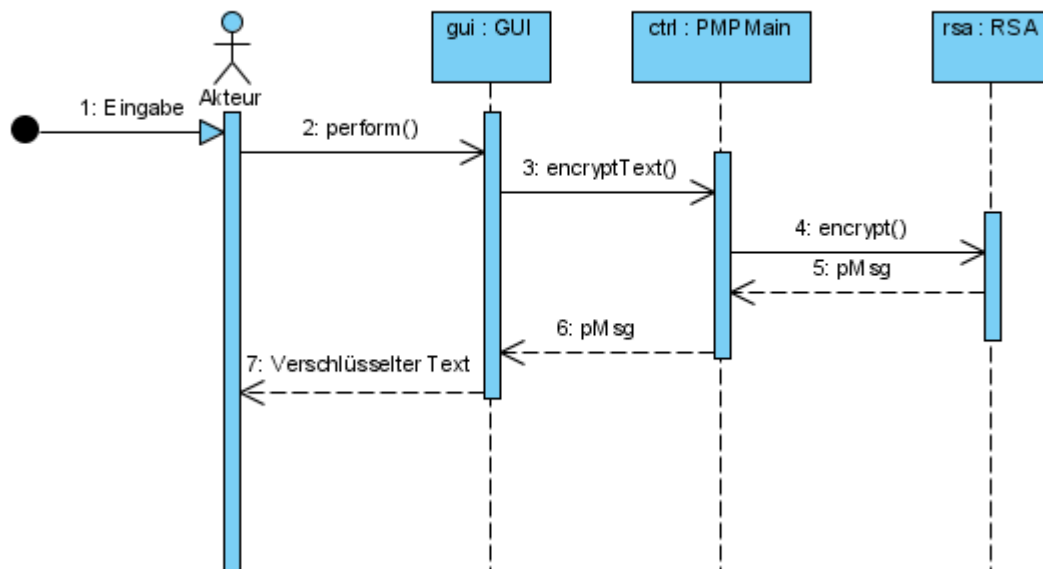
Die Klasse Splash zeigt einen kleinen Startbildschirm beim Starten des Programms an und versucht währenddessen über die Klasse PMPMain die erforderlichen Daten auszulesen

## SEQUENZDIAGRAMME

## VERSCHLÜSSELN EINES TEXTES

Beim Verschlüsseln eines Textes wird die Benutzereingabe wie folgt behandelt:

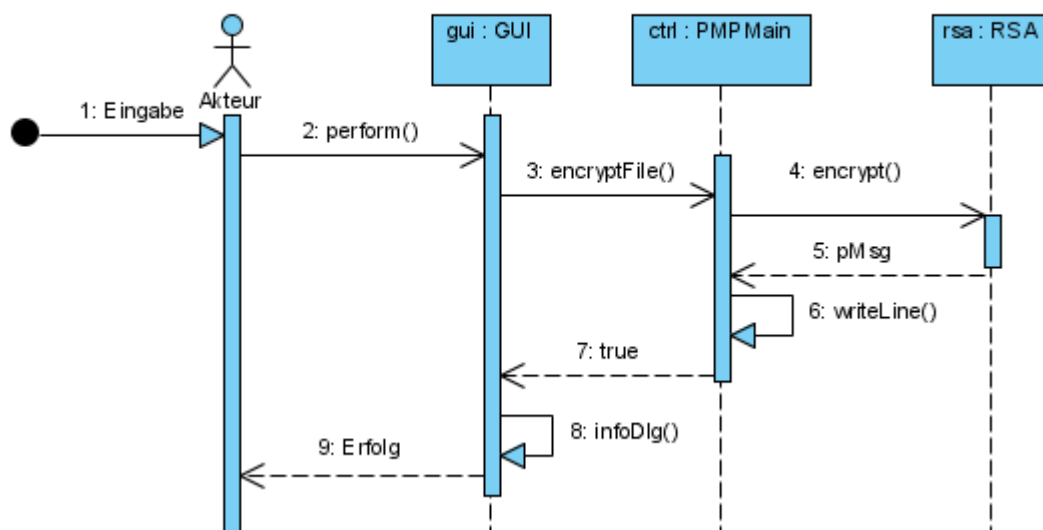
**sd Verschlüsseln**



## VERSCHLÜSSELN EINER DATEI

Das Verschlüsseln einer Datei läuft ähnlich ab:

**sd Verschlüsseln**



## QUELLETEXT DES PROGRAMMS

In diesem Teil kann man den Quelltext unseres Programms zum jetzigen Entwicklungsstand (Finale Version) einsehen.

**ANMERKUNG:** Aus Platzgründen wurden (wie auch in den Klassendiagrammen) jegliche Definitionen/Erzeugungen von Swing-GUI-Komponenten (JTextArea, JButton usw.) gekürzt. Dies verringert den Umfang des Quelltexts um mehr als 15 Seiten. Die Einsicht in die Funktionsweise des Programms wird dadurch nicht beeinträchtigt.

### PMPMAIN.JAVA

```
/*
 * PMPMain.java
 * Datum: 12/2006
 * Autor: Felix Bruckner
 * Beschreibung:
 * Die Main-Klasse des Seminarkurs-Programms, welche die Steuerung
 * des Programms uebernimmt.
 *
 */
//-----
package pmp;

import java.io.*;

import java.math.BigInteger;

//-----
public class PMPMain {
// Assoziationen
private Converter converter = new Converter();

private GUI gui;

private Prime prime = new Prime();

private RSA rsa = new RSA();

// IO-Objekte
private BufferedReader br;

private FileReader fr;

private FileWriter fw;

private BufferedWriter bw;

// -----
// Konstruktor
public PMPMain() {
    // Splashscreen...
    Thread splashThread = new Thread(new Splash(this));
    splashThread.start();

    // ... und anschließend GUI anzeigen
    gui = new GUI(this);
}
```

```

        gui.createAndShowGUI();
    }

    // -----
    // Main-Methode
    public static void main(String[] args) {
        PMPMain main = new PMPMain();
    }

    // -----
    // Operationen, um zwischen Oberflaeche und restlichen Klassen zu
    // kommunizieren

    // -----
    // Schlüsselpaar erzeugen
    public void generateKeyPair(String pTextToGenerateFrom) {
        // String in ASCII wandeln und anschließend splitten
        converter.split(converter.convertStringToASCII(pTextToGenerateFrom));

        // Die gesplitteten Zahlen mithilfe der Prime-Klasse
        // zur nächstgelgenden Primzahl erhöhen und als p und q
        // speichern
        BigInteger p = prime.nextPrime(converter.returnBigInt('a'));
        BigInteger q = prime.nextPrime(converter.returnBigInt('b'));

        // In der RSA Klasse ein neues Schlüsselpaar erzeugen und
        // dort abspeichern
        rsa.generateKeyPair(p, q);
    }

    // -----
    // Ver- und Entschlüsselungsoperationen

    // Nachricht verschlüsseln
    public String encryptText(String pMessage) {
        return rsa.encrypt(
            converter.convertStringToBigInt(converter
                .convertStringToASCII(pMessage)).toString());
    }

    // Nachricht entschlüsseln
    public String decryptText(String pMessage) {
        return converter.convertASCIIToString(new BigInteger(rsa
            .decrypt(new BigInteger(pMessage))));
    }

    // Datei verschlüsseln
    public boolean encryptFile(String pFilename, String pOutput) {
        String s = "";
        String str = "";
        File f = new File(pFilename);
        File of = new File(pOutput);
        try {
            br = new BufferedReader(new InputStreamReader(
                new FileInputStream(f)));
            while (null != (s = br.readLine())) {
                str = str + s + "\n";
            }
            br.close();
            String output = encryptText(str);

