

$2^2 - 1 = 3$ prim

$2^3 - 1 = 7$ prim

$2^5 - 1 = 31$ prim

$2^7 - 1 = 127$ prim

$2^{11} - 1 = 2047 = 23 \cdot 89$

The CrypTool Script

Cryptography, Mathematics and More

Prof. Bernhard Esslinger
and the CrypTool Development Team

10th Edition

Background reading
for the free e-learning program CrypTool
(with number theoretic code samples, using Sage)

The CrypTool Script: Cryptography, Mathematics and More

Background reading
for the free e-learning program CrypTool
(with number theoretic code samples, using Sage)

(10th edition – distributed with CrypTool version 1.4.30)

Copyright (c) Prof. Bernhard Esslinger (co-author and editor)
and the CrypTool Development Team, 1998-2009
Frankfurt am Main, Germany

www.cryptool.org

December 2, 2009

This is a free document, so the content of the document can be copied and distributed, also for commercial purposes — as long as the authors, title and the CrypTool web site (www.cryptool.org) are acknowledged. Naturally, citations from the CrypTool script are possible, as in all other documents.

The specific license for this document is the GNU Free Documentation Licence.

Copyright © 1998–2009 Bernhard Esslinger and the CrypTool Development Team. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

Source cover photograph: www.photocase.com, Andre Guenther

Typesetting software: L^AT_EX

Version control software: Subversion

Overview about the Content of the CrypTool Script

In this *script accompanying the program CrypTool* you will find predominantly mathematically oriented information on using cryptographic procedures. Also included are many sample code pieces written in the computer algebra system **Sage** (see appendix A.5). The main chapters have been written by various **authors** (see appendix A.2) and are therefore independent from one another. At the end of most chapters you will find references and web links.

The first chapter explains the principles of symmetric and asymmetric **encryption** and describes shortly the current decryption records of modern symmetric algorithms.

Because of didactic reasons the second chapter gives an exhaustive overview about **paper and pencil encryption methods**.

Big parts of this script are dedicated to the fascinating topic of **prime numbers** (chap. 3). Using numerous examples, **modular arithmetic** and **elementary number theory** (chap. 4) are introduced and applied in an exemplary manner for the **RSA procedure**.

By reading chapter 5 you'll gain an insight into the mathematical ideas and concepts behind **modern cryptography**.

Chapter 6 gives an overview about the status of attacks against modern **hash algorithms** and is then shortly devoted to **digital signatures**, which are an essential component of e-business applications.

Chapter 7 describes **elliptic curves**: they could be used as an alternative to RSA and in addition are extremely well suited for implementation on smartcards.

The last chapter **Crypto2020** discusses threats for existing cryptographic methods and introduces alternative research approaches to achieve long-term security of cryptographic schemes.

Whereas the *e-learning program* CrypTool motivates and teaches you how to use cryptography in practice, the *script* provides those interested in the subject with a deeper understanding of the mathematical algorithms used – trying to do it in an instructive way. If you are already a little bit familiar with this field of knowledge you can gain a fast overview about the functions delivered by CrypTool looking at the **menu tree** (see appendix A.1).

The authors would like to take this opportunity to thank their colleagues in the company and at the universities of Frankfurt, Gießen, Siegen, Karlsruhe and Darmstadt.

As with the e-learning program CrypTool, the quality of the script is enhanced by your suggestions and ideas for improvement. We look forward to your feedback.

Contents Overview

Overview	ii
Preface to the 10th Edition of the CrypTool Script	x
Introduction – How do the Script and the Program Play together?	xi
1 Encryption Procedures	1
2 Paper and Pencil Encryption Methods	14
3 Prime Numbers	50
4 Introduction to Elementary Number Theory with Examples	86
5 The Mathematical Ideas behind Modern Cryptography	166
6 Hash Functions and Digital Signatures	179
7 Elliptic Curves	187
8 Crypto 2020 — Perspectives for Long-Term Cryptographic Security	206
A Appendix	211
GNU Free Documentation License	240
List of Figures	248
List of Tables	249
List of Crypto Procedures	251
List of Sage Code Examples	252
Index	254

Contents

Overview	ii
Preface to the 10th Edition of the CrypTool Script	x
Introduction – How do the Script and the Program Play together?	xi
1 Encryption Procedures	1
1.1 Symmetric encryption	2
1.1.1 New results about cryptanalysis of AES	2
1.1.2 Current status of brute-force attacks on symmetric algorithms (RC5)	4
1.2 Asymmetric encryption	5
1.3 Hybrid procedures	6
1.4 Ciphers and cryptanalysis for educational purposes	7
1.5 Further details	7
1.6 Appendix: Examples using Sage	8
1.6.1 Mini-AES	8
Bibliography	10
Web links	13
2 Paper and Pencil Encryption Methods	14
2.1 Transposition ciphers	15
2.1.1 Introductory samples of different transposition ciphers	15
2.1.2 Column and row transposition ciphers	16
2.1.3 Further transposition algorithm ciphers	17
2.2 Substitution ciphers	19
2.2.1 Monoalphabetic substitution ciphers	19
2.2.2 Homophonic substitution ciphers	22
2.2.3 Polygraphic substitution ciphers	23
2.2.4 Polyalphabetic substitution ciphers	25
2.3 Combining substitution and transposition	26
2.4 Further methods	29

2.5	Appendix: Examples using Sage	32
2.5.1	Transposition ciphers	33
2.5.2	Substitution ciphers	37
2.5.3	Caesar cipher	38
2.5.4	Shift cipher	40
2.5.5	Affine cipher	41
2.5.6	Substitution with symbols	43
2.5.7	Vigenère cipher	45
2.5.8	Hill cipher	46
	Bibliography	48
3	Prime Numbers	50
3.1	What are prime numbers?	50
3.2	Prime numbers in mathematics	51
3.3	How many prime numbers are there?	52
3.4	The search for extremely large primes	53
3.4.1	The 20+ largest known primes (as of July 2009)	53
3.4.2	Special number types – Mersenne numbers and Mersenne primes	55
3.4.3	Challenge of the Electronic Frontier Foundation (EFF)	58
3.5	Prime number tests	59
3.6	Overview special number types and the search for a formula for primes	61
3.6.1	Mersenne numbers $f(n) = 2^n - 1$ for n prime	61
3.6.2	Generalized Mersenne numbers $f(k, n) = k \cdot 2^n \pm 1$ / Proth numbers	61
3.6.3	Generalized Mersenne numbers $f(b, n) = b^n \pm 1$ / Cunningham project	62
3.6.4	Fermat numbers $f(n) = 2^{2^n} + 1$	62
3.6.5	Generalized Fermat numbers $f(b, n) = b^{2^n} + 1$	63
3.6.6	Carmichael numbers	63
3.6.7	Pseudo prime numbers	63
3.6.8	Strong pseudo prime numbers	63
3.6.9	Idea based on Euclid's proof $p_1 \cdot p_2 \cdots p_n + 1$	64
3.6.10	As above but -1 except $+1$: $p_1 \cdot p_2 \cdots p_n - 1$	64
3.6.11	Euclidean numbers $e_n = e_0 \cdot e_1 \cdots e_{n-1} + 1$	64
3.6.12	$f(n) = n^2 + n + 41$	65
3.6.13	$f(n) = n^2 - 79 \cdot n + 1, 601$	65
3.6.14	Polynomial functions $f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0$	66
3.6.15	Catalan's conjecture	67
3.7	Density and distribution of the primes	67
3.8	Notes about primes	70

3.8.1	Proven statements / theorems about primes	70
3.8.2	Unproven statements / conjectures about primes	72
3.8.3	Open questions about twin primes	73
3.8.4	Further open questions	74
3.8.5	Quaint and interesting things around primes	74
3.9	Appendix: Number of prime numbers in various intervals	77
3.10	Appendix: Indexing prime numbers (n -th prime number)	78
3.11	Appendix: Orders of magnitude / dimensions in reality	79
3.12	Appendix: Special values of the binary and decimal system	80
3.13	Appendix: Examples using Sage	81
3.13.1	Check primality of integers generated by quadratic functions	81
	Bibliography	83
	Web links	85
	Acknowledgments	85
4	Introduction to Elementary Number Theory with Examples	86
4.1	Mathematics and cryptography	86
4.2	Introduction to number theory	87
4.2.1	Convention	89
4.3	Prime numbers and the first fundamental theorem of elementary number theory	90
4.4	Divisibility, modulus and remainder classes	92
4.4.1	The modulo operation – working with congruence	92
4.5	Calculations with finite sets	94
4.5.1	Laws of modular calculations	94
4.5.2	Patterns and structures	95
4.6	Examples of modular calculations	96
4.6.1	Addition and multiplication	96
4.6.2	Additive and multiplicative inverses	97
4.6.3	Raising to the power	100
4.6.4	Fast calculation of high powers	101
4.6.5	Roots and logarithms	102
4.7	Groups and modular arithmetic in \mathbb{Z}_n and \mathbb{Z}_n^*	102
4.7.1	Addition in a group	103
4.7.2	Multiplication in a group	103
4.8	Euler function, Fermat's little theorem and Euler-Fermat	105
4.8.1	Patterns and structures	105
4.8.2	The Euler function	105
4.8.3	The theorem of Euler-Fermat	106

4.8.4	Calculation of the multiplicative inverse	106
4.8.5	Fixpoints modulo 26	107
4.9	Multiplicative order and primitive roots	108
4.10	Proof of the RSA procedure with Euler-Fermat	111
4.10.1	Basic idea of public key cryptography	111
4.10.2	How the RSA procedure works	112
4.10.3	Proof of requirement 1 (invertibility)	113
4.11	Considerations regarding the security of the RSA algorithm	114
4.11.1	Complexity	115
4.11.2	Security parameters because of new algorithms	115
4.11.3	Forecasts about factorization of large integers	116
4.11.4	Status regarding factorization of concrete large numbers	119
4.11.5	Further current research about primes and factorization	124
4.12	Applications of asymmetric cryptography using numerical examples	127
4.12.1	One way functions	127
4.12.2	The Diffie-Hellman key exchange protocol	128
4.13	The RSA procedure with actual numbers	131
4.13.1	RSA with small prime numbers and with a number as message	131
4.13.2	RSA with slightly larger primes and an upper-case message	131
4.13.3	RSA with even larger primes and a text made up of ASCII characters	132
4.13.4	A small RSA cipher challenge (1)	134
4.13.5	A small RSA cipher challenge (2)	135
4.14	Appendix: gcd and the two algorithms of Euclid	138
4.15	Appendix: Forming closed sets	140
4.16	Appendix: Comments on modulo subtraction	140
4.17	Appendix: Base representation of numbers, estimation of length of digits	141
4.18	Appendix: Examples using Sage	143
4.18.1	Multiplication table modulo m	143
4.18.2	Fast exponentiation	143
4.18.3	Multiplicative order	144
4.18.4	Primitive roots	147
4.18.5	RSA examples with Sage	157
4.18.6	How many RSA keys exist within a given modulo range?	158
4.19	Appendix: List of the definitions and theorems formulated in this chapter	160
	Bibliography	161
	Web links	164
	Acknowledgments	165

5	The Mathematical Ideas behind Modern Cryptography	166
5.1	One way functions with trapdoor and complexity classes	166
5.2	Knapsack problem as a basis for public key procedures	168
5.2.1	Knapsack problem	168
5.2.2	Merkle-Hellman knapsack encryption	169
5.3	Decomposition into prime factors as a basis for public key procedures	170
5.3.1	The RSA procedure	170
5.3.2	Rabin public key procedure (1979)	172
5.4	The discrete logarithm as basis for public key procedures	172
5.4.1	The discrete logarithm in \mathbb{Z}_p	172
5.4.2	Diffie-Hellman key agreement	173
5.4.3	ElGamal public key encryption procedure in \mathbb{Z}_p^*	174
5.4.4	Generalised ElGamal public key encryption procedure	174
	Bibliography	177
	Web links	178
6	Hash Functions and Digital Signatures	179
6.1	Hash functions	180
6.1.1	Requirements for hash functions	180
6.1.2	Current attacks against hash functions like SHA-1	181
6.1.3	Signing with hash functions	182
6.2	RSA signatures	182
6.3	DSA signatures	183
6.4	Public key certification	183
6.4.1	Impersonation attacks	184
6.4.2	X.509 certificate	184
	Bibliography	185
7	Elliptic Curves	187
7.1	Elliptic curve cryptography – a high-performance substitute for RSA?	187
7.2	Elliptic curves – history	188
7.3	Elliptic curves – mathematical basics	190
7.3.1	Groups	190
7.3.2	Fields	191
7.4	Elliptic curves in cryptography	192
7.5	Operating on the elliptic curve	194
7.6	Security of elliptic-curve-cryptography: The ECDLP	197
7.7	Encryption and signing with elliptic curves	198
7.7.1	Encryption	198

7.7.2	Signing	199
7.7.3	Signature verification	199
7.8	Factorization using elliptic curves	199
7.9	Implementing elliptic curves for educational purposes	201
7.9.1	CrypTool	201
7.9.2	Sage	201
7.10	Patent aspects	203
7.11	Elliptic curves in use	203
	Bibliography	204
	Web links	205
8	Crypto 2020 — Perspectives for Long-Term Cryptographic Security	206
8.1	Widely used schemes	206
8.2	Preparation for tomorrow	207
8.3	New mathematical problems	208
8.4	New signatures	209
8.5	Quantum cryptography – a way out of the impasse?	209
8.6	Conclusion	209
	Bibliography	210
A	Appendix	211
A.1	CrypTool Menus	212
A.2	Authors of the CrypTool Script	214
A.3	Movies and Fictional Literature with Relation to Cryptography, Books for Kids with Simple Ciphers	216
A.3.1	For Grownups	216
A.3.2	For Kids	223
A.4	Learning Tool for Elementary Number Theory	225
A.5	Using Sage with this Script	231
	GNU Free Documentation License	240
	List of Figures	248
	List of Tables	249
	List of Crypto Procedures	251
	List of Sage Code Examples	252
	Index	254

Preface to the 10th Edition of the CrypTool Script

Starting in the year 2000 this script became part of the CrypTool v1 package. It is designed to accompany the program CrypTool by explaining some mathematical topics in more detail, but still in a way which is easy to understand.

In order to also enable developers/authors to work together independently the topics have been split up and for each topic an extra chapter has been written which can be read on its own. The later editorial work in TeX added cross linkages between different sections and footnotes describing where you can find the according functions within the CrypTool v1 program (see menu tree in appendix A.1). Naturally there are many more interesting topics in mathematics and cryptography which could be discussed in greater depth – therefore this is only one of many ways to do it.

The rapid spread of the Internet has lead to intensified research in the technologies involved, especially within the area of cryptography where a good deal of new knowledge has arisen.

This edition completely updated the TeX sources of the document, and of course the content of the script was corrected, amended and updated with some topics, e.g.:

- the search for the largest prime numbers (chap. 3.4),
- progress in cryptanalysis of hash algorithms (chap. 1.1.1) and
- the list of movies or novels, in which cryptography or number theory played major role (see appendix A.3); and where primes are used as hangers (see curioses in 3.8.5).
- Newly added is the appendix A.5 about using the computer algebra system Sage. This was added because Sage becomes more and more the standard open-source CAS system. Accordingly all samples written before in Pari-GP and Mathematica have been substituted with Sage code. Thanks to Nguyen and Massierer, a lot of new code samples could be added (see also chap. 7.9.2).

The first time the document was delivered with CrypTool was in version 1.2.01. Since then it has been expanded and revised in almost every new version of CrypTool.

I am deeply grateful to all the people helping with their impressive commitment who have made this global project so successful. Thanks also to the readers who sent us feedback.

I hope that many readers have fun with this script and that they get out of it more interest and greater understanding of this modern but also very ancient topic.

Bernhard Esslinger

Frankfurt (Germany), December 2009

Introduction – How do the Script and the Program Play together?

This script

This document is delivered together with the open-source program CrypTool.

The articles in this script are largely self-contained and can also be read independently of CrypTool.

Chapters 5 (Modern Cryptography) and 7 (Elliptic Curves) require a deeper knowledge in mathematics, while the other chapters should be understandable with a school leaving certificate.

The authors have attempted to describe cryptography for a broad audience – without being mathematically incorrect. We believe that this didactic pretension is the best way to promote the awareness for IT security and the readiness to use standardized modern cryptography.

The program CrypTool

CrypTool is an educational program with a comprehensive online help enabling you to use and analyse cryptographic procedures within a unified graphical user interface.

CrypTool is used world-wide for training in companies and teaching at schools and universities worldwide, and several universities are helping to further develop the project.

Acknowledgment

At this point I'd like to thank explicitly the following people who particularly contributed to CrypTool. They applied their very special talents and showed really great engagement:

- Mr. Henrik Koy
- Mr. Jörg-Cornelius Schneider
- Mr. Florian Marchal
- Dr. Peer Wichmann
- Staff of Prof. Claudia Eckert, Prof. Johannes Buchmann and Prof. Torben Weis.

Also I want to thank all the many people not mentioned here for their hard work (mostly carried out in their spare time).

Bernhard Esslinger

Frankfurt (Germany), November 2009

Chapter 1

Encryption Procedures

(Bernhard Esslinger, Joerg-Cornelius Schneider, May 1999; Updates Dec. 2001, Feb. 2003, June 2005, July 2007, November 2009)

Saying from India:

Explain it to me, I will forget it.

Show it to me, maybe I will remember it.

Let me do it, and I will be good at it.

This chapter introduces the topic in a more descriptive way without using too much mathematics.

The purpose of encryption is to change data in such a way that only an authorized recipient is able to reconstruct the plaintext. This allows us to transmit data without worrying about it getting into unauthorized hands. Authorized recipients possess a piece of secret information – called the key – which allows them to decrypt the data while it remains hidden from everyone else.

One encryption procedure has been mathematically proved to be secure, the *One Time Pad*. However, this procedure has several practical disadvantages (the key used must be randomly selected and must be at least as long as the message being protected), which means that it is hardly used except in closed environments such as for the hot wire between Moscow and Washington.

For all other procedures there is a (theoretical) possibility of breaking them. If the procedures are good, however, the time taken to break them is so long that it is practically impossible to do so, and these procedures can therefore be considered (practically) secure.

The book of Bruce Schneier [Schneier1996] offers a very good overview of the different algorithms. We basically distinguish between symmetric and asymmetric encryption procedures.

1.1 Symmetric encryption¹

For *symmetric* encryption sender and recipient must be in possession of a common (secret) key which they have exchanged before actually starting to communicate. The sender uses this key to encrypt the message and the recipient uses it to decrypt it.

All classical methods are of this type. Examples can be found within the program CrypTool, in chapter 2 (“Paper and Pencil Encryption Methods”) of this script or in [Nichols1996]. Now we want to consider more modern mechanisms.

The advantages of symmetric algorithms are the high speed with which data can be encrypted and decrypted. One disadvantage is the need for key management. In order to communicate with one another confidentially, sender and recipient must have exchanged a key using a secure channel before actually starting to communicate. Spontaneous communication between individuals who have never met therefore seems virtually impossible. If everyone wants to communicate with everyone else spontaneously at any time in a network of n subscribers, each subscriber must have previously exchanged a key with each of the other $n - 1$ subscribers. A total of $n(n - 1)/2$ keys must therefore be exchanged.

The most well-known symmetric encryption procedure is the DES-algorithm. The DES-algorithm has been developed by IBM in collaboration with the National Security Agency (NSA), and was published as a standard in 1975. Despite the fact that the procedure is relatively old, no effective attack on it has yet been detected. The most effective way of attacking consists of testing (almost) all possible keys until the right one is found (*brute-force-attack*). Due to the relatively short key length of effectively 56 bits (64 bits, which however include 8 parity bits), numerous messages encrypted using DES have in the past been broken. Therefore, the procedure can now only be considered to be conditionally secure. Symmetric alternatives to the DES procedure include the IDEA or Triple DES algorithms.

Up-to-the-minute procedure is the symmetric AES standard. The associated Rijndael algorithm was declared winner of the AES award on October 2nd, 2000 and thus succeeds the DES procedure.

More details about the AES algorithms and the AES candidates of the last round can be found within the online help of CrypTool².

1.1.1 New results about cryptanalysis of AES

Below you will find some results, which have recently called into question the security of the AES algorithm – from our point of view these doubts practically still remain unfounded. The following information is based on the original papers and the articles [Wobst-iX2002] and [Lucks-DuD2002].

AES with a minimum key length of 128 bit is still in the long run sufficiently secure against brute-force attacks – as long as the quantum computers aren’t powerful enough. When announced as new standard AES was immune against all known cryptanalytic attacks, mostly based on statistical considerations and earlier applied to DES: using pairs of clear and cipher

¹With CrypTool you can execute the following modern symmetric encryption algorithms (using the menu path **Crypt \ Symmetric (modern)**):

IDEA, RC2, RC4, DES (ECB), DES (CBC), Triple-DES (ECB), Triple-DES (CBC), MARS (AES candidate), RC6 (AES candidate), Serpent (AES candidate), Twofish (AES candidate), Rijndael (official AES algorithm).

²CrypTool online help: The index head-word **AES** leads to the 3 help pages: **AES candidates**, **The AES winner Rijndael** and **The Rijndael encryption algorithm**.

texts expressions are constructed, which are not completely at random, so they allow conclusions to the used keys. These attacks required unrealistically large amounts of intercepted data.

Cryptanalysts already label methods as “academic success” or as “cryptanalytic attack” if they are theoretically faster than the complete testing of all keys (brute force analysis). In the case of AES with the maximal key length (256 bit) exhaustive key search on average needs 2^{255} encryption operations. A cryptanalytic attack needs to be better than this. At present between 2^{75} and 2^{90} encryption operations are estimated to be performable only just for organizations, for example a security agency.

In their 2001-paper Ferguson, Schroepel and Whiting [Ferguson2001] presented a new method of symmetric codes cryptanalysis: They described AES with a closed formula (in the form of a continued fraction) which was possible because of the “relatively” clear structure of AES. This formula consists of around 1000 trillion terms of a sum - so it does not help concrete practical cryptanalysis. Nevertheless curiosity in the academic community was awakened. It was already known, that the 128-bit AES could be described as an over-determined system of about 8000 quadratic equations (over an algebraic number field) with about 1600 variables (some of them are the bits of the wanted key) – equation systems of that size are in practice not solvable. This special equation system is relatively sparse, so only very few of the quadratic terms (there are about 1,280,000 are possible quadratic terms in total) appear in the equation system.

The mathematicians Courtois and Pieprzyk [Courtois2002] published a paper in 2002, which got a great deal of attention amongst the cryptology community: The pair had further developed the XL-method (eXtended Linearization), introduced at Eurocrypt 2000 by Shamir et al., to create the so called XSL-method (eXtended Sparse Linearization). The XL-method is a heuristic technique, which in some cases manages to solve big non-linear equation systems and which was till then used to analyze an asymmetric algorithm (HFE). The innovation of Courtois and Pieprzyk was, to apply the XL-method on symmetric codes: the XSL-method can be applied to very specific equation systems. A 256-bit AES could be attacked in roughly 2^{230} steps. This is still a purely academic attack, but also a direction pointer for a complete class of block ciphers. The major problem with this attack is that until now nobody has worked out, under what conditions it is successful: the authors specify in their paper necessary conditions, but it is not known, which conditions are sufficient. There are two very new aspects of this attack: firstly this attack is not based on statistics but on algebra. So attacks seem to be possible, where only very small amounts of ciphertext are available. Secondly the security of a product algorithm³ does not exponentially increase with the number of rounds.

Currently there is a large amount of research in this area: for example Murphy and Robshaw presented a paper at Crypto 2002 [Robshaw2002a], which could dramatically improve cryptanalysis: the burden for a 128-bit key was estimated at about 2^{100} steps by describing AES as a special case of an algorithm called BES (Big Encryption System), which has an especially “round” structure. But even 2^{100} steps are beyond what is achievable in the foreseeable future. Using a 256 bit key the authors estimate that a XSL-attack will require 2^{200} operations.

More details can be found at:

³A ciphertext can be used as input for another encryption algorithm. A cascade cipher is build up as a composition of different encryption transformations. The overall cipher is called product algorithm or cascade cipher (sometimes depending whether the used keys are statistically dependent or not).

Cascading does not always improve the security.

This process is also used *within* modern algorithms: They usually combine simple and, considered at its own, cryptologically relatively insecure single steps in several rounds into an efficient overall procedure. Most block ciphers (e.g. DES, IDEA) are cascade ciphers.

Also serial usage of the same cipher with different keys (like with Triple-DES) is called cascade cipher.

<http://www.cryptosystem.net/aes>
<http://www.minrank.org/aes/>

So for 256-AES the attack is much more effective than brute-force but still far more away from any computing power which could be accessible in the short-to-long term.

The discussion is very controversial at the moment: Don Coppersmith (one of the inventors of DES) for example queries the practicability of the attack because XLS would provide no solution for AES [Coppersmith2002]. This implies that then the optimization of Murphy and Robshaw [Robshaw2002b] would not work.

In 2009 Biryukov und Khovratovich [Biryukov2009] published another theoretical attack on AES. This attack uses different methods from the ones described above. They applied methods from hash function cryptanalysis (local collisions and boomerang switching) to construct a related-key attack on AES-256. I. e. the attacker not only needs to be able to encrypt arbitrary data (chosen plain text), in addition he needs to be able to manipulate the unknown key (related-key).

Based on those assumptions, the effort to find a AES-256 key is reduced to 2^{119} time and 2^{77} memory. In the case of AES-192 the attack is even less practical, for AES-128 the authors do not provide an attack.

1.1.2 Current status of brute-force attacks on symmetric algorithms (RC5)

The current status of brute-force attacks on symmetric encryption algorithms can be explained with the block cipher RC5.

Brute-force (exhaustive search, trial-and-error) means to completely examine all keys of the key space: so no special analysis methods have to be used. Instead, the ciphertext is decrypted with all possible keys and for each resulting text it is checked, whether this is a meaningful clear text. A key length of 64 bit means at most $2^{64} = 18,446,744,073,709,551,616$ or about 18 trillion (GB) / 18 quintillion (US) keys to check⁴.

Companies like RSA Security provide so-called cipher challenges in order to quantify the security offered by well-known symmetric ciphers as DES, Triple-DES or RC5⁵. They offer prizes for those who manage to decipher ciphertexts, encrypted with different algorithms and different key lengths, and to unveil the symmetric key (under controlled conditions). So theoretical estimates can be confirmed.

It is well-known, that the “old” standard algorithm DES with a fixed key length of 56 bit is no more secure: This was demonstrated already in January 1999 by the Electronic Frontier Foundation (EFF). With their specialized computer Deep Crack they cracked a DES encrypted message within less than a day⁶.

The current record for strong symmetric algorithms unveiled a key 64 bit long. The algorithm used was RC5, a block cipher with variable key size.

The RC5-64 challenge has been solved by the distributed.net team after 5 years⁷. In total

⁴With CrypTool you can also try brute-force attacks of modern symmetric algorithms (using the menu path **Analysis \ Symmetric Encryption (modern)**): here the weakest knowledge of an attacker is assumed, he performs a ciphertext-only attack. To achieve a result in an appropriate time with a single PC you should mark not more than 20 bit of the key as unknown.

⁵<http://www.rsasecurity.com/rsalabs/challenges/secretkey/index.html>

In May 2007 RSA announced that they will not confirm the correctness of the not yet solved RC5-72 challenge.

⁶<http://www.rsasecurity.com/rsalabs/challenges/des3/index.html>

⁷<http://distributed.net/pressroom/news-20020926.html>

331,252 individuals co-operated over the internet to find the key⁸. More than 15 trillion (GB) / 15 quintillion (US) keys were checked, until they found the right key.

This makes clear, that symmetric algorithms (even if they have no cryptographic weakness) using keys of size 64 bit are no more appropriate to keep sensible data private.

There are also cipher challenges for asymmetric algorithms (please see chapter 4.11.4)⁹.

1.2 Asymmetric encryption¹⁰

In the case of *asymmetric* encryption each subscriber has a personal pair of keys consisting of a *secret* key and a *public* key. The public key, as its name implies, is made public, e.g. in a key directory on the Internet.

If Alice¹¹ wants to communicate with Bob, then she finds Bob's public key in the directory and uses it to encrypt her message to him. She then sends this ciphertext to Bob, who is then able to decrypt it again using his secret key. As only Bob knows his secret key, only he can decrypt messages addressed to him. Even Alice who sends the message cannot restore plaintext from the (encrypted) message she has sent. Of course, you must first ensure that the public key cannot be used to derive the private key.

Such a procedure can be demonstrated using a series of thief-proof letter boxes. If I have composed a message, I then look for the letter box of the recipient and post the letter through it. After that, I can no longer read or change the message myself, because only the legitimate recipient has the key for the letter box.

The advantage of asymmetric procedures is the easy key management. Let's look again at a network with n subscribers. In order to ensure that each subscriber can establish an encrypted connection to each other subscriber, each subscriber must possess a pair of keys. We therefore need $2n$ keys or n pairs of keys. Furthermore, no secure channel is needed before messages are transmitted, because all the information required in order to communicate confidentially can be sent openly. In this case, you simply have to pay attention to the accuracy (integrity and authenticity) of the public key. Disadvantage: Pure asymmetric procedures take a lot longer to perform than symmetric ones.

The most well-known asymmetric procedure is the RSA algorithm¹², named after its developers Ronald Rivest, Adi Shamir and Leonard Adleman. The RSA algorithm was published in 1978. The concept of asymmetric encryption was first introduced by Whitfield Diffie and Martin Hellman in 1976. Today, the ElGamal procedures also play a decisive role, particularly the Schnorr variant in the DSA (Digital Signature Algorithm).

⁸An overview of current distributed computing projects can be found here:
<http://distributedcomputing.info/>

⁹A wide spectrum of simple and complex crypto riddles are included in the "Mystery Twister C3" challenge.

¹⁰With CrypTool you can execute RSA encryption and decryption (using the menu path **Crypt \ Asymmetric**). In both cases you must select a RSA key pair. Only in the case of decryption the secret RSA key is necessary: so here you are asked to enter the PIN.

¹¹In order to describe cryptographic protocols participants are often named Alice, Bob, ... (see [Schneier1996, p. 23]). Alice and Bob perform all 2-person-protocols. Alice will initiate all protocols and Bob answers. The attackers are named Eve (eavesdropper) and Mallory (malicious active attacker).

¹²The RSA algorithm is extensively described in chapter 4.10 and later within this script. The RSA cryptosystem can be executed in many variations with CrypTool (using the menu path **Individual Procedures \ RSA Cryptosystem \ RSA Demonstration**). The topical research results concerning RSA are described in chapter 4.11.

1.3 Hybrid procedures¹³

In order to benefit from the advantages of symmetric and asymmetric techniques together, hybrid procedures are usually used (for encryption) in practice.

In this case the bulk data is encrypted using symmetric procedures: The key used for this is a secret session key generated by the sender randomly¹⁴ that is only used for this message.

This session key is then encrypted using the asymmetric procedure, and transmitted to the recipient together with the message.

Recipients can determine the session key using their private keys and then use the session key to encrypt the message.

In this way, we can benefit from the easy key management of asymmetric procedures (using public/private keys) and we benefit from the efficiency of symmetric procedures to encrypt large quantities of data (using secret keys).

¹³Within CrypTool you can find this technique using the menu path **Crypt \ Hybrid**: There you can follow the single steps and its dependencies with concrete numbers. The variant with RSA as the asymmetric algorithm is graphically visualized; the variant with ECC uses the standard dialogs. In both cases AES is used as the symmetric algorithm.

¹⁴An important part of cryptographically secure techniques is to generate random numbers. Within CrypTool you can check out different random number generators using the menu path **Indiv. Procedures \ Generate Random Numbers**. Using the menu path **Analysis \ Analyze Randomness** you can apply different test methods for random data to binary documents.

Up to now CrypTool has concentrated on cryptographically strong pseudo random number generators. Only the integrated Secude generator involves a "pure" random source.

*IETF*¹⁵:

There is an old saying inside the US National Security Agency (NSA):

”Attacks always get better; they never get worse.”

1.4 Ciphers and cryptanalysis for educational purposes

Compared to public-key ciphers based on mathematics like RSA, the structure of AES, and most other modern symmetric ciphers, is very complex and cannot be explained as easily as RSA.

So some simplified variants of modern symmetric ciphers like DES, IDEA or AES were developed for educational purposes in order to allow beginners to perform encryption and decryption by hand and gain a better understanding of how the algorithms work in detail. These simplified variants also help to understand and apply the according cryptanalysis methods.

One of these variants is e.g. S-AES (Simplified-AES) by Prof. Ed Schaefer and his students [Musa-etal2003]¹⁶. Another one is Mini-AES [Phan2002] (see chapter 1.6.1 “Mini-AES”):

- Edward F. Schaefer: *A Simplified Data Encryption Standard Algorithm* [Schaefer1996].
- Raphael Chung-Wei Phan: *Mini Advanced Encryption Standard (Mini-AES): A Testbed for Cryptanalysis Students* [Phan2002].
- Raphael Chung-Wei Phan: *Impossible differential cryptanalysis of Mini-AES* [Phan2003].
- Mohammad A. Musa, Edward F. Schaefer, Stephen Wedig: *A simplified AES algorithm and its linear and differential cryptanalyses* [Musa-etal2003].
- Nick Hoffman: *A SIMPLIFIED IDEA ALGORITHM* [Hoffman2006].
- S. Davod. Mansoori, H. Khaleghei Bizaki: *On the vulnerability of Simplified AES Algorithm Against Linear Cryptanalysis* [Mansoori-etal2007].

1.5 Further details

Beside the information you can find in the following chapters, in many other books and on a good number of websites, the online help of CrypTool also offers very many details about the symmetric and asymmetric encryption methods.

¹⁵<http://tools.ietf.org/html/rfc4270>

¹⁶See the two articles with the same title “Devising a Better Way to Teach and Learn the Advanced Encryption Standard” at the university’s news at

- <http://math.scu.edu/~eschaefer/getfile.pdf>

- <http://www.scu.edu/cas/research/cryptography.cfm>

1.6 Appendix: Examples using Sage

Below is Sage source code related to contents of the chapter 1 (“Encryption Procedures”).

Further details concerning cryptosystems within Sage (e.g. S-DES) can be found e.g. in the thesis of Minh Van Nguyen [Nguyen2009].

1.6.1 Mini-AES

The Sage module `crypto/block_cipher/miniaes.py` supports Mini-AES to allow students to explore the working of a modern block cipher.

Mini-AES, originally described at [Phan2002], is a simplified variant of the Advanced Encryption Standard (AES) to be used for cryptography education.

How to use Mini-AES is exhaustively described at the Sage reference page:
http://www.sagemath.org/doc/reference/sage/crypto/block_cipher/miniaes.html.

The following Sage code from the release tour of Sage 4.1¹⁷ calls the implementation of the Mini-AES.

Further example code for Mini-AES can be found in [Nguyen2009, chap. 6.5 and appendix D].

¹⁷See <http://mvngu.wordpress.com/2009/07/12/sage-4-1-released/>

Sage sample 1.1 Encryption and decryption with Mini-AES

```
# We can encrypt a plaintext using Mini-AES as follows:
sage: from sage.crypto.block_cipher.miniaes import MiniAES
sage: maes = MiniAES()
sage: K = FiniteField(16, "x")
sage: MS = MatrixSpace(K, 2, 2)
sage: P = MS([K("x^3 + x"), K("x^2 + 1"), K("x^2 + x"), K("x^3 + x^2")]); P

[ x^3 + x  x^2 + 1]
[ x^2 + x x^3 + x^2]
sage: key = MS([K("x^3 + x^2"), K("x^3 + x"), K("x^3 + x^2 + x"), K("x^2 + x + 1")]); key

[ x^3 + x^2  x^3 + x]
[x^3 + x^2 + x  x^2 + x + 1]
sage: C = maes.encrypt(P, key); C

[ x  x^2 + x]
[x^3 + x^2 + x  x^3 + x]

# Here is the decryption process:
sage: plaintext = maes.decrypt(C, key)
sage: plaintext == P
True

# We can also work directly with binary strings:
sage: from sage.crypto.block_cipher.miniaes import MiniAES
sage: maes = MiniAES()
sage: bin = BinaryStrings()
sage: key = bin.encoding("KE"); key
0100101101000101
sage: P = bin.encoding("Encrypt this secret message!")
sage: C = maes(P, key, algorithm="encrypt")
sage: plaintext = maes(C, key, algorithm="decrypt")
sage: plaintext == P
True

# Or work with integers n such that 0 <= n <= 15:
sage: from sage.crypto.block_cipher.miniaes import MiniAES
sage: maes = MiniAES()
sage: P = [n for n in xrange(16)]; P
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
sage: key = [2, 3, 11, 0]; key
[2, 3, 11, 0]
sage: P = maes.integer_to_binary(P)
sage: key = maes.integer_to_binary(key)
sage: C = maes(P, key, algorithm="encrypt")
sage: plaintext = maes(C, key, algorithm="decrypt")
sage: plaintext == P
True
```

Bibliography

- [Biryukov2009] Alex Biryukov, Dmitry Khovratovich,
Related-key Cryptanalysis of the Full AES-192 and AES-256,
2009
<http://eprint.iacr.org/2009/317>
- [Coppersmith2002] Don Coppersmith,
Re: Impact of Courtois and Pieprzyk results,
2002-09-19, “AES Discussion Groups” at
<http://aes.nist.gov/aes/>
- [Courtois2002] Nicolas Courtois, Josef Pieprzyk,
Cryptanalysis of Block Ciphers with Overdefined Systems of Equations,
received 10 Apr 2002, last revised 9 Nov 2002.
A different version, so called compact version of the first XSL attack, was published at
Asiacrypt Dec 2002.
<http://eprint.iacr.org/2002/044>
- [Ferguson2001] Niels Ferguson, Richard Schroepel, Doug Whiting,
A simple algebraic representation of Rijndael, Draft 2001/05/1,
<http://www.xs4all.nl/~vorpai/pubs/rdalgeq.html>
- [Hoffman2006] Nick Hoffman,
2006
A SIMPLIFIED IDEA ALGORITHM,
<http://www.nku.edu/~christensen/simplified%20IDEA%20algorithm.pdf>
- [Lucks-DuD2002] Stefan Lucks, Rüdiger Weis,
Neue Ergebnisse zur Sicherheit des Verschlüsselungsstandards AES,
In: DuD, Dec 2002.
- [Mansoori-et al2007] S. Davod. Mansoori, H. Khaleghi Bizaki,
On the vulnerability of Simplified AES Algorithm Against Linear Cryptanalysis,
In: IJCSNS International Journal of Computer Science and Network Security, Vol.7
No.7, July 2007, pp 257-263
See also:
http://paper.ijcsns.org/07_book/200707/20070735.pdf
- [Musa-et al2003] Mohammad A. Musa, Edward F. Schaefer, Stephen Wedig,
A simplified AES algorithm and its linear and differential cryptanalyses,
In: Cryptologia 17 (2), April 2003, pp 148-177.
See also:

- http://findarticles.com/p/articles/mi_qa3926/is_200304/ai_n9181658
<http://www.rose-hulman.edu/~holden/Preprints/s-aes.pdf>
<http://math.scu.edu/~eschaefer/> Ed Schaefer's Homepage
- [Nguyen2009] Minh Van Nguyen,
Exploring Cryptography Using the Sage Computer Algebra System,
 Victoria University, 2009,
 See Sage publications <http://www.sagemath.org/library-publications.html>
- [Nichols1996] Randall K. Nichols,
Classical Cryptography Course, Volume 1 and 2,
 Aegean Park Press 1996; or in 12 lessons online at
<http://www.fortunecity.com/skyscraper/coding/379/lesson1.htm>
- [Phan2002] Raphael Chung-Wei Phan,
Mini Advanced Encryption Standard (Mini-AES): A Testbed for Cryptanalysis Students,
 In: Cryptologia 26 (4), 2002, pp 283-306.
- [Phan2003] Raphael Chung-Wei Phan,
Impossible differential cryptanalysis of Mini-AES,
 In: Cryptologia, Oct 2003
http://findarticles.com/p/articles/mi_qa3926/is_200310/ai_n9311721/
- [Robshaw2002a] S.P. Murphy, M.J.B. Robshaw,
Essential Algebraic Structure within the AES,
 June 5, 2002, Crypto 2002,
<http://www.isg.rhul.ac.uk/~mrobshaw/rijndael/rijndael.html>
- [Robshaw2002b] S.P. Murphy, M.J.B. Robshaw,
Comments on the Security of the AES and the XSL Technique,
 September 26, 2002,
<http://www.isg.rhul.ac.uk/~mrobshaw/rijndael/rijndael.html>
- [Schaefer1996] Edward F. Schaefer,
A Simplified Data Encryption Standard Algorithm,
 In: Cryptologia 20 (1), 1996, pp 77-84
- [Schmeh2003] Klaus Schmeh,
Cryptography and Public Key Infrastructures on the Internet,
 John Wiley & Sons Ltd., Chichester 2003.
 An up-to-date, easy to read book, which also considers practical problems such as
 standardization or real existing software. In German the 4th edition was published in
 2009.
- [Schneier1996] Bruce Schneier,
Applied Cryptography, Protocols, Algorithms, and Source Code in C,
 Wiley 1994, 2nd edition 1996.
- [Stallings2006] William Stallings,
Cryptography and Network Security,
 Prentice Hall 2006,
<http://williamstallings.com/>

- [Stamp2007] Mark Stamp, Richard M. Low,
Applied Cryptanalysis: Breaking Ciphers in the Real World,
Wiley-IEEE Press 2007,
<http://cs.sjsu.edu/faculty/stamp/crypto/>
- [Swenson2008] Christopher Swenson,
Modern Cryptanalysis: Techniques for Advanced Code Breaking,
Wiley 2008.
- [Wobst-iX2002] Reinhard Wobst,
Angekratzt - Kryptoanalyse von AES schreitet voran,
In: iX, Dec 2002;
plus the reader's remark by Johannes Merkle in iX Feb 2003.

Web links

1. AES or Rijndael cryptosystem
<http://www.cryptosystem.net/aes>
<http://www.minrank.org/aes/>
2. AES discussion groups at NIST
<http://aes.nist.gov/aes>
3. distributed.net: “RC5-64 has been solved”
<http://distributed.net/pressroom/news-20020926.html>
4. RSA: “The RSA Secret Key Challenge”
<http://www.rsasecurity.com/rsalabs/challenges/secretkey/index.html>
5. RSA: “DES Challenge”
<http://www.rsasecurity.com/rsalabs/challenges/des3/index.html>
6. Further links can be found at the CrypTool homepage
<http://www.cryptool.org>

Chapter 2

Paper and Pencil Encryption Methods

(Christine Stötzzel, April 2004; Updates: B.+C. Esslinger, June 2005; Updates Minh Van Nguyen and B. Esslinger, November 2009)

Edgar Allan Poe: A Few Words on Secret Writing, 1841

Few persons can be made to believe that it is not quite an easy thing to invent a method of secret writing which shall baffle investigation. Yet it may be roundly asserted that human ingenuity cannot concoct a cipher which human ingenuity cannot resolve.

The following chapter provides a broad overview of paper and pencil methods¹ each with references to deeper information. All techniques that people can apply manually to en- and decipher a message are embraced by this term. These methods were and still are especially popular with secret services, as a writing pad and a pencil – in contrast to electronic aids – are totally unsuspecting.

The first paper and pencil methods already arose about 3000 years ago, but new procedures were developed during the past century, too. All paper and pencil methods are a matter of symmetric methods. Even the earliest encryption algorithms use the basic principles such as transposition, substitution, block construction and their combinations. Hence it is worthwhile to closely consider this “ancient” methods especially under didactic aspects.

Methods to be successful and wide-spread had to fulfill some attributes which are equally required for modern algorithms:

- Exhaustive description, almost standardization (including special cases, padding, etc.).
- Good balance between security and usability (because methods being too complicated were error-prone or unacceptably slow).

¹The footnotes to this chapter describe how the methods can be performed using CrypTool CrypTool 1. Additionally the last sub chapter (2.5) contains example code using the computer algebra system Sage.

2.1 Transposition ciphers

Encrypting a message by means of transposition does not change the original characters of this message, only their order is modified (transposition = exchange)².

2.1.1 Introductory samples of different transposition ciphers

- **Rail Fence**³ [Singh2001]: The characters of a message are alternately written in two (or more) lines, creating a zigzag pattern. The resulting ciphertext is read out line by line. This is more a children's method.

Plaintext⁴: an example of transposition

n	x	m	l	o	t	a	s	o	i	i	n
a	e	a	p	e	f	r	n	p	s	t	o

Table 2.1: Rail Fence cipher

Ciphertext⁵: NXMLO TASOI INAEA PEFRN PSTO

- **Scytale**⁶ [Singh2001]: This method was probably used since 600 B.C. – a description of how it operated is not known from before Plutarch (50-120 B.C.). A long strip of paper is wrapped around a wooden cylinder and then the message is written along the length of this strip. The ciphertext is produced by unwinding the strip.
- **Grille** [Goebel2003]: Both parties use identical stencils. Line by line, their holes are filled with plaintext that is read out column by column to produce the ciphertext. If there is plaintext left, the procedure is repeated⁷.
- **Turning grille** [Savard1999]: The German army used turning grilles during WW1⁸. A square grille serves as a stencil, a quarter of its fields being holes. The first part of the message is written on a piece of paper through these holes, then the grille is rotated by 90 degrees and the user can write down the second part of the message, etc. But this

²Another name used for transposition is permutation.

³This method can directly be found in CrypTool at the menu item **Crypt/Decrypt \ Symmetric (classic) \ Scytale / Rail Fence**. You can simulate this method also under the menu **Crypt/Decrypt \ Symmetric (classic) \ Permutation**: For a Rail Fence with 2 lines use as key “B,A” and accept the default settings (only one permutation, where your input is done line-by-line and the output is taken column-by-column). Using the key “A,B” would start the zigzag pattern below in the way, that the first letter is written into the first line instead of the second line.

⁴If the alphabet only uses 26 letters, we write the plaintext in small letters and the ciphertext in capital letters.

⁵The letters of the cleartext are – as used historically – grouped within blocks of 5 letters. It does not matter if the (constant) block length is different or no blank is inserted.

⁶This method can directly be found in CrypTool at the menu item **Crypt/Decrypt \ Symmetric (classic) \ Scytale / Rail Fence**. As this method is a special case of a simple columnar transposition, you also can simulate it in CrypTool under the menu **Crypt/Decrypt \ Symmetric (classic) \ Permutation**: For the Scytale within the dialog box only the first permutation is used. If the word has e.g. 4 angles use as key “1,2,3,4”. This is equivalent to write the text horizontally in blocks of 4 letters in a matrix and to read it out vertically. Because the key is in an ascending order, the Scytale is denoted as an identical permutation. And because writing and read-out is done only once it is a simple (and no double) permutation.

⁷This method cannot be simulated with a pure column transposition.

⁸The turning grille was already invented in 1881 by Eduard Fleissner von Wostrowitz.

A good visualization can be found under www.turning-grille.com.

method does only work, if the holes are chosen carefully: Every field has to be used, and no field may be used twice, either. The ciphertext is read out line by line.

In the example for a turning grille in the following table you can write 4 times 16 characters of the cleartext on a piece of paper:

O	-	-	-	-	O	-	-
-	-	-	O	O	-	-	O
-	-	-	O	-	-	O	-
-	-	O	-	-	-	-	-
-	-	-	-	O	-	-	-
O	-	O	-	-	-	O	-
-	O	-	-	-	-	-	O
-	-	-	O	O	-	-	-

Table 2.2: 8x8 turning grille

2.1.2 Column and row transposition⁹

- **Simple columnar transposition** [Savard1999]: First of all, a keyword is chosen, that is written above the columns of a table. This table is filled with the text to be encrypted line by line. Then the columns are rearranged by sorting the letters of the keyword alphabetically. Afterwards the columns are read out from left to right to build the ciphertext¹⁰.

Plaintext: an example of transposition

K	E	Y
a	n	e
x	a	m
p	l	e
o	f	t
r	a	n
s	p	o
s	i	t
i	o	n

Table 2.3: Simple columnar transposition

Transposition key: K=2; E=1; Y=3.

Ciphertext: NALFA PIOAX PORSS IEMET NOTN

- **AMSCO** [ACA2002]: The characters of the plaintext are written in alternating groups of one respectively two letters into a grille. Then the columns are swapped and the text can be read out.

⁹Most of the following methods can be simulated in CrypTool under the menu **Crypt/Decrypt \ Symmetric (classic) \ Permutation**.

¹⁰Using CrypTool: Choose a key for the 1st permutation, input line by line, permute and output column by column.

- **Double column transposition** [Savard1999] : Double columnar transposition was frequently used during WW2 and during the Cold War. Two simple columnar transpositions with different keys are executed successively¹¹.
- **Column transposition, General Luigi Sacco** [Savard1999]: The columns of a table are numbered according to the letters of the keyword. The plaintext is entered line by line, in the first line up to column number one, in the second line up to column number two, etc. Again, the ciphertext is read out in columns.

Plaintext: an example of transposition

C	O	L	U	M	N
1	5	2	6	3	4
a					
n	e	x			
a	m	p	l	e	
o	f	t	r	a	n
s	p				
o	s	i	t		
i	o	n			

Table 2.4: Columnar transposition (General Luigi Sacco)

Ciphertext: ANAOS OIEMF PSOXP TINLR TEAN

- **Column transposition, French army in WW1** [Savard1999]: After executing a simple columnar transposition, diagonal rows are read out.
- **Row transposition** [Savard1999]: The plaintext is divided into blocks of equal length and a keyword is chosen. Now the letters of the keyword are numbered and permutation is done only within each block according to this numbering¹².

2.1.3 Further transposition algorithm ciphers

- **Geometric figures** [Goebel2003]: Write the message into a grille following one pattern and read it out using another.
- **Union Route Cipher** [Goebel2003]: The Union Route Cipher derives from Civil War. This method does not rearrange letters of a given plaintext, but whole words. Particularly sensitive names and terms are substituted by codewords which are recorded in codebooks together with the existing routes. A route determines the size of a grille and the pattern that is used to read out the ciphertext. In addition, a number of filler words is defined.
- **Nihilist Transposition** [ACA2002]: Insert the plaintext into a square grille and write the same keyword above the columns and next to the lines. As this keyword is sorted alphabetically, the contents of the grille are rearranged, too. Read out the ciphertext line by line.

¹¹Using CrypTool: Choose a key for the 1st permutation, input line by line, permute and output column by column. Then choose a (different) key for the 2nd permutation, input line by line, permute and output column by column.

¹²Using CrypTool: Choose a key for 1st permutation, input line by line, permute column by column and output line by line.

Plaintext: an example of transposition

	W	O	R	D	S		D	O	R	S	W
W	a	n	e	x	a	D	s	p	o	i	s
O	m	p	l	e	o	O	e	p	l	o	m
R	f	t	r	a	n	R	a	t	r	n	f
D	s	p	o	s	i	S	n	i	o	-	t
S	t	i	o	n	-	W	x	n	e	a	a

Table 2.5: Nihilist transposition¹³

Ciphertext: SPOIS EPLOM ATRNF NIOTX NEAA

- **Cadenus** [ACA2002]: Cadenus is a form of columnar transposition that uses two keywords.

The 1st keyword is used to swap columns.

The 2nd keyword is used to define the initial letter of each column: this 2nd keyword is a permutation of the used alphabet. This permutation is written on the left of the first column. Afterwards, each column is moved (wrap-around) so that it begins with the letter, which is in the same line as the key letter of the first keyword within the second keyword. Ciphertext is read out line by line. See table 2.6.

Plaintext: cadenus is a form of columnar transposition using a keyword

	K	E	Y	E	K	Y	E	K	Y
A	c	a	d	a	c	d	s	a	a
D	e	n	u	n	e	u	s	r	p
X	s	i	s	i	s	s	i	f	i
K	a	f	o	f	a	o	u	l	o
C	r	m	o	m	r	o	n	n	s
W	f	c	o	c	f	o	k	t	g
N	l	u	m	u	l	m	w	n	e
S	n	a	r	a	n	r	d	o	o
Y	t	r	a	r	t	a	a	t	d
E	n	s	p	s	n	p	n	n	u
D	o	s	i	s	o	i	i	i	s
T	t	i	o	i	t	o	f	a	o
U	n	u	s	u	n	s	m	y	o
B	i	n	g	n	i	g	c	r	o
R	a	k	e	k	a	e	u	c	m
G	y	w	o	w	y	o	a	e	r
H	r	d	-	d	r	-	r	s	-

Table 2.6: Cadenus¹⁴

Ciphertext:

SAASR PIFIU LONNS KTGWN EDOOA TDNNU IISFA OMYOC ROUCM AERRS

¹³After filling the matrix with the cleartext you get the left block. After switching rows and columns you get the right block

2.2 Substitution ciphers

2.2.1 Monoalphabetic substitution ciphers

Monoalphabetic substitution assigns one character of the ciphertext alphabet to each plaintext character. This mapping remains unchanged during the whole process of encryption.

- **General monoalphabetic substitution / Random letter pairs**¹⁵ [Singh2001]: The substitution occurs by a given assignment of single letters.
- **Atbash**¹⁶ [Singh2001]: Replace the first letter of the alphabet by the last letter of the alphabet, the second one by the last but one, etc.
- **Shift cipher, for example Caesar cipher**¹⁷ [Singh2001]: Plaintext alphabet and ciphertext alphabet are shifted against each other by a determined number of letters. Using the Caesar cipher means shifting letters about three positions.

Plaintext: three positions to the right

Ciphertext: WKUHH SRVLWLRQV WR WKH ULJKW

- **Affine cipher**: This is a generalization of the shift cipher. A plaintext character is first substituted for another character and then the result is encrypted using the shift cipher. The affine cipher is so named because its encryption and decryption functions are affine or linear functions.
- **Substitution with symbols** [Singh2001], for instance the so-called “freemason cipher”: Each letter is replaced with a symbol.
- **Variants**: Fill characters, intentional mistakes [Singh2001].
- **Nihilist Substitution**¹⁸ [ACA2002]: Insert the alphabet into a 5x5-matrix to assign each letter the number built from row and column number. A keyword is chosen and placed above the columns of a second matrix (grille). The plaintext is written row by row into the grille. The ciphertext results from adding the numbers of the plaintext and the numbers of the keyword. Numbers between 100 and 110 are transformed to numbers between 00 and 10, so that each letter is represented by a two-digit number.

See table 2.7.

Plaintext: an example of substitution

¹⁴Within the 2nd block of three chars those chars are printed bold which are at the top of the 3rd block after applying the 2nd key word.

¹⁵This cipher can be simulated in CrypTool under the menu **Crypt/Decrypt \ Symmetric (classic) \ Substitution / Atbash**.

¹⁶This cipher can be simulated in CrypTool under the menu **Crypt/Decrypt \ Symmetric (classic) \ Substitution / Atbash**.

¹⁷In CrypTool this method can be found at three different places in the menu tree:

- **Crypt/Decrypt \ Symmetric (classic) \ Caesar / ROT13**
- **Analysis \ Symmetric Encryption (classic) \ Ciphertext only \ Caesar**
- **Indiv. Procedures \ Visualization of Algorithms \ Caesar**.

¹⁸An animation of this Nihilist method can be found in CrypTool at the menu item **Indiv. Procedures \ Visualization of Algorithms \ Nihilist**.

	1	2	3	4	5
1	S	U	B	T	I
2	O	N	A	C	D
3	E	F	G	H	K
4	L	M	P	Q	R
5	V	W	X	Y	Z

Matrix

K	E	Y
(35)	(31)	(54)
a	n	e
(58)	(53)	(85)
x	a	m
(88)	(54)	(96)
p	l	e
(78)	(72)	(85)
o	f	s
(56)	(63)	(65)
u	b	s
(47)	(44)	(65)
t	i	t
(49)	(46)	(68)
u	t	i
(47)	(55)	(69)
o	n	
(56)	(53)	

Table

Table 2.7: Nihilist substitution

Ciphertext: 58 53 85 88 54 96 78 72 85 56 63 65 47 44 65 49 46 68 47 55 69 56 53

- **Coding** [Singh2001]: In the course of time, codebooks were used again and again. A codebook assigns a codeword, a symbol or a number to every possible **word** of a message. Only if both parties hold identical codebooks and if the assignment of codewords to plaintext words is not revealed, a successful and secret communication can take place.
- **Nomenclature** [Singh2001]: A nomenclature is an encryption system that is based upon a ciphertext alphabet. This alphabet is used to encrypt the bigger part of the message. Particularly frequent or top-secret words are replaced by a limited number of codewords existing besides the ciphertext alphabet.
- **Map cipher** [ThinkQuest1999]: This method constitutes a combination of substitution and steganography¹⁹. Plaintext characters are replaced by symbols which are arranged in a map following certain rules
- **Straddling Checkerboard** [Goebel2003]: A 3x10 matrix is filled with the letters of the used alphabet and two arbitrary digits or special characters as follows: The different letters of a keyword and the remaining characters are written into the grille. The columns

¹⁹Instead of encrypting a message, pure steganography tries to conceal its existence.

are numbered 0 to 9, the second and the third line are numbered 1 and 2. Each plaintext character is replaced by the corresponding digit, respectively the corresponding pair of digits. As “1” and “2” are the first digits of the possible two-digit-numbers, they are not used as single digits.

See table 2.8.

Plaintext: an example of substitution

	0	1	2	3	4	5	6	7	8	9
1	K	-	-	E	Y	W	O	R	D	A
2	B	C	F	G	H	I	J	L	M	N
	P	Q	S	T	U	V	X	Z	.	/

Table 2.8: Straddling checkerboard with password “Keyword”

Ciphertext: 91932 69182 01736 12222 41022 23152 32423 15619

Besides, “1” and “2” are the most commonly used digits, but this feature is removed by the following technique.

It is ostentatious, how often the numbers 1 and 2 appear, but this will be fixed with the following version.

- **Straddling Checkerboard, variant** [Goebel2003] : This variant of the straddling checkerboard was developed by Soviet spies during WW2. Ernesto (Ché) Guevara and Fidel Castro allegedly used this cipher for their secret communication. A grille is filled with the alphabet (number of columns = length of keyword), and two arbitrary digits are chosen as reserved to indicate the second and third line of a 3x10-matrix (see above). Now the grille is traversed column by column and the single letters are transferred row by row into the matrix: For a faster encryption, the eight most common letters (ENIRSATO) are assigned the digits from 0 to 9, the reserved 2 digits are not assigned. The remaining letters are provided with combinations of digits one after another and are inserted into the grille.

See table 2.9.

Plaintext: an example of substitution

Grille

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

Matrix

	0	1	2	3	4	5	6	7	8	9
	A	T	E	-	N	O	R	-	I	S
3	K	J	B	L	U	Y	C	M	V	W
7	F	X	G	P	Z	H	Q	.	D	/

Table 2.9: Variant of the straddling checkerboard

Ciphertext: 04271 03773 33257 09343 29181 34185 4

- **Ché Guevara Cipher**: A special variant is the cipher used by Ché Guevara (with an additional substitution step and a slightly changed checkerboard):

- * The seven most frequent letters in Spanish are distributed in the first row.
- * Four instead of three rows are used.
- * So one could encrypt $10 * 4 - 4 = 36$ different characters.

- **Tri-Digital** [ACA2002]: A keyword with ten letters is used to create a numeric key by numbering its letters corresponding to their alphabetical order. This key is written above the columns of 3x10-matrix. This matrix is filled line by line with the alphabet as follows: The different letters of a keyword are inserted first, followed by the remaining letters. The last column is left out. Plaintext characters are substituted with numbers, the number of the last column is used to separate words.

- **Baconian Cipher** [ACA2002]: Assign a five-digit binary code to every letter and to 6 numbers or special characters (for example 00000 = A, 00001 = B, etc.) and replace the plaintext characters with this binary code. Now use a second, unsuspicious message to hide the ciphertext inside of it. This may happen by upper and lower case or italicized letters: e.g. all letters of the unsuspicious message below a binary “1” are capitalised.

See table 2.10.

message	F	I	G	H	T
ciphertext	00101	01000	00110	00111	10011
unsuspicious message	it is w	ar man	the su	ni ssh	in ing
Baconian Cipher	it Is W	aR man	thESu	niSSH	IniNG

Table 2.10: Baconian cipher

2.2.2 Homophonic substitution ciphers

Homophonic methods constitute a special form of monoalphabetic substitution. Each character of the plaintext alphabet is assigned several ciphertext characters.

- **Homophonic monoalphabetic substitution**²⁰ [Singh2001]: Each language has a typical frequency distribution of letters. To conceal this distribution, each plaintext letter is assigned several ciphertext characters. The number of ciphertext characters assigned depends on the frequency of the letter to be encrypted.
- **Beale cipher** [Singh2001]: The Beale cipher is a book cipher that numbers the words of a keytext. These numbers replace the cleartext letters by the words’ initial letters.

²⁰This cipher can be simulated in CrypTool under the menu **Crypt/Decrypt \Symmetric (classic)\Homophone**.

- **Grandpré Cipher** [Savard1999]: A square grille with 10 columns (other layouts are possible, too) is filled with ten words. The initial letters should result in an eleventh word. As columns and rows are numbered from 0 to 9, letters can be replaced by two-digit numbers. It is obvious that with the table having a hundred fields, most letters can be represented by more than one number. You should keep in mind that those ten words have to contain all letters of the plaintext alphabet.
- **Book cipher**: The words of a message are substituted by triples “page-line-position”. This method requires a detailed agreement of which book to use, especially regarding the edition (layout, error correction, etc.).

2.2.3 Polygraphic substitution ciphers

Polygraphic techniques do not work by replacing single characters, but by replacing whole groups of characters. In most cases, these groups are diagrams, trigrams or syllables.

- **“Great Chiffre”** [Singh2001]: This cipher was used by Louis XIV. and was not solved until the end of the nineteenth century. Cryptograms consisted of 587 different numbers, every number representing a syllable. The inventors of the “Great Chiffre” (Rossignol, father and son) constructed additional traps to increase security. For example, a number could assign a different meaning to or delete the preceding one.
- **Playfair**²¹ [Singh2001]: A 5x5-matrix is filled with the plaintext characters. For example, the different letters of a keyword are inserted first, followed by the remaining letters. The plaintext is divided into pairs, these digraphs are encrypted using the following rules:
 1. If both letters can be found in the same column, they are replaced by the letters underneath.
 2. If both letters can be found in the same row, take the letters to their right.
 3. If both letters of the digraph are in different columns and rows, the replacement letters are obtained by scanning along the row of the first letter up to the column where the other letter occurs and vice versa.
 4. Double letters are treated by special rules, if they appear in one digraph. They can be separated by a filler, for example.

See table 2.11.

Plaintext: plaintext letters are x encrypted in pairs

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

Table 2.11: 5x5 Playfair matrix

Ciphertext: SHBHM UWUZF KUUKC MBDWU DURDA VUKBG PQBHC M

²¹In CrypTool you can call this method under the menu **Crypt/Decrypt \ Symmetric (classic) \ Playfair**.

- **Trigraphic Playfair:** A 5x5-matrix is filled with the alphabet (see above) and the plaintext is divided into trigraphs. Trigraphs are encrypted according to the following rules:
 1. Three equal letters are substituted by three equal letters. It is the letter on the right underneath the original letter.
 2. A trigraph with two different letters is encrypted like a digraph in Playfair.
 3. If a trigraph contains three different characters, very complex rules come into effect. See [Savard1999]
- **Substituting digraphs by symbols** [Savard1999]: Giovanni Battista della Porta, 15th century. He created a 20x20-matrix that contained one symbol for every possible combination of letters (his alphabet did not comprise more than twenty letters).
- **Four square cipher** [Savard1999]: This method is similar to Playfair, because it is based on a system of coordinates whose four quadrants are each filled with the alphabet. The layout of letters can differ from quadrant to quadrant. To encipher a message, act in the following way: Look up the first plaintext letter in the first quadrant and the second one in the third quadrant. These two letters are opposite corners of a rectangle and the ciphertext letters can be found in quadrant number two and four.

See table 2.12.

Plaintext: plaintext letters are encrypted in pairs

d	w	x	y	m	E	P	T	O	L
r	q	e	k	i	C	V	I	Q	Z
u	v	h	p	s	R	M	A	G	U
a	l	b	z	n	F	W	Y	H	S
g	c	o	f	t	B	N	D	X	K
Q	T	B	L	E	v	q	i	p	g
Z	H	N	D	X	s	t	u	o	h
P	M	I	Y	C	n	r	d	x	y
V	S	K	W	O	b	l	w	m	f
U	A	F	R	G	c	z	k	a	e

Table 2.12: Four square cipher

Ciphertext: MWYQW XQINO VNKGC ZWPZF FGZPM DIICC GRVCS

- **Two square cipher** [Savard1999]: The two square cipher resembles the four square cipher, but the matrix is reduced to two quadrants. Are both letters of the digraph part of the same row, they are just exchanged. Otherwise, the plaintext letters are considered as opposite corners of a rectangle and substituted by the other vertices. Quadrants can be arranged horizontal and vertical.
- **Tri square cipher** [ACA2002]: Three quadrants are filled with the same alphabet. The first plaintext letter is looked up in the first quadrant and can be encrypted with every letter of that column. The second plaintext letter is looked up in the second quadrant (diagonally across) and can be encrypted with every letter of that row. Between these two ciphertext characters, the letter at the intersection point is set.
- **Dockyard Cipher** [Savard1999]: Used by the German navy during WW2.

2.2.4 Polyalphabetic substitution ciphers

Concerning polyalphabetic substitution, the assignment of ciphertext characters to plaintext characters is not static, but changes during the process of encryption (depending on the key).

- **Vigenère**²² [Singh2001]: Each plaintext character is encrypted with a different ciphertext alphabet that is determined by the characters of a keyword (the so-called Vigenère-Tableau serves auxiliary means). If the plaintext is longer than the key, the latter is repeated.

See table 2.13.

Plaintext:	the	alphabet	is	changing
Key:	KEY	KEYKEYKE	YK	EYKEYKEY
Ciphertext:	DLC	KPNREZOX	GC	GFKRESRE

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

Table 2.13: Vigenère tableau

- **Interrupted key:** The key is not repeated continuously, but starts again with every new word of the message.
- **Autokey** [Savard1999]: After using the agreed key, use the message itself as a key. See table 2.14.

Plaintext:	the	alphabet	is	changing
Key:	KEY	THEALPHA	BE	TISCHANG
Ciphertext:	DLC	TSTHLQLT	JW	VPSPNIAM

Table 2.14: Autokey

- **Progressive key** [Savard1999]: The key changes during the process of encryption. With every repetition, the characters of the keyword are shifted about one position. “KEY” becomes “LFZ”.
- **Gronsfeld** [Savard1999]: Variant of Vigenère that uses a numeric key.

²²In CrypTool you can call this method under the menu **Crypt/Decrypt \ Symmetric (classic) \ Vigenère**.

- **Beaufort** [Savard1999]: Variant of Vigenère, the key is subtracted, not added. The ciphertext alphabets may be written backwards.
- **Porta** [ACA2002]: Variant of Vigenère with only 13 alphabets. As a consequence, two letters of the keyword are assigned the same ciphertext alphabet and the first and the second half of the alphabet are reciprocal.
- **Slidefair** [ACA2002]: This method can be used as a variant of Vigenère, Gronsfeld or Beaufort. Slidefair does encrypt digraphs according to the following rules: Look up the first letter in the plaintext alphabet above the tableau. Then look up the second one in the row belonging to the corresponding keyword letter. These two letters make up opposite corners of an imaginary rectangle. The letters at the two remaining corners substitute the digraph.

- **Superposition**

- **Book cipher**: A keytext (for example out of a book) is added to the plaintext.
- **Superposition with numbers**: A sequence or a number of sufficient length (for example pi) is added.
- **Phillips** [ACA2002]: The alphabet is filled into a square table with 5 columns. Seven more tables are generated by first shifting the first row one position towards the bottom, then shifting the second row towards the bottom. The plaintext is divided into blocks of five which are encrypted with one matrix each. Letters are substituted by the ones on their right and underneath.
- **Ragbaby** [ACA2002]: Construct an alphabet with 24 characters. Then number the plaintext characters, starting the numeration of the first word with “1”, the numeration of the second one with “2” and so forth. Number 25 corresponds to number 1. Each letter of the message is encrypted by shifting it the corresponding positions to the right.

alphabet: KEYWORDABCFGHILMNPSTUVXZ

Plaintext:	t h e	a l p h a b e t	i s	c h a n g i n g
Numbering:	1 2 3	2 3 4 5 6 7 8 9	3 4	4 5 6 7 8 9 10 11
Ciphertext:	U L O	C P V P I M C O	N X	I P I Z T X Y X

Table 2.15: Ragbaby

2.3 Combining substitution and transposition

In the history of cryptography one often comes across combinations of the previous mentioned methods.

- **ADFG(V)X**²³ [Singh2001]: ADFG(V)X-encryption was developed in Germany during WW1. The alphabet is filled into a 5x5 or 6x6 matrix, and columns and rows are marked with the letters ADFGX and V, depending on the size of the grille. Each plaintext character is substituted by the corresponding pair of letters. Finally, a (row-) transposition cipher is performed on the resulting text.

²³In CrypTool you can call this method under the menu **Crypt/Decrypt \ Symmetric (classic) \ ADFGVX**.

- **Fractionation** [Savard1999]: Generic term for all kinds of methods that encrypt one plaintext character by several ciphertext characters and then apply a transposition cipher to this ciphertext so that ciphertext characters originally belonging to each other are separated.

- **Bifid/Polybius square/checkerboard** [Goebel2003]: Bifid encryption is the basic form of fractionation. A 5x5 matrix is filled with the plaintext alphabet (see Playfair encryption), rows and columns are numbered, so that each cleartext character can be substituted by a pair of digits. Mostly the plaintext is divided into blocks of equal length. The length of blocks (here 5) is another configuration parameter of this cipher. Block-by-block all line numbers are read out first, followed by all numbers naming the columns. To obtain the ciphertext, the digits are pairwise transformed into letters again. The numbers can be any permutation of (1,2,3,4,5), which is one key of configuration parameter of this cipher. Instead of numbering rows and columns, a keyword can be used, too.

See table 2.16.

	2	4	5	1	3
1	K	E	Y	W	O
4	R	D	A	B	C
2	F	G	H	I	L
3	M	N	P	Q	S
5	T	U	V	X	Z

Plaintext:	combi	nings	ubsti	tutio	nandt	ransp	ositi
Rows:	41342	32323	54352	55521	34345	44333	13252
Columns:	33211	41443	41321	24213	45442	25435	33121

Table 2.16: Bifid

41342 32323 54352 55521 34345 44333 13252 33211 41443 41321 24213 45442 25435
33121

Ciphertext: BNLLL UPHVI NNUCS OHLMW BDNOI GINUR HCZQI

- **Trifid** [Savard1999]: 27 characters (alphabet + 1 special character) may be represented by a triple consisting of the digits 1 to 3. The message to be encrypted is divided into blocks of three and the relevant triple is written underneath each plaintext character as a column. The resulting numbers below the plaintext blocks are read out line by line and are substituted with the corresponding characters.
- **Bazeries** [ACA2002]: The plaintext alphabet is filled into a 5x5-matrix column by column, a second matrix is filled line by line with a keyword (a number smaller than a million) followed by the remaining letters of the alphabet. Then the message is divided into blocks of arbitrary length and their characters' order is inverted. Finally, each letter is substituted – according to its position in the original matrix – by its counterpart in the second matrix.

See table 2.17.

Plaintext: combining substitution and transposition

Keyword: 900.004 (nine hundred thousand and four)

a	f	l	q	v	N	I	E	H	U
b	g	m	r	w	D	R	T	O	S
c	h	n	s	x	A	F	B	C	G
d	i	o	t	y	K	L	M	P	Q
e	k	p	u	z	V	W	X	Y	Z

com	bini	ngs	ub	stitu	tiona	ndt	ran	sposi	ti	on
mo c	inib	sgn	bu	utits	anoit	tdn	nar	isops	it	no
TMA	LBLD	CRB	DY	YPLPC	NBMLP	PKB	BNO	LCMXC	LP	BM

Table 2.17: Bazeries

- **Digrafid** [ACA2002]: To substitute digraphs, the following table is used (to simplify matters, the alphabet is used in its original form). Look up the first letter of the digraph in the horizontal alphabet and write down the column number. Then look up the second letter in the vertical alphabet and write down the corresponding line number. Between these two numbers, the number at the intersection point is set. Afterwards, the triple are written vertically underneath the digraphs that are arranged in groups of three. The three digit numbers arising horizontally are transformed back into digraphs.

Remark: This cipher only works with complete blocks of 3 pairs of cleartext characters. For a complete description, it is necessary to explain how sender and receiver handle texts which fill in the last block only 1-5 characters. The possibilities range from ignoring a last and incomplete block to padding it with random characters or with characters predefined in advance.

See table 2.18.

1	2	3	4	5	6	7	8	9		
A	B	C	D	E	F	G	H	I	1	2 3
J	K	L	M	N	O	P	Q	R	4	5 6
S	T	U	V	W	X	Y	Z	.	7	8 9
									A	J S 1
									B	K T 2
									C	L U 3
									D	M V 4
									E	N W 5
									F	O X 6
									G	P Y 7
									H	Q Z 8
									I	R . 9

co mb in	in gs ub	st it ut	io na nd	tr an sp	os it io
3 4 9	9 7 3	1 9 3	9 5 5	2 1 1	6 9 9
2 4 2	2 3 7	9 3 9	2 4 4	8 2 8	6 3 2
6 2 5	5 1 2	2 2 2	6 1 4	9 5 7	1 2 6
LI KB FN	.C BY EB	SU I. BK	RN KD FD	BA HQ RP	X. FT AO

Table 2.18: Digrafid

- **Nicodemus** [ACA2002]: First of all, a simple columnar transposition is carried out. Before reading out the columns, the message is encrypted additionally by Vigenère (all letters of a column are enciphered with the corresponding keyword letter). The ciphertext is read out in vertical blocks.

See table 2.19.

Plaintext: combining substitution and transposition

K	E	Y	E	K	Y	E	K	Y
c	o	m	o	c	m	S	M	K
b	i	n	i	b	n	M	L	L
i	n	g	n	i	g	R	S	E
s	u	b	u	s	b	Y	C	Z
s	t	i	t	s	i	X	C	G
t	u	t	u	z	t	Y	J	R
i	o	n	o	i	n	S	S	L
a	n	d	n	a	d	R	K	B
t	r	a	r	t	a	V	D	Y
n	s	p	s	n	p	W	X	N
o	s	i	s	o	i	W	Y	G
t	i	o	i	t	o	M	D	N

Table 2.19: Nicodemus

Ciphertext: SMRYX MLSCC KLEZG YSRVW JSKDX RLBYN WMYDG N

2.4 Further methods

- **“Pinprick encryption”** [Singh2001]: For centuries, this simple encryption method has been put into practice for different reasons (actually steganography). During the Victorian Age, for example, small holes underneath letters in newspaper articles marked the characters of a plaintext, as sending a newspaper was much more cheaper than the postage on a letter.
- **Stencil**: Stencils (Cardboard with holes) are also known as “Cardinal-Richelieu-Key”. Sender and receiver have to agree upon a text. Above this text, a stencil is laid and the letters that remain visible make up the ciphertext.
- **Card games** [Savard1999]: The key is created by means of a pack of cards and rules that are agreed upon in advance. All methods mentioned in this paragraph are designed as paper and pencil methods, i.e. they are applicable without electronic aid. A pack of cards is unsuspecting to outsiders, shuffling the deck provides a certain amount of coincidence, cards can be transformed into numbers easily and a transposition cipher can be carried out without any further aid.
 - **Solitaire (Bruce Schneier)**²⁴ [Schneier1999]: Sender and receiver have to own a deck of cards shuffled in the same manner. A key stream is generated that has to consist of as many characters as the message to be encrypted.