```

```

        try {
            fw = new FileWriter(of);
            bw = new BufferedWriter(fw);
            bw.write(output);
            bw.close();
            return true;
        } catch (Exception e) {
            gui.errDlg("Konnte nicht in Datei schreiben");
            return false;
        }
    } catch (Exception e) {
        gui.errDlg("FEHLER: " + e);
        return false;
    }
}

// Datei entschlüsseln
public boolean decryptFile(String pFilename, String pOutput) {
    String str = "";
    File of = new File(pOutput);
    try {
        str = readLine(pFilename);
        br.close();
        String output = decryptText(str);

        try {
            fw = new FileWriter(of);
            bw = new BufferedWriter(fw);
            bw.write(output);
            bw.close();
            return true;
        } catch (Exception e) {
            gui.errDlg("Konnte nicht in Datei schreiben");
            return false;
        }
    } catch (Exception e) {
        gui.errDlg("FEHLER: " + e);
        return false;
    }
}

// -----
// Operationen für die Komm. zwischen den Klassen RSA und GUI

// Privaten Key abrufen
public String getPrivateKey() {
    try {
        return rsa.getPrivateKey().toString();
    } catch (Exception e) {
        return "Kein Schlüssel gefunden";
    }
}

// Public Key abrufen
public String getPublicKey() {
    if (rsa.getPublicKey().toString().equals("null:null")) {
        return "Kein Schlüssel gefunden";
    } else {
        return rsa.getPublicKey().toString();
    }
}

```

```

    }
}

// Öffentlichen Schlüssel setzen
public void setPublicKey(String pPublicKey) {
    rsa.setPublicKey(pPublicKey);
}

// Privaten Schlüssel setzen
public void setPrivateKey(String pPrivateKey) {
    rsa.setPrivateKey(pPrivateKey);
}

// -----
// Schlüssel IO

// Schlüssel speichern
public boolean storeKeys() {
    try {
        writeLine(getPublicKey(), System.getProperty("user.dir")
            + "\\data\\public.key");
        writeLine(getPrivateKey(), System.getProperty("user.dir")
            + "\\data\\private.key");
        return true;
    } catch (Exception e) {
        return false;
    }
}

// Eigenen Schlüssel veröffentlichen
public boolean publishOwnKey(String pFilename, String pKey) {
    try {
        writeLine(getPublicKey(), System.getProperty("user.dir")
            + "\\..\shared\\keys\\" + pFilename);
        return true;
    } catch (Exception e) {
        return false;
    }
}

// Schlüssel laden
public boolean readKeys() {
    // Wenn eine der Dateien 0 Byte groß ist oder nicht existiert,
    abbruch
    try {
        if (readLine(
            (System.getProperty("user.dir") +
            "\\data\\public.key"))
            .equals("")
            || readLine(
                System.getProperty("user.dir")
                +
            "\\data\\private.key").equals("")) {
            return false;
        } else {
            try {

                setPublicKey(readLine(System.getProperty("user.dir")
                    + "\\data\\public.key"));
            }
        }
    }
}

```

```

        setPrivateKey(readLine(System.getProperty("user.dir")
            + "\\data\\private.key"));
        return true;
    } catch (Exception e) {
        gui
            .errDlg("Konnte Schluessel zwar laden,
jedoch NICHT setzen!\n"
Dateien beschaedigt.\n"
            + "Moeglicherweise .key-
            + e);
        return false;
    }
} catch (Exception e) {
    return false;
}
}

// -----
// Hilfoperationen
// Erste Zeile aus einer Datei lesen
public String readLine(String pF) {
    String str;
    File vf = new File(pF);
    if (vf.length() == 0) {
        return "";
    }
    try {
        fr = new FileReader(vf);
        br = new BufferedReader(fr);
        str = br.readLine();
    } catch (Exception e) {
        System.out.println(e);
        return "";
    }
    return str;
}

// Eine Zeile in eine Datei schreiben
public boolean writeLine(String pStr, String pF) {
    try {
        fw = new FileWriter(pF);
        bw = new BufferedWriter(fw);
        bw.write(pStr);
        bw.close();
        return true;
    } catch (Exception e) {
        return false;
    }
}

// -----
// Versionsinformation
public String getVersion() {
    return readLine(System.getProperty("user.dir") +
"\\data\\version.def");
}
}

```

## RSA.JAVA

```
/*
 * RSA.java
 * Datum: 12/2006
 * Autor: Felix Bruckner
 * Beschreibung:
 * Implementierung des RSA Algorithmus
 *
 * Referenzen für die Mathematik hinter dem RSA Algorithmus:
 * http://www.wiwi.uni-
 bielefeld.de/StatCompSci/lehre/material\_spezifisch/statalg00/rsa/node9.html
 */
package pmp;

import java.math.BigInteger;
import java.util.Random;
import java.util.regex.*;

public class RSA {
    private BigInteger n, d, e, p, q;

    private Pattern pattern = Pattern.compile(":");

    public RSA() {
    }

    public void generateKeyPair(BigInteger pP, BigInteger pQ) {
        p = pP;
        q = pQ;

        n = p.multiply(q);

        BigInteger m = (p.subtract(BigInteger.ONE)).multiply(q
            .subtract(BigInteger.ONE));
        e = new BigInteger("3");
        while (m.gcd(e).intValue() > 1)
            e = e.add(new BigInteger("2"));
        d = e.modInverse(m);
    }

    public String encrypt(BigInteger message) {
        return message.modPow(e, n).toString();
    }

    public String decrypt(BigInteger message) {
        return message.modPow(d, n).toString();
    }

    public String getPrivateKey() {

        return d.toString();

    }

    public String getPublicKey() {
        return e + ":" + n;
    }

    public void setPublicKey(String pStr) {
```

```
        String[] str = pattern.split(pStr);
        e = new BigInteger(str[0]);
        n = new BigInteger(str[1]);
    }

    public void setPrivateKey(String pStr) {
        d = new BigInteger(pStr);
    }
}
```