²⁴In CrypTool you can call this method under the menu **Crypt/Decrypt \ Symmetric (classic) \ Solitaire**.

The algorithm to generate the key is based on a shuffled deck of 54 cards (Ace, 2 - 10, jack, queen, king in four suits and two jokers). The pack of cards is held face up:

1. Swap the first joker with the card beneath it.
2. Move the second joker two cards down.
3. Now swap the cards above the first joker with those below the second one.
4. Look at the bottom card and convert it into a number from 1 to 53 (bridge order of suits: clubs, diamonds, hearts, spades; joker = 53). Write down this number and count down as many cards starting with the top card. These cards are swapped with the remaining cards, only the bottom card remains untouched.
5. Look at the top card and convert it into a number, too. Count down as many cards starting with the top card.
6. Write down the number of the following card. This card is converted into your first keystream character. As we need numbers from 1 to 26 to match the letters of our alphabet, clubs and hearts correspond to the numbers 1 to 13, diamonds and spades to 14 to 26. If your output card is a joker, start again.

For each keystream character you like to generate, these six steps have to be carried out. This procedure is – manually – very lengthy (4 h for 300 characters, dependant on your exercise) and requires high concentration.

Encryption takes place by addition modulo 26. Encryption is relatively fast compared to the key stream generation.

- **Mirdek (Paul Crowley)** [Crowley2000]: Even though this method is quite complicated, the author provides a very good example to illustrate the procedure.
- **Playing Card Cipher (John Savard)** [Savard1999]: This algorithm uses a shuffled deck of 52 cards (no joker). Separate rules describe how to shuffle the deck. A keystream is created via the following steps:

1. The pack of cards lies in front of the user, top down. Cards are turned up and dealt out in a row until the total of the cards is 8 or more.
2. If the last card dealt out is a J, Q or K, write down its value, otherwise write down the sum of the cards dealt out (a number between 8 and 17). In a second row, deal out that number of cards.
3. The remaining cards are dealt out in rows under the second row. The first one ends under the lowest card of the top row, the second one under the next lowest card, and so on. If there are two identical cards, red is lower than black.
4. The cards dealt out under step 3 are collected column by column, starting with the column under the lowest card. The first card that is picked up becomes the bottom card (face up).
5. The cards dealt out in step 1 and 2 are picked up, beginning with the last card.
6. The deck is turned over, the top card is now the bottom card (face down). Afterwards, steps 1 to 6 are repeated twice.

To generate a keystream character, write down the first card not being J, Q or K. Count down that number of cards. The card selected has to be between 1 and 10. Now repeat these steps beginning with the last card. These two numbers are added and the last digit of the sum is your keystream character.

- **VIC cipher** [Savard1999]: This is a highly complicated but relatively secure paper and pencil method. It has been developed and applied by Soviet spies. Amongst other things,

the user had to create ten pseudo-random numbers out of a date, the first words of a sentence and any five-digit number. A straddling checkerboard is part of the encryption, too. A detailed description can be found under [Savard1999].

2.5 Appendix: Examples using Sage

In the following section some classic ciphers are implemented using the open source computer algebra system **Sage**²⁵. The code was tested with Sage version 4.2. All ciphers are explained in chapter 2 (“Paper and Pencil Encryption Methods”).

To make the sample code²⁶ easier to understand, we used the structure and the naming conventions shown in the graphics below:

- Encryption consists of the two steps encoding and enciphering.
 - Encoding adapts the letters in the given plaintext P to the case defined in the given alphabet, and all non-alphabet characters are filtered out.
 - Enciphering creates the ciphertext C.
- Decryption also consists of two steps: deciphering and decoding.
 - Decoding is only necessary if the symbols in the alphabet are not ASCII characters.

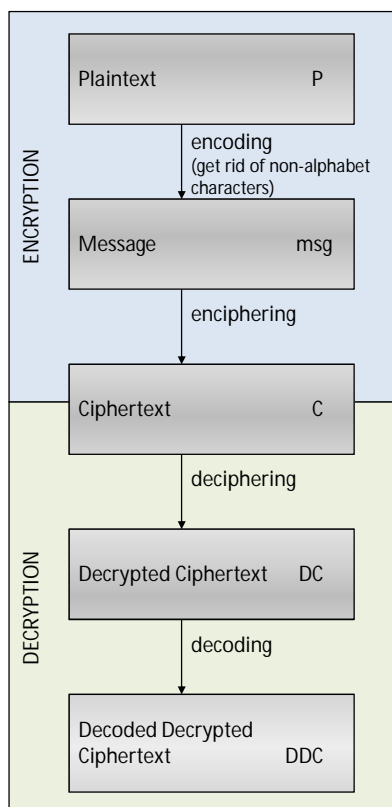


Figure 2.1: Structure and naming convention of the Sage cipher code examples

²⁵A first introduction to the CAS Sage can be found in the appendix A.5.

²⁶Further examples with Sage concerning classic crypto methods can be found e.g.:

- at http://www.sagemath.org/doc/constructions/linear_codes.html#classical-ciphers
- in the thesis of Minh Van Nguyen [Nguyen2009]

2.5.1 Transposition ciphers

Transposition ciphers are implemented in the Sage class

```
sage.crypto.classical.TranspositionCryptosystem
```

To construct and work with a transposition cipher, we first need to determine the alphabet that contains the symbols used to build the space of our plaintext and ciphertext. Typically, this alphabet will be the upper-case letters of the English alphabet, which can be accessed via the function

```
sage.monoids.string_monoid.AlphabeticStrings
```

We then need to decide on the block length of a block permutation, which is the length of the row vector to be used in the simple columns transposition. This row vector is our key, and it specifies a permutation of a plaintext.

The following first example of transposition ciphers has block length 14, and the key is build in a way, that every letter in the plaintext is shifted to the right by two characters, with wrap around at the end of the block. That is the encryption process. The decryption process is shifting each letter of the ciphertext to the left by $14 - 2 = 12$.

Sage sample 2.1 Simple Transposition by shifting (key and inverse key explicitly given)

```
sage: # transposition cipher using a block length of 14
sage: T = TranspositionCryptosystem(AlphabeticStrings(), 14)
sage: # given plaintext
sage: P = "a b c d e f g h i j k l m n"
sage: # encryption key
sage: key = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1, 2]
sage:
sage: # encode plaintext (get rid of non-alphabet chars, convert lower-case to upper-case)
sage: msg = T.encoding(P)
sage: # encrypt plaintext by shifting to the left by 2 letters (do it in two steps)
sage: E = T(key)
sage: C = E(msg); C
CDEFGHIJKLMNOPAB
sage:
sage: # decrypt ciphertext by shifting to the left by 12 letters
sage: keyInv = [13, 14, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
sage: D = T(keyInv)
sage: D(C)
ABCDEFGHIJKLMN
sage:
sage: # Representation of key and inverse key as permutations
sage: E
(1,3,5,7,9,11,13)(2,4,6,8,10,12,14)
sage: D
(1,13,11,9,7,5,3)(2,14,12,10,8,6,4)
```

The second example of transposition ciphers is also a simple shifting column transposition. But now the code is a little bit more automated: The keys are generated from the shift parameter.

Sage sample 2.2 Simple Transposition by shifting (key and inverse key constructed with “range”

```
sage: # transposition cipher using a block length of 14, code more variable
sage: keylen = 14
sage: shift = 2
sage: A = AlphabeticStrings()
sage: T = TranspositionCryptosystem(A, keylen)
sage:
sage: # construct the plaintext string from the first 14 letters of the alphabet plus blanks
sage: # plaintext    = "A B C D E F G H I J K L M N"
sage: A.gens()
(A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z)
sage: P=' '
sage: for i in range(keylen): P=P + " " + str(A.gen(i))
.....:
sage: P
' A B C D E F G H I J K L M N'
sage:
sage: # encryption key
sage: # key = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1, 2]
sage: key = [(i+shift).mod(keylen) + 1 for i in range(keylen)]; key
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1, 2]
sage:
sage: # encode plaintext (get rid of non-alphabet chars)
sage: msg = T.encoding(P)
sage: # encrypt plaintext by shifting to the left by 2 letters (do it in one step)
sage: C = T.enciphering(key, msg); C
CDEFGHIJKLMNOPAB
sage:
sage: # decrypt ciphertext by shifting to the left by 12 letters
sage: # keyInv = [13, 14, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
sage: shiftInv=keylen-shift;
sage: keyInv = [(i+shiftInv).mod(keylen) + 1 for i in range(keylen)]; keyInv
[13, 14, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
sage: DC = T.deciphering(keyInv, C); DC
ABCDEFGHIJKLMNOPQRSTUVWXYZ
sage:
sage: # decryption using the "deciphering method with key" instead of "enciphering with keyInv"
sage: # using the deciphering method requires to change the type of the variable key
sage: DC = T.deciphering(T(key).key(), C); DC
ABCDEFGHIJKLMNOPQRSTUVWXYZ
sage:
sage: # representation of key and inverse key as permutations
sage: T(key)
(1,3,5,7,9,11,13)(2,4,6,8,10,12,14)
sage: T(key).key()
(1,3,5,7,9,11,13)(2,4,6,8,10,12,14)
sage: T(keyInv)
(1,13,11,9,7,5,3)(2,14,12,10,8,6,4)
```

In the third example of transposition ciphers we use an arbitrary permutation as key in the encryption and decryption processes in order to scramble the characters within each block (block length = number of columns in a simple column transposition). If the block length is n , then the key must be a permutation on n symbols. The following example uses the method `random_key()` of the class `TranspositionCryptosystem`. Each call to `random_key()` produces a different key. Note that therefore your results (key and ciphertext) may be different from the following example.

Sage sample 2.3 Simple Column Transposition with randomly generated (permutation) key

```
sage: # Remark: Enciphering here requires, that the length of msg is a multiple of keylen
sage: keylen = 14 # length of key
sage: A = AlphabeticStrings()
sage: T = TranspositionCryptosystem(A, keylen); T
Transposition cryptosystem on Free alphabetic string monoid on A-Z of block length 14
sage:
sage: P = "a b c d e f g h i j k l m n o p q r s t u v w x y z a b"
sage: key = T.random_key(); key
(1,2,3,13,6,5,4,12,7)(11,14)
sage: msg = T.encoding(P); msg
ABCDEFGHIJKLMN O P Q R S T U V W X Y Z A B
sage: C = T.enciphering(key, msg); C
BCMLDEAHIJNGFKPQAZRSOVWXBUTY
sage: # decryption using the "deciphering method with key" instead of "enciphering with keyInv"
ssage: DC = T.deciphering(key, C); DC
ABCDEFGHIJKLMN O P Q R S T U V W X Y Z A B
sage:
sage: # Just another way of decryption: Using "enciphering" with the inverse key
sage: keyInv = T.inverse_key(key); keyInv
(1,7,12,4,5,6,13,3,2)(11,14)
sage: DC = T.enciphering(keyInv, C); DC
ABCDEFGHIJKLMN O P Q R S T U V W X Y Z A B
sage:
sage: # Test correctness of decryption
sage: msg == DC
True
```

The fourth example of transposition ciphers additionally shows the key space of a simple column transposition.

Sage sample 2.4 Simple Column Transposition (showing the key_space)

```
sage: keylen = 14      # length of key
sage: A = AlphabeticStrings()
sage: T = TranspositionCryptosystem(A, keylen); T
Transposition cryptosystem on Free alphabetic string monoid on A-Z of block length 14
sage: T.key_space()
Symmetric group of order 14! as a permutation group
sage: # Remark: The key space is not quite correct as also permutations shorter than keylen are counted.
sage:
sage: P = "a b c d e f g h i j k l m n o p q r s t u v w x y z a b"
sage: key = T.random_key(); key
(1,2,7)(3,9)(4,5,10,12,8,13,11)(6,14)
sage: msg = T.encoding(P); msg
ABCDEFGHIJKLMNOPQRSTUVWXYZAB
sage:
sage: # enciphering in one and in two steps
sage: C = T.enciphering(key, msg); C
BGIEJNAMCLDHKFPUWSXBOAQZRVYT
sage:
sage: enc = T(key); enc.key()
(1,2,7)(3,9)(4,5,10,12,8,13,11)(6,14)
sage: C = enc(msg); C
BGIEJNAMCLDHKFPUWSXBOAQZRVYT
sage:
sage: # deciphering
sage: DC = T.deciphering(key, C); DC
ABCDEFGHIJKLMNOPQRSTUVWXYZAB
```

2.5.2 Substitution ciphers

Substitution cryptosystems are implemented in Sage in the class

`sage.crypto.classical.SubstitutionCryptosystem`

The following code sample uses Sage to construct a substitution cipher with a random key. A random key can be generated using the method `random_key()` of the class `SubstitutionCryptosystem`. Different keys determine different substitution ciphers. So each call to `random_key()` returns different results.

Sage sample 2.5 Monoalphabetic Substitution with randomly generated key

```
sage: # plaintext/ciphertext alphabet
sage: A = AlphabeticStrings()
sage: S = SubstitutionCryptosystem(A)
sage:
sage: P = "Substitute this with something else better."
sage: key = S.random_key(); key
INZDHFUXJPATQOYLKSWGVECMRB
sage:
sage: # method encoding can be called from A or from T
sage: msg = A.encoding(P); msg
SUBSTITUTETHISWITHSOMETHINGELSEBETTER
sage: C = S.encrypting(key, msg); C
WVNWGJGVGHGXJWCJGXWYQHGXJOUHTWHNHGGHS
sage:
sage: # We now decrypt the ciphertext to recover our plaintext.
sage:
sage: DC = S.decrypting(key, C); DC
SUBSTITUTETHISWITHSOMETHINGELSEBETTER
sage: msg == DC
True
```

2.5.3 Caesar cipher

The following example uses Sage to construct a Caesar cipher.

Sage sample 2.6 Caesar (substitution by shifting the alphabet; key explicitly given, step-by-step approach)

```
sage: # plaintext/ciphertext alphabet
sage: A = AlphabeticStrings()
sage: P = "Shift the alphabet three positions to the right."
sage:
sage: # construct Caesar cipher
sage: S = SubstitutionCryptosystem(A)
sage: key = A([3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, \
....:         20, 21, 22, 23, 24, 25, 0, 1, 2])
sage:
sage: # encrypt message
sage: msg      = A.encoding(P); msg
SHIFTTHEALPHABETTHREEPOSITIONSTOTHERIGHT
sage: encrypt = S(key); encrypt
DEFGHIJKLMNOPQRSTUVWXYZABC
sage: C       = encrypt(msg); C
VKLIWWKHDOSKDEHWWKUHHRSRVLWLRQVWRWKHULJKW
sage:
sage: # Next, we recover the plaintext.
sage: # decrypt message
sage: keyInv = A([23, 24, 25, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, \
....:         14, 15, 16, 17, 18, 19, 20, 21, 22])
sage: decrypt = S(keyInv); decrypt
XYZABCDEFGHIJKLMNQRSTUUVW
sage: DC      = decrypt(C); DC
SHIFTTHEALPHABETTHREEPOSITIONSTOTHERIGHT
sage: msg == DC
True
```

The second Caesar sample does the same, but the code is more sophisticated/automated/flexible.

Sage sample 2.7 Caesar (substitution by shifting the alphabet; substitution key generated)

```
sage: # plaintext/ciphertext alphabet
sage: A = AlphabeticStrings()
sage: keylen = len(A.gens()); keylen
26
sage: shift = 3
sage: P = "Shift the alphabet three positions to the right."
sage:
sage: # construct Caesar cipher
sage: S = SubstitutionCryptosystem(A)
sage: S
Substitution cryptosystem on Free alphabetic string monoid on A-Z
sage: # key = A([3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, \
sage: #      20, 21, 22, 23, 24, 25, 0, 1, 2])
sage: key = [(i+shift).mod(keylen) for i in range(keylen)];
sage: key = A(key); key
DEFGHIJKLMNOPQRSTUVWXYZABC
sage: len(key)
26
sage:
sage: # encrypt message
sage: msg = A.encoding(P); msg
SHIFTTHEALPHABETTHREEPOSITIONSTOTHERIGHT
sage: C = S.enciphering(key, msg); C
VKLIWWKHDOSKDEHWWKUHHRSRVLWLRQVWRWKHULJKW
sage:
sage: # Next, we recover the plaintext.
sage: # decrypt message
sage: # keyInv = A([23, 24, 25, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, \
sage: #      14, 15, 16, 17, 18, 19, 20, 21, 22])
sage: shiftInv=keylen-shift;
sage: keyInv = [(i+shiftInv).mod(keylen) for i in range(keylen)];
sage: keyInv = A(keyInv); keyInv
XYZABCDEFGHIJKLMNPOQRSTUVWXYZ
sage: DC = S.enciphering(keyInv, C); DC
SHIFTTHEALPHABETTHREEPOSITIONSTOTHERIGHT
sage:
sage: # Just another way of decryption: Using "deciphering" with the key
sage: DC = S.deciphering(key, C); DC
SHIFTTHEALPHABETTHREEPOSITIONSTOTHERIGHT
sage:
sage: msg == DC
True
```

2.5.4 Shift cipher

The shift cipher can also be thought of as a generalization of the Caesar cipher. While the Caesar cipher restricts us to shift exactly three positions along an alphabet, the shift cipher allows us to shift any number of positions along the alphabet.

In the above samples we applied the `SubstitutionCryptosystem` and build Caesar as a special kind of substitution. In contrast here Caesar can be build as a special kind of the shift cipher.

The shift cipher is implemented directly in the Sage class

```
sage.crypto.classical.ShiftCryptosystem
```

In the following example, we construct a shift cipher over the capital letters of the English alphabet. We then encrypt a plaintext P by shifting it 12 positions along the alphabet. Finally, we decrypt the ciphertext C and make sure that the result (DC) is indeed the original plaintext. Shifting is a special way of substitution.

Sage sample 2.8 A shift cipher over the upper-case letters of the English alphabet

```
sage: # construct Shift cipher directly
sage: shiftcipher = ShiftCryptosystem(AlphabeticStrings()); shiftcipher
Shift cryptosystem on Free alphabetic string monoid on A-Z
sage: P = shiftcipher.encoding("Shift me any number of positions."); P
SHIFTMEANYNUMBEROFPOSITIONS
sage: key = 12 # shift can be any integer number
sage:
sage: # shift the plaintext by 12 positions to get the ciphertext
sage: C = shiftcipher.enciphering(key, P); C
ETURFYQMZKZGYNQDARBAEUFUAZE
sage:
sage: # decrypt the ciphertext and ensure that it is the original plaintext
sage: DC = shiftcipher.deciphering(key, C); DC
SHIFTMEANYNUMBEROFPOSITIONS
sage: DC == P
True
```

The Caesar cipher is simply a shift cipher whose shifting key is 3. In the next example, we use the shift cipher to create a Caesar cipher over the capital letters of the English alphabet.

Sage sample 2.9 Constructing the Caesar cipher using the shift cipher

```
sage: # create a Caesar cipher
sage: caesarcipher = ShiftCryptosystem(AlphabeticStrings())
sage: P = caesarcipher.encoding("Shift the alphabet by three positions to the right."); P
SHIFTTHEALPHABETBYTHREEPOSITIONSTOTHERIGHT
sage:
sage: key = 3 # shift the plaintext by exactly 3 positions
sage: C = caesarcipher.enciphering(key, P); C
VKLIWWKHDOSKDEHWEBWKUHSRVLWLRQVWRWKHULJKW
sage:
sage: # decrypt the ciphertext and ensure that it is the original plaintext
sage: DC = caesarcipher.deciphering(key, C); DC
SHIFTTHEALPHABETBYTHREEPOSITIONSTOTHERIGHT
sage: DC == P
True
```

2.5.5 Affine cipher

The affine cipher is implemented in the Sage class

```
sage.crypto.classical.AffineCryptosystem
```

In the following example, we construct an affine cipher with key $(3, 13)$ and use this key to encrypt a given plaintext P . The plaintext is then decrypted and the result DC is compared to the original plaintext.

Sage sample 2.10 An affine cipher with key $(3, 13)$

```
sage: # create an affine cipher
sage: affineCipher = AffineCryptosystem(AlphabeticStrings()); affineCipher
Affine cryptosystem on Free alphabetic string monoid on A-Z
sage: P = affineCipher.encoding("The affine cryptosystem.")
sage: P
THEAFFINECRYPTOSYSTEM
sage:
sage: # encrypt the plaintext using the key (3, 13)
sage: a, b = (3, 13)
sage: C = affineCipher.enciphering(a, b, P)
sage: C
SIZNCCLAZTMHGSDPHPSZX
sage:
sage: # decrypt the ciphertext and make sure that it is equivalent to the original plaintext
sage: DC = affineCipher.deciphering(a, b, C)
sage: DC
THEAFFINECRYPTOSYSTEM
sage: DC == P
True
```

We can also construct a shift cipher using the affine cipher. To do so, we need to restrict keys of the affine cipher be of the form $(1, b)$ where b is any non-negative integer. For instance, we can work through Sage example 2.8 on page 40 as follows:

Sage sample 2.11 Constructing a shift cipher using the affine cipher

```
sage: # construct a shift cipher
sage: shiftcipher = AffineCryptosystem(AlphabeticStrings()); shiftcipher
Affine cryptosystem on Free alphabetic string monoid on A-Z
sage: P = shiftcipher.encoding("Shift me any number of positions.")
sage: P
SHIFTMEANYNUMBEROFPOSITIONS
sage:
sage: # shift the plaintext by 12 positions to get the ciphertext
sage: a, b = (1, 12)
sage: C = shiftcipher.enciphering(a, b, P)
sage: C
ETURFYQMZKZGYNQDARBAEUFUAZE
sage:
sage: # decrypt the ciphertext and ensure that it is the original plaintext
sage: DC = shiftcipher.deciphering(a, b, C); P
SHIFTMEANYNUMBEROFPOSITIONS
sage: DC == P
True
```

We can also use the affine cipher to create the Caesar cipher. To do so, the encryption/decryption key must be $(1, 3)$. In the next example, we work through Sage example 2.9 on page 40 using the affine cipher.

Sage sample 2.12 Constructing the Caesar cipher using the affine cipher

```
sage: # create a Caesar cipher
sage: caesarcipher = AffineCryptosystem(AlphabeticStrings())
sage: P = caesarcipher.encoding("Shift the alphabet by three positions to the right.")
sage: P
SHIFTTHEALPHABETBYTHREEPOSITIONSTOTHERIGHT
sage:
sage: # shift the plaintext by 3 positions
sage: a, b = (1, 3)
sage: C = caesarcipher.enciphering(a, b, P)
sage: C
VKLIWWKHDOSKDEHWEBWKUHHSRVLWLRQVVRWKHULJKW
sage:
sage: # decrypt the ciphertext and ensure that it is the original plaintext
sage: DC = caesarcipher.deciphering(a, b, C)
sage: DC
SHIFTTHEALPHABETBYTHREEPOSITIONSTOTHERIGHT
sage: DC == P
True
```

2.5.6 Substitution with symbols

In the following Sage example the symbols are from the binary number system. A monoalphabetic substitution cipher with a binary alphabet has very little security: Because the plaintext/ciphertext alphabet has only the two elements 0 and 1, there are only two keys possible: (0 1) and (1 0).

Remark: The key of a general substitution cipher contains all symbols of the alphabet exactly once.

Sage sample 2.13 Monoalphabetic substitution with a binary alphabet

```
sage: # the plaintext/ciphertext alphabet
sage: B = BinaryStrings()
sage: # substitution cipher over the alphabet B; no keylen argument possible
sage: S = SubstitutionCryptosystem(B); S
Substitution cryptosystem on Free binary string monoid
sage: # To get a substitute for each symbol, key has always the length of the alphabet
sage: key = S.random_key(); key
10
sage: len(key)
2
sage: P = "Working with binary numbers."
sage: # encryption
sage: msg = B.encoding(P); msg
01010111011011110111001001101011011010010110111001100111001000000111011101101\
00101110100011010000010000001100010011010010110111001100001011100100111100100\
1000000110111001110101011011010110001001100101011100100111001100101110
sage: C = S.encrypting(key, msg); C
1010100010010000100011011001010010010110100100011001100011011111000100010010\
110100010111001011110111110011101100101101001000110011110100011011000011011\
01111110010001100010101001001001110110011010100011011000110011010001
sage: # decryption
sage: DC = S.decrypting(key, C); DC
01010111011011110111001001101011011010010110111001100111001000000111011101101\
00101110100011010000010000001100010011010010110111001100001011100100111100100\
10000001101110011101010110110110001001100101011100100111001100101110
sage: msg == DC
True
```

Remark: Currently `S` has no attribute `key`, and I found no way to transform the binary sequence `DC` back to ASCII.

The second sample of a monoalphabetic substitution with symbols uses a larger alphabet as plaintext/ciphertext space as the first sample. Here the hexadecimal number system is used as substitution alphabet.

Sage sample 2.14 Monoalphabetic substitution with a hexadecimal alphabet (and decoding in Python)

```
sage: A = HexadecimalStrings()
sage: S = SubstitutionCryptosystem(A)
sage: key = S.random_key(); key
2b56a4e701c98df3
sage: len(key)
16
sage: # Number of possible keys
sage: factorial(len(key))
20922789888000
sage: P = "Working with a larger alphabet."
sage:
sage: msg = A.encoding(P); msg
576f726b696e6720776974682061206c617267657220616c7068616265742e
sage: C = S.enciphering(key, msg); C
47e375e9e1efe75277e17ae052eb52e8eb75e7e47552ebe872e0ebe5e47a5f
sage: DC = S.deciphering(key, C); DC
576f726b696e6720776974682061206c617267657220616c7068616265742e
sage: msg == DC
True
sage:
sage: # Conversion hex back to ASCII:
sage: # - AlphabeticStrings() and HexadecimalStrings() don't have according methods.
sage: # - So we used Python directly.
sage: import binascii
sage: DDC = binascii.a2b_hex(repr(DC)); DDC
'Working with a larger alphabet.'
sage:
sage: P == DDC
True
```

2.5.7 Vigenère cipher

The Vigenère cipher is implemented in the Sage class

```
sage.crypto.classical.VigenereCryptosystem
```

For our ciphertext/plaintext space, we can work with the upper-case letters of the English alphabet, the binary number system, the octal number system, or the hexadecimal number system. Here is an example using the class `AlphabeticStrings`, which implements the English capital letters.

Sage sample 2.15 Vigenère cipher

```
sage: # construct Vigenere cipher
sage: keylen = 14
sage: A = AlphabeticStrings()
sage: V = VigenereCryptosystem(A, keylen); V
Vigenere cryptosystem on Free alphabetic string monoid on A-Z of period 14
sage:
sage: # alternative could be a given key: key = A('ABCDEFGHIJKLMN'); key
sage: key = V.random_key(); key
WSSSEEGVVAARUD
sage: len(key)
14
sage:
sage: # encoding
sage: P = "The Vigenere cipher is polyalphabetic."
sage: len(P)
38
sage: msg = V.encoding(P); msg      # alternative: msg = A.encoding(P); msg
THEVIGENERECIPHERISPOLYALPHABETIC
sage:
sage: # encryption [2 alternative ways (in two steps or in one): both work]
sage: # encrypt = V(key); encrypt
sage: # C = encrypt(msg); C
sage: C = V.enciphering(key, msg); C
PZWNMKKIZRETCSDWJAWTUGTALGBDXWLAG
sage:
sage: # decryption
sage: DC = V.deciphering(key, C); DC
THEVIGENERECIPHERISPOLYALPHABETIC
sage: msg == DC
True
```

2.5.8 Hill cipher

The Hill [Hill1929, Hill1931] or matrix cipher is more mathematically sophisticated than other ciphers mentioned in this chapter. The encryption/decryption key of this cipher is an invertible square matrix, and the plaintext/ciphertext is processed also as a matrix. The encryption and decryption processes use matrix operations. The Hill cipher is implemented in the Sage class

```
sage.crypto.classical.HillCryptosystem
```

In the following example our plaintext/ciphertext space is the capital letters of the English alphabet. In the Hill cipher, each letter of this alphabet is assigned a unique integer modulo 26.

Sage sample 2.16 Hill cipher

```
sage: # construct a Hill cipher
sage: keylen = 19      # keylen = 3  # Alternative key length with non-random small key
sage: A = AlphabeticStrings()
sage: H = HillCryptosystem(A, keylen); H
Hill cryptosystem on Free alphabetic string monoid on A-Z of block length 19
sage:
sage: # To create key non-randomly, HKS is necessary [even H.key_space() is not enough].
sage: # HKS = H.key_space()
sage: # key = HKS([[1,0,1],[0,1,1],[2,2,3]]); key
sage:
sage: # Random key creation
sage: key = H.random_key(); key
[10 7 5 2 0 6 10 23 15 7 17 19 18 2 9 12 0 10 11]
[23 1 1 10 4 9 21 1 25 22 19 8 17 22 15 8 12 25 22]
[ 4 12 16 15 1 12 24 5 9 13 5 15 8 21 23 24 22 20 6]
[ 5 11 6 7 3 12 8 9 21 20 9 4 16 18 10 3 2 23 18]
[ 8 22 14 14 20 13 21 19 3 13 2 11 13 23 9 25 25 6 8]
[24 25 8 24 7 18 3 20 6 11 25 5 6 19 7 24 2 4 10]
[15 25 11 1 4 7 11 24 20 2 18 4 9 8 12 19 24 0 12]
[14 6 2 9 11 20 13 4 10 11 4 23 14 22 14 16 9 12 18]
[12 10 21 5 21 15 16 17 19 20 1 1 15 5 0 2 23 4 14]
[21 15 15 16 15 20 4 10 25 7 15 4 7 12 24 9 19 10 6]
[25 15 2 3 17 23 21 16 8 18 23 4 22 11 15 19 6 0 15]
[14 23 9 3 18 15 10 18 7 5 12 23 11 9 22 21 20 4 14]
[ 3 6 8 13 20 16 11 1 13 10 4 21 25 15 12 3 0 11 18]
[21 25 14 6 11 3 21 0 19 17 5 8 5 4 9 2 23 19 15]
[ 8 11 9 11 20 15 6 1 3 18 18 22 16 17 6 3 15 11 2]
[21 15 5 22 2 9 0 4 22 10 2 10 19 19 17 19 1 21 4]
[ 7 17 9 2 15 5 14 3 6 9 12 12 22 15 8 4 21 14 19]
[19 14 24 19 7 5 22 22 13 14 7 18 17 19 25 2 1 23 6]
[ 2 6 14 22 17 7 23 6 22 7 13 20 0 14 23 17 6 1 12]
sage:
sage: # encoding and encryption
sage: P = "Hill or matrix cipher uses matrix operations."
sage: len(P)
45
sage: # implementation requires: Length of msg is a multiple of matrix dimension (block_length)
sage: msg = H.encoding(P); msg
HILLORMATRIXCIPHERUSESMATRIXOPERATIONS
sage: len(msg)
38
sage:
sage: # encryption
sage: C = H.encrypting(key, msg); C
CRWCKPRVYXNBRZTNZCTQWFSWDWBCBABGMNEHVP
sage:
sage: # decryption
sage: DC = H.decrypting(key, C); DC
HILLORMATRIXCIPHERUSESMATRIXOPERATIONS
sage: msg == DC
True
sage:
sage: # alternative decryption using inverse matrix
sage: keyInv = H.inverse_key(key); keyInv
[ 6 23 1 23 3 12 17 22 6 16 22 14 18 3 1 10 21 16 20]
[18 23 15 25 24 23 7 4 10 7 21 7 9 0 13 22 5 5 23]
...
[10 11 12 6 11 17 13 9 19 16 14 24 4 8 5 16 18 20 1]
[19 16 16 21 1 19 7 12 3 18 1 17 7 10 24 21 7 16 11]
sage: DC = H.encrypting(keyInv, C); DC
HILLORMATRIXCIPHERUSESMATRIXOPERATIONS
```

Bibliography

- [ACA2002] American Cryptogram Association,
Length and Standards for all ACA Ciphers,
2002.
<http://www.cryptogram.org/cdb/aca.info/aca.and.you/chap08.html#>
- [Bauer1995] Friedrich L. Bauer,
Entzifferte Geheimnisse, Springer, 1995.
- [Bauer2000] Friedrich L. Bauer,
Decrypted Secrets, Springer 1997, 2nd edition 2000.
- [Crowley2000] Paul Crowley,
Mirdek: A card cipher inspired by "Solitaire",
2000.
<http://www.ciphergoth.org/crypto/mirdek/>
- [DA1999] Data encryption page of the ThinkQuest Team 27158 for ThinkQuest 1999
(no update since 1999, no search possibility),
1999.
<http://library.thinkquest.org/27158/>
- [Goebel2003] Greg Goebel,
Codes, Ciphers and Codebreaking,
2003.
<http://www.vectorsite.net/ttcode.htm>
- [Hill1929] Lester S. Hill,
"Cryptography in an Algebraic Alphabet," *The American Mathematical Monthly*,
36(6):306–312, 1929.
- [Hill1931] Lester S. Hill,
"Concerning Certain Linear Transformation Apparatus of Cryptography," *The American Mathematical Monthly*, 38(3):135–154, 1931.
- [Nichols1996] Randall K. Nichols,
Classical Cryptography Course, Volume 1 and 2,
Aegean Park Press 1996; or in 12 lessons online at
<http://www.fortunecity.com/skyscraper/coding/379/lesson1.htm>
- [Nguyen2009] Minh Van Nguyen,
Exploring Cryptography Using the Sage Computer Algebra System,
Victoria University, 2009,
See Sage publications <http://www.sagemath.org/library-publications.html>

- [Savard1999] John J. G. Savard,
A Cryptographic Compendium,
1999.
<http://www.quadibloc.com/crypto/jscrypt.htm>
- [Schmeh2004] Klaus Schmeh,
Die Welt der geheimen Zeichen. Die faszinierende Geschichte der Verschlüsselung,
W3L Verlag Bochum, 1. Auflage 2004.
- [Schmeh2007] Klaus Schmeh,
Codeknacker gegen Codemacher. Die faszinierende Geschichte der Verschlüsselung,
W3L Verlag Bochum, 2. Auflage 2007.
This is most current among the books dealing in a comprehensive manner with the history of cryptology. It contains a small collection of solved and unsolved crypto riddles. One of the challenges deals with a double column transposition using two long keys, which are differently in addition.
- [Schneier1999] Bruce Schneier,
The Solitaire Encryption Algorithm,
version 1.2, 1999.
<http://www.schneier.com/solitaire.html>
- [Singh2001] Simon Singh,
Geheime Botschaften. Die Kunst der Verschlüsselung von der Antike bis in die Zeiten des Internet,
dtv, 2001.
- [ThinkQuest1999] ThinkQuest Team 27158,
Data Encryption,
1999.
<http://library.thinkquest.org/27158/>

Chapter 3

Prime Numbers

(Bernhard Esslinger, May 1999; Updates Nov. 2000, Dec. 2001, June 2003, May 2005, March 2006, June 2007, November 2009)

*Albert Einstein*¹:
Progress requires exchange of knowledge.

3.1 What are prime numbers?

Prime numbers are whole, positive numbers greater than or equal to 2 that can only be divided by 1 and themselves. All other natural numbers greater than or equal to 2 can be formed by multiplying prime numbers.

The *natural* numbers $\mathbb{N} = \{1, 2, 3, 4, \dots\}$ thus comprise

- the number 1 (the unit value)
- the primes and
- the composite numbers.

Prime numbers are particularly important for 3 reasons:

- In number theory, they are considered to be the basic components of natural numbers, upon which numerous brilliant mathematical ideas are based.
- They are of extreme practical importance in modern cryptography (public key cryptography). The most common public key procedure, invented at the end of the 1970's, is RSA encryption. Only using (large) prime numbers for particular parameters can you guarantee that an algorithm is secure, both for the RSA procedure and for even more modern procedures (digital signature, elliptic curves).
- The search for the largest known prime numbers does not have any practical usage known to date, but requires the best computers, is an excellent benchmark (possibility for determining the performance of computers) and leads to new calculation methods on many computers
(see also: <http://www.mersenne.org/prime.htm>).

¹Albert Einstein, German physicist and Nobel Prize winner, Mar 14, 1879 – Apr 14, 1955.

Many people have been fascinated by prime numbers over the past two millennia. Ambition to make new discoveries about prime numbers has often resulted in brilliant ideas and conclusions. The following section provides an easily comprehensible introduction to the basics of prime numbers. We will also explain what is known about the distribution (density, number of prime numbers in particular intervals) of prime numbers and how prime number tests work.

3.2 Prime numbers in mathematics

Every whole number has a factor. The number 1 only has one factor, itself, whereas the number 12 has the six factors 1, 2, 3, 4, 6, 12. Many numbers can only be divided by themselves and by 1. With respect to multiplication, these are the “atoms” in the area of numbers. Such numbers are called prime numbers.

In mathematics, a slightly different (but equivalent) definition is used.

Definition 3.2.1. *A whole number $p \in \mathbf{N}$ is called prime if $p > 1$ and p only possesses the trivial factors ± 1 and $\pm p$.*

By definition, the number 1 is not a prime number. In the following sections, p will always denote a prime number.

The sequence of prime numbers starts with

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, \dots .