## CONVERTER.JAVA

```

/*
 * Convert.java
 * Datum: 12/2006
 * Autor: Felix Bruckner
 * Beschreibung:
 * UnterstÃtzende-Klasse fÃ¼r den Seminarkurs, sie wandelt und
 * teilt/verkettet Strings in bzw, aus ASCII Code (um).
 */
//-----
--
package pmp;

import java.math.BigInteger;
import java.util.Random;

//-----
--
public class Converter {
    // Zufallszahl von 1-9
    private Random rand = new Random(); // Diese Zufallszahl wird benutzt,

    private int rnd = rand.nextInt(9); // um eine fuehrende 0 zu verhindern.

    // String 1
    private String a = "";

    // String 2
    private String b = "";

    // -----
    // Konstruktor
    public Converter() {
        a = a + rnd;
        b = b + rnd;
    } // public Converter()

    // -----
    // String in einen String aus ASCII Codes umwandeln
    public String convertStringToASCII(String pStr) {

        String str = "";
        for (int i = 0; i < pStr.length(); i++) {
            char c = pStr.charAt(i);

            if ((int) c >= 100) {
                str = str + (int) c;
            } else {
                str = str + "0" + (int) c;
            }
        }
        return str;
    } // public String convertStringToASCII(String pStr)

    // -----
    // ASCII Codes in String umwandeln
    public String convertASCIIToString(BigInteger pBigInt) {
        String myTemp = "" + pBigInt.toString();
        String str = "";

```

```

for (int i = 1; i <= myTemp.length() - 1; i = i + 3) {
    int z = i;
    String temp = myTemp.substring(z, z + 3);
    int c = Integer.valueOf(temp).intValue();
    str = str + new Character((char) c).toString();
}

return str; // String zurÃ¼ckgeben
} // public String convertASCIIToString(BigInteger pBigInt)

// -----
// String teilen
public boolean split(String pStr) {
    String str = this.convertStringToASCII(pStr);
    // String in a und b aufteilen
    int sw = 0;

    for (int i = 0; i < str.length(); i = i + 3) {
        String temp = str.substring(i, i + 3);
        switch (sw) {
            case 0:
                a = a + temp;
                sw++;
                break;
            case 1:
                b = b + temp;
                sw = 0;
                break;
        }
    }
    return true;
} // public boolean split(String pStr)

// -----
// String in BigInt umwandeln
public BigInteger convertStringToBigInt(String pStr) {
    String str = rnd + pStr;
    BigInteger bigInt = new BigInteger(str);
    return bigInt;
}

// -----
// Funktion, um die beiden Strings in BigInts umzuwandeln
// und zurÃ¼ckzugeben
public BigInteger returnBigInt(char pStrName) {
    BigInteger converted;

    switch (pStrName) {
        case 'a':
            converted = new BigInteger(a.toString());
            break;
        case 'b':
            converted = new BigInteger(b.toString());
            break;
        default:
            converted = new BigInteger("0");
    }
    return converted;
} // public BigInteger returnBigInt(char pStrName)
} // public class Converter

```

## PRIME.JAVA

```

/*
 * Convert.java
 * Datum: 12/2006
 * Autor: Felix Bruckner
 * Beschreibung:
 * Primzahlen-Klasse, die aus gegebenen BigInteger-Zahlen
 * die nächstliegende Primzahl nach dem Miller-Rabin Test
 * sucht.
 *
 * Code lose angelehnt an die JSP CoreServlets von Sun Microsystems Press:
 * http://www.coreservlets.com/
 */
//-----
package pmp;

import java.math.BigInteger;

//-----
public class Prime {

// Da BigInteger nicht mit int umgehen kann, werden hier
// zwei Konstanten mit den BigIntegern 1 und 2 angelegt
    private static final BigInteger ONE = new BigInteger("1");

    private static final BigInteger TWO = new BigInteger("2");

// -----
// Nächstgelegene Primzahl nach dem mit BigInteger mitgeliefertem
// Miller-Rabin Test suchen.
// Die Wahrscheinlichkeit, dass die gegebene Primzahl
// keine ist, liegt bei 1/2^200.
    public BigInteger nextPrime(BigInteger pBigInt) {
        if (isEven(pBigInt))
            pBigInt = pBigInt.add(ONE);
        else
            pBigInt = pBigInt.add(TWO);
        // Primzahl mit einer Fehlerwahrscheinlichkeit von 1/2^100
erzeugen
        if (pBigInt.isProbablePrime(100))
            return (pBigInt);
        else
            return (nextPrime(pBigInt));
    } // public static BigInteger nextPrime(BigInteger pBigInt)

// -----
// Überprüfen ob die gegebene Zahl gerade ist
    private static boolean isEven(BigInteger numberToCheck) {
        return (numberToCheck.mod(TWO).equals(BigInteger.ZERO));
    } // private static boolean isEven(BigInteger numberToCheck)
} // public class Prime