The first 100 numbers include precisely 25 prime numbers. After this, the percentage of primes constantly decreases. Prime numbers can be factorized in a uniquely *trivial* way:

$$5 = 1 \cdot 5, \quad 17 = 1 \cdot 17, \quad 1,013 = 1 \cdot 1,013, \quad 1,296,409 = 1 \cdot 1,296,409.$$

All numbers that have 2 or more factors not equal 1 are called *composite* numbers. These include

$$4 = 2 \cdot 2, \quad 6 = 2 \cdot 3$$

as well as numbers that *look like primes*, but are in fact composite:

$$91 = 7 \cdot 13, \quad 161 = 7 \cdot 23, \quad 767 = 13 \cdot 59.$$

Theorem 3.2.1. *Each whole number m greater than 1 possesses a lowest factor greater than 1. This is a prime number p . Unless m is a prime number itself, then: p is less than or equal to the square root of m .*

All whole numbers greater than 1 can be expressed as a product of prime numbers — in a unique way. This is the claim of the **1st fundamental theorem of number theory** (= fundamental theorem of arithmetic = fundamental building block of all positive integers).

Theorem 3.2.2. *Each element n of the natural numbers greater than 1 can be written as the product $n = p_1 \cdot p_2 \dots p_m$ of prime numbers. If two such factorizations*

$$n = p_1 \cdot p_2 \cdot \dots \cdot p_m = p'_1 \cdot p'_2 \cdot \dots \cdot p'_{m'}$$

are given, then they can be reordered such that $m = m'$ and for all i : $p_i = p'_i$. (p_1, p_2, \dots, p_m are called the prime factors of n).

In other words: each natural number other than 1 can be written as a product of prime numbers in precisely one way, if we ignore the order of the factors. The factors are therefore unique (the *expression as a product of factors* is unique)! For example,

$$60 = 2 \cdot 2 \cdot 3 \cdot 5 = 2^2 \cdot 3^1 \cdot 5^1.$$

And this — other than changing the order of the factors — is the only way in which the number 60 can be factorized. If you allow numbers other than primes as factors, there are several ways of factorizing integers and the **uniqueness** is lost:

$$60 = 1 \cdot 60 = 2 \cdot 30 = 4 \cdot 15 = 5 \cdot 12 = 6 \cdot 10 = 2 \cdot 3 \cdot 10 = 2 \cdot 5 \cdot 6 = 3 \cdot 4 \cdot 5 = \dots$$

The following section is aimed more at those familiar with mathematical logic: The 1st fundamental theorem only appears to be obvious. We can construct numerous other sets of numbers (i.e. other than positive whole numbers greater than 1), for which numbers in the set cannot be expressed uniquely as a product of the prime numbers of the set: In the set $M = \{1, 5, 10, 15, 20, \dots\}$ there is no equivalent to the fundamental theorem under multiplication. The first five prime numbers of this sequence are 5, 10, 15, 20, 30 (note: 10 is prime, because 5 is not a factor of 10 in this set — the result is not an element of the given basic set M). Because the following applies in M :

$$100 = 5 \cdot 20 = 10 \cdot 10$$

and 5, 10, 20 are all prime numbers in this set, the expression as a product of prime factors is not unique here.

3.3 How many prime numbers are there?

For the natural numbers, the primes can be compared to elements in chemistry or the elementary particles in physics (see [Blum1999, p. 22]).

Although there are only 92 natural chemical elements, the number of prime numbers is unlimited. Even the Greek, Euclid² knew this in the third century B.C.

Theorem 3.3.1 (Euclid³). *The sequence of prime numbers does not discontinue. Therefore, the quantity of prime numbers is infinite.*

His proof that there is an infinite number of primes is still considered to be a brilliant mathematical consideration and conclusion today (**proof by contradiction**). He assumed that there is only a finite number of primes and therefore a largest prime number. Based on this assumption, he drew logical conclusions until he obtained an obvious contradiction. This meant that something must be wrong. As there were no mistakes in the chain of conclusions, it could only be the assumption that was wrong. Therefore, there must be an infinite number of primes!

²Euclid, a Greek mathematician of 4th and 3rd century B.C. He worked at the Egyptian academy of Alexandria and wrote “The Elements”, the most well known systematically textbook of the Greek mathematics.

³The common usage of the term does not denote Euclid as the inventor of the theorem rather; the true inventor is merely not as prominent. The theorem has already been distinguished and proven in Euclid’s Elements (Book IX, theorem 20). The phraseology is remarkable due to the fact that the word infinite is not used. The text reads as followed

Οί πρώτοι ἀριθμοὶ πλείους εἰσὶ παντὸς τοῦ προτεθέντος πλήθους πρώτων ἀριθμῶν,

the English translation of which is: the prime numbers are more than any previously existing amount of prime numbers.

Euclid's proof by contradiction goes as follows:

Proof

Assumption: There is a *finite* number of primes.

Conclusion: Then these can be listed $p_1 < p_2 < p_3 < \dots < p_n$, where n is the (finite) number of prime numbers. p_n is therefore the largest prime. Euclid now looks at the number $a = p_1 \cdot p_2 \cdot \dots \cdot p_n + 1$. This number cannot be a prime number because it is not included in our list of primes. It must therefore be divisible by a prime, i.e. there is a natural number i between 1 and n , such that p_i divides the number a . Of course, p_i also divides the product $a - 1 = p_1 \cdot p_2 \cdot \dots \cdot p_n$, because p_i is a factor of $a - 1$. Since p_i divides the numbers a and $a - 1$, it also divides the difference of these numbers. Thus: p_i divides $a - (a - 1) = 1$. p_i must therefore divide 1, which is impossible.

Contradiction: Our assumption was false.

Thus there is an *infinite* number of primes (Cross-reference: overview under 3.9 of the number of prime numbers in various intervals). \square

Here we should perhaps mention yet another fact which is initially somewhat surprising. Namely, in the prime numbers sequence p_1, p_2, \dots , gaps between prime numbers can have an individually determined length n . It is undeniable that under the n succession of natural numbers

$$(n+1)! + 2, \dots, (n+1)! + (n+1),$$

none of them is a prime number since in order, the numbers $2, 3, \dots, (n+1)$ are comprised respectively as real divisors. ($n!$ means the product of the first n natural numbers therefore $n! = n * (n-1) * \dots * 2 * 1$).

3.4 The search for extremely large primes

The largest prime numbers known today have several millions digits, which is too big for us to imagine. The number of elementary particles in the universe is estimated to be “only” a 80-digit number (See: overview under 3.11 of various orders of magnitude / dimensions).

3.4.1 The 20+ largest known primes (as of July 2009)

The following table contains the current record primes and a description of its particular number type⁴:

⁴An up-to-date version can be found in the internet at <http://primes.utm.edu/largest.html> and at <http://primes.utm.edu/merzenne/index.html>.

	Definition	Decimal Digits	When	Description
1	$2^{43,112,609} - 1$	12,978,189	2008	Mersenne, 47th known
2	$2^{42,643,801} - 1$	12,837,064	2009	Mersenne, 46th known
3	$2^{37,156,667} - 1$	11,185,272	2008	Mersenne, 45th known
4	$2^{32,582,657} - 1$	9,808,358	2006	Mersenne, 44th known
5	$2^{30,402,457} - 1$	9,152,052	2005	Mersenne, 43rd known
6	$2^{25,964,951} - 1$	7,816,230	2005	Mersenne, 42nd known
7	$2^{24,036,583} - 1$	7,235,733	2004	Mersenne, 41st known
8	$2^{20,996,011} - 1$	6,320,430	2003	Mersenne, 40th known
9	$2^{13,466,917} - 1$	4,053,946	2001	Mersenne, M-39
10	$19,249 \cdot 2^{13,018,586} + 1$	3,918,990	2007	Generalized Mersenne ⁵
11	$27,653 \cdot 2^{9,167,433} + 1$	2,759,677	2005	Generalized Mersenne
12	$28,433 \cdot 2^{7,830,457} + 1$	2,357,207	2004	Generalized Mersenne
13	$2^{6,972,593} - 1$	2,098,960	1999	Mersenne, M-38
14	$5,359 \cdot 2^{5,054,502} + 1$	1,521,561	2003	Generalized Mersenne
15	$4,847 \cdot 2^{3,321,063} + 1$	999,744	2005	Generalized Mersenne
16	$3 \cdot 2^{3,136,255} - 1$	944,108	2007	Generalized Mersenne
17	$2^{3,021,377} - 1$	909,526	1998	Mersenne, M-37
18	$2^{2,976,221} - 1$	895,932	1997	Mersenne, M-36
19	$222,361 \cdot 2^{2,854,840} + 1$	859,398	2006	Generalized Mersenne
20	$1,372,930^{131,072} + 1$	804,474	2003	Generalized Fermat ⁶
21	$1,361,244^{131,072} + 1$	803,988	2004	Generalized Fermat
22	$1,176,694^{131,072} + 1$	795,695	2003	Generalized Fermat
23	$342,673 \cdot 2^{2,639,439} - 1$	794,556	2007	Generalized Mersenne

Table 3.1: The 20+ largest known primes and its particular number types (as of July 2009)

The largest known prime is a Mersenne prime, found by the GIMPS project (chapter 3.4.2).

Within the largest known primes there are also numbers of the type generalized Mersenne number (chapter 3.6.2) and generalized Fermat numbers (chapter 3.6.5).

3.4.2 Special number types – Mersenne numbers and Mersenne primes

Almost all known huge prime numbers are special candidates, called *Mersenne numbers*⁷ of the form $2^p - 1$, where p is a prime. Not all Mersenne numbers are prime:

$$\begin{array}{ll} 2^2 - 1 = 3 & \Rightarrow \text{prime} \\ 2^3 - 1 = 7 & \Rightarrow \text{prime} \\ 2^5 - 1 = 31 & \Rightarrow \text{prime} \\ 2^7 - 1 = 127 & \Rightarrow \text{prime} \\ 2^{11} - 1 = 2,047 = 23 \cdot 89 & \Rightarrow \text{NOT prime!} \end{array}$$

Even Mersenne knew that not all Mersenne numbers are prime (see exponent $p = 11$). A prime Mersenne number is called Mersenne prime number.

However, he is to be thanked for the interesting conclusion that a number of the form $2^n - 1$ cannot be a prime number if n is a composite number:

Theorem 3.4.1 (Mersenne). *If $2^n - 1$ is a prime number, then n is also a prime number.*

Proof

The theorem of Mersenne can be proved by contradiction. We therefore assume that there exists a composite natural number n (with real factorization) $n = n_1 \cdot n_2$, with the property that $2^n - 1$ is a prime number.

From

$$\begin{aligned} (x^r - 1)((x^r)^{s-1} + (x^r)^{s-2} + \dots + x^r + 1) &= ((x^r)^s + (x^r)^{s-1} + (x^r)^{s-2} + \dots + x^r) \\ &\quad - ((x^r)^{s-1} + (x^r)^{s-2} + \dots + x^r + 1) \\ &= (x^r)^s - 1 = x^{rs} - 1, \end{aligned}$$

we conclude

$$2^{n_1 n_2} - 1 = (2^{n_1} - 1)((2^{n_1})^{n_2-1} + (2^{n_1})^{n_2-2} + \dots + 2^{n_1} + 1).$$

Because $2^n - 1$ is a prime number, one of the above two factors on the right-hand side must be equal to 1. This is the case if and only if $n_1 = 1$ or $n_2 = 1$. But this contradicts our assumption.

⁶This number was found within the distributed computing project “Seventeen or Bust” (SoB) (<http://www.seventeenorbust.com>) at March 26, 2007. While the well known GIMPS project (chapter 3.4.2) searches for bigger and bigger of the infinitely many primes, there is a chance, that the SoB project could have been completed its task sometime.

The SoB project tries to prove computationally, that the number $k = 78,557$ is the smallest Sierpinski number (John Selfridge proved in 1962, that 78,557 is a Sierpinski number).

The famous Polish mathematician Waclaw Sierpinski (1882 to 1969) proved in 1960, that there exist infinitely many odd integers k , which fulfill the following property: For all Sierpinski numbers k it is true: All numbers $N = k \cdot 2^n + 1$ are composite for all integers $n \geq 1$ (Sierpinski’s Composite Number Theorem, <http://mathworld.wolfram.com/SierpinskisCompositeNumberTheorem.html>).

When the project started in 2002 there have been 17 possible candidates < 78557 (this is the reason for the project’s name “Seventeen or Bust”). It is sufficient to find one single counter-example, to exclude a candidate k , which means to find a single $n \geq 1$, where $N = k \cdot 2^n + 1$ is prime. So it is only a byproduct of this task that this also generates new monster primes.

⁶Generalized Fermat number: $1,372,930^{131,072} + 1 = 1,372,930^{(2^{17})} + 1$

⁷Marin Mersenne, French priest and mathematician, Sep 08, 1588 – Sep 01, 1648.

Therefore the assumption is false. This means that there exists no composite number n , such that $2^n - 1$ is a prime. \square

Unfortunately this theorem only applies in one direction (the inverse statement does not apply, no equivalence): that means that there exist prime exponent for which the Mersenne number is **not** prime (see the above example $2^{11} - 1$, where 11 is prime, but $2^{11} - 1$ not).

Mersenne claimed that $2^{67} - 1$ is a prime number. There is also a mathematical history behind this claim: it first took over 200 years before Edouard Lucas (1842-1891) proved that this number is composite. However, he argued indirectly and did not name any of the factors. Then Frank Nelson Cole⁸ showed in 1903 which factors make up this composite number:

$$2^{67} - 1 = 147,573,952,589,676,412,927 = 193,707,721 \cdot 761,838,257,287.$$

He admitted to having worked 20 years on the factorization (expression as a product of prime factors)⁹ of this 21-digit decimal number!

Due to the fact that the exponents of the Mersenne numbers do not use all natural numbers, but only the primes, the *experimental space* is limited considerably. The currently known Mersenne prime numbers have the exponents¹⁰

2; 3; 5; 7; 13; 17; 19; 31; 61; 89; 107; 127; 521; 607; 1,279; 2,203; 2,281; 3,217;
4,253; 4,423; 9,689; 9,941; 11,213; 19,937; 21,701; 23,207; 44,497; 86,243; 110,503;
132,049; 216,091; 756,839; 859,433; 1,257,787; 1,398,269; 2,976,221; 3,021,377;
6,972,593; 13,466,917; 20,996,011; 24,036,583; 25,964,951; 30,402,457;
32,582,657; 37,156,667; 43,112,609; 42,643,801.

Thus 47 Mersenne prime numbers are currently known.

The 19th number with the exponent 4,253 was the first with at least 1,000 digits in decimal system (the mathematician Samuel Yates coined the expression *titanic* prime for this; it was discovered by Hurwitz in 1961); the 27th number with the exponent 44,497 was the first with at least 10,000 digits in the decimal system (Yates coined the expression *gigantic* prime for this. These names are now long outdated).

For the first 39 Mersenne prime numbers we know that this list is complete. The exponents until the 40th Mersenne prime number have not yet been checked completely¹¹.

M-37 – January 1998

The 37th Mersenne prime,

$$2^{3,021,377} - 1$$

⁸Frank Nelson Cole, American mathematician, Sep. 20, 1861 – May 26, 1926.

⁹Using CrypTool you can factorize numbers in the following way: menu **Indiv. Procedures \ RSA Cryptosystem \ Factorization of a Number**.

CrypTool can factorize in a reasonable time numbers no longer than 250 bit. Numbers bigger than 1024 bits are currently not accepted by CrypTool.

The current factorization records are listed in chapter 4.11.4.

¹⁰The following page from Landon Curt Noll contains all Mersenne primes including its date of discovery and its value as number and as word: <http://www.isthe.com/chongo/tech/math/prime/mersenne.html>

Also see: <http://www.utm.edu/>.

¹¹The current status of the check can be found at: <http://www.mersenne.org/status.htm>.

Hints, how the primality of a number can be checked, are in chapter 3.5, prime number tests.

was found in January 1998 and has 909,526 digits in the decimal system, which corresponds to 33 pages in the newspaper!

M-38 – June 1999

The 38th Mersenne prime, called M-38,

$$2^{6,972,593} - 1$$

was discovered in June 1999 and has 2,098,960 digits in the decimal system (that corresponds to around 77 pages in the newspaper).

M-39 – December 2001

The 39th Mersenne prime, called M-39,

$$2^{13,466,917} - 1,$$

was published at December 6, 2001 – more exactly, the verification of this number, found at November 14, 2001 by the Canadian student Michael Cameron, was successfully completed. This number has about 4 million decimal digits (exactly 4,053,946 digits). Trying only to print this number

$$(924947738006701322247758 \dots 1130073855470256259071)$$

would require around 200 pages in the Financial Times.

Right now (July 2009) all prime exponents smaller than 18,000,989 have been tested and double-checked¹². So we can be certain, that this is really the 39th Mersenne prime number and that there are no smaller undiscovered Mersenne primes (it is common usage to use the notation M-*nn* not until it is proven, that the *nn*-th known Mersenne prime is really the *nn*-th Mersenne prime).

GIMPS

The GIMPS project (Great Internet Mersenne Prime Search) was founded in 1996 by George Woltman to search for new largest Mersenne primes (<http://www.mersenne.org>). Further explanations about this number type can be found under Mersenne numbers and Mersenne primes.

Right now the GIMPS project has discovered 13 largest Mersenne primes so far, including the largest known prime number at all.

The following table contains these Mersenne record primes^{13,14}.

Dr. Richard Crandall discovered the advanced transform algorithm used by the GIMPS program. George Woltman implemented Crandall's algorithm in machine language, thereby producing a prime-search program of unprecedented efficiency, and that work led to the successful GIMPS project.

¹²See home page of the GIMPS project: http://www.mersenne.org/report_milestones.

¹³An up-to-date version can be found in the internet at <http://www.mersenne.org/history.htm>.

¹⁴Always, when a new record is published in the respective forums the same and often ironic discussions start: Does this kind of research have a deeper sense? Can this result be applied for anything useful? The answer is, that we don't know it yet. In fundamental research one cannot see at once whether and how it brings mankind forward.

Definition	Decimal Digits	When	Who
$2^{43,112,609} - 1$	12,978,189	August 23, 2008	Edson Smith
$2^{42,643,801} - 1$	12,837,064	April 12, 2009	Odd Magnar Strindmo
$2^{37,156,667} - 1$	11,185,272	September 6, 2008	Hans-Michael Elvenich
$2^{32,582,657} - 1$	9,808,358	September 4, 2006	Curtis Cooper/Steven Boone
$2^{30,402,457} - 1$	9,152,052	December 15, 2005	Curtis Cooper/Steven Boone
$2^{25,964,951} - 1$	7,816,230	February 18, 2005	Martin Nowak
$2^{24,036,583} - 1$	7,235,733	May 15, 2004	Josh Findley
$2^{20,996,011} - 1$	6,320,430	November 17, 2003	Michael Shafer
$2^{13,466,917} - 1$	4,053,946	November 14, 2001	Michael Cameron
$2^{6,972,593} - 1$	2,098,960	June 1, 1999	Nayan Hajratwala
$2^{3,021,377} - 1$	909,526	January 27, 1998	Roland Clarkson
$2^{2,976,221} - 1$	895,932	August 24, 1997	Gordon Spence
$2^{1,398,269} - 1$	420,921	November 1996	Joel Armengaud

Table 3.2: The largest primes found by the GIMPS project (as of July 2009)

On June 1st, 2003 a possible Mersenne prime was reported to the GIMPS server, which was checked afterwards as usual, before it was to be published. Unfortunately mid June the initiator and GIMPS project leader George Woltman had to tell, that two independent verification runs proved the number was composite. This was the first false positive report of a client in 7 years.

Now more than 130,000 volunteers, amateurs and experts, participate in the GIMPS project. They connect their computers into the so called “primenet”, organized by the company entropia.

3.4.3 Challenge of the Electronic Frontier Foundation (EFF)

This search is also spurred on by a competition started by the non-profit organization EFF (Electronic Frontier Foundation) using the means of an unknown donor. The participants are rewarded with a total of 500,000 USD if they find the longest prime number. In promoting this project, the unknown donor is not looking for the quickest computer, but rather wants to draw people’s attention to the opportunities offered by *cooperative networking*
<http://www.eff.org/awards/coop>

The discoverer of M-38 received 50,000 USD from the EFF for discovering the first prime with more than 1 million decimal digits.

For the next prize of 100,000 USD offered by EFF for a proven prime with more than 10 million decimal digits, Edson Smith qualified, who found the number $2^{43,112,609} - 1$ within the GIMPS project.

According to the EFF rules for their prizes they offer in the next stage 150,000 USD for a proven prime with more than 100 million decimal digits.

Edouard Lucas (1842-1891) held the record for the longest prime number for over 70 years by proving that $2^{127} - 1$ is prime. No new record is likely to last that long.

3.5 Prime number tests¹⁵

In order to implement secure encryption procedures we need extremely large prime numbers (in the region of $2^{2,048}$, i.e. numbers with 600 digits in the decimal system!).

If we look for the prime factors in order to decide whether a number is prime, then the search takes too long, if even the smallest prime factor is enormous. Factorizing numbers using systematic computational division or using the sieve of Eratosthenes is only feasible using current computers for numbers with up to around 20 digits in the decimal system. The biggest number factorized into its 2 almost equal prime factors has 200 digits (see RSA-200 in chapter 4.11.4).

However, if we know something about the *construction* of the number in question, there are extremely highly developed procedures that are much quicker. These procedures can determine the primality attribute of a number, but they cannot determine the prime factors of a number, if it is compound.

In the 17th century, Fermat¹⁶ wrote to Mersenne that he presumed that all numbers of the form

$$f(n) = 2^{2^n} + 1$$

are prime for all whole numbers $n \geq 0$ (see below, chapter 3.6.4).

As early as in the 19th century, it was discovered that the 29-digit number

$$f(7) = 2^{2^7} + 1$$

is not prime. However, it was not until 1970 that Morrison/Billhart managed to factorize it.

$$\begin{aligned} f(7) &= 340, 282, 366, 920, 938, 463, 463, 374, 607, 431, 768, 211, 457 \\ &= 59, 649, 589, 127, 497, 217 \cdot 5, 704, 689, 200, 685, 129, 054, 721 \end{aligned}$$

Despite Fermat was wrong with this supposition, he is the originator of an important theorem in this area: Many rapid prime number tests are based on the (little) Fermat theorem put forward by Fermat in 1640 (see chapter 4.8.3).

Theorem 3.5.1 (“little” Fermat). *Let p be a prime number and a be any whole number, then for all a*

$$a^p \equiv a \pmod{p}.$$

This could also be formulated as follows:

Let p be a prime number and a be any whole number that is not a multiple of p (also $a \not\equiv 0 \pmod{p}$), then $a^{p-1} \equiv 1 \pmod{p}$.

If you are not used to calculate with remainders (modulo), please simply accept the theorem or first read chapter 4 “Introduction to Elementary Number Theory with Examples”. What is important here is that this sentence implies that if this equation is not met for any whole number a , then p is not a prime! The tests (e.g. for the first formulation) can easily be performed using the *test basis* $a = 2$.

¹⁵With the educational tool for number theory **NT** you can apply the tests of Fermat and of Miller-Rabin: See learning units 3.2 and 3.3, pages 3-11/11.

NT can be called in CrypTool via the menu path **Indiv. Procedures \ Number Theory Interactive \ Learning Tool for Number Theory**. See appendix A.4.

¹⁶Pierre de Fermat, French mathematician, Aug 17, 1601 – Jan 12, 1665.

This gives us a criterion for non-prime numbers, i.e. a negative test, but no proof that a number a is prime. Unfortunately Fermat's theorem does not apply — otherwise we would have a simple proof of the prime number property (or to put it in other words, we would have a simple prime number criterion).

Pseudo prime numbers

Numbers n that have the property

$$2^n \equiv 2 \pmod{n}$$

but are not prime are called *pseudo prime numbers* (i.e. the exponent is not a prime). The first pseudo prime number is

$$341 = 11 \cdot 31.$$

Carmichael numbers

There are pseudo prime numbers n that pass the Fermat test

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

with all bases a which are relatively prime to n [$\gcd(a, n) = 1$], despite these numbers n are not prime: These numbers are called *Carmichael numbers*. The first of these is

$$561 = 3 \cdot 11 \cdot 17.$$

Sample: The number to be tested is 561. Because $561 = 3 \cdot 11 \cdot 17$ it is:
 The test condition $a^{560} \pmod{561} = 1$ is satisfied for $a = 2, 4, 5, 7, \dots$,
 but not for $a = 3, 6, 9, 11, 12, 15, 17, 18, 21, 22, \dots$.
 This means the test condition must not be satisfied for multiples of the prime factors 3, 11 or 17.
 The test applied for $a = 3$ results in: $3^{560} \pmod{561} = 375$.
 The test applied for $a = 5$ results in: $5^{560} \pmod{561} = 1$.

Strong pseudo prime numbers

A stronger test is provided by Miller/Rabin¹⁷: it is only passed by so-called *strong pseudo prime numbers*. Again, there are strong pseudo prime numbers that are not primes, but this is much less often the case than for (simple) pseudo prime numbers or for Carmichael numbers. The smallest strong pseudo prime number base 2 is

$$15,841 = 7 \cdot 31 \cdot 73.$$

¹⁷In 1976 an efficient probabilistic primality test was published by Prof. Rabin, based on a number theoretic result of Prof. Miller from the year before.

Prof. Miller worked at the Carnegie-Mellon University, School of Computer Science. Prof. Rabin, born in 1931, worked at the Harvard and Hebrew University.

If you test all 4 bases, 2, 3, 5 and 7, you will find only one strong pseudo prime number up to $25 \cdot 10^9$, i.e. a number that passes the test and yet is not a prime number.

More extensive mathematics behind the Rabin test delivers the probability that the number examined is prime (such probabilities are currently around 10^{-60}).

Detailed descriptions of tests for finding out whether a number is prime can be found on Web sites such as:

<http://www.utm.edu/research/primes/merzenne.shtml>

<http://www.utm.edu/research/primes/prove/index.html>

3.6 Overview special number types and the search for a formula for primes

There are currently no useful, open (i.e. not recursive) formulae known that only deliver prime numbers (recursive means that in order to calculate the function the same function is used with a smaller variable). Mathematicians would be happy if they could find a formula that leaves gaps (i.e. does not deliver all prime numbers) but does not deliver any composite (non-prime) numbers.

Ideally, we would like, for the number n , to immediately be able to obtain the n -th prime number, i.e. for $f(8) = 19$ or for $f(52) = 239$.

Ideas for this can be found at

http://www.utm.edu/research/primes/notes/faq/p_n.html.

Cross-reference: the table under 3.10 contains the precise values for the n th prime numbers for selected n .

For “prime number formulae” usually very special types of numbers are used. The following enumeration contains the most common ideas for “prime number formulae”, and what our current knowledge is about very big elements of the number series: Is their primality proven? If their are compound numbers could their prime factors be determined?

3.6.1 Mersenne numbers $f(n) = 2^n - 1$ for n prime

As shown above, this formula seems to deliver relatively large prime numbers but - as for $n = 11$ [$f(n) = 2,047$] - it is repeatedly the case that the result even with prime exponents is not prime. Today, all the Mersenne primes having less than around 4,000,000 digits are known (M-39):

<http://perso.wanadoo.fr/yves.gallot/primes/index.html>

3.6.2 Generalized Mersenne numbers $f(k, n) = k \cdot 2^n \pm 1$ for n prime and k small prime / Proth numbers¹⁸

This first generalization of the Mersenne numbers creates the so called Proth numbers. There are (for small k) extremely quick prime number tests (see [Knuth1981]). This can be performed in practice using software such as the Proths software from Yves Gallot:

¹⁸Their names come from the French farmer Franois Proth (1852-1879). More famous as the Proth primes is the related Sierpinski problem: Find all numbers k , so that $k \cdot 2^n + 1$ is composite for all $n \in \mathbb{N}$. See table 3.1.

<http://www.prothsearch.net/index.html>.

3.6.3 Generalized Mersenne numbers $f(b, n) = b^n \pm 1$ / The Cunningham project

This is another possible generalisation of the Mersenne numbers. The **Cunningham project** determines the factors of all composite numbers that are formed as follows:

$$f(b, n) = b^n \pm 1 \quad \text{for } b = 2, 3, 5, 6, 7, 10, 11, 12$$

(b is not equal to multiples of bases already used, such as 4, 8, 9).

Details of this can be found at:

<http://www.cerias.purdue.edu/homes/ssw/cun>

3.6.4 Fermat numbers¹⁹ $f(n) = 2^{2^n} + 1$

As mentioned above in chapter 3.5, Fermat wrote to Mersenne regarding his assumption, that all numbers of this type are primes. This assumption was disproved by Euler (1732). The prime 641 divides $f(5)$ ²⁰.

$f(0) = 2^{2^0} + 1 = 2^1 + 1$	$= 3$	\mapsto prime
$f(1) = 2^{2^1} + 1 = 2^2 + 1$	$= 5$	\mapsto prime
$f(2) = 2^{2^2} + 1 = 2^4 + 1$	$= 17$	\mapsto prime
$f(3) = 2^{2^3} + 1 = 2^8 + 1$	$= 257$	\mapsto prime
$f(4) = 2^{2^4} + 1 = 2^{16} + 1$	$= 65,537$	\mapsto prime
$f(5) = 2^{2^5} + 1 = 2^{32} + 1$	$= 4,294,967,297 = 641 \cdot 6,700,417$	\mapsto NOT prime!
$f(6) = 2^{2^6} + 1 = 2^{64} + 1$	$= 18,446,744,073,709,551,617$	
	$= 274,177 \cdot 67,280,421,310,721$	\mapsto NOT prime!
$f(7) = 2^{2^7} + 1 = 2^{128} + 1$	$=$ (see page 59)	\mapsto NOT prime!

Within the project “Distributed Search for Fermat Number Dividers” offered by Leonid Durman there is also progress in finding new monster primes:

<http://www.fermatsearch.org/>

This website links to other web pages in Russian, Italian and German.

The discovered factors can be compound integers or primes.

On February 22, 2003 John Cosgrave discovered

- the largest composite Fermat number to date and
- the largest prime non-simple Mersenne number so far with 645,817 decimal digits.

The Fermat number

$$f(2, 145, 351) = 2^{(2^{2,145,351})} + 1$$

¹⁹The Fermat prime numbers play a role in circle division. As proven by Gauss a regular p -edge can only be constructed with the use of a pair of compasses and a ruler, when p is a Fermat prime number.

²⁰Surprisingly this number can easily be found by using Fermat’s theorem (see e.g. [Scheid1994, p. 176])

is divisible by the prime

$$p = 3 * 2^{2,145,353} + 1$$

At that time this prime p was the largest known prime generalized Mersenne number and the 5th largest known prime number at all.

This work was done using NewPGen from Paul Jobling's, PRP from George Woltman's, Proth from Yves Gallot's programs and also the Proth-Gallot group at St. Patrick's College, Dublin.

More details are in

http://www.fermatsearch.org/history/cosgrave_record.htm/

3.6.5 Generalized Fermat numbers²¹ $f(b, n) = b^{2^n} + 1$

Generalized Fermat numbers are more numerous than Mersenne numbers of a equal size and many of them are waiting to be discovered to fill the big gaps between the Mersenne primes already found or still undiscovered. Progress in number theory made it possible that numbers, where the representation is not limited to the base 2, can be tested at almost the same speed than a Mersenne number.

Yves Gallot wrote the program Proth.exe to investigate generalized Fermat numbers.

Using this program at February 16, 2003 Michael Angel discovered the largest of them till then with 628,808 digits, which at that time became the 5th largest known prime number:

$$b^{2^{17}} + 1 = 62,722^{131,072} + 1.$$

More details are in

<http://primes.utm.edu/top20/page.php?id=12>

3.6.6 Carmichael numbers

As mentioned above in chapter 3.5 not all Carmichael numbers are prime.

3.6.7 Pseudo prime numbers

See above in chapter 3.5.

3.6.8 Strong pseudo prime numbers

See above in chapter 3.5.

²¹The base of this power is no longer restricted to 2. Even more generic would be: $f(b, c, n) = b^{c^n} \pm 1$

3.6.9 Idea based on Euclid's proof $p_1 \cdot p_2 \cdots p_n + 1$

This idea is based on Euclid's proof that there are infinite many prime numbers.

$$\begin{array}{lll}
2 \cdot 3 + 1 & = 7 & \mapsto \text{prime} \\
2 \cdot 3 \cdot 5 + 1 & = 31 & \mapsto \text{prime} \\
2 \cdot 3 \cdot 5 \cdot 7 + 1 & = 211 & \mapsto \text{prime} \\
2 \cdot 3 \cdots 11 + 1 & = 2,311 & \mapsto \text{prime} \\
2 \cdot 3 \cdots 13 + 1 & = 59 \cdot 509 & \mapsto \text{NOT prime!} \\
2 \cdot 3 \cdots 17 + 1 & = 19 \cdot 97 \cdot 277 & \mapsto \text{NOT prime!}
\end{array}$$

3.6.10 As above but -1 except $+1$: $p_1 \cdot p_2 \cdots p_n - 1$

$$\begin{array}{lll}
2 \cdot 3 - 1 & = 5 & \mapsto \text{prime} \\
2 \cdot 3 \cdot 5 - 1 & = 29 & \mapsto \text{prime} \\
2 \cdot 3 \cdots 7 - 1 & = 11 \cdot 19 & \mapsto \text{NOT prime!} \\
2 \cdot 3 \cdots 11 - 1 & = 2,309 & \mapsto \text{prime} \\
2 \cdot 3 \cdots 13 - 1 & = 30,029 & \mapsto \text{prime} \\
2 \cdot 3 \cdots 17 - 1 & = 61 \cdot 8,369 & \mapsto \text{NOT prime!}
\end{array}$$

3.6.11 Euclidean numbers $e_n = e_0 \cdot e_1 \cdots e_{n-1} + 1$ with $n \geq 1$ and $e_0 := 1$

e_{n-1} is not the $(n-1)$ th prime number, but the number previously found here. Unfortunately this formula is not open but recursive. The sequence starts with

$$\begin{array}{lll}
e_1 = 1 + 1 & = 2 & \mapsto \text{prime} \\
e_2 = e_1 + 1 & = 3 & \mapsto \text{prime} \\
e_3 = e_1 \cdot e_2 + 1 & = 7 & \mapsto \text{prime} \\
e_4 = e_1 \cdot e_2 \cdot e_3 + 1 & = 43 & \mapsto \text{prime} \\
e_5 = e_1 \cdot e_2 \cdots e_4 + 1 & = 13 \cdot 139 & \mapsto \text{NOT prime!} \\
e_6 = e_1 \cdot e_2 \cdots e_5 + 1 & = 3,263,443 & \mapsto \text{prime} \\
e_7 = e_1 \cdot e_2 \cdots e_6 + 1 & = 547 \cdot 607 \cdot 1,033 \cdot 31,051 & \mapsto \text{NOT prime!} \\
e_8 = e_1 \cdot e_2 \cdots e_7 + 1 & = 29,881 \cdot 67,003 \cdot 9,119,521 \cdot 6,212,157,481 & \mapsto \text{NOT prime!}
\end{array}$$

Also e_9, \dots, e_{17} are composite, which means that this formula is not particularly useful.

Comment:

However, what is special about these numbers is that any pair of them does not have a common factor other than 1^{22} . Therefore they are *relatively prime*.

²²This can easily be shown via the following rule for the *greatest common divisor* \gcd with $\gcd(a, b) = \gcd(b - \lfloor b/a \rfloor \cdot a, a)$.

We have for $i < j$:

$\gcd(e_i, e_j) \leq \gcd(e_1 \cdots e_i \cdots e_{j-1}, e_j) = \gcd(e_j - e_1 \cdots e_i \cdots e_{j-1}, e_1 \cdots e_i \cdots e_{j-1}) = \gcd(1, e_1 \cdots e_i \cdots e_{j-1}) = 1$.
See page 138.

3.6.12 $f(n) = n^2 + n + 41$

This sequence starts off very *promisingly*, but is far from being a proof.

$f(0) = 41$	\mapsto prime
$f(1) = 43$	\mapsto prime
$f(2) = 47$	\mapsto prime
$f(3) = 53$	\mapsto prime
$f(4) = 61$	\mapsto prime
$f(5) = 71$	\mapsto prime
$f(6) = 83$	\mapsto prime
$f(7) = 97$	\mapsto prime
\vdots	
$f(33) = 1,163$	\mapsto prime
$f(34) = 1,231$	\mapsto prime
$f(35) = 1,301$	\mapsto prime
$f(36) = 1,373$	\mapsto prime
$f(37) = 1,447$	\mapsto prime
$f(38) = 1,523$	\mapsto prime
$f(39) = 1,601$	\mapsto prime
$f(40) = 1681 = 41 \cdot 41$	\mapsto NOT prime!
$f(41) = 1763 = 41 \cdot 43$	\mapsto NOT prime!

The first 40 values are prime numbers (which have the obvious regularity that their difference starts with 2 and increases by 2 each time), but the 41th and 42th values are not prime numbers. It is easy to see that $f(41)$ cannot be a prime number: $f(41) = 41^2 + 41 + 41 = 41(41 + 1 + 1) = 41 \cdot 43$.

3.6.13 $f(n) = n^2 - 79 \cdot n + 1,601$

This function²³ delivers prime numbers for all values from $n = 0$ to $n = 79$. Unfortunately $f(80) = 1,681 = 11 \cdot 151$ is not a prime number. To this date, no function has been found that delivers more prime numbers in a row. On the other hand, each prime occurs twice (first in the decreasing then in the increasing sequence), which means that the algorithm delivers a total of 40 different prime values (these are the same ones as delivered by the function in chapter 3.6.12)²⁴.

²³See Appendix A of this chapter for the source code to compute the table using Sage.

²⁴Another quadratic polynom, which delivers these primes, is: $f(n) = n^2 - 9 \cdot n + 61$.

Among the first 1000 sequence elements more than 50% are prime (See Appendix A of this chapter).

$f(0) = 1.601 \mapsto \text{prim}$	$f(26) = 223 \mapsto \text{prim}$
$f(1) = 1.523 \mapsto \text{prim}$	$f(27) = 197 \mapsto \text{prim}$
$f(2) = 1.447 \mapsto \text{prim}$	$f(28) = 173 \mapsto \text{prim}$
$f(3) = 1.373 \mapsto \text{prim}$	$f(29) = 151 \mapsto \text{prim}$
$f(4) = 1.301 \mapsto \text{prim}$	$f(30) = 131 \mapsto \text{prim}$
$f(5) = 1.231 \mapsto \text{prim}$	$f(31) = 113 \mapsto \text{prim}$
$f(6) = 1.163 \mapsto \text{prim}$	$f(32) = 97 \mapsto \text{prim}$
$f(7) = 1.097 \mapsto \text{prim}$	$f(33) = 83 \mapsto \text{prim}$
$f(8) = 1.033 \mapsto \text{prim}$	$f(34) = 71 \mapsto \text{prim}$
$f(9) = 971 \mapsto \text{prim}$	$f(35) = 61 \mapsto \text{prim}$
$f(10) = 911 \mapsto \text{prim}$	$f(36) = 53 \mapsto \text{prim}$
$f(11) = 853 \mapsto \text{prim}$	$f(37) = 47 \mapsto \text{prim}$
$f(12) = 797 \mapsto \text{prim}$	$f(38) = 43 \mapsto \text{prim}$
$f(13) = 743 \mapsto \text{prim}$	$f(39) = 41 \mapsto \text{prim}$
$f(14) = 691 \mapsto \text{prim}$	$f(40) = 41 \mapsto \text{prim}$
$f(15) = 641 \mapsto \text{prim}$	$f(41) = 43 \mapsto \text{prim}$
$f(16) = 593 \mapsto \text{prim}$	$f(42) = 47 \mapsto \text{prim}$
$f(17) = 547 \mapsto \text{prim}$	$f(43) = 53 \mapsto \text{prim}$
$f(18) = 503 \mapsto \text{prim}$	\dots
$f(19) = 461 \mapsto \text{prim}$	$f(77) = 1.447 \mapsto \text{prim}$
$f(20) = 421 \mapsto \text{prim}$	$f(78) = 1.523 \mapsto \text{prim}$
$f(21) = 383 \mapsto \text{prim}$	$f(79) = 1.601 \mapsto \text{prim}$
$f(22) = 347 \mapsto \text{prim}$	$f(80) = 41 \cdot 41 \mapsto \text{NOT prim!}$
$f(21) = 383 \mapsto \text{prim}$	$f(81) = 41 \cdot 43 \mapsto \text{NOT prim!}$
$f(22) = 347 \mapsto \text{prim}$	$f(82) = 1.847 \mapsto \text{prim}$
$f(23) = 313 \mapsto \text{prim}$	$f(83) = 1.933 \mapsto \text{prim}$
$f(24) = 281 \mapsto \text{prim}$	$f(84) = 43 \cdot 47 \mapsto \text{NOT prim!}$
$f(25) = 251 \mapsto \text{prim}$	

3.6.14 Polynomial functions $f(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x^1 + a_0$ (a_i in \mathbb{Z} , $n \geq 1$)

There exists no such polynomial that for all x in \mathbb{Z} only delivers prime values. For a proof of this, please refer to [Padberg1996, p. 83 f.], where you will also find further details about prime number formulae.

This means there is no hope in looking for further formulae (functions) similar to that in chap. 3.6.12 or chap. 3.6.13.

3.6.15 Catalan's conjecture²⁵

Catalan conjectured that C_4 is a prime:

$$\begin{aligned}C_0 &= 2, \\C_1 &= 2^{C_0} - 1, \\C_2 &= 2^{C_1} - 1, \\C_3 &= 2^{C_2} - 1, \\C_4 &= 2^{C_3} - 1, \dots\end{aligned}$$

(see <http://www.utm.edu/research/primes/merzenne.shtml> under “Conjectures and Unsolved Problems”).

This sequence is also defined recursively and increases extremely quickly. Does it only consist of primes?

$$\begin{array}{ll}C(0) = 2 & \mapsto \text{prime} \\C(1) = 2^2 - 1 = 3 & \mapsto \text{prime} \\C(2) = 2^3 - 1 = 7 & \mapsto \text{prime} \\C(3) = 2^7 - 1 = 127 & \mapsto \text{prime} \\C(4) = 2^{127} - 1 = 170, 141, 183, 460, 469, 231, 731, 687, 303, 715, 884, 105, 727 & \mapsto \text{prime}\end{array}$$

It is not (yet) known whether C_5 and all higher elements are prime, but this is not very likely. In any case, it has not been proved that this formula delivers only primes.

3.7 Density and distribution of the primes

As Euclid discovered, there is an infinite number of primes. However, some infinite sets are *denser* than others.

Within the set of natural numbers, there is an infinite number of even, uneven and square numbers. How to compare the “density” of two infinite sets is shown with even and square numbers.

The following proves that the even numbers are distributed more densely than square ones:²⁶

- the size of the n th element:
The n th element of the even numbers is $2n$; the n th element of the square numbers is n^2 . Because for all $n > 2$: $2n < n^2$, the n th even number occurs much earlier than the n th square number.
- the numbers of values that are less than or equal to a certain *maximum value* x in \mathbb{R} are:
There are $\lfloor x/2 \rfloor$ such even numbers and $\lfloor \sqrt{x} \rfloor$ square numbers. Because for all $x > 6$ the value $\lfloor x/2 \rfloor$ is greater than the largest integer smaller or equal to the square root of x , the even numbers are distributed more densely.

²⁵Eugene Charles Catalan, Belgian mathematician, May 5, 1814–Feb 14, 1894.

After him the so-called *Catalan numbers* $A(n) = (1/(n+1)) * (2n)!/(n!)^2$
= 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, ... are named.

²⁶Whereas in colloquial language you often can hear, that “there are more” even numbers than square ones, mathematicians say, that from both there are infinitely many, that their sets are equivalent to \mathbb{N} (so both are infinite and countable, i.e. one can assign to each even number and to each square number an integer), but that the set of even numbers is denser than the set of square numbers.

The value of the n -th prime $P(n)$

Theorem 3.7.1. *For large n : The value of the n -th prime $P(n)$ is asymptotic to $n \cdot \ln(n)$, i.e. the limit of the relation $P(n)/(n \cdot \ln n)$ is equal to 1 if n tends to infinity.*

For $n > 5$, $P(n)$ lies between $2n$ and n^2 . This means that there are fewer prime numbers than even natural numbers but more prime numbers than square numbers²⁷.

The number of prime numbers $PI(x)$

The definition is similar for the number of prime numbers $PI(x)$ that do not exceed the maximum value x :

Theorem 3.7.2. *$PI(x)$ is asymptotic to $x/\ln(x)$.*

This is the **prime number theorem**. It was put forward by Legendre²⁸ and Gauss²⁹ but not proved until over 100 years later.

Cross-reference: The overview under 3.9 shows the number of prime numbers in various intervals.

These formulae, which only apply when n tends to infinity, can be replaced by more precise formulae. For $x \geq 67$:

$$\ln(x) - 1,5 < x/PI(x) < \ln(x) - 0,5$$

Given that we know $PI(x) = x/\ln x$ only for very large x (x tending towards infinity), we can create the following overview:

x	$\ln(x)$	$x/\ln(x)$	$PI(x)$ (counted)	$PI(x)/(x/\ln(x))$
10^3	6.908	144	168	1.160
10^6	13.816	72,386	78,498	1.085
10^9	20.723	48,254,942	50,847,534	1.054

For a binary number³⁰ x of the length of 250 bits (2^{250} is approximately $= 1.809251 \cdot 10^{75}$) it is:

$$PI(x) = 2^{250}/(250 \cdot \ln 2) \text{ is approximately } = 2^{250}/173.28677 = 1.045810 \cdot 10^{73}.$$

We can therefore expect that the set of numbers with a bit length of less than 250 contains approximately 10^{73} primes (a reassuring result?!).

We can also express this as follows: Let us consider a *random* natural number n . Then the probability that this number is prime is around $1/\ln(n)$. For example, let us take numbers in the region of 10^{16} . Then we must consider $16 \cdot \ln 10 = 36,8$ numbers (on average) until we find a prime. A precise investigation shows: There are 10 prime numbers between $10^{16} - 370$ and $10^{16} - 1$.

Under the heading *How Many Primes Are There at*

<http://www.utm.edu/research/primes/howmany.shtml>

²⁷Please refer to the table 3.10

²⁸Adrien-Marie Legendre, French mathematician, Sep 18, 1752 – Jan 10, 1833.

²⁹Carl Friedrich Gauss, German mathematician and astronomer, Apr 30, 1777–Feb 23, 1855.

³⁰Number written in the binary system consists only of the digits 0 and 1.

you will find numerous other details.

Using the following Web site:

<http://www.math.Princeton.EDU/~arbooker/nthprime.html>

you can easily determine $PI(x)$.

The **distribution** of primes displays several irregularities for which no “system” has yet been found: On the one hand, many occur closely together, like 2 and 3, 11 and 13, 809 and 811, on the other hand large gaps containing no primes also occur. For example, no primes lie between 113 and 127, 293 and 307, 317 and 331, 523 and 541, 773 and 787, 839 and 853 as well as between 887 and 907.

For details, please see:

<http://www.utm.edu/research/primes/notes/gaps.html>

This is precisely part of what motivates mathematicians to discover its secrets.

Sieve of Eratosthenes

An easy way of calculating all $PI(x)$ primes less than or equal to x is to use the sieve of Eratosthenes. In the 3rd century B.C., he found an extremely easy, automatic way of finding this out. To begin with, you write down all numbers from 2 to x , circle 2, then cross out all multiples of 2. Next, you circle the lowest number that hasn't been circled or crossed out (3) and again cross out all multiples of this number, etc. You only need to continue until you reach the largest number whose square is less than or equal to x .³¹

Apart from 2, prime numbers are never even. Apart from 2 and 5, prime numbers never end in 2, 5 or 0. So you only need to consider numbers ending in 1, 3, 7, 9 anyway (there are infinite primes ending in these numbers; see [Tietze1973, vol. 1, p. 137]).

You can now find a large number of finished programs on the Internet - often complete with source code - allowing you to experiment with large numbers yourself (see chapter 3.6). You also have access to large databases that contain either a large number of primes or the factorization of numerous composite numbers.

Further interesting topics regarding prime numbers

This chapter 3 didn't consider other number theory topics such as divisibility rules, modulus calculation, modular inverses, modular powers, modular roots, Chinese remainder theorem, Euler Phi function or perfect numbers. Some of these topics are considered in the **next chapter** (chapter 4).

³¹With the educational tool for number theory **NT** you can apply the sieve of Eratosthenes in a computer-aided and guided way: Enter you own number and do the sieving step by step: See learning unit 1.2, pages 6/21 and 7/21.

NT can be called in CrypTool via the menu path **Indiv. Procedures \ Number Theory Interactive \ Learning Tool for Number Theory**. See appendix A.4.

3.8 Notes about primes

The following notes list some interesting theorems, conjectures and open questions about primes, but also some quaint things and overviews.

3.8.1 Proven statements / theorems about primes

- For each number n in \mathbf{N} there are n consecutive natural numbers that are not primes. A proof of this can be found in [Padberg1996, p. 79].
- Paul Erdős³² proved: Between each random number not equal to 1 and its double, there is at least one prime. He was not the first to prove this theorem, but proved it in a much simpler manner than those before him.
- There is a real number a such that the function $f : \mathbf{N} \rightarrow \mathbb{Z}$ where $n \mapsto a^{3^n}$ only delivers primes for all n (see [Padberg1996, p. 82]). Unfortunately, problems arise when we try to determine a (see below).
- There are arithmetic prime sequences of arbitrary length³³.

In 1923 the famous British mathematician Godfrey Harold Hardy³⁴ compiled the conjecture, that there are arithmetic sequences of arbitrary length, which consist of primes only. This conjecture was proven in 2004 by two young American mathematicians.

At some point every school child learns about arithmetic number series. These are sequences of numbers, for which the difference between any 2 consecutive numbers is equal or constant (an arithmetic sequence must have at least three elements but can also have indefinitely many). In the sample sequence 5, 8, 11, 14, 17, 20 the difference between the series's elements is 3 and the length of the sequence is 6.

Arithmetic series have been known for millennia and one would think they have no more secrets. They get more interesting again, if we impose additional constraints on the series elements - as the prime example shows.

E.g. 5, 17, 29, 41, 53 is an arithmetic prime series which consists of 5 elements and the difference between the elements is always 12.

The sequence is not extendable - the next would be 65, but 65 is not prime (65 is the product of 5 and 13).

How many elements are possible within an arithmetic prime number sequence? Around 1770 the French Joseph-Louis Lagrange and the British Edward Waring investigated this question. In 1923 the famous British mathematician Godfrey Harold Hardy and his colleague John Littlewood theorized, that there is no upper limit for the number of elements. But they could not prove this. In 1939 more progress was achieved. The Dutch mathematician Johannes van der Corput was able to prove that there are infinitely many different arithmetic prime number sequences with exactly three elements. Two examples are 3, 5, 7 and 47, 53, 59.

The longest arithmetic prime number sequence known today contains 23 elements.

³²Paul Erdős, Hungarian mathematician, Mar 26, 1913–Sep 20, 1996.

³³Sources:

- <http://primes.utm.edu/glossary/page.php?sort=ArithmeticSequence> Original source

- German magazine GEO 10 / 2004: "Experiment mit Folgen"

- <http://www.faz.net> "Hardys Vermutung – Primzahlen ohne Ende" by Heinrich Hemme (July 06, 2004)

³⁴Godfrey Harold Hardy, British mathematician, Feb 7, 1877–Dec 1, 1947.

Elements	First element	Distance	When	Discovered by
22	11,410,337,850,553	4,609,098,694,200	1993	Paul A. Pritchard, Andrew Moran, Anthony Thyssen
22	376,859,931,192,959	18,549,279,769,020	2003	Markus Frind
23	56,211,383,760,397	44,546,738,095,860	2004	Markus Frind, Paul Jobling, Paul Underwood

Table 3.3: The longest arithmetic prime number sequences (as of May 2005)

As a team, the two young³⁵ mathematicians Ben Green and Terence Tao, were able in 2004 to prove Hardy’s conjecture, which had puzzled mathematicians for over 80 years: It states, that for any arbitrary length there exists an arithmetic prime number series. Additionally they managed to prove, that for any given length there are infinitely many different series.

Green and Tao intended to proof that there are infinitely many arithmetic sequences of length four. For this they considered sets of numbers consisting of primes and so called “near primes”. These are numbers with a small set of divisors like numbers which are the product of two primes - these numbers are called “half primes” . Thus they managed to considerably simplify their work because about near primes there already existed a lot of useful theorems. Finally they discovered that the results of their theorem were far more reaching than they had assumed and so they were able to prove Hardy’s conjecture.

Any one who believes that it is easy to use Green’s and Tao’s 49 page proof to compute arithmetic prime number series of arbitrary length will soon become disappointed, because the proof is non-constructive. It is a so called proof of existence. This means that these mathematicians have shown “only” that these series exist, but not how to find them in practice.

This means that in the set of the natural numbers there is e.g. a series of one billion primes, which all have the same distance; and there are infinitely many of them. But these sequences lie extremely far beyond the numbers we usually use (“far outside”).

If someone wants to discover such sequences he should consider the following thought. The length of a sequence determines the minimal common distance between the single primes of the sequence. Given a sequence with 6 elements the distance between them has to be 30 or a multiple of 30. The number 30 results as the product of all primes smaller than the length of the sequence. So its the product of all primes smaller than 6: $2 * 3 * 5 = 30$. If you look for a sequence with 15 elements, then the common distance is at least $2 * 3 * 5 * 7 * 11 * 13 = 30.030$.

This means that the length of an arithmetic prime sequence can be arbitrary big, but the distance between the elements cannot be any arbitrary number. E.g. there is no arithmetic prime sequence with the distance 100, because 100 cannot be divided by 3.

³⁵Hardy wrote in his memoirs in 1940, that mathematics - more than all other arts and sciences - is an activity for young people.

At that time 27-years-old Ben Green from the University of British Columbia in Vancouver and 29-year-old Terence Tao from the University of California in Los Angeles seem to confirm Hardy.

If you take the sequences above (with the lengths of 22 and 23) and look at the factors of their distances, you get:

$$4,609,098,694,200 = 2^3 * 3 * 5^2 * 7 * 11 * 13 * 17 * 19 * 23 * 1033$$

$$18,549,279,769,020 = 2^2 * 3 * 5 * 7^2 * 11 * 13 * 17 * 19 * 23 * 5939$$

$$44,546,738,095,860 = 2^2 * 3 * 5 * 7 * 11 * 13 * 17 * 19 * 23 * 99,839$$

Further restriction: If you look at arithmetic prime sequences, which fulfill the *additional* requirement, that all primes are consecutive, then its getting even more complicated. At the website of Chris Caldwell³⁶ you can find further hints: the longest known arithmetic prime sequence, consisting only of directly consecutive primes, has a length of 10 and the distance is $210 = 2 * 3 * 5 * 7$.

3.8.2 Unproven statements / conjectures about primes

- Christian Goldbach³⁷ conjectured: Every even natural number greater than 2 can be represented as the sum of two prime numbers. Computers have verified³⁸ the Goldbach conjecture for all even numbers up to $15 * 10^{17}$ (as at July 2009) but no general proof has yet been found³⁹.
- Bernhard Riemann⁴⁰ put forward a formula for the distribution of primes that would further improve the estimate. However, this has neither been proved nor disproved so far.

³⁶<http://primes.utm.edu/glossary/page.php?sort=ArithmeticSequence>

³⁷Christian Goldbach, German mathematician, Mar 18, 1690–Nov 20, 1764.

³⁸It is generally accepted today, that the **Goldbach Conjecture** is true, i. e. valid for all even natural numbers greater than 2. In 1999, mathematician Jörg Richstein from the computer sciences institute at the University of Giessen, studied even numbers up to 400 billion ($4 * 10^{14}$) and found no contradictory example ([Richstein1999]).

In the meantime further progress was made: see

<http://www.ieeta.pt/~tos/goldbach.html>,

http://en.wikipedia.org/wiki/Goldbach's_conjecture,

<http://primes.utm.edu/glossary/page.php/GoldbachConjecture.html>.

Nevertheless, this does not provide us with general proof.

The fact is that despite all efforts, Goldbach's conjecture has to date not been proven. This leads one to believe that since the pioneer work of the Austrian mathematician Kurt Gödel is well-known, not every true mathematical theorem is provable (see <http://www.mathematik.ch/mathematiker/goedel.html>). Perhaps Goldbach's conjecture was correct, but in any case the proof will never be found. Conversely, that will presumably also remain unproven.

³⁹The English publisher *Faber* and the American publisher *Bloomsbury* issued in 2000 the 1992 published book "Uncle Petros and Goldbach's Conjecture" by Apostolos Doxiadis. It's the story of an old maths professor who fails to prove a more than 250 year old puzzle. To boost the sales figures the English and American publishers have offered a prize of 1 million USD, if someone can prove the conjecture – which should be published by 2004 in a well-known mathematical journal.

Surprisingly only British and American citizens are allowed to participate.

The theorem which has come closest so far to Goldbach's conjecture was proved by Chen Jing-Run in 1966 in a way which is somewhat hard to understand: Each even integer greater than 2 is the sum of one prime and of the product of two primes. E.g.: $20 = 5 + 3 * 5$.

Most of the research about the Goldbach conjecture is collected in the book: "Goldbach Conjecture", ed. Wang Yuan, 1984, World scientific Series in Pure Maths, Vol. 4.

Especially this conjecture makes it clear, that even today we do not have a complete understanding of the deeper connections between addition and multiplication of natural numbers.

⁴⁰Bernhard Riemann, German mathematician, Sep 17, 1826–Jul 20, 1866.

- Benford's law⁴¹ does not apply to primes.

According to Benford's law, also called the first-digit law, the single digits in lists of numbers from many (but not all) real-life sources of data, are distributed in a non-uniform way. Especially the leading digit is much more often the digit 1 than any other digit.

Which empirical data applies to this "law" is not completely clear yet. Timo Eckhardt analyzed in his thesis in 2008 extensively attributes of prime numbers. Among other things all primes until 7,052,046,499 were described with different bases of the positional notation.

Comparing the bases 3 to 10 the deviation from Benford's law was lowest with base 3. Comparing the first digit for base 10 all digits are almost equally distributed. Analyzing bigger bases showed strong differences.

3.8.3 Open questions about twin primes

Twin primes are prime numbers whose difference is 2. Examples include 5 and 7, or 101 and 103, or $1,693,965 \cdot 2^{66,443} \pm 1$.

The biggest known twin pair nowadays is

$$65,516,468,355 \cdot 2^{333,333} \pm 1$$

with 100,355 decimal digits.⁴²

Remark: Triplet primes, however, only occur once: 3, 5, 7. For all other sets of three consecutive uneven numbers, one of them is always divisible by 3 and thus not a prime.

Open questions are:

- The number of twin primes: infinitely many or a limited number?
- Does a formula exist for calculating the number of twin primes per interval?

A big step towards the solution of this problem was made by Dan Goldston, János Pintz and Cem Yildirim in 2003⁴³. The three mathematicians were investigating the distribution of prime numbers. They could proof, that

$$\liminf_{n \rightarrow \infty} \frac{p_{n+1} - p_n}{\log p_n} = 0,$$

where p_n denotes the n -th prime number.

This means that the smallest limit point (\liminf) of the sequence $\frac{p_{n+1} - p_n}{\log p_n}$ equals zero.

⁴¹http://en.wikipedia.org/wiki/Benford%27s_law,
http://arxiv.org/PS_cache/arxiv/pdf/0906/0906.2789v1.pdf.

⁴²<http://primes.utm.edu>, October 2009

⁴³D. A. Goldston: "Gaps Between Primes"
<http://www.math.sjsu.edu/~goldston/OberwolfachAbstract.pdf>
 See also:

- D. A. Goldstone: "Are There Infinitely Many Twin Primes?"
<http://www.math.sjsu.edu/~goldston/twinprimes.pdf>
- K. Soundararajan: "Small Gaps Between Prime Numbers: The Work Of Goldston-Pintz-Yildirim"
<http://www.ams.org/bull/2007-44-01/S0273-0979-06-01142-6/S0273-0979-06-01142-6.pdf>

A point is called limit point of a sequence, if there lie in any arbitrarily small neighbourhood of that point infinitely many elements of the sequence.

$\log p_n$ is about the average distance between the prime p_n and the next prime p_{n+1} .

Hence, the term above implies, that there are infinitely many consecutive primes with a gap between them which is arbitrarily small compared to the expected average gap.

Moreover, it was proofed, that

$$p_{n+1} - p_n < (\log p_n)^{8/9}$$

holds true for infinitely many primes⁴⁴.

Those results could be the basis for the proof, that infinitely many twin primes exist.

3.8.4 Further open questions

- The above proof of the function $f : N \rightarrow Z$ with $n \mapsto a^{3^n}$ only guarantees the existence of such a number a . How can we determine this number a and will it have a value, making the function also of some practical interest?
- Is there an infinite number of Mersenne prime numbers?
- Is there an infinite number of Fermat prime numbers?
- Does a polynomial time algorithm exist for calculating the prime factors of a number (see [Klee1997, p. 167])? This question can be divided into the three following questions:
 - Does a polynomial time algorithm exist that decides whether a number is prime? This question has been answered by the AKS algorithm (see chapter 4.11.5, “Primes in P”: Primality testing is polynomial).
 - Does a polynomial time algorithm exist that calculates for a composite number from how many prime factors it is made up (without calculating these factors)?
 - Does a polynomial time algorithm exist that calculates for a composite number n a non-trivial (i.e. other than 1 and n) factor of n ?⁴⁵

At the end of chapter 4.11.4, section RSA-200 you can see the dimensions of the numbers where the current algorithms testing for primality and calculating the factorization deliver results.

3.8.5 Quaint and interesting things around primes⁴⁶

Primes are not only a very active and serious research area in mathematics. Also a lot of people think about them in their free time and outside the scientific research.

⁴⁴c't magazine 2003, no. 8, page 54

⁴⁵Please compare chapters 4.11.5 and 4.11.4.

⁴⁶Further curious things about primes may be found at:

- <http://primes.utm.edu/curios/home.php>

- <http://www.primzahlen.de/files/theorie/index.htm>.

Recruitment at Google in 2004

In summer 2004 the company Google used the number e ⁴⁷ to attract potential employees⁴⁸.

On a prominent billboard in California's Silicon Valley on July 12 there appeared the following mysterious puzzle:

(first 10 digit prime in consecutive digits of e).com

Finding the first 10 digit prime in the decimal expansion of e is not easy, but with various software tools, one can determine that the answer is

7, 427, 466, 391

Then if you visited the website *www.7427466391.com*, you were presented with an even more difficult puzzle. Figuring this second puzzle out took you to a web page that asks you, to submit your CV to Google. The ad campaign got high attention.

Presumably Google's conceit was that if you're smart enough to solve the puzzles, you're smart enough to work for them. Of course some days after the launch, anyone who really wanted to discover the answers without incurring a headache could merely do a Google search for them, since many solvers immediately posted their solutions online.⁴⁹

Contact [Directed by Robert Zemeckis, 1997] – Primes helping to contact aliens

The movie originated from Carl Sagan's book with the same title.

After years of unavailing search the radio astronomer Dr. Ellie Arroway (Jodie Foster) discovers signals from the solar system Vega, 26 light years away. These signals contain the primes in the right order and without a gap. This makes the hero confident, that this message is different from the radio signals which permanently hit earth and which are random and of cosmic origin (radio galaxies, pulsars). In an unmasking scene a politician asks her after that, why these intelligent aliens didn't just speak English ...

Doing communication with absolute strange and unknown beings from deep space is very hard especially because of 2 reasons: First the big distance and therefore the long transfer time makes it impossible to exchange within an average lifetime more than one message in each direction one after the other. Secondly the first contact must give the receiver of the radio signals a good chance to notice the message and to categorize it as something from intelligent beings. Therefore the aliens send numbers at the beginning of their message, which can be considered as the easiest part of any higher language, and which are not too trivial: so they chose the sequence of primes. These special numbers play such a fundamental role in mathematics that

⁴⁷The base of the natural logarithm e is approximately 2.718 281 828 459. This is one of the most important numbers in all of mathematics like complex analysis, finance, physics and geometry. Now it was used the first time – as far as I know – for marketing or recruitment.

⁴⁸Most of this information is taken from the article “e-number crunching” by John Allen Paulos in TheGuardian, Sept. 30, 2004, and from the web:

- <http://www.mkaz.com/math/google/>
- <http://epramono.blogspot.com/2004/10/7427466391.html>
- <http://mathworld.wolfram.com/news/2004-10-13/google/>
- <http://www.math.temple.edu/~paulos/>

⁴⁹The second level of the puzzle, which involved finding the 5th term of a given number sequence had nothing to do with primes any more.

one can assume that they are well known to each species who has the technical know-how to receive radio waves.

The aliens then send a plan to build a mysterious machine ...

3.9 Appendix: Number of prime numbers in various intervals

Ten-sized intervals		Hundred-sized intervals		Thousand-sized intervals	
Interval	Number	Interval	Number	Interval	Number
1-10	4	1-100	25	1-1000	168
11-20	4	101-200	21	1001-2000	135
21-30	2	201-300	16	2001-3000	127
31-40	2	301-400	16	3001-4000	120
41-50	3	401-500	17	4001-5000	119
51-60	2	501-600	14	5001-6000	114
61-70	2	601-700	16	6001-7000	117
71-80	3	701-800	14	7001-8000	107
81-90	2	801-900	15	8001-9000	110
91-100	1	901-1000	14	9001-10000	112

Table 3.4: How many primes exist within the first intervals of tens?

Interval	Number	Average number per 1000
1 - 10,000	1,229	122.900
1 - 100,000	9,592	95.920
1 - 1,000,000	78,498	78.498
1 - 10,000,000	664,579	66.458
1 - 100,000,000	5,761,455	57.615
1 - 1,000,000,000	50,847,534	50.848
1 - 10,000,000,000	455,052,512	45.505

Table 3.5: How many primes exist within the first intervals of dimensions?

3.10 Appendix: Indexing prime numbers (n -th prime number)

Index	Precise value	Rounded value	Comment
1	2	2	
2	3	3	
3	5	5	
4	7	7	
5	11	11	
6	13	13	
7	17	17	
8	19	19	
9	23	23	
10	29	29	
100	541	541	All prime numbers up to 1E+07 were known at the beginning of the 20th century.
1,000	7,917	7,917	
664,559	9,999,991	9.99999E+06	
1E+06	15,485,863	1.54859E+07	
6E+06	104,395,301	1.04395E+08	
1E+07	179,424,673	1.79425E+08	
1E+09	22,801,763,489	2.28018E+10	
1E+12	29,996,224,275,833	2.99962E+13	
			This prime was discovered in 1959.

Table 3.6: List of particular n -th prime numbers

Comment:

With gaps, extremely large prime numbers were discovered at an early stage.

Web links:

<http://www.math.Princeton.EDU/~arbooker/nthprime.html>

http://www.utm.edu/research/primes/notes/by_year.html.

3.11 Appendix: Orders of magnitude / dimensions in reality

In the description of cryptographic protocols and algorithms, numbers occur that are so large or so small that they are inaccessible to our intuitive understanding. It may therefore be useful to provide comparative numbers from the real world around us so that we can develop a feeling for the security of cryptographic algorithms. Some of the numbers listed below originate from [Schwenk1996] and [Schneier1996, p.18].

Probability that you will be hijacked on your next flight	$5.5 \cdot 10^{-6}$
Annual probability of being hit by lightning	10^{-7}
Probability of 6 correct numbers in the lottery	$7.1 \cdot 10^{-8}$
Risk of being hit by a meteorite	$1.6 \cdot 10^{-12}$
Time until the next ice age (in years)	$14,000 = (2^{14})$
Time until the sun dies (in years)	$10^9 = (2^{30})$
Age of the earth (in years)	$10^9 = (2^{30})$
Age of the universe (in years)	$10^{10} = (2^{34})$
Number of molecules within one waterdrop	$10^{20} = (2^{63})$
Number of bacteria living on earth	$10^{30.7} = (2^{102})$
Number of the earth's atoms	$10^{51} = (2^{170})$
Number of the sun's atoms	$10^{57} = (2^{190})$
Number of atoms in the universe (without dark material)	$10^{77} = (2^{265})$
Volume of the universe (in cm^3)	$10^{84} = (2^{280})$

Table 3.7: Likelihoods and dimensions from physics and everyday life

3.12 Appendix: Special values of the binary and decimal system

These values can be used to evaluate from a key length in bit the corresponding number of possible keys and the search effort (if assumed, that e.g. one million keys can be tested within one second).

Binary system	Decimal system
2^{10}	1024
2^{40}	$1.09951 \cdot 10^{12}$
2^{56}	$7.20576 \cdot 10^{16}$
2^{64}	$1.84467 \cdot 10^{19}$
2^{80}	$1.20893 \cdot 10^{24}$
2^{90}	$1.23794 \cdot 10^{27}$
2^{112}	$5.19230 \cdot 10^{33}$
2^{128}	$3.40282 \cdot 10^{38}$
2^{150}	$1.42725 \cdot 10^{45}$
2^{160}	$1.46150 \cdot 10^{48}$
2^{192}	$6,27710 \cdot 10^{57}$
2^{250}	$1.80925 \cdot 10^{75}$
2^{256}	$1.15792 \cdot 10^{77}$
2^{320}	$2.13599 \cdot 10^{96}$
2^{512}	$1.34078 \cdot 10^{154}$
2^{768}	$1.55252 \cdot 10^{231}$
2^{1024}	$1.79769 \cdot 10^{308}$
2^{2048}	$3.23170 \cdot 10^{616}$

Table 3.8: Special values of the binary and decimal systems

Such tables can easily be calculated using computer algebra systems. Here is a code sample for Sage:

Sage sample 3.1 Special values of the binary and decimal systems

```
E = [10, 40, 56, 64, 80, 90, 112, 128, 150, 160, 192, 256, 1024, 2048]
for e in E:
    # print "2^" + str(e), "---", 1.0*(2^e)
    print "2^%4d" % e , " --- ", RR(2^e).n(24)
....:
2^ 10 --- 1024.00
2^ 40 --- 1.09951e12
2^ 56 --- 7.20576e16
2^ 64 --- 1.84467e19
2^ 80 --- 1.20893e24
2^ 90 --- 1.23794e27
2^ 112 --- 5.19230e33
2^ 128 --- 3.40282e38
2^ 150 --- 1.42725e45
2^ 160 --- 1.46150e48
2^ 192 --- 6.27710e57
2^ 256 --- 1.15792e77
2^1024 --- 1.79769e308
2^2048 --- 3.23170e616
```

3.13 Appendix: Examples using Sage

Below is Sage source code related to contents of the chapter 3 (“Prime Numbers”).

3.13.1 Check primality of integers generated by quadratic functions

The following Sage code verifies the primality of integers generated by the function $f(n) = n^2 - 9n + 61$. The code defines a function called `quadratic_prime_formula()` that takes three arguments:

- **start** — An integer which is the lower bound for integers in the sequence `start, start + 1, start + 2, ..., end - 1, end`.
- **end** — An integer which is the upper bound for the integers in the sequence `start, start + 1, start + 2, ..., end - 1, end`.
- **verbose** — (default: `True`) a flag to signify whether to print a message indicating the primality of an integer generated by $f(n)$.

A meaningful modification of this code is to use another function, of which the primality of its function values should be checked.

Sage sample 3.2 Verify the primality of integers generated by a quadratic function

```
def quadratic_prime_formula(start, end, verbose=True):
    print "N -- N^2 - 9*N + 61"
    P = 0 # the number of primes between start and end
    for n in xrange(start, end + 1):
        X = n^2 - 9*n + 61
        if is_prime(X):
            P += 1
            if verbose:
                print str(n) + " -- " + str(X) + " is prime"
        else:
            if verbose:
                print str(n) + " -- " + str(X) + " is NOT prime"
    print "Number of primes: " + str(P)
    print "Percentage of primes: " + str(float((P * 100) / (end - start + 1)))
```

With the following function call we compute the values of $f(n) = n^2 - 9n + 61$ for $n = 0, 1, 2, \dots, 50$ and verify the primality of the generated integers:

```
sage: quadratic_prime_formula(0, 50)
N -- N^2 - 9*N + 61
0 -- 61 is prime
1 -- 53 is prime
2 -- 47 is prime
3 -- 43 is prime
4 -- 41 is prime
5 -- 41 is prime
6 -- 43 is prime
7 -- 47 is prime
8 -- 53 is prime
9 -- 61 is prime
```

```

10 -- 71 is prime
11 -- 83 is prime
12 -- 97 is prime
13 -- 113 is prime
14 -- 131 is prime
15 -- 151 is prime
16 -- 173 is prime
17 -- 197 is prime
18 -- 223 is prime
19 -- 251 is prime
20 -- 281 is prime
21 -- 313 is prime
22 -- 347 is prime
23 -- 383 is prime
24 -- 421 is prime
25 -- 461 is prime
26 -- 503 is prime
27 -- 547 is prime
28 -- 593 is prime
29 -- 641 is prime
30 -- 691 is prime
31 -- 743 is prime
32 -- 797 is prime
33 -- 853 is prime
34 -- 911 is prime
35 -- 971 is prime
36 -- 1033 is prime
37 -- 1097 is prime
38 -- 1163 is prime
39 -- 1231 is prime
40 -- 1301 is prime
41 -- 1373 is prime
42 -- 1447 is prime
43 -- 1523 is prime
44 -- 1601 is prime
45 -- 1681 is NOT prime
46 -- 1763 is NOT prime
47 -- 1847 is prime
48 -- 1933 is prime
49 -- 2021 is NOT prime
50 -- 2111 is prime
Number of primes: 48
Percentage of primes: 94.1176470588

```

The last two lines of the output contain a small statistics. You can see that $f(n)$ generates 48 primes when $0 \leq n \leq 50$, which is approximately 94% of the values generated by $f(n)$.

For larger sequences, it is impractical to print all single messages indicating the primality of integers. In the following Sage session, we count the number of primes generated by $f(n)$ where $0 \leq n \leq 1000$ and suppress primality messages.

```

sage: quadratic_prime_formula(0, 1000, False)
N -- N^2 - 9*N + 61
Number of primes: 584
Percentage of primes: 58.3416583417

```

Bibliography

- [Aaronson2003] Scott Aaronson,
The Prime Facts: From Euclid to AKS,
<http://www.scottaaronson.com/writings/prime.pdf>
After I had completed this article, I did come across the fine paper by Scott Aaronson, which also offers a didactically very well-done introduction to this topic. It is humorous and easy to read but at the same time precise and erudite.
- [Bartholome1996] A. Bartholomé, J. Rung, H. Kern,
Zahlentheorie für Einsteiger, Vieweg 1995, 2nd edition 1996.
- [Blum1999] W. Blum,
Die Grammatik der Logik, dtv, 1999.
- [Bundschuh1998] Peter Bundschuh,
Einführung in die Zahlentheorie, Springer 1988, 4th edition 1998.
- [Crandell2001] Richard Crandell, Carl Pomerance,
Prime Numbers. A Computational Perspective, Springer, 2001.
- [Doxiadis2000] Apostolos Doxiadis,
Uncle Petros and the Goldbach's Conjecture,
Faber/Bloomsbury, 2000.
- [Graham1989] R.E. Graham, D.E. Knuth, O. Patashnik,
Concrete Mathematics, Addison-Wesley, 1989.
- [Klee1997] V. Klee, S. Wagon,
Ungelöste Probleme in der Zahlentheorie und der Geometrie der Ebene,
Birkhäuser Verlag, 1997.
- [Knuth1981] Donald E. Knuth,
The Art of Computer Programming, vol 2: Seminumerical Algorithms,
Addison-Wesley, 1969, 2nd edition 1981.
- [Lorenz1993] F. Lorenz,
Algebraische Zahlentheorie, BI Wissenschaftsverlag, 1993.
- [Oppliger2005] Rolf Oppliger
Contemporary Cryptography, Artech House, 2005,
<http://www.esecurity.ch/Books/cryptography.html>.
- [Padberg1996] F. Padberg,
Elementare Zahlentheorie, Spektrum Akademischer Verlag 1988, 2nd edition 1996.