```

## GUI.JAVA

```
/*
 * GUI.java
 * Datum: 4/2006
 * Autor: Felix Bruckner
 * Beschreibung:
 * Finale Oberfläche
 */
//-----
// Package
package pmp;

//-----
// Imports
import javax.swing.*;

import java.io.*;
import java.awt.*;
import java.awt.event.*;

//-----
// Klasse GUI
public class GUI extends JPanel {
//-----
// Assoziationen
private PMPMain ctrl;

//-----
// GUI
JFrame frame;

// Objekte+GUI-Elemente
/*
    Gekürzt
*/

//-----
// Konstruktor
public GUI(PMPMain pControl) {

    // LayoutManager deaktivieren
    super.setLayout(null);

    // Assoziation zur Steuerung
    ctrl = pControl;

    // Komponenten der GUI hinzufügen
    addComponents();
}

//-----
// Komponenten der GUI hinzufügen
private void addComponents() {
/*    GUI Elemente - Gekürzt */

//-----
// Überprüfung, ob Schlüssel existieren

    logMsg("PMP " + ctrl.getVersion() + " gestartet.");
```

```

    if (ctrl.readKeys()) {
        lblLoadStatus.setForeground(Color.GREEN);
        lblLoadStatus.setText("Schlüssel erfolgreich geladen.");
        keysLoaded(true);
        logMsg("Schlüssel gefunden und geladen");
    } else {
        lblLoadStatus.setForeground(Color.RED);
        lblLoadStatus
            .setText("Keine Schlüssel gefunden. Bitte zuerst
Schlüsselpaar erzeugen.");
        keysLoaded(false);
        logMsg("Keine Schlüssel gefunden.");
    }
}

//-----
// Schlüsselverzeichnis-Komponenten
private void createDialogComponents() {
    File f = new File(System.getProperty("user.dir")
        + "\\..\\shared\\keys\\");
    File[] fileArray = f.listFiles();
    String[] strArray = new String[fileArray.length];
    for (int i = 0; i < fileArray.length; i++) {
        strArray[i] = fileArray[i].getName();
    }
    /* GUI Elemente - Gekürzt */
}

private void reloadList() {
    // does nothing
}

//-----
// Fenster/Contentpane erstellen + anzeigen
public void createAndShowGUI() {
    // Fenster erstellen
    UIManager.put("swing.boldMetal", Boolean.FALSE);
    try {

        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        e.printStackTrace();
    }
    frame = new JFrame("PMP (Version " + ctrl.getVersion() + ")");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setResizable(false);
    frame.setPreferredSize(new Dimension(806, 614));

    // Contentpane
    JComponent newContentPane = new GUI(ctrl);
    newContentPane.setOpaque(true);
    frame.getContentPane().add(new GUI(ctrl));

    // Hauptfenster genau in der Mitte des Bildschirms öffnen
    GraphicsEnvironment ge = GraphicsEnvironment
        .getLocalGraphicsEnvironment();
    Point center = ge.getCenterPoint();
    Rectangle bounds = ge.getMaximumWindowBounds();

```

```

    int w = Math.max(bounds.width / 2, Math.min(frame.getWidth(),
        bounds.width));
    int h = Math.max(bounds.height / 2, Math.min(frame.getHeight(),
        bounds.height));
    int x = center.x - w / 2, y = center.y - h / 2;
    frame.setBounds(x, y, w, h);
    if (w == bounds.width && h == bounds.height)
        frame.setExtendedState(Frame.MAXIMIZED_BOTH);
    frame.validate();

    // 3 Sek. warten
    try {
        Thread.sleep(1000);
    } catch (Exception e) {
        System.exit(0);
    }

    // Fenster anzeigen
    frame.pack();
    frame.setVisible(true);
}

//-----
// Überprüfen ob Schlüsselpaar existiert
public void keysLoaded(boolean pSuccess) {
    if (pSuccess) {
        tabbedEnc.setEnabled(true);
        tabbedDec.setEnabled(true);
        btnEncOk.setEnabled(true);
        btnDecOk.setEnabled(true);
    } else {
        tabbedEnc.setEnabled(false);
        tabbedDec.setEnabled(false);
        btnEncOk.setEnabled(false);
        btnDecOk.setEnabled(false);
    }
}

//-----
// Methode um Bilder zu laden
protected static ImageIcon createImageIcon(String path) {
    java.net.URL imgURL = GUI.class.getResource(path);
    if (imgURL != null) {
        return new ImageIcon(imgURL);
    } else {
        return null;
    }
}

//-----
// ActionListener
private void addButtonListener(JButton b) {
    b.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
            perform(ae.getSource());
        }
    });
}

//-----

```

```

// Fehler-Dialog
public void errDlg(String pMsg) {
    JOptionPane.showMessageDialog(this, pMsg, "Problem",
        JOptionPane.ERROR_MESSAGE);
    return;
}

//-----
// Info-Dialog
public void infoDlg(String pMsg) {
    JOptionPane.showMessageDialog(this, pMsg, "Hinweis",
        JOptionPane.INFORMATION_MESSAGE);
    return;
}

//-----
// Log
private void logMsg(String pMsg) {
    taLog.append(pMsg + "\n");
    taLog.selectAll();
    taLog.setCaretPosition(taLog.getDocument().getLength());
}

//-----
// GUI-Operationen (-> Button-Klicks) werden hier verarbeitet
public void perform(Object pObj) {
    //-----
    // Schlüssel erzeugen
    if (pObj == btnGenKeyPair) {
        System.out.print(taGKInputText.getText().length()+"\n");
        if((taGKInputText.getText().length())>256){
            errDlg("Maximal 256 Zeichen erlaubt.");
        } else {
            long time = System.currentTimeMillis();
            ctrl.generateKeyPair(taGKInputText.getText());
            taPrivateKey.setText(ctrl.getPrivateKey());
            taPublicKey.setText(ctrl.getPublicKey());
            ctrl.storeKeys();
            keysLoaded(true);
            infoDlg("Schlüssel wurden innerhalb von "
                + ((double) (System.currentTimeMillis() -
time) / 1000)
                + " Sek. erzeugt und können in den jeweiligen
Tabs abgerufen werden!");
            logMsg("Schlüssel erzeugt und gespeichert.");
            taPubKey.setText(ctrl.getPublicKey());
        }
    }
}

//-----
// Log an-/ausschalten
else if (pObj == btnLog) {
    if (taLog.isVisible()) {
        taLog.setVisible(false);
    } else {
        taLog.setVisible(true);
    }
}

//-----
// Schlüssel-Verzeichnis