- [Pieper1983] H. Pieper,
Zahlen aus Primzahlen, Verlag Harri Deutsch 1974, 3rd edition 1983.
- [Richstein1999] J. Richstein,
Verifying the Goldbach Conjecture up to $4 \cdot 10^{14}$, Mathematics of Computation 70, 2001,
 p. 1745-1749).
- [Scheid1994] Harald Scheid,
Zahlentheorie, BI Wissenschaftsverlag, 2nd edition, 1994.
- [Schneier1996] Bruce Schneier,
Applied Cryptography, Protocols, Algorithms, and Source Code in C,
 Wiley and Sons, 2nd edition 1996.
- [Schroeder1999] M.R. Schroeder,
Number Theory in Science and Communication,
 Springer 1984, 3rd edition 1997, Corrected Printing 1999.
- [Schwenk1996] J. Schwenk
Conditional Access, in taschenbuch der telekom praxis 1996,
 Hrgb. B. Seiler, Verlag Schiele und Schön, Berlin.
- [Shoup2005] Victor Shoup
A Computational Introduction to Number Theory and Algebra,
 Cambridge University Press, 2005,
<http://shoup.net/ntb/>.
- [Tietze1973] H. Tietze,
Gelöste und ungelöste mathematische Probleme,
 Verlag C. H. Beck 1959, 6th edition 1973.

Web links

1. GIMPS (Great Internet Mersenne-Prime Search)
www.mersenne.org is the home page of the GIMPS project,
<http://www.mersenne.org/prime.htm>
2. The Proth Search Page with the Windows program by Yves Gallot
<http://www.utm.edu/research/primes/programs/gallot/index.html>
3. Generalized Fermat Prime Search
<http://primes.utm.edu/top20/page.php?id=12>
4. Distributed Search for Fermat Number Divisors
<http://www.fermatsearch.org/>
5. At the University of Tennessee you will find extensive research results about prime numbers.
<http://www.utm.edu/>
6. The best overview about prime numbers is offered from my point of view by “The Prime Pages” from professor Chris Caldwell.
<http://www.utm.edu/research/primes>
7. Descriptions e.g. about prime number tests
<http://www.utm.edu/research/primes/mersenne.shtml>
<http://www.utm.edu/research/primes/prove/index.html>
8. Showing the n -th prime number
http://www.utm.edu/research/primes/notes/by_year.html
9. The supercomputer manufacturer SGI Cray Research not only employed brilliant mathematicians but also used the prime number tests as benchmarks for its machines.
http://www.isthe.com/chongo/tech/math/prime/prime_press.html
10. The Cunningham Project,
<http://www.cerias.purdue.edu/homes/ssw/cun/>
11. <http://www.eff.org/awards/coop>
12. <http://www.math.Princeton.EDU/~arbooker/nthprime.html>
13. Goldbach conjecture verification project von Toms Oliveira e Silva,
<http://www.ieeta.pt/~tos/goldbach.html>
14. <http://www.mathematik.ch/mathematiker/goedel.html>

Acknowledgments

I would like to take this opportunity to thank Mr. Henrik Koy and Mr. Roger Oyono for their very constructive proof-reading of the first versions of this article.

Chapter 4

Introduction to Elementary Number Theory with Examples

(Bernhard Esslinger, July 2001; Updates: Dec. 2001, June 2002, May 2003, May 2005, March 2006, June 2007, July 2009)

This “introduction” is for people with a mathematical interest. There is no more pre-knowledge necessary than what you learn in the secondary school.

We intentionally had “beginners” in mind; we did not take the approach of mathematical textbooks, called “introduction”, which cannot be understood at the first reading further than page 5 and which have the real purpose to deliver all information that special monographs can be read.

4.1 Mathematics and cryptography

A large proportion of modern, asymmetric cryptography is based on mathematical knowledge – on the properties (“laws”) of whole numbers, which are investigated in elementary number theory. Here, the word “elementary” means that questions raised in number theory are essentially rooted in the set of natural and whole numbers.

Further mathematical disciplines currently used in cryptography include (see [Bauer1995, p. 2], [Bauer2000, p. 3]) :

- Group theory
- Combination theory
- Complexity theory
- Ergodic theory
- Information theory.

Number theory or arithmetic (the emphasis here is more on the aspect of performing calculations with numbers) was established by Carl Friedrich Gauss¹ as a special mathematical

¹Carl Friedrich Gauss, German mathematician and astronomer, Apr 30, 1777–Feb 23, 1855.

discipline. Its elementary features include the greatest common divisor² (gcd), congruence (remainder classes), factorization, the Euler-Fermat theorem and primitive roots. However, the most important aspect is prime numbers and their multiplicative operation.

For a long time, number theory was considered to be the epitome of pure research, the ideal example of research in the ivory tower. It delved into “the mysterious laws of the realm of numbers”, giving rise to philosophical considerations as to whether it described elements that exist everywhere in nature or whether it artificially constructed elements (numbers, operators and properties).

We now know that patterns from number theory can be found everywhere in nature. For example, the ratio of rotating counterclockwise and rotating clockwise spirals in a sunflower is equal to two consecutive Fibonacci numbers³, for example 21 : 34.

Also, at the latest when number theory was applied in modern cryptography, it became clear that a discipline that had been regarded as purely theoretical for centuries actually had a practical use. Today, experts in this field are in great demand on the job market.

Applications in (computer) security now use cryptography because this mathematical discipline is simply better and easier to prove than all other “creative” substitution procedures that have been developed over the course of time and better than all sophisticated physical methods such as those used to print bank notes [Beutelspacher1996, p. 4].

This article explains the basics of elementary number theory in a way that you can easily understand. It provides numerous examples and very rarely goes into any proofs (these can be found in mathematical textbooks).

The goal is not to exhaustively explain the number theory findings, but to show the essential procedures. The volume of the content is so oriented that the reader can understand and apply the RSA method.

For this purpose we will use both theory and examples to explain how to perform calculations in finite sets and describe how these techniques are applied in cryptography. Particular attention will be paid to the traditional Diffie-Hellman (DH) and RSA public key procedures.

Additionally I added some qualified statements about the security of the RSA algorithm.

Carl Friedrich Gauss:

Mathematics is the queen of sciences and number theory is the queen of mathematics.

4.2 Introduction to number theory

Number theory arose from interest in positive whole numbers $1, 2, 3, 4, \dots$, also referred to as the set of natural numbers *natural numbers* \mathbb{N} . These are the first mathematical constructs used by human civilization. According to Kronecker⁴, they are a creation of God. In Dedekind’s⁵

²This article deals with the gcd (greatest common divisor) in appendix 4.14.

³The sequence of Fibonacci numbers $(a_i)_{i \in \mathbb{N}}$ is defined by the “recursive” rule $a_1 := a_2 := 1$ and for all numbers $n = 1, 2, 3, \dots$ we define $a_{n+2} := a_{n+1} + a_n$. This historical sequence can be found in many interesting forms in nature (for example, see [Graham1994, p. 290 ff] or the website of Ron Knott, which is devoted to Fibonacci numbers). A lot is known about the Fibonacci sequence and it is used today as an important tool in mathematics.

⁴Leopold Kronecker, German mathematician, Dec 7, 1823 – Dec 29, 1891

⁵Julius Wilhelm Richard Dedekind, German mathematician, Oct 6, 1831 – Feb 12, 1916.

opinion, they are a creation of the human intellect. Dependent upon one's ideology, this is an unsolvable contradiction or one and the same thing.

In ancient times, no distinction was made between number theory and numerology, which attributed a mystical significance to specific numbers. In the same way as astronomy and chemistry gradually detached themselves from astrology and alchemy during the Renaissance (from the 14th century), number theory also separated itself from numerology.

Number theory has always been a source of fascination – for both amateurs and professional mathematicians. In contrast to other areas of mathematics, many of the problems and theorems in number theory can be understood by non-experts. On the other hand, the solutions to these problems or the prove to the theorems often resisted to the mathematicians for a very long time. It is therefore one thing to pose good questions but quite another matter to find the answer. One example of this is what is known as Fermat's Last (or large) theorem⁶.

Up until the mid 20th century, number theory was considered to be the purest area of mathematics, an area that had no practical use in the real world. This changed with the development of computers and digital communication, as number theory was able to provide several unexpected solutions to real-life tasks. At the same time, advances in information technology allowed specialists in number theory to make huge progress in factorizing large numbers, finding new prime numbers, testing (old) conjectures and solving numerical problems that were previously impossible to solve. Modern number theory is made up of areas such as:

- Elementary number theory
- Algebraic number theory
- Analytic number theory
- Geometric number theory
- Combinatorial number theory
- Numeric number theory
- Probability theory.

All of the different areas are concerned with questions regarding whole numbers (both positive and negative whole numbers plus zero). However, they each have different methods of dealing with them.

This article only deals with the area of elementary number theory.

⁶One of the things we learn in mathematics at school is Pythagoras' theorem, which states the following for a right-angle triangle: $a^2 + b^2 = c^2$, where a and b are the lengths of the sides containing the right angle and c is the length of the hypotenuse. Fermat famously proposed that $a^n + b^n \neq c^n$ for $a, b, c \in \mathbb{N}$ and whole-number exponents $n > 2$. Unfortunately, the letter in which Fermat made the claim did not have enough space for him to prove it. The theorem was not proven until over 300 years later [Wiles1994, p. 433-551].

4.2.1 Convention

Unless stated otherwise:

- The letters $a, b, c, d, e, k, n, m, p, q$ are used to present whole numbers.
- The letters i and j represent natural numbers.
- The letters p always represents a prime number.
- The sets $\mathbb{N} = \{1, 2, 3, \dots\}$ and $\mathbb{Z} = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$ are the *natural* and *whole* numbers respectively.

Joanne K. Rowling⁷

This isn't magic – it's logic – a puzzle. A lot of the greatest wizards haven't got an ounce of logic.

4.3 Prime numbers and the first fundamental theorem of elementary number theory

Many of the problems in elementary number theory are concerned with prime numbers.

Every whole number has divisors or factors. The number 1 has just one – itself, whereas the number 12 has the six factors 1, 2, 3, 4, 6 and 12⁸. Many numbers are only divisible by themselves and by 1. When it comes to multiplication, these can be regarded as the “atoms” in the realm of numbers.

Definition 4.3.1. *Prime numbers are natural numbers greater than 1 that can only be divided by 1 and themselves.*

By definition, 1 is not a prime number.

If we write down the prime numbers in ascending order (prime number sequence), then we get:

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, \dots

The first 100 numbers include precisely 25 prime numbers. After this, the percentage of primes decreases, but never reaches zero.

We come across whole numbers that are prime fairly often. In the last decade only, three years were prime: 1993, 1997 and 1999. If they were rare, cryptography would not be able to work with them to the extent it does.

Prime numbers can be factorized in a unique (“*trivial*”) way:

$$\begin{aligned}5 &= 1 * 5 \\17 &= 1 * 17 \\1013 &= 1 * 1013 \\1,296,409 &= 1 * 1,296,409.\end{aligned}$$

Definition 4.3.2. *Natural numbers greater than 1 that are not prime are called **composite numbers**. These have at least two factors other than 1.*

⁷Joanne K. Rowling, “Harry Potter and the Philosopher’s Stone”, Bloomsbury, (c) 1997, chapter “Through the trapdoor”, p. 307, by Hermine.

⁸Due to the fact that 12 has so many factors, this number – and multiples of this number – is often found in everyday life: the 12-hour scale on clocks, the 60 minutes in an hour, the 360-degree scale for measuring angles, etc. If we divide these scales into segments, the segments often turn out to be whole numbers. These are easier to use in mental arithmetic than fractions.

Examples of the decomposition of such numbers into prime factors:

$$\begin{aligned} 4 &= 2 * 2 \\ 6 &= 2 * 3 \\ 91 &= 7 * 13 \\ 161 &= 7 * 23 \\ 767 &= 13 * 59 \\ 1029 &= 3 * 7^3 \\ 5324 &= 22 * 11^3. \end{aligned}$$

Theorem 4.3.1. *Each composite number a has a lowest factor greater than 1. This factor is a prime number p and is less than or equal to the square root of a .*

All whole numbers greater than 1 can be expressed as a product of prime numbers — in a *unique* way.

This is the claim of the 1st *fundamental theorem of number theory* (= fundamental theorem of arithmetic = fundamental building block of all positive integers). This was formulated precisely for the first time by Carl Friedrich Gauss in his *Disquisitiones Arithmeticae* (1801).

Theorem 4.3.2. Gauss 1801 *Every even natural number greater than 1 can be written as the product of prime numbers. Given two such decompositions $a = p_1 * p_2 * \dots * p_n = q_1 * q_2 * \dots * q_m$, these can be resorted such that $n = m$ and, for all i , $p_i = q_i$.*

In other words: Each natural number other than 1 can be written as a product of prime numbers in precisely one way, if we ignore the order of the factors. The factors are therefore unique (the “expression as a product of factors” is unique)!

For example, $60 = 2 * 2 * 3 * 5 = 2^2 * 3 * 5$. And this — other than changing the order of the factors — is the only way in which the number 60 can be factorized.

If you allow numbers other than primes as factors, there are several ways of factorizing integers and the *uniqueness* is lost:

$$60 = 1 * 60 = 2 * 30 = 4 * 15 = 5 * 12 = 6 * 10 = 2 * 3 * 10 = 2 * 5 * 6 = 3 * 4 * 5 = \dots$$

The 1st fundamental theorem only appears to be obvious. We can construct numerous other sets of numbers⁹ for which numbers in the set *cannot* be expressed uniquely as a product of the prime numbers of the set.

In order to make a mathematical statement, therefore, it is important to state not only the operation for which it is defined but also the basic set on which the operation is defined.

For more details on prime numbers (e.g. how “Fermat’s Little Theorem” can be used to test extremely large numbers to determine whether they are prime), please refer to the article on prime numbers, chapter 3 in this script.

⁹These sets are formed especially from the set of natural numbers. An example of this can be found in this script on page 52 at the end of chapter 3.2.

4.4 Divisibility, modulus and remainder classes¹⁰

If whole numbers are added, subtracted or multiplied, the result is always another whole number.

The division of two whole numbers does not always result in a whole number. For example, if we divide 158 by 10 the result is the decimal number 15.8, which is not a whole number!

If, however, we divide 158 by 2 the result 79 is a whole number. In number theory we express this by saying that 158 is *divisible* by 2 but not by 10. In general, we say:

Definition 4.4.1. *A whole number n is **divisible** by a whole number d if the quotient n/d is a whole number c such that $n = c * d$.*

n is called a *multiple* of d , whereas d is called a *divisor* or *factor* of n .

The mathematical notation for this is $d|n$ (read “ d divides n ”). The notation $d \nmid n$ means that d does not divide the number n .

In our example therefore: $10 \nmid 158$ but $2|158$.

4.4.1 The modulo operation – working with congruence

When we investigate divisibility, it is only the remainder of the division that is important. When dividing a number n by m , we often use the following notation:

$$\frac{n}{m} = c + \frac{r}{m},$$

where c is a whole number and r is a number with the values $0, 1, \dots, m-1$. This notation is called division with remainder, whereby c is called the whole-number “quotient” and r is the “remainder” of the division.

Example:

$$\frac{19}{7} = 2 + \frac{5}{7} \quad (m = 7, c = 2, r = 5)$$

What do the numbers 5, 12, 19, 26, \dots have in common for division by 7? The remainder is always $r = 5$. For division by 7, only the following remainders are possible:

$$r = 0, 1, 2, \dots, 6.$$

The numbers that result in the same remainder r when divided by 7 are combined to form the “remainder class r modulo 7”. Two numbers a and b belonging to the same remainder class modulo 7 are said to be “congruent modulo 7”. Or in general:

Definition 4.4.2. *The remainder class r modulo m is the set of all whole numbers a that have the same remainder r when divided by m .*

¹⁰With the educational tool for number theory **NT** you can have a playful view at the calculation with congruences, discussed in this and the next chapter (see learning unit 2.1, pages 2-9/40).

NT can be called in CrypTool via the menu path **Indiv. Procedures \ Number Theory Interactive \ Learning Tool for Number Theory**. See appendix A.4.

Example:

Remainder class 0 modulo 4 =

$$\{x|x = 4 * n; n \in \mathbb{Z}\} = \{\dots, -16, -12, -8, -4, 0, 4, 8, 12, 16, \dots\}$$

Remainder class 3 modulo 4 =

$$\{x|x = 4 * n + 3; n \in \mathbb{Z}\} = \{\dots, -13, -9, -5, -1, 3, 7, 11, 15, \dots\}$$

As only the remainders $0, 1, 2, \dots, m-1$ are possible for division modulo m , modular arithmetic works with finite sets. For each modulo m there are precisely m remainder classes.

Definition 4.4.3. *Two numbers $a, b \in \mathbb{N}$ are said to be congruent modulo $m \in \mathbb{N}$ if and only if they have the same remainder when divided by m .*

We write: $a \equiv b \pmod{m}$ (read a is congruent b modulo m), which means that a and b belong to the same remainder class. The modulo is therefore the divisor. This notation was introduced by Gauss. Although the divisor is usually positive, a and b can also be any whole numbers.

Example:

$19 \equiv 12 \pmod{7}$, because the remainders are equal: $19/7 = 2$ remainder 5 and $12/7 = 1$ remainder 5.

$23103 \equiv 0 \pmod{453}$, because $23103/453 = 51$ remainder 0 and $0/453 = 0$ remainder 0.

Theorem 4.4.1. *$a \equiv b \pmod{m}$ if and only if, the difference $(a - b)$ is divisible by m , i.e. if $q \in \mathbb{Z}$ exists with $(a - b) = q * m$.*

These two statements are therefore equivalent.

Therefore: If m divides the difference, there exists a whole number q such that: $a = b + q * m$. As an alternative to the congruence notation, we can also use the divisibility notation: $m|(a - b)$.

Example Example of equivalent statements:

$35 \equiv 11 \pmod{3} \iff 35 - 11 \equiv 0 \pmod{3}$, where $35 - 11 = 24$ is divisible by 3 without remainder while $35 : 3$ and $11 : 3$ leave the remainder 2.

Comment:

The above equivalence does not apply to the sum $(a + b)$!

Example:

$11 \equiv 2 \pmod{3}$, therefore $11 - 2 = 9 \equiv 0 \pmod{3}$; but $11 + 2 = 13$ is not divisible by 3. The statement in theorem 4.4.1 does not even apply to sums in one direction. It is correct for sums only if the remainder is 0 and only in the following direction: If a divisor divides both summands with no remainder, it also divides the sum with no remainder.

We can apply the above equivalence in theorem 4.4.1 if we need a quick and easy method of determining whether large numbers are divisible by a certain number.

Example: Is 69,993 divisible by 7?

The number can be written in the form of a difference in which it is clear that each operand is divisible by 7: $69,993 = 70,000 - 7$. Therefore, the difference is also divisible by 7.

Although these considerations and definitions may seem to be rather theoretical, we are so familiar with them in everyday life that we no longer think about the formal procedure. For example, the 24 hours on a clock are represented by the numbers $1, 2, \dots, 12$. We obtain the

hours after 12 noon as the remainder of a division by 12 and know immediately that 2 o'clock in the afternoon is the same as 14.00.

This “modular” arithmetic (based on division remainders) forms the basis of asymmetric encryption procedures. Cryptographic calculations are therefore not based on real numbers, as the calculations you performed at school, but rather on character strings with a limited length, in other words on positive whole numbers that cannot exceed a certain value. This is one of the reasons why we choose a large number m and “calculate modulo m ”. That is, we ignore whole-number multiples of m and, rather than working with a number, we only work with the remainder when this number is divided by m . The result is that all results are in the range 0 to $m - 1$.

4.5 Calculations with finite sets

4.5.1 Laws of modular calculations

From algebra theorems it follows that essential parts of the conventional calculation rules are kept when we proceed to modular calculations over a basic set \mathbb{Z} . For example, addition remains commutative. The same goes for multiplication modulo m . The result of a division¹¹ is not a fraction but rather a whole number between 0 and $m - 1$.

The known laws apply:

1. Associative law:

$$\begin{aligned} ((a + b) + c) \pmod{m} &\equiv (a + (b + c)) \pmod{m}. \\ ((a * b) * c) \pmod{m} &\equiv (a * (b * c)) \pmod{m}. \end{aligned}$$

2. Commutative law:

$$\begin{aligned} (a + b) \pmod{m} &\equiv (b + a) \pmod{m}. \\ (a * b) \pmod{m} &\equiv (b * a) \pmod{m}. \end{aligned}$$

The associative law and the commutative law apply to both addition and multiplication.

3. Distributive law:

$$(a * (b + c)) \pmod{m} \equiv (a * b + a * c) \pmod{m}.$$

4. Reducibility:

$$\begin{aligned} (a + b) \pmod{m} &\equiv (a \pmod{m} + b \pmod{m}) \pmod{m}. \\ (a * b) \pmod{m} &\equiv (a \pmod{m} * b \pmod{m}) \pmod{m}. \end{aligned}$$

When adding or multiplying the order in which the modulo operation is performed does not matter.

5. Existence of an identity (neutral element):

$$\begin{aligned} (a + 0) \pmod{m} &\equiv (0 + a) \pmod{m} \equiv a \pmod{m}. \\ (a * 1) \pmod{m} &\equiv (1 * a) \pmod{m} \equiv a \pmod{m}. \end{aligned}$$

6. Existence of an inverse element:

For all whole numbers a and m there exists a whole number $-a$ such that:

$$(a + (-a)) \pmod{m} \equiv 0 \pmod{m} \quad (\text{additive inverse}).$$

¹¹When dividing modulo m we can one use co-prime numbers because because other numbers have the same property as zero. See footnote 15 in chapter 4.6.1.

For each a ($a \not\equiv 0 \pmod{p}$) where p is prime there exists a whole number a^{-1} , such that:
 $(a * a^{-1}) \pmod{p} \equiv 1 \pmod{p}$ (multiplicative inverse).

7. Closeness¹²:

$$a, b \in G \implies (a + b) \in G.$$

$$a, b \in G \implies (a * b) \in G.$$

8. Transitivity:

$$[a \equiv b \pmod{m}, b \equiv c \pmod{m}] \implies [a \equiv c \pmod{m}].$$

4.5.2 Patterns and structures

In general mathematicians investigate “Structures”. They ask e.g. at $a * x \equiv b \pmod{m}$, which values x can take for given values of a , b , m .

Especially the case is investigated, where the result b of this operation is the neutral element. Then x is the inverse of a regarding this operation.

¹²The property of closeness is always defined in relation to an operation in a set. See chapter 4.15 “Appendix: Forming closed sets”.

*Seneca*¹³:

The way of theory is long, it is short and effective by examples.

4.6 Examples of modular calculations

As we have already seen:

For two natural numbers a and m , $a \bmod m$ denotes the remainder obtained when we divide a by m . This means that $a \bmod m$ is always a number between 0 and $m - 1$.

For example, $1 \equiv 6 \equiv 41 \equiv 1 \pmod{5}$ because the remainder is always 1. Another example is: $2000 \equiv 0 \pmod{4}$ because 4 divides 2000 with no remainder.

Modular arithmetic only contains a limited quantity of non-negative numbers. The number of these is specified by a modulus m . If the modulo is $m = 5$, then only the 5 numbers in the set $\{0, 1, 2, 3, 4\}$ are used.

A calculation result larger than 4 is then reduced “modulo 5”. In other words, it is the remainder when the result is divided by 5. For example, $2 * 4 \equiv 8 \equiv 3 \pmod{5}$ because 3 is the remainder when we divide 8 by 5.

4.6.1 Addition and multiplication

The following shows

- the addition table¹⁴ $\pmod{5}$ and
- the multiplication tables¹⁵ (table 4.1) for mod 5 (table 4.2) and mod 6 (table 4.3).

Example of an addition table

The result when we add 3 and 4 $\pmod{5}$ is calculated as follows: Calculate $3 + 4 = 7$ and keep subtracting 5 from the result until the result is less than the modulo: $7 - 5 = 2$. Therefore: $3 + 4 \equiv 2 \pmod{5}$.

Example of a multiplication table:

The result of the multiplication $4 * 4 \pmod{5}$ is calculated as follows: $4 * 4 = 16$ and subtract 5 until the result is less than the modulus.

$$16 - 5 = 11; 11 - 5 = 6; 6 - 5 = 1.$$

The table directly shows that $4 * 4 \equiv 1 \pmod{5}$ because $16 : 5 = 3$ remainder 1. Multiplication is defined on the set \mathbb{Z} excluding 0.

¹³Lucius Annaeus Seneca, philosophical writer and poet, 4 B. C. – 65 A. D.

¹⁴Comment on subtraction modulo 5:

$$2 - 4 = -2 \equiv 3 \pmod{5}.$$

It is therefore not true modulo 5 that $-2 = 2$ (see chapter 4.16 “Appendix: Comments on modulo subtraction”).

¹⁵Comment on division modulo 6:

Due to the special role of zero as the identity for addition, division by zero is not permitted:

For all a it is $a * 0 = 0$, because $a * 0 = a * (0 + 0) = a * 0 + a * 0$. Obviously 0 has no inverse regarding the multiplication, because if there would be one, it must be $0 = 0 * 0^{-1} = 1$. Also see footnote 11.

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

Table 4.1: Addition table modulo 5

*	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

Table 4.2: Multiplication table modulo 5

4.6.2 Additive and multiplicative inverses

You can use the tables to read the inverses for each number in relation to addition and multiplication.

The inverse of a number is the number that gives the result 0 when the two numbers are added and 1 when they are multiplied. Thus, the inverse of 4 for addition mod 5 is 1 and the inverse of 4 for multiplication mod 5 is 4 itself, because

$$\begin{aligned} 4 + 1 &= 5 \equiv 0 \pmod{5}; \\ 4 * 4 &= 16 \equiv 1 \pmod{5}. \end{aligned}$$

The inverse of 1 for multiplication mod 5 is 1, while the inverse modulo 5 of 2 is 3 and, since multiplication is commutative, the inverse of 3 is again 2.

If we take a random number and add or multiply another number (here 4) and then add¹⁶ or multiply the corresponding inverse (1 or 4) to the interim result (1 or 3), then the end result is the same as the initial value.

Example:

$$\begin{aligned} 2 + 4 &\equiv 6 \equiv 1 \pmod{5}; & 1 + 1 &\equiv 2 \equiv 2 \pmod{5}, \\ 2 * 4 &\equiv 8 \equiv 3 \pmod{5}; & 3 * 4 &\equiv 12 \equiv 2 \pmod{5}. \end{aligned}$$

In the set $\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$ for the addition and in the set \mathbb{Z}_5^* for the multiplication, all numbers have a **unique** inverse modulo 5.

In the case of modular addition, this is true for every modulo (not just for 5).

This is not the case, however, for modular multiplication.

Theorem 4.6.1. *A natural number a from the set $\{1, \dots, m-1\}$ has one inverse if and only*

¹⁶In general $x + y + (-y) \equiv x \pmod{m}$ [$(-y)$ = additive inverse of $y \pmod{m}$].

if this number and the modulo m are co-prime¹⁷, in other words if a and m have no common prime factors.

Since $m = 5$ is prime, the numbers 1 to 4 are relatively prime to 5 and **each** of these numbers has a multiplicative inverse in mod 5.

Table 4.3 shows as a counterexample the multiplication table for mod 6 (since the modulus $m = 6$ is not prime, not all elements from $\mathbb{Z}_6 \setminus \{0\}$ are relatively prime to 6).

*	1	2	3	4	5
1	1	2	3	4	5
2	2	4	0	2	4
3	3	0	3	0	3
4	4	2	0	4	2
5	5	4	3	2	1

Table 4.3: Multiplication table modulo 6

In addition to 0, the numbers 2, 3 and 4 also have no unique inverse (we can also say they have **no** inverse, because the elementary property of an inverse is uniqueness).

The numbers 2, 3 and 4 have the factor 2 or 3 in common with the modulus 6. Only the numbers 1 and 5, which are relatively prime to 6, have multiplicative inverses, namely themselves.

The number of numbers that are relatively prime to the modulus m is the same as the number of numbers that have a multiplicative inverse (see the Euler function $J(m)$ below).

For the two moduli 5 and 6 used in the multiplication tables, this means: the modulus 5 is a prime number itself. In mod 5, therefore, there are exactly $J(5) = 5 - 1 = 4$ numbers that are relatively prime to the modulus, that is all numbers from 1 to 4.

Since 6 is not a prime number, we write it as a product of its factors: $6 = 2 * 3$. In mod 6, therefore, there are exactly $J(6) = (2 - 1) * (3 - 1) = 1 * 2 = 2$ numbers that have a multiplicative inverse, that is 1 and 5.

Although it may seem difficult to calculate the table of multiplicative inverses for large moduli (this only applies to the areas of the table shaded dark grey), we can use Fermat's Little Theorem to create a simple algorithm for this [Pfleeger1997, p. 80]. Quicker algorithms are described, for instance, in [Knuth1998]¹⁸.

Cryptographically not only the unique nature of the inverse is important, but also that the set of possible values has been exhausted.

¹⁷Two whole numbers a and b are co-prime if and only if $\gcd(a, b) = 1$.

If p is prime and a is a random whole number that is not a multiple of p , then p and a are co-prime.

Further name to the topic co-prime (with $a_i \in \mathbb{Z}, i = 1, \dots, n$):

1. a_1, a_2, \dots, a_n are *relatively prime*, if $\gcd(a_1, \dots, a_n) = 1$.
2. An even stronger request for more than two numbers is:
 a_1, \dots, a_n are *in pairs relatively prime*, if for all $i = 1, \dots, n$ and $j = 1, \dots, n$ with $i \neq j$: $\gcd(a_i, a_j) = 1$.

Example:

2, 3, 6 are relatively prime, because $\gcd(2, 3, 6) = 1$. They are not in pairs relatively prime, because $\gcd(2, 6) = 2 > 1$.

¹⁸Using Euclid's extended theorem (extended gcd), we can calculate the multiplicative inverse and determine whether numbers have an inverse (see appendix 4.14). Alternatively, we can also use the primitive roots.

Theorem 4.6.2. For $a, i \in \{1, \dots, m-1\}$ with $\gcd(a, m) = 1$, then the product $a * i \bmod m$ takes for a certain number a all values from $\{1, \dots, m-1\}$ (exhaustive permutation of the length $m-1$)¹⁹.

The following three examples²⁰ illustrate the properties of multiplicative inverses.

Multiplication table mod 17 (table 4.4) shows for $i = 1, 2, \dots, 18$ the following:

$(5 * i)/17 = a$ remainder r and high-lighted $5 * i \equiv 1 \pmod{17}$,

$(6 * i)/17 = a$ remainder r and high-lighted $6 * i \equiv 1 \pmod{17}$.

We need to **find** the i for which the product remainder $a * i$ modulo 17 with $a = 5$ or $a = 6$ has the value 1.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$5 * i$	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90
remainder	5	10	15	3	8	13	1	6	11	16	4	9	14	2	7	12	0	5
$6 * i$	6	12	18	24	30	36	42	48	54	60	66	72	78	84	90	96	102	108
remainder	6	12	1	7	13	2	8	14	3	9	15	4	10	16	5	11	0	6

Table 4.4: Multiplication table modulo 17 (for $a = 5$ and $a = 6$)

Between $i = 1, \dots, m$, all values between $0, \dots, m-1$ occur for the remainders, because both 5 and 6 are also relatively prime to the modulus $m = 17$.

The multiplicative inverse of 5 (mod 17) is 7, while the inverse of 6 (mod 17) is 3.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$5 * i$	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90
remainder	5	10	2	7	12	4	9	1	6	11	3	8	0	5	10	2	7	12
$6 * i$	6	12	18	24	30	36	42	48	54	60	66	72	78	84	90	96	102	108
remainder	6	12	5	11	4	10	3	9	2	8	1	7	0	6	12	5	11	4

Table 4.5: Multiplication table modulo 13 (for $a = 5$ and $a = 6$)

Between $i = 1, \dots, m$, all values between $0, \dots, m-1$ occur for the remainders, because both 5 and 6 are relatively prime to the modulus $m = 13$.

The multiplicative inverse of 5 (mod 13) is 8, while the inverse of 6 (mod 13) is 11.

The following table 4.6 contains an example, where the modulus m and the number $a = 6$ are *not* relatively prime.

We have calculated $(5 * i) \pmod{12}$ and $(6 * i) \pmod{12}$. Between $i = 1, \dots, m$, not all values between $0, \dots, m-1$ occur and 6 does not have an inverse mod 12, because 6 and the modulus $m = 12$ are not co-prime.

The multiplicative inverse of 5 (mod 12) is 5. The number 6 has no inverse (mod 12).

¹⁹See also theorem 4.9.1 in chapter 4.9, Multiplicative order and primitive roots.

²⁰See chapter 4.18.1 “Multiplication table modulo m” for the source code to compute the tables using Sage.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$5 * i$	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90
remainder	5	10	3	8	1	6	11	4	9	2	7	0	5	10	3	8	1	6
$6 * i$	6	12	18	24	30	36	42	48	54	60	66	72	78	84	90	96	102	108
remainder	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0

Table 4.6: Multiplication table modulo 12 (for $a = 5$ and $a = 6$)

4.6.3 Raising to the power

In modular arithmetic, raising to the power is defined as repeated multiplication – as usual except that multiplication is now slightly different. We can even apply the usual rules, such as:

$$\begin{aligned} a^{b+c} &= a^b * a^c, \\ (a^b)^c &= a^{b*c} = a^{c*b} = (a^c)^b. \end{aligned}$$

Modular powers work in the same way as modular addition and modular multiplication:

$$3^2 = 9 \equiv 4 \pmod{5}.$$

Even consecutive powers work in the same way:

Example 1:

$$(4^3)^2 = 64^2 \equiv 4096 \equiv 1 \pmod{5}.$$

(1) We can speed up²¹ the calculation by reducing the **interim results** modulo 5 but we need to take care because *not* everything will then work in the same way as in standard arithmetic.

$$\begin{aligned} (4^3)^2 &\equiv (4^3 \pmod{5})^2 \pmod{5} \\ &\equiv (64 \pmod{5})^2 \pmod{5} \\ &\equiv 4^2 \pmod{5} \\ &\equiv 16 \equiv 1 \pmod{5}. \end{aligned}$$

(2) In standard arithmetic, consecutive powers can be reduced to a single power by multiplying the exponents:

$$(4^3)^2 = 4^{3*2} = 4^6 = 4096.$$

This is not quite as simple in modular arithmetic because this would give:

$$(4^3)^2 \equiv 4^{3*2 \pmod{5}} \equiv 4^6 \pmod{5} \equiv 4^1 \equiv 4 \pmod{5}.$$

But as we saw above, the correct result is 1!!

(3) Therefore, the rule is slightly different for consecutive powers in modular arithmetic: We do not multiply the exponents in $(\text{mod } m)$ but rather in $(\text{mod } J(m))$.

²¹The time required to calculate the multiplication of two numbers normally depends on the length of the numbers. We can observe this if we use the school method to calculate, for instance, $474 * 228$. The time required increases in a quadratic square manner, because we need to multiply $3 * 3$ numbers. The numbers become considerably smaller if we reduce the interim result.

Using $J(5) = 4$ gives:

$$(4^3)^2 \equiv 4^{3*2 \pmod{J(5)}} \equiv 4^{6 \pmod{4}} \equiv 4^2 \equiv 16 \equiv 1 \pmod{5}.$$

This delivers the correct result.

Theorem 4.6.3. $(a^b)^c \equiv a^{b*c \pmod{J(m)}} \pmod{m}$.

Example 2:

$$3^{28} = 3^{4*7} \equiv 3^{4*7 \pmod{10}} \equiv 3^8 \equiv 6561 \equiv 5 \pmod{11}.$$

4.6.4 Fast calculation of high powers

RSA encryption and decryption²² entails calculating high powers modulo m . For example, the calculation $(100^5) \pmod{3}$ exceeds the 32-bit long integer number range provided we calculate a^n by actually multiplying a with itself n times in line with the definition. In the case of extremely large numbers, even a fast computer chip would take longer than the age of the universe to calculate a single exponential. Luckily, there is an extremely effective shortcut for calculating exponentials (but not for calculating logarithms).

If the expression is divided differently using the rules of modular arithmetic, then the calculation does not even exceed the 16-bit short integer number range:

$$(a^5) \equiv (((a^2 \pmod{m})^2 \pmod{m}) * a) \pmod{m}.$$

We can generalize this by representing the exponent as a binary number. For example, the naive method would require 36 multiplications in order to calculate a^n for $n = 37$. However, if we write n in the binary representation as $100101 = 1 * 2^5 + 1 * 2^2 + 1 * 2^0$, then we can rewrite the expression as: $a^{37} = a^{2^5+2^2+2^0} = a^{2^5} * a^{2^2} * a^{2^0}$

Example 3: $87^{43} \pmod{103}$.