```

```

else if (pObj == btnKeyDirectory) {
    GraphicsEnvironment ge = GraphicsEnvironment
        .getLocalGraphicsEnvironment();
    Point center = ge.getCenterPoint();
    Rectangle bounds = ge.getMaximumWindowBounds();

    dialog = new JDialog(frame, "Schlüsselverzeichnis", true);
    dialog.setLayout(new GridLayout(1, 1));
    int w = Math.max(bounds.width / 2, Math.min(dialog.getWidth(),
        bounds.width));
    int h = Math.max(bounds.height / 2,
Math.min(dialog.getHeight(),
        bounds.height));
    int x = center.x - w / 2, y = center.y - h / 2;
    dialog.setBounds(x + 100, y + 70, 480, 400);
    dialog.setResizable(false);

    dialog.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
    keypane = new ImagePanel(new
ImageIcon("data/img/iface_bg.png")
        .getImage());
    keypane.setPreferredSize(new Dimension(550, 500));
    keypane.setVisible(true);
    dialog.add(keypane);
    createDialogComponents();
    dialog.setVisible(true);
} else if (pObj == btnKeyDirClose) {
    dialog.dispose();
} else if (pObj == btnKeyDirReload) {
    reloadList();
}

else if (pObj == btnKeyDirSaveOwnKey) {
    String str = "";
    str = JOptionPane
        .showInputDialog(
            this,
            "Geben Sie ihren Namen oder einen Alias
ein.\nUnter diesem Namen wird der Schlüssel veröffentlicht.\nAchtung:
Bereits vorhandene Schlüssel werden überschrieben",
            "Schlüssel veröffentlichen",
            JOptionPane.INFORMATION_MESSAGE);

    if (str.equals(""))
        errDlg("Konnte Schlüssel nicht veröffentlichen:\nKein
Name wurde eingegeben");
    else {
        ctrl.publishOwnKey(str, taPubKey.getText());
        infoDlg("Schlüssel wurde unter dem Namen '" + str
            + "' veröffentlicht!");
        dialog.dispose();
    }
} else if (pObj == btnKeyDirApply) {
    taPubKey2.setText(ctrl.readLine(System.getProperty("user.dir")
        + "\\..\\shared\\keys\\")
        + keyDir.getSelectedValue().toString());
    taPubKey.setText(ctrl.readLine(System.getProperty("user.dir")
        + "\\..\\shared\\keys\\")
        + keyDir.getSelectedValue().toString());
}
}
//-----

```

```

// Text verschlüsseln
    else if (pObj == btnEncOk) {
        if (taPubKey.getText().equals("")) {
            errDlg("Kein öffentlicher Schlüssel angegeben");
        } else {
            if ((taEncText.getText().length()) > 256) {
                errDlg("Maximal 256 Zeichen erlaubt.");
            } else {
                long time = System.currentTimeMillis();
                ctrl.setPublicKey(taPubKey.getText());

                taEncText.setText(ctrl.encryptText(taEncText.getText()));
                logMsg("Text wurde innerhalb von "
                    + ((double) (System.currentTimeMillis()
- time) / 1000)
                    + " Sek. verschlüsselt.");
                taDecText.setText(taEncText.getText());
            }
        }
        ctrl.readKeys();
    }

// Text entschlüsseln
    else if (pObj == btnDecOk) {
        taDecText.setText(ctrl.decryptText(taDecText.getText()));
        taEncText.setText(taDecText.getText());
        logMsg("Text wurde entschlüsselt.");
    }

//-----
// Datei verschlüsseln

    else if (pObj == btnEncOkFile) {
        if (taPubKey2.getText().equals("")) {
            errDlg("Kein öffentlicher Schlüssel angegeben");
        } else {
            long time = System.currentTimeMillis();
            ctrl.setPublicKey(taPubKey2.getText());
            if (ctrl.encryptFile(tfInputFileE.getText(),
tfOutputFileE
                .getText())) {
                infoDlg("Die Datei wurde innerhalb von "
                    + ((double) System.currentTimeMillis()
- time) / 1000)
                    + " Sek. erfolgreich verschlüsselt!");
            }
        }
        ctrl.readKeys();
    }

// Datei öffnen
    else if (pObj == btnOpenFile) {
        JFileChooser openFile = new JFileChooser();
        openFile.setCurrentDirectory(new File(System
            .getProperty("user.dir")
            + "\\test\\"));
        openFile.setFileFilter(new javax.swing.filechooser.FileFilter()
{
            public boolean accept(File f) {
                filename = f.getName();
            }
        });
    }
}

```

```

        return filename.endsWith(".txt")
            || filename.endsWith(".TXT")
            || filename.endsWith(".rtf")
            || filename.endsWith(".RTF") ||
f.isDirectory();
    }

    public String getDescription() {
        return "Text-Dokumente (*.txt, *.rtf)";
    }
});
openFile.showOpenDialog(this);
tfInputFileE.setText(openFile.getSelectedFile().getPath());
tfOutputFileE.setText(tfInputFileE.getText() + ".enc");
btnSaveFile.setEnabled(true);
}

// Datei schließen
else if (pObj == btnSaveFile) {
    JFileChooser saveFile = new JFileChooser();
    saveFile.setCurrentDirectory(new File(tfInputFileE.getText()));
    saveFile.setFileFilter(new javax.swing.filechooser.FileFilter()
{
        public boolean accept(File f) {
            filename = f.getName();
            return filename.endsWith(".enc") ||
f.isDirectory();
        }

        public String getDescription() {
            return "Verschlüsselte Datei (*.enc)";
        }
    });
    saveFile.showSaveDialog(this);
    tfOutputFileD.setText(saveFile.getSelectedFile().getPath());
}

//-----
// Dateien entschlüsseln

// Datei öffnen
else if (pObj == btnOpenFileD) {
    JFileChooser openFile = new JFileChooser();
    openFile.setCurrentDirectory(new File(System
        .getProperty("user.dir")
        + "\\test\\"));
    openFile.setFileFilter(new javax.swing.filechooser.FileFilter()
{
        public boolean accept(File f) {
            filename = f.getName();
            return filename.endsWith(".enc") ||
f.isDirectory();
        }

        public String getDescription() {
            return "Verschlüsselte Datei (*.enc)";
        }
    });
    openFile.showOpenDialog(this);
    tfInputFileD.setText(openFile.getSelectedFile().getPath());
    tfOutputFileD.setText(openFile.getSelectedFile().getPath())

```

```
        .substring(0,
openFile.getSelectedFile().getPath().length() - 4));
        btnSaveFileD.setEnabled(true);
    }

    // Datei speichern
    else if (pObj == btnSaveFileD) {
        JFileChooser saveFile = new JFileChooser();
        saveFile.setCurrentDirectory(new File(tfInputFileD.getText()));
        saveFile.showSaveDialog(this);
        tfOutputFileD.setText(saveFile.getSelectedFile().getPath());
    }

    // Datei entschlüsseln
    else if (pObj == btnDecOkFile) {
        if (ctrl.decryptFile(tfInputFileD.getText(), tfOutputFileD
            .getText())) {
            infoDlg("Die Datei wurde erfolgreich entschlüsselt!");
        } //endif
    } // endif
} // public void perform(Object pObj)
} // public class GUI
//eof
```