Since $43 = 32 + 8 + 2 + 1$, 103 is prime, $43 < J(103)$

and the squares $\pmod{103}$ can be calculated beforehand

$$\begin{aligned} 87^2 &\equiv 50 \pmod{103}, \\ 87^4 &\equiv 50^2 \equiv 28 \pmod{103}, \\ 87^8 &\equiv 28^2 \equiv 63 \pmod{103}, \\ 87^{16} &\equiv 63^2 \equiv 55 \pmod{103}, \\ 87^{32} &\equiv 55^2 \equiv 38 \pmod{103}. \end{aligned}$$

We have²³:

$$\begin{aligned} 87^{43} &\equiv 87^{32+8+2+1} \pmod{103} \\ &\equiv 87^{32} * 87^8 * 87^2 * 87 \pmod{103} \\ &\equiv 38 * 63 * 50 * 87 \equiv 85 \pmod{103}. \end{aligned}$$

²²See chapter 4.10 (Proof of the RSA procedure with Euler-Fermat) and chapter 4.13 (The RSA procedure with actual numbers).

²³See chapter 4.18.2 “Fast exponentiation” for source code implementing the square and multiply method in Sage, which can be used to reproduce the calculations above.

The powers $(a^2)^k$ can be determined easily by means of repeated squaring. As long as a does not change, a computer can calculate them beforehand and – if enough memory is available – save them. In order to then find a^n in each individual case, it now only needs to multiply those $(a^2)^k$ for which there is a one in the k -th position of the binary representation of n . The typical effort is then reduced from 2^{600} to $2 * 600$ multiplications! This frequently used algorithm is called “Square and Multiply”.

4.6.5 Roots and logarithms

The inverses of the powers are also defined. The roots and logarithms are again whole numbers. Yet in contrast to the usual situation, they are not only difficult to calculate but, in the case of large numbers, cannot be calculated at all within a reasonable amount of time.

Let us take the equation $a \equiv b^c \pmod{m}$.

a) Taking the logarithm (determining c) – Discrete logarithm problem:²⁴

If we know a and b of the three numbers a , b and c that meet this equation, then every known method of finding c is approximately just as time-consuming as trying out all m possible values for c one after the other. For a typical m of the order of magnitude of 10^{180} for 600-digit binary numbers, this is a hopeless task. More precisely, for suitably large numbers m , the time required according to current knowledge is proportional to $\exp(C * (\log m [\log \log m]^2)^{1/3})$ with a constant $C > 1$.

b) Calculating the root (determining b):

The situation is similar if b is the unknown variable and we know the values of a and c : If we know the Euler function of m , $J(m)$, then we can easily²⁵ calculate d with $c * d \equiv 1 \pmod{J(m)}$ and use theorem 4.6.3 to obtain:

$$a^d \equiv (b^c)^d \equiv b^{c*d} \equiv b^{c*d \pmod{J(m)}} \equiv b^1 \equiv b \pmod{m}$$

the c -th root b of a .

If $J(m)$ cannot be determined²⁶, it is difficult to calculate the c -th root. This forms the basis for the security assumption used by the RSA encryption system (see chapter 4.10 or chapter 5.3.1).

The time required for inverting addition and multiplication, on the other hand, is simply proportional to $\log m$ or $(\log m)^2$. Powers (for a number x calculate x^a with a fixed) and exponents (for a number x calculate a^x with a fixed) are therefore typical one way functions (compare chapters 5.1 and 4.12.1).

4.7 Groups and modular arithmetic in \mathbb{Z}_n and \mathbb{Z}_n^*

Mathematical “groups” play a decisive role in number theory and cryptography. We only talk of groups if, for a defined set and a defined relation (an operation such as addition or multiplication), the following properties are fulfilled:

²⁴Further details about the discrete logarithm problem can be found in chapter 5.4.

²⁵See chapter 4.14 “Appendix: gcd and the two algorithms of Euclid”.

²⁶According to the first fundamental theorem of number theory and theorem 4.8.4, we can determine $J(m)$ by reducing m to prime factors.

- The set is closed
- A neutral element exists
- An inverse element exists for each element
- The associative law applies.

The abbreviated mathematical notation is $(G, +)$ or $(G, *)$.

Definition 4.7.1. \mathbb{Z}_n :

\mathbb{Z}_n comprises all numbers from 0 to $n - 1$: $\mathbb{Z}_n = \{0, 1, 2, \dots, n - 2, n - 1\}$.

\mathbb{Z}_n is an often used finite group of the natural numbers. It is sometimes also called the *remainder set R modulo n* .

For example, 32-bit computers (standard PCs) only directly work with whole numbers in a finite set, that is the value range $0, 1, 2, \dots, 2^{32} - 1$.

This value range is equivalent to the set $\mathbb{Z}_{2^{32}}$.

4.7.1 Addition in a group

If we define the operation $\text{mod}+$ on such a set where

$$a \text{ mod } + b := (a + b) \text{ (mod } n),$$

then the set \mathbb{Z}_n together with the relation $\text{mod}+$ is a group because the following properties of a group are valid for all elements in \mathbb{Z}_n :

- $a \text{ mod } + b$ is an element of \mathbb{Z}_n (the set is closed),
- $(a \text{ mod } + b) \text{ mod } + c \equiv a \text{ mod } + (b \text{ mod } + c)$ ($\text{mod}+$ is associative),
- the neutral element is 0.
- each element $a \in \mathbb{Z}_n$ has an inverse for this operation, namely $n - a$ (because $a \text{ mod } + (n - a) \equiv a + (n - a) \text{ (mod } n) \equiv n \equiv 0 \text{ (mod } n)$).

Since the operation is commutative, i.e. $(a \text{ mod } + b) = (b \text{ mod } + a)$, this structure is actually a “commutative group”.

4.7.2 Multiplication in a group

If we define the operation mod^* on the set \mathbb{Z}_n where

$$a \text{ mod } * b := (a * b) \text{ (mod } n),$$

then \mathbb{Z}_n together with this operation is **usually not a group** because not all properties are fulfilled for each n .

Example:

- In \mathbb{Z}_{15} , for example, the element 5 does not have an inverse. That is to say, there is no a with $5 * a \equiv 1 \text{ (mod } 15)$. Each modulo product with 5 on this set gives 5, 10 or 0.

- b) In $\mathbb{Z}_{55} \setminus \{0\}$, for example, the elements 5 and 11 do not have multiplicative inverses. That is to say, there is no $a \in \mathbb{Z}_{55}$ such that $5 * a \equiv 1 \pmod{55}$ and no a such that $11 * a \equiv 1 \pmod{55}$. This is because 5 and 11 are not relatively prime to 55. Each modulo product with 5 on this set gives 5, 10, 15, ..., 50 or 0. Each modulo product with 11 on this set gives 11, 22, 33, 44 or 0.

On the other hand, there are subsets of \mathbb{Z}_n that form a group with the operation mod^* . If we choose all elements in \mathbb{Z}_n that are relatively prime to n , then this set forms a group with the operation mod^* . We call this set \mathbb{Z}_n^* .

Definition 4.7.2. \mathbb{Z}_n^* :

$$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \gcd(a, n) = 1\}.$$

\mathbb{Z}_n^* is sometimes also called the reduced remainder set R' modulo n .

Example: For $n = 10 = 2 * 5$ the following applies:

full remainder set $R = \mathbb{Z}_n = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

reduced remainder set $R' = \mathbb{Z}_n^* = \{1, 3, 7, 9\} \longrightarrow J(n) = 4$.

Comment:

R' or \mathbb{Z}_n^* is always a genuine subset of R or \mathbb{Z}_n because 0 is always an element of R but never an element of R' . Since 1 and $n - 1$ are always relatively prime to n , they are always elements of both sets.

If we select a random element in \mathbb{Z}_n^* and multiply it by every other element in \mathbb{Z}_n^* , then the products²⁷ are all in \mathbb{Z}_n^* , and the results are also a unique permutation of the elements in \mathbb{Z}_n^* . Since 1 is always an element of \mathbb{Z}_n^* , there is a unique “partner” in this set such that the product is 1. In other words:

Theorem 4.7.1. *Each element in \mathbb{Z}_n^* has a multiplicative inverse.*

Example: $a = 3$ modulo 10 with $\mathbb{Z}_n^* = \{1, 3, 7, 9\}$ it holds that $a^{-1} = 7$:

$$\begin{aligned} 3 &\equiv 3 * 1 \pmod{10}, \\ 9 &\equiv 3 * 3 \pmod{10}, \\ 1 &\equiv 3 * 7 \pmod{10}, \\ 7 &\equiv 3 * 9 \pmod{10}. \end{aligned}$$

The unique invertibility is an essential condition for cryptography (see section 4.10).

²⁷This is due to the fact that \mathbb{Z}_n^* is closed with respect to the multiplication and due to the gcd property:

$[a, b \in \mathbb{Z}_n^*] \Rightarrow [(a * b) \pmod{n}] \in \mathbb{Z}_n^*$, exactly:

$[a, b \in \mathbb{Z}_n^*] \Rightarrow [\gcd(a, n) = 1, \gcd(b, n) = 1] \Rightarrow [\gcd(a * b, n) = 1] \Rightarrow [(a * b) \pmod{n}] \in \mathbb{Z}_n^*$.

Eric Berne²⁸:

Mathematical game theory postulates players who respond rationally. Transactional game theory, on the other hand, deals with games that are not rational, perhaps even **irrational** and thereby closer to reality.

4.8 Euler function, Fermat's little theorem and Euler-Fermat

4.8.1 Patterns and structures

As mathematicians investigate the structure $a * x \equiv b \pmod{m}$ (see chapter 4.5.2), so they are interested in the structure $x^a \equiv b \pmod{m}$.

Again here they are interested in the case, if $b = 1$ (value of the multiplicative inverse) and if $b = x$ (the function has a fixpoint).

4.8.2 The Euler function

Given n , the number of numbers from the set $\{1, \dots, n-1\}$ that are relatively prime to n is equal to the value of the Euler²⁹ function $J(n)$.

Definition 4.8.1. *The Euler function³⁰ $J(n)$ specifies the number of elements in \mathbb{Z}_n^* .*

$J(n)$ also specifies how many whole numbers have multiplicative inverses in mod n . $J(n)$ can be calculated if we know the prime factors of n .

Theorem 4.8.1. *For a prime number, the following is true: $J(p) = p - 1$.*

Theorem 4.8.2. *If m is the product of two distinct primes, then:*

$$J(p * q) = (p - 1) * (q - 1) \quad \text{or} \quad J(p * q) = J(p) * J(q).$$

This case is important for the RSA procedure.

Theorem 4.8.3. *If $n = p_1 * p_2 * \dots * p_k$ where p_1 to p_k are distinct prime numbers (i.e. no factor occurs more than once), then the following is true (as a generalization of theorem 4.8.2):*

$$J(n) = (p_1 - 1) * (p_2 - 1) * \dots * (p_k - 1).$$

Theorem 4.8.4. *In general, the following is true for every prime number p and every n in \mathbb{N} :*

1. $J(p^n) = p^{n-1} * (p - 1).$

2. *If $n = p_1^{e_1} * p_2^{e_2} * \dots * p_k^{e_k}$, where p_1 to p_k are distinct prime numbers, then:*

$$J(n) = [(p_1^{e_1-1}) * (p_1 - 1)] * \dots * [(p_k^{e_k-1}) * (p_k - 1)] = n * [(p_1 - 1)/p_1] * \dots * [(p_k - 1)/p_k].$$

²⁸Eric Berne, "Games People Play", rororo, (c) 1964, page 235.

²⁹Leonhard Euler, Swiss mathematician, Apr 15, 1707 – Sep 18, 1783

³⁰Often written as the Euler phi function $\Phi(n)$.

Example:

- $n = 70 = 2 * 5 * 7 \implies$ using theorem 4.8.3: $J(n) = 1 \cdot 4 \cdot 6 = 24$.
- $n = 9 = 3^2 \implies$ using theorem 4.8.4: $J(n) = 3^1 \cdot 2 = 6$, because $\mathbb{Z}_9^* = \{1, 2, 4, 5, 7, 8\}$.
- $n = 2,701,125 = 3^2 * 5^3 * 7^4 \implies$ using theorem 4.8.4:

$$J(n) = [3^1 * 2] * [5^2 * 4] * [7^3 * 6] = 1,234,800.$$

4.8.3 The theorem of Euler-Fermat

In order to prove the RSA procedure, we need Fermat's theorem and its generalisation (Euler-Fermat theorem) – please see chapter 3.5.

Theorem 4.8.5. Fermat's Little Theorem³¹ *Let p be a prime number and a be a random whole number, then:*

$$a^p \equiv a \pmod{p}.$$

An alternative formulation of Fermat's Little Theorem is as follows: Let p be a prime number and a be a random whole number that is relatively prime to p , then:

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

Theorem 4.8.6. Euler-Fermat theorem (generalization of Fermat's Little Theorem)
For all elements a in the group \mathbb{Z}_n^ (i.e. a and n are natural numbers that are co-prime):*

$$a^{J(n)} \equiv 1 \pmod{n}.$$

This theorem states that if we raise a group element (here a) to the power of the order of the group (here $J(n)$), we always obtain the neutral element for multiplication (the number 1).

The 2nd formulation of Fermat's Little Theorem is derived directly from Euler's theorem if n is a prime number.

If n is the product of two prime numbers, we can - in certain cases - use Euler's theorem to calculate the result of a modular power very quickly. We have: $a^{(p-1)*(q-1)} \equiv 1 \pmod{pq}$.

Examples for calculating a modular power:

- With $2 = 1 * 2$ and $6 = 2 * 3$ where 2 and 3 are both prime; $J(6) = 2$ because only 1 and 5 are relatively prime to 6, we obtain the equation $5^2 \equiv 5^{J(6)} \equiv 1 \pmod{6}$, without having to calculate the power.
- With $792 = 22 * 36$ and $23 * 37 = 851$ where 23 and 37 are both prime, it follows for $31 \in \mathbb{Z}_{851}^*$ that $31^{792} \equiv 31^{J(23*37)} \equiv 31^{J(851)} \equiv 1 \pmod{851}$.

4.8.4 Calculation of the multiplicative inverse

Another interesting application is a special case of determining the multiplicative inverses using the Euler-Fermat theorem (multiplicative inverses are otherwise determined using the extended Euclidean algorithm).

³¹Pierre de Fermat, French mathematician, Aug 17, 1601 – Jan 12, 1665.

Example:

Find the multiplicative inverse of 1579 modulo 7351.

According to Euler-Fermat: $a^{J(n)} \equiv 1 \pmod{n}$ for all a in \mathbb{Z}_n^* . If we divide both sides by a , we get: $a^{J(n)-1} \equiv a^{-1} \pmod{n}$. For the special case that the modulo is prime, we have $J(n) = p - 1$. Therefore, the modular inverse is

$$a^{-1} = a^{J(n)-1} \equiv a^{(p-1)-1} \equiv a^{p-2} \pmod{p}.$$

For our example, this means:

Since the modulus 7351 is prime, $p - 2 = 7349$.
 $1579^{-1} \equiv 1579^{7349} \pmod{7351}.$

By cleverly breaking down the exponent, we can calculate this power relatively easily (see Section 4.6.4 Fast calculation of high powers):

$$\begin{aligned} 7349 &= 4096 + 2048 + 1024 + 128 + 32 + 16 + 4 + 1 \\ 1579^{-1} &\equiv 4716 \pmod{7351}. \end{aligned}$$

4.8.5 Fixpoints modulo 26

According to theorem 4.6.3, the arithmetic operations of modular expressions are performed in the exponents modulo $J(n)$ rather than modulo n ³².

In $a^{e*d} \equiv a^1 \pmod{n}$, if we wish to determine the inverses for the factor e in the exponent, we need to calculate modulo $J(n)$.

Example: (with reference to the RSA algorithm)

If we calculate modulo 26, which set can e and d come from?

Solution: We have $e * d \equiv 1 \pmod{J(26)}$.

The reduced remainder set $R' = \mathbb{Z}_{26}^* = \{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25\}$ are the elements in \mathbb{Z}_{26} , which have a multiplicative inverse, that is which are relatively prime to 26.

The reduced remainder set R'' contains only the elements of R' that are relatively prime to $J(n) = 12$: $R'' = \{1, 5, 7, 11\}$.

For every e in R'' there exists a d in R'' such that $a \equiv (a^e)^d \pmod{n}$.

For every e in R'' , there exists therefore precisely one element (not necessarily different from e) such that $e * d \equiv 1 \pmod{J(26)}$.

For all e that are relatively prime to $J(n)$ we could calculate d as follows using the Euler-Fermat theorem: For $a^{J(n)} \equiv 1 \pmod{n}$ is the same as saying $a^{J(n)-1} \equiv a^{-1} \pmod{n}$. Therefore

$$d \equiv e^{-1} \pmod{J(n)} \equiv e^{J(n)-1} \pmod{J(n)}.$$

The problems of factorizing $n = pq$ with $q \neq p$ and finding $J(n)$ have a similar degree of difficulty and if we find a solution for one of the two problems, we also have a solution for the other³³ (please compare requisition 3 in section 4.10.1).

³²For the following example, we will adopt the usual practice for the RSA procedure of using “ n ” rather than “ m ” to denote the modulus.

³³If we know the factors of $n = p * q$ with $p \neq q$, then $J(n) = (p - 1) * (q - 1) = n - (p + q) + 1$. Additionally the

4.9 Multiplicative order and primitive roots³⁴

The multiplicative order and the primitive root are two useful constructs (concepts) in elementary number theory.

Mathematicians often ask, in which conditions the repeated application of an operation results in the neutral element (compare Patterns and Structures, chapter 4.8.1).

For the i -times successive modular multiplication of a number a with $i = 1, \dots, m-1$ the product is the neutral element of the multiplication if and only if a and m are relatively prime.

Definition 4.9.1. The **multiplicative order** $\text{ord}_m(a)$ of a whole number $a \pmod{m}$ (where a and m are co-prime) is the smallest whole number i for which $a^i \equiv 1 \pmod{m}$.

The following table shows that in a multiplicative group (here \mathbb{Z}_{11}^*) not all numbers necessarily have the same order. The orders in this case are 1, 2, 5 and 10 and we notice that:

1. The orders are all factors of 10.
2. The numbers $a = 2, 6, 7$ and 8 have the order 10 - we say that these numbers have the **maximum order** in \mathbb{Z}_{11}^* .

Example 1:

The following table 4.7³⁵ shows the values $a^i \pmod{11}$ for the exponents $i = 1, 2, \dots, 10$ and for the bases $a = 1, 2, \dots, 10$ as well as the resulting value $\text{ord}_{11}(a)$ for each a /

	i=1	i=2	i=3	i=4	i=5	i=6	i=7	i=8	i=9	i=10	$\text{ord}_{11}(a)$
a=1	1	1	1	1	1	1	1	1	1	1	1
a=2	2	4	8	5	10	9	7	3	6	1	10
a=3	3	9	5	4	1	3	9	5	4	1	5
a=4	4	5	9	3	1	4	5	9	3	1	5
a=5	5	3	4	9	1	5	3	4	9	1	5
a=6	6	3	7	9	10	5	8	4	2	1	10
a=7	7	5	2	3	10	4	6	9	8	1	10
a=8	8	9	6	4	10	3	2	5	7	1	10
a=9	9	4	3	5	1	9	4	3	5	1	5
a=10	10	1	10	1	10	1	10	1	10	1	2

Table 4.7: Values of $a^i \pmod{11}$, $1 \leq a, i < 11$ and according order of $a \pmod{m}$

Table 4.7 shows, for example, that the order of 3 modulo 11 has the value 5.

factors p and q are solutions of the quadratic equation $x^2 - (p+q)x + pq = 0$.

If only n and $J(n)$ are known, then it is: $pq = n$ and $p+q = n - J(n) + 1$. So you get p and q by solving the equation

$$x^2 + (J(n) - n - 1)x + n = 0.$$

³⁴With the educational tool for number theory **NT** you can have a playful experience with primitive roots (see learning unit 2.2, pages 10-14/40 and 24-40/40).

NT can be called in CrypTool via the menu path **Indiv. Procedures \ Number Theory Interactive \ Learning Tool for Number Theory**. See appendix A.4.

³⁵See chapter 4.18.3 “Multiplicative order” for the source code to generate the table using Sage.

Definition 4.9.2. If a and m are co-prime and if $\text{ord}_m(a) = J(m)$ (i.e. a has maximum order), then we say that a is a **primitive root** of m .

A number a is not a primitive root for every modulo m . In the table 4.7, only $a = 2, 6, 7$ and 8 is a primitive root with respect to mod 11 ($J(11) = 10$).

Using the primitive roots, we can clearly establish the conditions for which powers modulo m have a unique inverse and the calculation in the exponents is manageable.

The following two tables show the multiplicative orders and primitive roots modulo 45 and modulo 46.

Example 2:

The following table 4.8³⁶ shows the values $a^i \text{ mod } 45$ for the exponents $i = 1, 2, \dots, 12$ and for the bases $a = 1, 2, \dots, 12$ as well as the resulting value $\text{ord}_{45}(a)$ for each a .

$a \setminus i$	1	2	3	4	5	6	7	8	9	10	11	12	$\text{ord}_{45}(a)$	$J(45)$
1	1	1	1	1	1	1	1	1	1	1	1	1	1	24
2	2	4	8	16	32	19	38	31	17	34	23	1	12	24
3	3	9	27	36	18	9	27	36	18	9	27	36	—	24
4	4	16	19	31	34	1	4	16	19	31	34	1	6	24
5	5	25	35	40	20	10	5	25	35	40	20	10	—	24
6	6	36	36	36	36	36	36	36	36	36	36	36	—	24
7	7	4	28	16	22	19	43	31	37	34	13	1	12	24
8	8	19	17	1	8	19	17	1	8	19	17	1	4	24
9	9	36	9	36	9	36	9	36	9	36	9	36	—	24
10	10	10	10	10	10	10	10	10	10	10	10	10	—	24
11	11	31	26	16	41	1	11	31	26	16	41	1	6	24
12	12	9	18	36	27	9	18	36	27	9	18	36	—	24

Table 4.8: Values of $a^i \text{ mod } 45, 1 \leq a, i < 13$

$J(45)$ is calculated using theorem 4.8.4: $J(45) = J(3^2 * 5) = 3^1 * 2 * 4 = 24$.

Since 45 is not a prime, there is no “multiplicative order” for all values of a (for all numbers that are not relatively prime to 45 : 3, 5, 6, 9, 10, 12, \dots , because $45 = 3^2 * 5$).

Example 3:

Is 7 a primitive root modulo 45?

The necessary, but not sufficient requirement/condition $\text{gcd}(7, 45) = 1$ is fulfilled. Table 4.8 shows that the number $a = 7$ is not a primitive root of 45, because $\text{ord}_{45}(7) = 12 \neq 24 = J(45)$.

Example 4:

The following table 4.9³⁷ answers the question as to whether the number $a = 7$ is a primitive root of 46. The necessary, but not sufficient requirement/condition $\text{gcd}(7, 46) = 1$ is fulfilled.

$J(46)$ is calculated using theorem 4.8.2: $J(46) = J(2 * 23) = 1 * 22 = 22$. The number 7 is a primitive root of 46, because $\text{ord}_{46}(7) = 22 = J(46)$.

³⁶See chapter 4.18.3 “Multiplicative order” for the source code to generate the table using Sage.

³⁷See chapter 4.18.3 “Multiplicative order” for the source code to generate the table using Sage.

$a \backslash i$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	ord
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	4	8	16	32	18	36	26	6	12	24	2	4	8	16	32	18	36	26	6	12	24	2	–
3	3	9	27	35	13	39	25	29	41	31	1	3	9	27	35	13	39	25	29	41	31	1	3	11
4	4	16	18	26	12	2	8	32	36	6	24	4	16	18	26	12	2	8	32	36	6	24	4	–
5	5	25	33	27	43	31	17	39	11	9	45	41	21	13	19	3	15	29	7	35	37	1	5	22
6	6	36	32	8	2	12	26	18	16	4	24	6	36	32	8	2	12	26	18	16	4	24	6	–
7	7	3	21	9	17	27	5	35	15	13	45	39	43	25	37	29	19	41	11	31	33	1	7	22
8	8	18	6	2	16	36	12	4	32	26	24	8	18	6	2	16	36	12	4	32	26	24	8	–
9	9	35	39	29	31	3	27	13	25	41	1	9	35	39	29	31	3	27	13	25	41	1	9	11
10	10	8	34	18	42	6	14	2	20	16	22	36	38	12	28	4	40	32	44	26	30	24	10	–
11	11	29	43	13	5	9	7	31	19	25	45	35	17	3	33	41	37	39	15	27	21	1	11	22
12	12	6	26	36	18	32	16	8	4	2	24	12	6	26	36	18	32	16	8	4	2	24	12	–
13	13	31	35	41	27	29	9	25	3	39	1	13	31	35	41	27	29	9	25	3	39	1	13	11
14	14	12	30	6	38	26	42	36	44	18	22	32	34	16	40	8	20	4	10	2	28	24	14	–
15	15	41	17	25	7	13	11	27	37	3	45	31	5	29	21	39	33	35	19	9	43	1	15	22
16	16	26	2	32	6	4	18	12	8	36	24	16	26	2	32	6	4	18	12	8	36	24	16	–
17	17	13	37	31	21	35	43	41	7	27	45	29	33	9	15	25	11	3	5	39	19	1	17	22
18	18	2	36	4	26	8	6	16	12	32	24	18	2	36	4	26	8	6	16	12	32	24	18	–
19	19	39	5	3	11	25	15	9	33	29	45	27	7	41	43	35	21	31	37	13	17	1	19	22
20	20	32	42	12	10	16	44	6	28	8	22	26	14	4	34	36	30	2	40	18	38	24	20	–
21	21	27	15	39	37	41	33	3	17	35	45	25	19	31	7	9	5	13	43	29	11	1	21	22
22	22	24	22	24	22	24	22	24	22	24	22	24	22	24	22	24	22	24	22	24	22	24	22	–
23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	23	–

Table 4.9: Values of $a^i \bmod 46, 1 \leq a, i < 23$

Theorem 4.9.1. ^{38,39} Given a modulus n and a number a , relative prime to n the following holds:

The set $\{a^i \bmod n \mid i = 1, \dots, J(n)\}$ equals the multiplicative group Z_n^* if and only if $\text{ord}_n(a) = J(n)$.

The multiplicative group Z_n^* contains all values from 1 to $n - 1$.

³⁸For prime moduli p all a with $0 < a < p$ are of order $J(p) = p - 1$. Compare table 4.8 for an example. In this case $a^i \bmod p$ goes through all the values $1, \dots, p - 1$. Exhausting all possible values of the set is an important cryptographic proposition (compare theorem 4.6.2). This determines a permutation $\pi(p - 1)$.

³⁹Table 4.9 demonstrates that for composite moduli n not all a are of maximal order $J(n)$. In this example only 5, 7, 11, 15, 17, 19 and 21 are of order 22.

4.10 Proof of the RSA procedure with Euler-Fermat

Using the Euler-Fermat theorem, we can “prove” the RSA^{40,41} procedure in the group \mathbb{Z}_n^* .

4.10.1 Basic idea of public key cryptography

The basic idea behind public key cryptography is that all participants possess a different pair of keys (P and S) and the public keys for all recipients are published. You can retrieve the public key P for a recipient from a directory just as you would look up some one’s phone number in the phone book. Furthermore, each recipient has a secret key S that is needed in order to decrypt the message and that is not known to anyone else. If the sender wishes to send a message M , he encrypts it using the public key P of the recipient before sending it:

The ciphertext C is determined as $C = E(P; M)$, where E (encryption) is the encryption rule. The recipient uses his private key S to decrypt the message with the decryption rule $D : M = D(S; C)$.

In order to ensure that this system works for every message M , the following four **requirements** must be met:

1. $D(S; E(P; M)) = M$ for every M (invertibility) and M takes “very many” of its possible values.
2. All (S, P) pairs are different for all participants (i.e. lots of them are needed).
3. The time required to derive S from P is at least as high as the time required to decrypt M with no knowledge of S .
4. Both C and M can be calculated relatively easily.

The 1st requirement is a general condition for all cryptographic encryption algorithms.

The 2nd requirement can easily be met because there is a “very” large number of prime numbers⁴² and because this can be ensured by a central office that issues certificates.

It is this last requirement that makes the procedure actually usable. This is because it is possible to calculate the powers in a linear amount of time (because there is a restriction on the length of the numbers).

Although Whitfield Diffie and Martin Hellman formulated the general method as early as 1976, the actual procedure that met all four requirements was only discovered later by Rivest, Shamir and Adleman.

⁴⁰The RSA procedure is the most common asymmetric cryptography procedure. Developed in 1978 by Ronald Rivest, Adi Shamir and Leonard Adleman, it can be used both for signatures and for encryption. Cryptographers always associate this procedure with the abbreviation “**RSA**” – the following remark is meant with humor to show that each letter combination can be used with several meanings: In Britain the “Royal Society for the encouragement of Arts, Manufactures & Commerce” is commonly known as the “RSA”.

⁴¹In literature and in movies not only classic but also modern cryptographic methods have been used (see appendix A.3).

⁴²According to the **prime number theorem** (chapter 3.7.2, p. 68) of Legendre and Gauss there are approximately $n/\ln(n)$ prime numbers up to the number n . This means, for example, that there are $6.5 * 10^{74}$ prime numbers under $n = 2^{256}$ ($= 1.1 * 10^{77}$) and $3.2 * 10^{74}$ prime numbers under $n = 2^{255}$. Between 2^{255} and 2^{256} there are therefore $3.3 * 10^{74}$ prime numbers with precisely 256 bits. This large number is also the reason why we cannot simply save them all.

4.10.2 How the RSA procedure works

The individual steps for implementing the RSA procedure can be described as follows (see [Eckert2003, p. 213 ff] and [Sedgewick1990, p. 338 ff]). Steps 1 to 3 constitute key generation, steps 4 and 5 are the encryption, and steps 6 and 7 are the decryption:

1. Select two distinct random prime numbers^{43,44} p and q and calculate $n = p * q$ ⁴⁵.
The value n is called the RSA modulus⁴⁶.
2. Select an arbitrary $e \in \{2, \dots, n - 1\}$ such that⁴⁷:
 e is relatively prime to $J(n) = (p - 1) * (q - 1)$.
We can then “throw away” p and q .⁴⁸
3. Select $d \in \{1, \dots, n - 1\}$ with $e * d \equiv 1 \pmod{J(n)}$, i.e. d is the multiplicative inverse of e modulo $J(n)$ ^{49,50}. We can then “throw away” $J(n)$.
 $\rightarrow (n, e)$ is the public key P .
 $\rightarrow (n, d)$ is the private key S (only d must be kept secret).
4. For encryption, the message represented as a (binary) number is divided into parts such that each part of the number is less than n .
5. Encryption of the plaintext (or the parts of it) $M \in \{1, \dots, n - 1\}$:

$$C = E((n, e); M) := M^e \pmod{n}.$$
6. For decryption, the ciphertext represented as a binary number is divided into parts such that each part of the number is less than n .

⁴³Compaq introduced the so-called multi-prime method with high marketing effort in 2000. n was the product of two big and one relative small prime: $n = o * p * q$. With theorem 4.8.3 we get: $J(n) = (o - 1) * (p - 1) * (q - 1)$. This method did not assert itself yet.

One reason probably is, that Compaq claimed a patent on it. Generally there is less understanding in Europe and with the Open Source Initiative, that one can claim patents on algorithms. But there is really no understanding outside the U.S., that one can get a patent for a special case (3 factors) of an algorithm (RSA), although the patent for the general case was almost expired.

⁴⁴If the two primes p and q are equal then $(m^e)^d \equiv m \pmod{n}$ is not true for all $m < n$ (although $e * d \equiv 1 \pmod{J(n)}$ is fulfilled). **Example:**

If $n = 5^2$ then according to theorem 4.8.4 it is $J(n) = 5 * 4 = 20$, $e = 3$, $d = 7$, $e * d = 21 \equiv 1 \pmod{J(n)}$. But it is $(5^3)^7 \equiv 0 \pmod{25}$.

⁴⁵The GISA (German Information Security Agency) recommends, to choose the prime factors p and q almost the same, but not too close:

$$0.5 < |\log_2(p) - \log_2(q)| < 30.$$

They recommend to generate the primes independently and check that the restriction is fulfilled (see [GISA2002]).

⁴⁶In CrypTool the RSA modulo is denoted with a capital “ N ”.

⁴⁷It is recommended by cryptanalytic reasons, but not necessary to make RSA work, to select e such that:
 $\max(p, q) < e < J(n) - 1$.

⁴⁸The procedure also allows us to select d freely and then calculate e . However, this has practical disadvantages. We usually want to be able to encrypt messages “quickly”, which is why we choose a public exponent e such that it has a short bit length compared to the modulus n and as few binary ones as possible (e.g. $2^{16} + 1$). So a fast exponentiation is possible when encrypting. We want to select the publicly known e to be an advantageous value that allows the exponential calculation to be performed quickly during encryption. The prime numbers 3, 17 and 65537 have proved to be particularly practical for this purpose. The most often used number is $65537 = 2^{16} + 1$, or in binary: $10 \dots 0 \dots 01$ (this number is prime and therefore relatively prime to many other numbers).

⁴⁹For reasons of security, d should not be too small.

⁵⁰We start by determining either d or e depending on the implementation.

7. Decryption of the ciphertext (or the parts of it) $C \in \{1, \dots, n-1\}$:

$$M = D((n, d); C) := C^d \pmod{n}.$$

The numbers d, e and n are usually extremely large (e. g. d and e 300 bits, n 600 bits).

Comment:

The security of the RSA algorithm depends as with all public key methods on the difficulty to calculate the private key d from the public key (n, e) .

Concretely for the RSA method does this mean:

1. It is hard to calculate $J(n)$ for big compounds n and
2. It is hard to calculate the prime factors of big compounds n .

4.10.3 Proof of requirement 1 (invertibility)

For pairs of keys (n, e) and (n, d) that possess fixed properties in steps 1 to 3 of the RSA procedure, the following must be true for all $M < n$:

$$M \equiv (M^e)^d \pmod{n} \quad \text{with} \quad (M^e)^d = M^{e*d}.$$

This means that the deciphering algorithm above works correctly.

We therefore need to show that:

$$M^{e*d} \equiv M \pmod{n}.$$

We will show this in 3 steps (see [Beutelspacher1996, p. 131ff]).

Step 1:

In the first step we show that: $M^{e*d} \equiv M \pmod{p}$. This results from the requirements and from Euler-Fermat (theorem 4.8.6). Since $n = p * q$ and $J(p * q) = (p-1) * (q-1)$ and since e and d are selected in such a way that $e * d \equiv 1 \pmod{J(n)}$, there is a whole number k such that: $e * d = 1 + k * (p-1) * (q-1)$.

$$\begin{aligned} M^{e*d} &\equiv M^{1+k*J(n)} \equiv M * M^{k*J(n)} \equiv M * M^{k*(p-1)*(q-1)} \pmod{p} \\ &\equiv M * (M^{p-1})^{k*(q-1)} \pmod{p} \quad \text{based on little Fermat: } M^{p-1} \equiv 1 \pmod{p} \\ &\equiv M * (1)^{k*(q-1)} \pmod{p} \\ &\equiv M \pmod{p} \end{aligned}$$

The requirement for using the simplified Euler-Fermat theorem (theorem 4.8.5) was that M and p are relatively prime.

Since this is not true in general, we need to consider the case when M and p are not relatively prime. Since p is a prime number, this implies that p is a factor of M . But this means:

$$M \equiv 0 \pmod{p}.$$

If p is a factor of M , then p is also a factor of M^{e*d} . Therefore:

$$M^{e*d} \equiv 0 \pmod{p}.$$

Since p is a factor of both M and $Me * d$, it is also a factor of their difference:

$$(M^{e*d} - M) \equiv 0 \pmod{p}.$$

And therefore our conjecture is also true in this special case.

Step 2:

In exactly the same way we prove that: $M^{e*d} \equiv M \pmod{q}$.

Step 3:

We now combine the conjectures from step 1 and 2 for $n = p * q$ to show that:

$$M^{e*d} \equiv M \pmod{n} \text{ for all } M < n.$$

From step 1 and 2 we have $(M^{e*d} - M) \equiv 0 \pmod{p}$ and $(M^{e*d} - M) \equiv 0 \pmod{q}$. Therefore, p and q are both factors of the same number $z = (M^{e*d} - M)$. Since p and q are **distinct** prime numbers, their product must also be a factor of this number z . Thus:

$$(M^{e*d} - M) \equiv 0 \pmod{p * q} \text{ or } M^{e*d} \equiv M \pmod{p * q} \text{ or } M^{e*d} \equiv M \pmod{n}.$$

□

Comment 1:

We can also condense the three steps if we use the theorem 4.8.6 (Euler-Fermat) - i.e. not the simplified theorem where $n = p$ and which corresponds to Fermat's Little Theorem:

$$(M^e)^d \equiv M^{e*d} \equiv M^{(p-1)(q-1)*k+1} \equiv \left(\underbrace{M^{(p-1)(q-1)}}_{\equiv M^{J(n)} \equiv 1 \pmod{n}} \right)^k * M \equiv 1^k * M \equiv M \pmod{n}.$$

Comment 2:

When it comes to signing messages, we perform the same operations but first use the secret key d , followed by the public key e . The RSA procedure can also be used to create digital signatures, because:

$$M \equiv (M^d)^e \pmod{n}.$$

4.11 Considerations regarding the security of the RSA algorithm⁵¹

There have always been discussions about the suitability of the RSA algorithm for digital signatures and encryption, e. g. after publications of breakthroughs in factorization. Nevertheless the RSA algorithm has become a de-facto standard since it was published more than 20 years ago (compare 7.1).

The security of the RSA algorithm rests — as with all cryptographic methods — on the following 4 central pillars:

- the complexity of the number theoretical problem on which the algorithm is based (here factorization of big numbers),

⁵¹Major parts of chapters 4.11.1 and 4.11.2 follow the article “Vorzüge und Grenzen des RSA-Verfahrens” written by F. Bourseau, D. Fox and C. Thiel [Bourseau2002].