## IMAGEPANEL.JAVA

```
/*
 * ImagePanel.java
 * Datum: 4/2007
 * Autor: Felix Bruckner
 * Beschreibung:
 * Hilfsklasse für JPanels mit Hintergrundbildern
 *
 * Klasse übernommen von
 * http://www.java2s.com/Code/Java/Swing-JFC/Panelwithbackgroundimage.htm
 */
package pmp;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Image;

import javax.swing.ImageIcon;
import javax.swing.JPanel;

class ImagePanel extends JPanel {

    private Image img;

    public ImagePanel(final String img) {
        this(new ImageIcon(img).getImage());
    }

    public ImagePanel(final Image img) {
        this.img = img;

        final Dimension size = new Dimension(img.getWidth(null),
img.getHeight(null));
        setMinimumSize(size);
        setMaximumSize(size);
        setSize(size);
        setLayout(null);
    }

    public void paintComponent(final Graphics g) {
        g.drawImage(img, 0, 0, null);
    }
}
```

## SPLASH.JAVA

```
/*
 * Splash.java
 * Datum: 4/2007
 * Autor: Felix Bruckner
 * Beschreibung:
 * Zeigt einen Splashscreen mit Versionsinformationen etc an.
 *
 */
package pmp;

import java.awt.Color;
```

```
import java.awt.Graphics;
import java.awt.Image;
import java.awt.color.*;
import javax.swing.JWindow;

public class Splash extends JWindow implements Runnable {
    private PMPMain ctrl;

    public Splash(PMPMain pControl) {
        ctrl = pControl;
    }

    public void run() {
        setSize(320, 180);
        setLocationRelativeTo(null);
        setVisible(true);

        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            dispose();
        }
        dispose();
    }

    public void paint(Graphics g) {
        Image splashImage = getToolkit().getImage(
            System.getProperty("user.dir") +
            "\\data\\img\\splash.png");
        g.drawImage(splashImage, 0, 0, this);
        g.setColor(new Color(255, 255, 255));
        g.drawString("Initialisieren...", 10, 165);
        g.setColor(new Color(100, 100, 100));
        g.drawString("PMP Version" + ctrl.getVersion(), 210, 15);
    }
}
```

## GLOSSAR

---

#

### 3DES

Andere Bezeichnung für →Triple DES.

---

A

### AES

Steht für Advanced Encryption Standard und wurde 2000 als Nachfolger von →3DES bzw →DES eingeführt. Der verwendete Algorithmus →Rijndael besitzt variable Blockgrößen und Schlüssellängen von 128 bis 256 Bit, was ein sehr hohes Maß an Sicherheit gewährt<sup>55</sup>

### ASYMMETRISCHES VERFAHREN

Als asymmetrisches Verfahren (auch Public-Key-Verfahren) bezeichnet man Kryptosysteme, in welchen der Sender als auch Empfänger einen öffentlichen (→Public Key) und privaten Schlüssel (→Private Key) besitzen. Dies erzeugt mehr Sicherheit als bei →symmetrischen Verfahren. Siehe hierzu auch das Kapitel: Hauptthema RSA und PGP.<sup>56</sup>

---

B

### BLOCKCHIFFRE

---

<sup>55</sup> Vgl.: [http://de.wikipedia.org/wiki/Advanced\\_Encryption\\_Standard](http://de.wikipedia.org/wiki/Advanced_Encryption_Standard)

<sup>56</sup> Vgl.: <http://www.philippbauer.de/info/info/asymmetrische-verschlueselung/>

Eine Blockchiffre ist ein →symmetrisches Verfahren zur Ver- und Entschlüsselung. Jeweils gleichlange (z.B. 64 Bit) Blöcke an Daten werden hierbei mit einem festen →Schlüssel verschlüsselt. Meist wird als Blocklänge 64 Bit gewählt, um eine Dechiffrierung zu erschweren. <sup>57</sup>

## D

---

### DES

Der Data Encryption Standard ist ein sehr weitverbreiteter →symmetrischer Verschlüsselungsalgorithmus. Im Jahre 1976 wurde er offizieller Standard für die US-Regierung. Siehe hierzu das Kapitel Die Vorläufer von RSA und PGP

## E

---

### EINWEGFUNKTION

Unter einer Einwegfunktion versteht man eine Funktion, die man durchführen, jedoch nur sehr schwierig wieder rückgängig machen kann.

## F

---

### FIAT-SHAMIR-VERFAHREN

Das Fiat-Shamir-Verfahren oder auch Fiat-Shamir-Protokoll ist ein Verfahren zur Authentisierung. Es wird in aller Kürze im Kapitel Zero-Knowledge – Die Mathematik hinter Zero-Knowledge beschrieben.

### FINGERPRINT

Siehe PGP - Schlüsselverwaltung

## G

---

### GEHEIME KANÄLE

---

<sup>57</sup> Vgl.:

<http://www.cryptoshop.com/de/knowledgebase/cryptography/symmetric/blockcipher.php>

Bei „geheimen Kanälen“ unterscheidet man zwischen physikalisch (technisch abhörsichere Übertragung) oder organisatorisch (Überbringung durch einen vertrauenswürdigen Boten) geheimen Kanälen.<sup>58</sup>

## H

---

### HTTPS

**H**yper**T**ext **T**ransfer **P**rotocol **S**ecure wurde von Netscape entwickelt und dient zur Verschlüsselung von übertragenen Daten zwischen Browser und WWW. Als Verfahren wird SSL eingesetzt, welches mithilfe von symmetrischen Verfahren wie →DES, →Triple DES und →AES Datenpakete verschlüsselt.

## I

---

### IDEA

Der International Data Encryption Algorithm wurde 1992 veröffentlicht und besitzt eine Schlüssellänge von 128 Bit. IDEA ist von ein einer Schweizer Firma patentiert und somit nicht frei verfügbar. Eine genaue Beschreibung befindet sich im Kapitel Nachfolger - IDEA .<sup>59</sup>

## M

---

### MODUL-ARITHMETIK

siehe RSA –Modulare Arithmetik

## O

---

### ÖFFENTLICHE KANÄLE

Die öffentlichen Kanäle sind das Gegenteil von den Geheimen Kanälen, sprich nicht abhörgeschützt.

## P

---

---

<sup>58</sup> Vgl.: [BEU 1]

<sup>59</sup> Vgl.: [http://www.regenechsen.de/phpwcms/index.php?krypto\\_idea](http://www.regenechsen.de/phpwcms/index.php?krypto_idea)

## PRIVATE KEY

Ein privater (geheimer) Schlüssel wird in →asymmetrischen Verfahren eingesetzt und bezeichnet den →Schlüssel, der nur der Person bekannt ist, die Daten an andere verschlüsselt. Siehe hierzu das Kapitel: Hauptthema RSA und PGP.

## PROTOKOLL

Als Protokoll bezeichnet man einen vorab, nach bestimmten Regeln ausgemachten Ablauf, Vorgang oder Prozess, den man mithilfe von bestimmten Informationen durchführen kann.<sup>60</sup>

## PUBLIC KEY

Der Public Key (öffentlicher Schlüssel) ist der Schlüssel eines Schlüsselpaares, welcher jedem zugänglich gemacht werden kann (muss), damit andere Personen eine Nachricht mithilfe von →asymmetrischen Verfahren an den Besitzer des Public Keys verschlüsseln können. Siehe hierzu das Kapitel: Hauptthema RSA und PGP

## PUBLIC-KEY-VERFAHREN

Siehe →asymmetrisches Verfahren

## R

---

### RC2

Ein von Ronald Rivest entwickeltes Verfahren, welches →Blockchiffrierung mit variabler Schlüssellänge einsetzt. Entwickelt wurde es als möglicher Ersatz für →DES.<sup>61</sup>

### RC4

Wurde 1987 von Ronald Rivest als Nachfolger zu →RC2 entwickelt. Das Verfahren verwendet eine einfach →Stromchiffre, welche noch heute für Internetstandards wie →WEP/WPA, →HTTPS, →SSH eingesetzt wird.<sup>62</sup>

---

<sup>60</sup> Vgl.: <http://de.wikipedia.org/wiki/Protokoll>

<sup>61</sup> Vgl.: <http://tools.ietf.org/html/rfc2268>

## RIJNDAEL

Rijndael ist ein →Blockchiffre welcher im →AES zum Einsatz kommt.  
Vergleiche hierzu das Kapitel Nachfolger - Rijndael

## RIVEST CIPHER

Andere Bezeichnung für →RC2, →RC4

## RON'S CODE

Andere Bezeichnung für →RC2, →RC4

## RSA

Siehe Kapitel: Hauptthema: PGP und RSA.

## S

---

## SCHLÜSSEL

Ein Schlüssel in der Kryptologie bezeichnet eine Zeichenfolge, mit der man Daten verschlüsseln und wieder entschlüsseln kann. Dabei wird von öffentlichen Schlüsseln (→Public Key) und geheimen Schlüsseln (→Private Key) unterschieden.

## SSH

Die **Secure Shell** eine Möglichkeit sich auf UNIX-basierenden Systemen entfernt einzuloggen und es zu administrieren bzw. Programme auszuführen. Ein SSH Server und Client sind auf jedem UNIX / POSIX / Linux-System bereits vorinstalliert. Bemerkenswert ist der große Umfang an Verschlüsselungsmöglichkeiten: Für übertragene Passwörter wird auf →RSA und →RC4 gesetzt. Des Weiteren unterstützt SSH noch viele weitere Verschlüsselungs-Standards, wie z.B. →AES, →DES, →IDEA und noch einige weitere, weniger verbreitete.<sup>63</sup>

---

<sup>62</sup> Vgl.: <http://www.wisdom.weizmann.ac.il/~itsik/RC4/rc4.html>

<sup>63</sup> Vgl. <http://de.wikipedia.org/wiki/SSH>

## STEGANOGRAPHIE

Mit diesem Begriff bezeichnet man das Verstecken von geheimen Nachrichten in belanglosen anderen.

## STROMCHIFFRE (STREAM CIPHER)

Eine Stromchiffre ist eine →symmetrische, kontinuierliche Ver- und Entschlüsselung eines Datenstroms, welche im Gegensatz zu →Blockchiffren Bit für Bit oder Zeichen für Zeichen ver/entschlüsselt.

## SUBSTITUTION

Damit ist das Verschieben von Buchstaben innerhalb eines Textes gemeint, um so den Text für den Unwissenden unleserlich zu machen.

## SYMMETRISCHES VERFAHREN

Bei symmetrischen Verfahren wird im Gegensatz zu →asymmetrischen nur ein →Schlüssel verwendet um Daten zu ver- und entschlüsseln. Man teilt die symmetrischen Verfahren in →Blockchiffren und →Stromchiffren auf. Bekannte symmetrische Verfahren sind z.B. →AES, →DES, →IDEA. Siehe hierzu auch das Kapitel **Die Vorläufer von RSA und PGP**.

## T

---

## TRANSPOSITION

Bei dieser Verschlüsselungsmethode werden die Buchstaben innerhalb eines Textes nach einem bestimmten Muster vermischt. Somit wird ein Text unleserlich gemacht.

## TRIPLE DES

Eine (komplizierte) Erweiterung des →DES-Algorithmus, welches die Schlüssellänge von 56 auf bis zu 168 Bit erweitert (3 Mal so viel, daher der Name) und wird als fast genauso sicher wie moderne 128 Bit-Verschlüsselungen angesehen, ist jedoch aufgrund des verdreifachten Rechenaufwands viel langsamer.<sup>64</sup>

---

<sup>64</sup> Vgl.: [http://www.regenechsen.de/phpwcms/index.php?krypto\\_des](http://www.regenechsen.de/phpwcms/index.php?krypto_des)

## W

---

### WEP

Wired Equivalent Privacy ist ein veralteter Standard für WLAN. Er wurde von →WPA ersetzt.

### WPA

Wi-Fi Protected Access ist eine Verschlüsselungsmethode für WLAN und heute Standard, nachdem sich →WEP als zu unsicher erwiesen hatte. Das Verfahren basiert auf der →RC4-Stromchiffre.

## Z

---

### ZERO-KNOWLEDGE

Zero Knowledge ist ein kryptografisches Protokoll, welches zur Authentifizierung eingesetzt wird, indem man seinen Kommunikationspartner darauf prüft ob er „Geheimnis“ kennt, ohne möglichst Informationen über das „Geheimnis“ selbst preiszugeben. Dieses Verfahren wird im Kapitel **Zero-Knowledge** genauer behandelt.

## ABBILDUNGSVERZEICHNIS

Alle gekennzeichneten Abbildungen sind eigene Darstellungen, welche nur für diese Arbeit angefertigt wurden.

Abb. 1: Beispielhafte Schlüsselerzeugung mit dem Diffie- Hellman- Schlüsselaustausch

Abb. 2: Typischer Ablauf einer symmetrischen Verschlüsselung

Abb. 3: Typischer Ablauf einer asymmetrischen Verschlüsselung

Abb. 4: Schlüsselerzeugung

Abb. 5: Die Entstehung des Web of Trust

Abb. 6: Die Funktionsweise von PGP

Abb. 7: Ver- und Entschlüsselung mit PGP

## QUELLENANGABEN

### LITERATURVERZEICHNIS

**[WOBS] Wobst**, Reinhard: Abenteuer Kryptologie, Methoden, Risiken und Nutzen der Datenverschlüsselung, 2. überarbeitete Aufl., Bonn; Reading, Massachusetts [u.a.], Adison Wesley, 1998.

**[BEU1] Beutelspacher**, Albrecht; **Schwenk**, Jörg; **Wolfenstetter, Klaus-Dieter**: Moderne Verfahren der Kryptographie, von RSA zu Zero-Knowledge, 5. Aufl., Wiesbaden, Vieweg, 2004.

**[BEU2] Beutelspacher, Albrecht**: Geheimsprachen, Geschichte und Techniken, 4. Aufl., München, Beck, 2005.

**[WELS] Welschenbach**, Michael: Kryptographie in C und C++, 2. Aufl., Berlin, Springer, 2001.

**[BURN] Burnett**, Steve; **Paine**, Stephen: Kryptographie, RSA Security's Official Guide, 1. Aufl., Bonn, mitp, 2001.

**[SING] Singh**, Simon: Geheime Botschaften – Die Kunst der Verschlüsselung von der Antike bis in die Zeiten des Internets, 6. Auflage, München, DTV, 2005.

**[ZIMM] Creutzig**, Christopher; **Buhl**, Andreas; **Zimmermann**, Philipp: PGP, Pretty Good Privacy, der Briefumschlag für Ihre elektronische Post, 4. Aufl., Bielefeld, Art d'Ameublement, 1999.

### WEBSITES

#### Wikipedia:

<http://de.wikipedia.org/> und <http://en.wikipedia.org/>

#### CryptoLounge:

<http://www.kuno-kohn.de/crypto/crypto/>

#### FH Wedel: Kryptographie:

<http://www.fh-wedel.de/~si/seminare/ws96/ausarbeitung/sicherh/sicherh2.htm>

#### Bundesamt für Sicherheit in der Informationstechnik:

[http://www.bsi-fuer-buerger.de/schuetzen/07\\_0301.htm#asym](http://www.bsi-fuer-buerger.de/schuetzen/07_0301.htm#asym)

#### SUN Microsystems:

<http://research.sun.com/people/>

#### Public Encryption for the Masses:

<http://www.elektronikschule.de/~grupp/pgp/>

#### FH Esslingen: Die Geschichte der Kryptologie:

<http://www.it.fht-esslingen.de/~schmidt/vorlesungen/kryptologie/seminar/historie/History.html>

**Universität Bielefeld: Asymmetrische/ moderne Kryptographie :**

[http://www.wiwi.uni-bielefeld.de/StatCompSci/lehre/material\\_spezifisch/statalg00/rsa/rsa.html](http://www.wiwi.uni-bielefeld.de/StatCompSci/lehre/material_spezifisch/statalg00/rsa/rsa.html)

**TU Freiberg:**

[http://www.mathe.tu-freiberg.de/~dempe/schuelerpr\\_neu/hellmann.htm](http://www.mathe.tu-freiberg.de/~dempe/schuelerpr_neu/hellmann.htm)

**Otto-von-Guericke-Universität Magdeburg: Institut für Verteilte Systeme**

<http://www-ivs.cs.uni-magdeburg.de/>

**Living Internet: Public Key Cryptography (PKC) History**

[http://www.livinginternet.com/i/is\\_crypt\\_pkc\\_inv.htm](http://www.livinginternet.com/i/is_crypt_pkc_inv.htm)

**Answers.com**

<http://www.answers.com/>

**AT-mix: RSA**

<http://www.at-mix.de/rsa.htm>

**Hamburger Schulserver: Die Geschichte des RSA-Verfahrens**

<http://www.hh.schule.de/julius-leber-schule/melatob/historyrsa.html>

**Cyberlaw**

<http://www.cyberlaw.com/rsa.html>

**Diplomarbeit: Modulare Arithmetik**

<http://home.datacomm.ch/th.aes/Daten/Html/Modularitmetik.html>

**TU Darmstadt: Seminar 98/99 Verifikation digitaler Signaturen**

[http://www.informatik.tu-darmstadt.de/BS/Lehre/Sem98\\_99/T11/index.html#tth\\_sEc3](http://www.informatik.tu-darmstadt.de/BS/Lehre/Sem98_99/T11/index.html#tth_sEc3)

**Forschungszentrum Jülich: PGP-Schlüsselmanagement**

[http://www.fz-juelich.de/zam/docs/bhb/bhb\\_html/d0141/keymang1.htm](http://www.fz-juelich.de/zam/docs/bhb/bhb_html/d0141/keymang1.htm)

**Weizmann Insitute of Science Israel: Zero-Knowledge: a tutorial by Oded Goldreich**

<http://www.wisdom.weizmann.ac.il/~oded/zk-tut02.html>

**Regenechsen: Ideas come true: IDEA**

[http://www.regenechsen.de/phpwcms/index.php?krypto\\_idea](http://www.regenechsen.de/phpwcms/index.php?krypto_idea)

**CSRC / NIST: AES**

<http://csrc.nist.gov/CryptoToolkit/aes/rijndael/>

**Universität Paderborn: Research Group Algorithmic Mathematics**

<http://www-math.uni-paderborn.de/~aggathen/rijndael/2001/flussvisualisierung/>

**Universität Koblenz: Namenskonventionen**

<http://www.uni-koblenz.de/~daniel/Namenskonventionen.html>

**Cryptoshop: Blockchiffren**

<http://www.cryptoshop.com/de/knowledgebase/cryptography/symmetric/blockcipher.php>

## REFERENZEN, WELCHE ZUR PROGRAMMIERUNG VERWENDET WURDEN:

### **Mersenne Primes: History, Theorems and Lists:**

<http://primes.utm.edu/mersenne/index.html>

### **JSP CoreServlets, Sun Microsystems Press:**

<http://www.coreservlets.com/>

### **Using RSA Encryption with Java:**

<http://www.aviransplace.com/index.php/archives/2004/10/12/using-rsa-encryption-with-java/1/>

### **Real's Java How-To: Convert from type X to type Y:**

<http://www.rgagnon.com/javadetails/java-0004.html>

### **Sun Developers Network:**

<http://java.sun.com/developer/JDCTechTips/2002/tt0806.html> bzw.

<http://java.sun.com/developer/allgemein>

**EHRENWÖRTLICHE ERKLÄRUNG**

Wir erklären hiermit an Eides Statt, dass wir die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt haben.

Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

---

Ort, Datum

---

Unterschrift

---

Ort, Datum

---

Unterschrift