- the election of fitting parameters (here the length of the module N),
- the adequate usage of the algorithm and key generation and
- the correct implementation of the algorithm.

Usage and key generation are well understood today. Implementation based on long integer arithmetic is very easy.

The following sections examine the RSA algorithm with respect to the first two points.

4.11.1 Complexity

Successful decryption or forgery of a signature — without knowing the private key — requires calculating the e -th root mod n . The private key, this is the multiplicative inverse of $e \bmod J(n)$, can be easily determined if $J(n)$ is known. $J(n)$ again can be calculated from the prime factors of n . Breaking of RSA therefore cannot be more difficult than factorization of the module n .

The best factorization method known today is a further development of the General Number Field Sieve (GNFS), which was originally devised to factor only numbers of a special form (like Fermat numbers). The complexity of solving the factorization problem with the GNFS is asymptotically

$$O(l) = e^{c \cdot (l \cdot \ln 2)^{1/3} \cdot (\ln(l \cdot \ln(2)))^{2/3} + o(l)}$$

Please refer to:

- A. Lenstra, H. Lenstra: *The development of the Number Field Sieve* [Lenstra1993].
- Robert D. Silverman: *A Cost-Based Security Analysis of Symmetric and Asymmetric Key Lengths* [Silverman2000].

This formula shows, that the factorization problem belongs to the class of problems with sub-exponential time complexity (i. e. time complexity grows asymptotically not as fast as exponential functions like e^l or 2^l , but strictly slower, e. g. like $e^{\sqrt{l}}$). This classification is all that is currently known; it does not preclude the possibility that the factorization problem can be solved in polynomial time (see 4.11.5).

$O(l)$ is the average number of processor steps depending on the bit length l of the number n to be factorized. For the best currently known factorization algorithm the constant $c = (64/9)^{1/173} = 1923$.

The inverse proposition, that the RSA algorithm can be broken only by factorization of n , is still not proven. Most number theorists consider the “RSA problem” and the factorization problem equivalent in terms of time complexity.

Please refer to: *Handbook of Applied Cryptography* [Menezes2001].

4.11.2 Security parameters because of new algorithms

Factorization algorithms⁵² The complexity is basically determined by the length l of the module n . Higher values for this major parameter are oriented at the possibilities of the current

⁵¹With the educational tool for number theory **NT** you can gather more experience with current factorization algorithms (see learning unit 5.1-5.5, pages 1-15/15).

NT can be called in CrypTool via the menu path **Indiv. Procedures \ Number Theory Interactive \ Learning Tool for Number Theory**. See appendix A.4.

algorithms for factorization:

- In 1994 a 129-digit RSA module (428 bit), published in 1977, was factorized by a distributed implementation of the Quadratic Sieve algorithm (QS), developed 1982 by Pomerance. This effort took 8 months.

Please refer to:

C. Pomerance: *The quadratic sieve factoring algorithm* [Pomerance1984].

- In 1999 a 155-digit module (512 bit) was factored with an implementation of the General Number Field Sieve algorithm (GNFS), developed by Buhler, Lenstra and Pomerance. The GNFS is more efficient than QS if n is longer than about 116 decimal digits. This effort took 5 months.

Please refer to:

J.P. Buhler, H.W. Lenstra, C. Pomerance: *Factoring integers with the number field sieve* [Buhler1993].

This made evident that a module length of 512 bit no longer prevents from attackers.

And also past 1999 further factorization progress was made up to now (see RSA-200 in chapter 4.11.4).

Lattice base reduction algorithms

The module length is not the only parameter relevant for security. Beneath requirements from implementation and engineering the sizes and the proportions of the parameters e , d and N are relevant.

According attacks based on lattice reductions are a real threat for (too) simple implementations of RSA. These attacks can be structured into the following four categories:

- Attacks against very small public keys e (e.g. $e = 3$).
- Attacks against relatively small private exponents d (e.g. $d < N^{0.5}$).
- Factorization of the modulus N , if one of the factors p or q is *partly* known.
- Attacks requiring, that a *part* of the private key d is known.

A good overview about the current status of publications concerning these attacks can be found in the diploma thesis of Matthias Schneider [SchneiderM2004].

4.11.3 Forecasts about factorization of large integers

Within the last 20 years a lot of progress has been made. Estimations about the future development of the ability to factor RSA modules vary and depend on some assumptions:

- progression in computing performance (Moore's law: every 18 month the computing power will double) and in grid computing.
- development of new algorithms.

Within the last years the module bit length feasible for factorization increased — even without new algorithms — by 10 bit per year. Larger numbers require not only more time to be factored, but also huge RAM storage for the solutions matrix being used by the best algorithms known today. This need for storage grows like the square root of the computation time, i. e. also sub-exponentially. Because RAM availability increased exponentially in the recent decades, it seems that this should not be the limiting factor.

An estimation of the evolution of secure key lengths was done by Lenstra/Verheul in 1999 [Lenstra1999] (compare figure 7.1 in chapter 7.1).

Within the article [Bourseau2002] Dirk Fox⁵³ published his prognosis of an almost linear factorization progression, if all influencing factors are included: Each year the module length feasible for factorization increases by 20 bit on average. So his forecast was below the more optimistic estimations of GISA and NIST.

This forecast by Dirk Fox from the year 2001 seems to prove true by the factorization record of RSA-200 (see chapter 4.11.4). His estimation for the year 2005, to achieve a bit length of 660 bit, was almost a precision landing (compare figure 4.1).

If the forecast withstand in the future then the factorization of an RSA modulus of 1024 bit can be expected in 15 years.

⁵³His company Secorvo Ltd also delivered a statement on the recommendation for key length selection published by the GISA (German Information Security Agency). Chapter 2.3.1 of this statement contains a competent and understandable discussion of RSA security (this document exists – to my knowledge – only in German): <http://www.secorvo.de/publikat/stellungnahme-algorithmenempfehlung-020307.pdf>

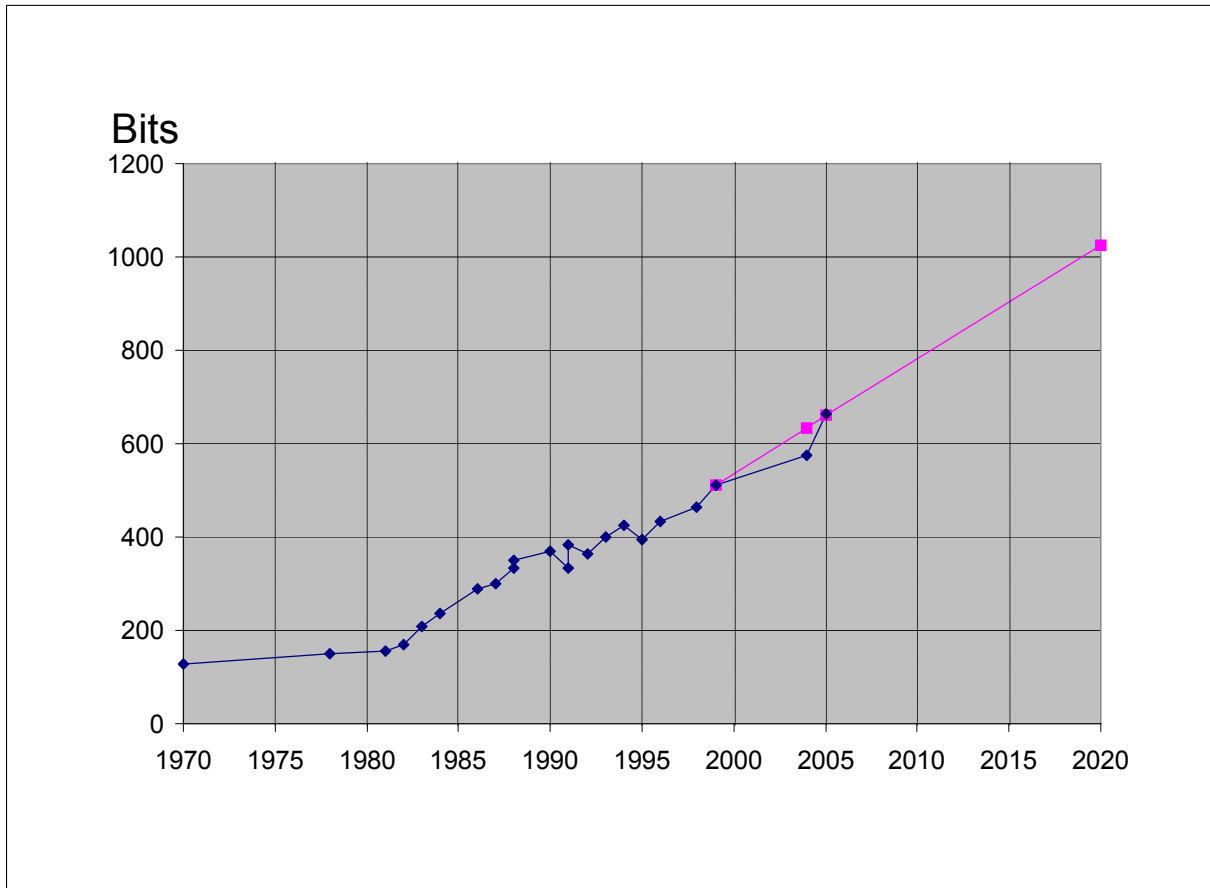


Figure 4.1: Forecast about future factorization records compared with current results (from Secorvo)

*Hermann Hesse*⁵⁴:

To let the possible happen, you again and again have to try the impossible.

4.11.4 Status regarding factorization of concrete large numbers

An exhaustive overview about the factoring records of composed integers using different methods can be found on the following web pages:

<http://www.crypto-world.com>

http://www.tutorgig.com/ed/RSA_number The RSA Factoring Challenge

http://en.wikipedia.org/wiki/Integer_factorization_records

The current record (as of May 2005) obtained using the GNFS method (General Number Field Sieve) factorized a general 200 decimal digit into its both prime factors.

The last records⁵⁵ with factorization algorithms for composed numbers are listed in the table 4.10.

	Decimal digits	Binary digits	Factored on	Factored by
C307	307	1017	May, 2007	Jens Franke et al.
RSA-200	200	663	May, 2005	Jens Franke et al.
RSA640 ⁵⁶	193	600	November, 2005	Jens Franke et al.
C176	176	583	May, 2005	Kazumaro Aoki et al.
RSA576	174	576	December, 2003	Jens Franke et al.
RSA-160	160	530	April, 2003	Jens Franke et al.
C158	158	523	January, 2002	Jens Franke et al.
RSA-155	155	512	August, 1999	Herman te Riele et al.

Table 4.10: The current factoring records (as of July 2009)

Below the last records are explained in more detail. The two methods, GNFS and SNFS, used to do so are shortly illustrated at the following web pages:

⁵⁴Hermann Hesse, German/Swiss writer and Nobel Prize winner, July 2, 1877 – August 9, 1962.

⁵⁵The 'RSA numbers' are certain large semiprime numbers (i.e., numbers with exactly two prime factors). They were generated and published by the company RSA Security and they form the basis of the RSA Factoring Challenge, in which factorizations for these numbers are sought.

See <http://www.rsa.com/rsalabs/node.asp?id=2092>.

RSA Labs offers its challenges since the beginning of the 1990th. The first RSA Factoring Challenge labeled the numbers, from RSA-100 to RSA-500, according to their number of decimal digits; the second RSA Factoring Challenge labeled the numbers according to their number of binary digits. Within the second challenge cash prizes were offered for successful factorizations of RSA576 to RSA2048 (RSA576, RSA640 etc. using 64 bit steps upwards — An exception to this is RSA617, which was created prior to the change in the numbering scheme). But the RSA challenges ended ahead of time in 2007, RSA Inc. retracted the prize.

The 'C numbers' originate from the Cunningham project: <http://www.cerias.purdue.edu/homes/ssw/cun/>.

⁵⁶A research group of the GISA solved this challenge which was awarded with 20,000 US dollar using the GNFS method. The researchers needed about five months to divide this number into its both 320 bit long prime factors.

The researchers around Professor Jens Franke (from the University of Bonn, the GISA and the CWI) do not aim on getting cash prizes but in extending the research limits. So statements about the necessary length of a secure RSA modulus are more well-founded.

The next bigger challenge is RSA-704. See <http://www.heise.de/newsticker/meldung/print/65957>.

http://en.wikipedia.org/wiki/Special_number_field_sieve
http://en.wikipedia.org/wiki/General_number_field_sieve

RSA-155

On August 22, 1999 researchers from the Netherlands found the solution of this RSA challenge. They factorized a 155-digit number into its both 78-digit primes (see chapter 4.11.2).

This 512 bit RSA-155 meant to reach a kind of *magic* border.

C158

On January 18, 2002 researchers at the German University of Bonn⁵⁷ factorized a 158-digit decimal number into its both prime factors (these are build with 73 and 86 decimal digits) using the GNFS method (General Number Field Sieve).

This record got much less attention within the press than the solution of RSA-155.

The task of the researchers from Bonn was not initiated by a challenge, but they wanted to find the last prime factors of the integer $2^{953} - 1$ (see “Wanted List” of the Cunningham Project⁵⁸).

The 6 smaller prime factors, already found before have been:

3, 1907, 425796183929,
1624700279478894385598779655842584377,
3802306738549441324432139091271828121 and
128064886830166671444802576129115872060027.

The first 3 factors can be easily computed⁵⁹. The next three prime factors were found by P. Zimmerman⁶⁰, T. Grandlund⁶¹ and R. Harley during the years 1999 and 2000 using the elliptic curve factorization method.

The last remaining factor, called “C158”, was known to be composite by then, but its factors were not known (the following 3 lines contain one number):

39505874583265144526419767800614481996020776460304936
45413937605157935562652945068360972784246821953509354
4305870490251995655335710209799226484977949442955603

The factorization of C158 resulted in the following two 73- and 86-digit prime factors:

3388495837466721394368393204672181522815830368604993048084925840555281177

and

1165882340667125990314837655838327081813101
2258146392600439520994131344334162924536139.

So now all 8 prime factors of $2^{953} - 1$ have been found.

⁵⁷http://www.ercim.org/publication/Ercim_News/enw49/franke.html, 2002-01

⁵⁸Cunningham project: <http://www.cerias.purdue.edu/homes/ssw/cun/>

⁵⁹E.g. using CrypTool via menu **Indiv. Procedures \ RSA Cryptosystem \ Factorization of a Number**. CrypTool can factorize in a reasonable time numbers no longer than 250 bit. Numbers bigger than 1024 bits are currently not accepted by CrypTool.

⁶⁰<http://www.loria.fr/~zimmerma/ecmnet>

⁶¹<http://www.swox.se/gmp/>

Links:

<http://www.loria.fr/~zimmerma/records/gnfs158>
<http://www.crypto-world.com/FactorRecords.html>
<http://www.crypto-world.com/announcements/c158.txt>

RSA-160

On January 18, 2002 researchers at the German University of Bonn⁶² factorized a 160-digit number into its both prime factors (these are build with each 80 decimal digits) using the GNFS method (General Number Field Sieve).

The computations for the factorization of RSA-160 also took place at the German Information Security Agency (GISA) in Bonn⁶³.

The 160-digit decimal number origins from the old challenge list of RSADSI. This number was retracted after RSA-155 (RSA512) had been factorized successfully. The prime factors of RSA-160 were still unknown. So this record of the team of Prof. Franke provides the solution of the old challenge, for which RSADSI didn't award a price anymore.

The composite number called "RSA-160" is (the following 3 lines contain one number):

215274110271888970189601520131282542925777358884567598017049
767677813314521885913567301105977349105960249790711158521430
2079314665202840140619946994927570407753

The factorization of RSA-160 resulted in the following two prime factors:

$p = 45427892858481394071686190649738831$
656137145778469793250959984709250004157335359

and

$q = 47388090603832016196633832303788951$
973268922921040957944741354648812028493909367

The calculations took place between December 2002 and April 2003.

RSA-200

On May 9, 2005 the research group of Prof. Jens Franke at the German University of Bonn⁶⁴ announced, that they achieved a new world record in number factorization together with their colleagues of the Amsterdam Centrum voor Wiskunde en Informatica.

⁶²<http://www.loria.fr/~zimmerma/records/rsa160>
<http://www.loria.fr/~zimmerma/records/factor.html>
<http://www.crypto-world.com/FactorWorld.html>

⁶³Every year the GISA creates a paper to describe which crypto algorithms are feasible to generate digital signatures according to the German signature law – under participation of experts from economy and science. To review signature methods based on the factorization problem the GISA also co-operates with researchers from the University of Bonn. Further information about crypto algorithms can be found on the web page of GISA:
<http://www.bsi.bund.de/esig/basics/techbas/krypto/index.htm>.

⁶⁴<http://www.loria.fr/~zimmerma/records/rsa200>

They factorized a 200-digit number into its both prime factors (these are build with each 100 decimal digits) using the GNFS method (General Number Field Sieve).

The composite number called “RSA-200” is (the following 3 lines contain one number):

```
2799783391122132787082946763872260162107044678695542853756000992932
6128400107609345671052955360856061822351910951365788637105954482006
576775098580557613579098734950144178863178946295187237869221823983
```

The factorization of RSA-200 resulted in the following two prime factors:

```
p = 35324619344027701212726049781984643686711974001976
25023649303468776121253679423200058547956528088349
```

and

```
q = 7925869954478330333470858414800596877379758573642
19960734330341455767872818152135381409304740185467
```

The calculations took place between December 2003 and May 2005. The factorization done by the group around Bahr, Böhm, Franke, Kleinjung, Montgomery and te Riele lasted almost 17 months. The operating expense of the calculations was about 120,000 MIPS-years⁶⁵.

C307 / M1039

In May 2007 Prof. Franke, Prof. Kleinjung (University of Bonn), the Japanese telecommunication company NTT and Prof. Arjen Lenstra (Polytechnical University of Lausanne) announced, that they managed to factorize a 307 digit decimal number into its both prime factors with the SNFS method (Special Number Field Sieve) within 11 months (the two factors have 80 and 227 decimal digits).

The task of the researchers was not initiated by a challenge, but they wanted to find the last prime factors of the Mersenne number $2^{1039} + 1$ (see “Wanted List” of the Cunningham Project⁶⁶).

The number $2^{1039} - 1$ consists of 3 prime factors: The smallest one, $p_7 = 5080711$ was already known.⁶⁷

⁶⁵A MIPS-year (MY) is the quantity of operations a machine can perform in one year, if the machine constantly achieves one million integer operations per second (MIPS). For illustration: a INTEL Pentium 100 processor achieves about 50 MIPS. To factorize a 2048 bit module it is estimated to need about $8.5 \cdot 10^{40}$ MY.

⁶⁶Cunningham project: <http://www.cerias.purdue.edu/homes/ssw/cun/>
Cunningham table: <http://homes.cerias.purdue.edu/~ssw/cun/pmain1206>

The numbers in the Cunningham table have the following syntax:

“(2,n)-” means $2^n - 1$; “(2,n)+” means $2^n + 1$.

To describe the magnitude one writes $p < n >$ or $c < n >$: “n” is the number of decimal digits and “p” and “c” tell, whether the number is prime or composite.

$2^{1039} - 1 = p_7 * c_{307} = p_7 * p_{80} * p_{227}$

It is explained more precisely at the CUN page:

“2651+ means $2^{651} + 1$ and the size (c209 means 209 decimal digits) of the number which was factored. Then come the new factor(s), the discoverer and the method used. Recently, only the multiple polynomial quadratic sieve (ppmpqs), the elliptic curve method (ecm) and the number field sieve (nfs) have been used. ‘hmpqs’ stands for hypercube multiple polynomial quadratic sieve. Under ‘new factors’, ‘p90’ means a 90-digit prime and ‘c201’ is a 201-digit composite number.”.

⁶⁷This one can also be found using CrypTool via menu **Indiv. Procedures \ RSA Cryptosystem \ Factor-**

To complete this the second factor (co-divider) “C307” had to be factorized: Till then it was only known, that the last remaining factor was composite, but it was unknown, how many prime factors it had and what are the prime factors. The following 5 lines contain one number:

$C307 = 1159420574072573064369807148876894640753899791702017724986868353538$
 $8224838599667566080006095408005179472053993261230204874402860435302$
 $8619141014409345351233471273967988850226307575280937916602855510550$
 $0425810771176177610094137970787973806187008437777186828680889844712$
 $822002935201806074755451541370711023817$

The factorization of C307 resulted in the following two 80- and 2276-digit prime factors:

$p_{80} = 558536666199362912607492046583159449686465270184$
 $88637648010052346319853288374753$

and

$p_{227} = 207581819464423827645704813703594695162939708007395209881208$
 $387037927290903246793823431438841448348825340533447691122230$
 $281583276965253760914101891052419938993341097116243589620659$
 $72167481161749004803659735573409253205425523689.$

So now the number $2^{1039} - 1$ is completely factorized in its 3 prime factors.

Links:

<http://www.loria.fr/~zimmerma/records/21039->
<http://www.crypto-world.com/announcements/m1039.txt>
<http://www.crypto-world.com/FactorAnnouncements.html>
[http://www1.uni-bonn.de/pressDB/jsp/pressemitteilungsdetails.jsp?](http://www1.uni-bonn.de/pressDB/jsp/pressemitteilungsdetails.jsp?detailjahr=2007&detail=160)
[detailjahr=2007&detail=160](http://www1.uni-bonn.de/pressDB/jsp/pressemitteilungsdetails.jsp?detailjahr=2007&detail=160)

Size of factorized numbers compared to primality proven numbers

As you notice the factorized compound numbers built of 2 prime factors are much smaller than the especially structured numbers, for which primality tests are able to decide whether these numbers are prime or not (see chapters 3.4, 3.5 and 3.6).

Length in bits of the current world records:

$[RSA-200 \text{ Number}] \longleftrightarrow [43rd \text{ known Mersenne Prime}]$
 $663 \longleftrightarrow 30,402,457$

ization of a Number — with the algorithms of Brent, Williams or Lenstra, which are “relatively” good to separate small factors.

CrypTool can factorize in a reasonable time numbers no longer than 250 bit.

4.11.5 Further current research about primes and factorization

Prime numbers are part of very many topical research areas in number theory and computer science. Progress made with factorization is bigger than was estimated 5 years ago – this is not only due to faster computers but also new knowledge.

The security of the RSA algorithm is based on the empirical observation that factoring large numbers is a hard problem. A module n (typically, 1024 bit) can be easily constructed as the product of two large primes p, q (typically, 500–600 bit each), by calculating $n = pq$. However, it is a hard problem to extract p, q from n . Without knowing p or q , the private key cannot be calculated.

Thus, any progress in efficiency of factorizing large integers will effect the security of the RSA. As a consequence, the underlying primes p, q and, thus, the module n (1024 bit as of today) have to be increased. In case of a quantum leap in factorization, the RSA algorithm might be compromised.

Bernstein's paper and its implication on the security of the RSA algorithm

In his paper “Circuits for integer factorization: a proposal” (<http://cr.yp.to/djb.html>), published November 2001, D. J. Bernstein [Bernstein2001] addresses the problem of factorizing large integers. Therefore, his results are of relevance from a RSA point of view. As a main result Bernstein claims that the implementation of the General Number Field Sieve algorithm (GNFS) can be improved to factor, with the same effort as before, integers with three times more digits.

We note that the definition of *effort* is a crucial point: Bernstein claims that effort is the product of time and costs of the machine (including the memory used). The gist of the paper lies in the fact that he can reduce a big part of factorizing to sorting. Using Schimmler's scheme, sorting can be optimized by massive parallel computing. At the end of section 3 Bernstein explains this effect: The costs of m^2 parallel computers with a constant amount of memory is a constant times m^2 . The costs of a computer with a single processor and memory of size m^2 is also of the order of m^2 , but with a different constant factor. With m^2 processors in parallel, sorting of m^2 numbers (with Schimmler's scheme) can be achieved in time m , while a m^2 -memory computer needs time of the order of m^2 . Decreasing memory and increasing the number of processors, the computing time can be reduced by a factor $1/m$ without additional effort in terms of total costs. In section 5 it is said that massive parallel computing can also increase efficiency of factorizing using Lenstra's elliptic-curve-method (a search algorithm has costs that increase in a quadratic square manner instead of cubically).

We note that all results achieved so far are asymptotic results. This means that they only hold in the limit n to infinity. Unfortunately, there is no upper limit for the residual error (i.e. the difference between the real and the asymptotic value) for finite n — a problem which has already been addressed by the author. As a consequence, one cannot conclude whether the costs (in the sense of Bernstein) for factorizing 1024–2048-bit RSA modules can be significantly reduced.

There is no doubt that Bernstein's approach is innovative. However, the reduction of computing time under constant costs comes along with a massive use of parallel computing — a scenario which seems not to be realistic yet. For example, formally 1 sec computing time on one machine and $1/1,000,000$ sec time parallel computing time on 1,000,000 machines might have same costs. In reality, it is much harder to realize the second situation, and Bernstein does not take into account the fixed costs, in particular for building a network between all these

computers.

Although distributed computing over a large network might help to overcome this problem, realistic costs for data transfer have to be taken into account — a point which was not addressed in Bernstein’s proposal.

As long as there is neither (low cost) hardware nor a distributed computing approach (based on Bernstein’s ideas), there should not be a problem for RSA. It has to be clarified from which magnitude of n on Bernstein’s method could lead to a significant improvement (in the sense of the asymptotic result).

Arjen Lenstra, Adi Shamir et. al. analyzed the paper of Bernstein [Lenstra2002]. In summary they expect a factorization improvement on how much longer the bit length of the keys could be with a factor of 1.17 (instead of factor 3 as proposed by Bernstein).

The abstract of their paper “Analysis of Bernstein’s Factorization Circuit” says:

“... Bernstein proposed a circuit-based implementation of the matrix step of the number field sieve factorization algorithm. We show that under the non-standard cost function used in [1], these circuits indeed offer an asymptotic improvement over other methods but to a lesser degree than previously claimed: for a given cost, the new method can factor integers that are 1.17 times larger (rather than 3.01). We also propose an improved circuit design based on a new mesh routing algorithm, and show that for factorization of 1024-bit integers the matrix step can, under an optimistic assumption about the matrix size, be completed within a day by a device that costs a few thousand dollars. We conclude that from a practical standpoint, the security of RSA relies exclusively on the hardness of the relation collection step of the number field sieve.”

RSA Security’s analysis of the Bernstein paper [RSA Security 2002] from April, 8 2002 also – as expected – concludes, that RSA is still not compromised.

This is still an ongoing discussion.

When this section was written (June 2002) nothing was publicly known about, how far there exist implementations of his theoretical onsets and how much financing there was for his research project.

Links:

<http://cr.yp.to/djb.html>
<http://www.counterpane.com/crypto-gram-0203.html#6>
<http://www.math.uic.edu>

The TWIRL device

In January 2003 Adi Shamir and Eran Tromer from the Weizmann Institute of Science published a preliminary draft called “*Factoring Large Numbers with the TWIRL Device*” raising concerns about the security of key sizes till 1024 bits [Shamir2003].

Their abstract summarizes their results very well: “The security of the RSA cryptosystem depends on the difficulty in factoring large integers. The best current factoring algorithm is the Number Field Sieve (NFS), and its most difficult part is the sieving step. In 1999 a large distributed computation involving thousands of workstations working for many months managed to factor a 512-bit RSA key, but 1024-bit keys were believed to be safe for the next 15-20 years. In this paper we describe a new hardware implementation of the NFS sieving step ... which is 3-4 orders of magnitude more cost effective than the best previously published designs Based on a detailed analysis of all the critical components (but without an actual implementation),

we believe that the NFS sieving step for 1024-bit RSA keys can be completed in less than a year with a \$10M device, and that the NFS sieving step for 512-bit RSA keys can be completed in less than ten minutes with a \$10K device. Coupled with recent results about the difficulty of the NFS matrix step ... this raises some concerns about the security of these key sizes.”

A detailed explanation from these two authors also can be found in the RSA Laboratories CryptoBytes [Shamir2003a].

The 3-page article in the DuD issue of June 2003 [Weis2003] contains a very good explanation, how the attack using the Generalized Number Field Sieve (GNFS) works and which progress is made, to factorize numbers. At GNFS we can distinguish 2 general steps: The sieve step (relation collecting) and the matrix reduction. Besides the sieve step is highly parallelizable, it dominates the overall calculation burden. Shamir and Tromer haven’t built a TWIRL device yet, but the estimated costs of 10 till 50 million Euro (in order to factorize a 1024-bit number) is not prohibitive for secret agencies or big criminal organizations, because the “costs for a single espionage satellite is estimated e.g. to be several billion USD”. The authors therefore recommend, to get as soon as possible rid of today used sensible RSA, Diffie-Hellman or ElGamal keys up to 1024 bit and to use then keys of at least 2048 bit length. The planned T CPA/Palladium hardware will use 2048-bit RSA keys!

So recommendations like the ones from the GISA (German Information Security Agency) to use higher key lengths are very valid.

“Primes in P”: Primality testing is polynomial

In August 2002 the three Indian researchers M. Agrawal, N. Kayal and N. Saxena published the paper “*PRIMES in P*” about a new primality testing algorithm called AKS [Agrawal2002]. They discovered a polynomial time deterministic algorithm for determining if a number is prime or not.

The importance of this discovery is that it provides number theorists with new insights and opportunities for further research. Lots of people over centuries have been looking for a polynomial time test for primality, and this result is a major theoretic breakthrough. It shows that new results can be generated from already known facts.

But even its authors note that other known algorithms may be faster (for example ECPP). The new algorithm works on any integer. For example the GIMPS project uses the Lucas-Lehmer primality test which takes advantage of the special properties of Mersenne numbers. This makes the Lucas-Lehmer test much faster, allowing to test numbers with millions of digits while general purpose algorithms are limited to numbers with a few thousand digits.

Current research results on this topic can be found at:

<http://www.mersenne.org/>

<http://fatphil.org/maths/AKS/> Original paper in English

<http://ls2-www.cs.uni-dortmund.de/lehre/winter200203/kt/material/primes.ps>

Good explanation in German by Thomas Hofmeister.

Joanne K. Rowling⁶⁸:

It is our choices, that show what we truly are, far more than our abilities.

4.12 Applications of asymmetric cryptography using numerical examples

The results of modular arithmetic are used extensively in modern cryptography. Here we will provide a few examples from cryptography using small⁶⁹ numbers.

Enciphering a text entails applying a function (mathematical operation) to a character string (number) to generate a different number. Deciphering entails reversing this function, in other words using the distorted image that the function has created from the plaintext in order to restore the original image. For example, the sender could take the plaintext M of a confidential message and add a secret number, the key S , to obtain the ciphertext C :

$$C = M + S.$$

The recipient can reconstruct the plaintext by reversing this operation, in other words by subtracting S :

$$M = C - S.$$

Adding S reliably makes the plaintext impossible to read. However, this encryption is rather weak, because all an interceptor needs to do to calculate the key is obtain a plaintext and the associated ciphertext

$$S = C - M,$$

and can then read any subsequent messages encrypted using S .

The essential reason for this is that subtraction is just as simple an operation as addition.

4.12.1 One way functions

If the key is to be impossible to determine even with knowledge of both the plaintext and the ciphertext, we need a function that is, on the one hand, relatively easy to calculate – we don't want to have problems encrypting messages. On the other hand, the inverse function should exist (otherwise information would be lost during encryption), but should be de facto incalculable.

What are possible candidates for such a **one way function**? We could take multiplication rather than addition, but even primary school children know that the inverse function, division, is only slightly more difficult than multiplication itself. We need to go one step higher in the hierarchy of calculation methods. It is still relatively simple to calculate the power of a number, but the corresponding two reverse functions – *taking roots* (find b in the equation $a = b^c$ when a and c are known) and *calculating logarithms* (find c in the above equation when a and b are known) are so complicated that pupils normally do not learn them at school.

Although a certain structure can still be recognised for addition and multiplication, raising numbers to the power of another or calculating exponentials totally mixes up all the numbers.

⁶⁸Joanne K. Rowling, "Harry Potter and the Chamber of Secrets", Bloomsbury, 1998, last chapter "Dobby's reward", p. 245, by Dumbledore.

⁶⁹In the RSA procedure, we call numbers "small" if the bit lengths are much less than 1024 bits (i.e. 308 decimal points). In practice, 1024 bits is currently the minimum length for a secure Certification Authority RSA modulus.

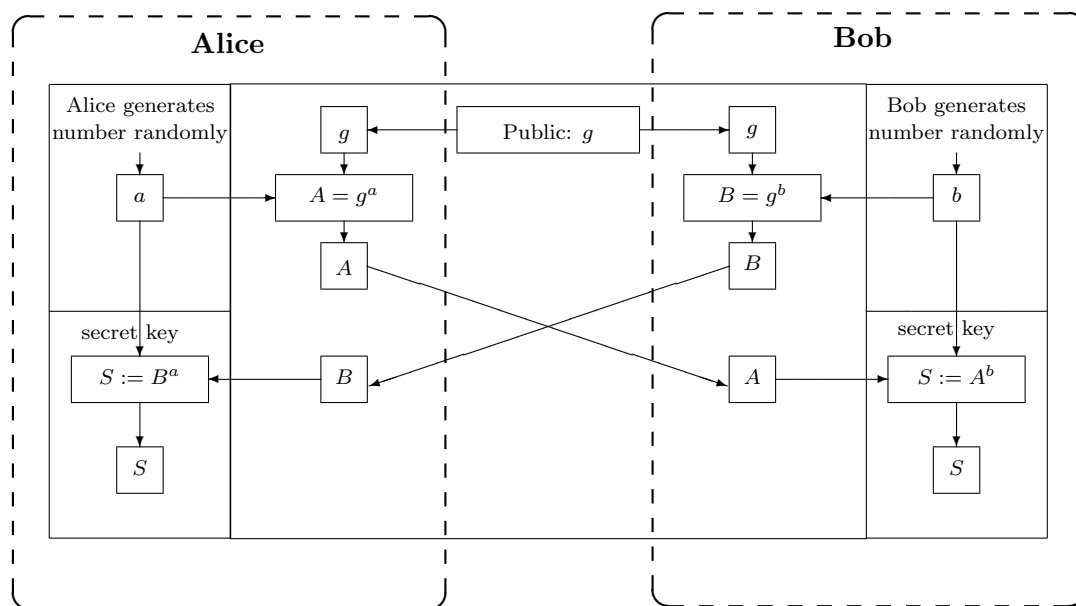
Knowing a few values of the function doesn't tell us much about the function as a whole (in contrast to addition and multiplication).

4.12.2 The Diffie-Hellman key exchange protocol

Whitfield Diffie, Martin E. Hellman and Ralph Merkle developed this DH key exchange protocol in Stanford in 1976⁷⁰.

Alice and Bob⁷¹ use a one way function to obtain a key S , the session key, for subsequent correspondence. This is then a secret that is only known to the two of them. Alice selects a random number a and keeps it secret. She applies a one way function to a to calculate the number $A = g^a$ and sends it to Bob. He does the same, by selecting a secret random number b , calculating $B = g^b$ and sending it to Alice. The number g is random and can be publicly known. Alice applies the one way function together with her secret number a to B , while Bob does the same with his secret number b and the received number A .

The result S is the same in each case because the one way function is commutative: $(g^a)^b = (g^b)^a$. But even Bob cannot reconstruct Alice's secret number a from the data available to him, while Alice cannot determine Bob's secret number b . And a perpetrator who knows g and has intercepted both A and B cannot use this knowledge to determine a, b or S .



Procedure:

Alice and Bob want to negotiate a secret session key S via a channel that may be intercepted.

1. They select a prime number p and a random number g and exchange this information openly.

⁷⁰With CrypTool this exchange protocol has been visualized: you can execute the single steps with concrete numbers using menu **Indiv. Procedures \ Protocols \ Diffie-Hellman Demonstration**.

⁷¹Bob and Alice are the default names used for the two authorized participants in a protocol (see [Schneier1996, p. 23]).

2. Alice now selects a , a random number less than p and keeps it secret.
Similarly, Bob selects b , a random number less than p and keeps it secret.
3. Alice now calculates $A \equiv g^a \pmod{p}$.
Bob calculates $B \equiv g^b \pmod{p}$.
4. Alice sends the result A to Bob.
Bob sends the result B to Alice.
5. In order to now determine the session key to be used by both, they both separately raise the respective results they have received to the power of their secret random number modulo p . This means:
 - Alice calculates $S \equiv B^a \pmod{p}$ and
 - Bob calculates $S \equiv A^b \pmod{p}$.

Even if a spy intercepts g, p , and the interim results A and B , he cannot use these in order to determine the used session key used – due to the difficulty of calculating the discrete logarithm⁷².

We will now use an example with (unrealistically) small numbers to illustrate this.

Example using small numbers:

1. Alice and Bob select $g = 11, p = 347$.
2. Alice selects $a = 240$, Bob selects $b = 39$ and they keep a and b secret.
3. Alice calculates $A \equiv g^a \equiv 11^{240} \equiv 49 \pmod{347}$.
Bob calculates $B \equiv g^b \equiv 11^{39} \equiv 285 \pmod{347}$.
4. Alice sends Bob: $A \equiv 49$,
Bob sends Alice: $B \equiv 285$.
5. Alice calculates $B^a \equiv 285^{240} \equiv 268 \pmod{347}$,
Bob calculates $A^b \equiv 49^{39} \equiv 268 \pmod{347}$.

Alice and Bob can now communicate securely using their shared session key. Even if spies were to intercept everything transferred via the connection: $g = 11, p = 347, A = 49$ and $B = 285$, they would not be able to calculate the secret key.

Comment:

In this example using such small numbers, it is easily possible to calculate the discrete logarithms, but with large numbers the discrete logarithm problem^{73,74} is extremely difficult to solve.

⁷²Further details about the discrete logarithm problem can be found in chapter 5.4.

⁷³You can use Sage to determine the discrete logarithm x that solves the equation $11^x \equiv 49 \pmod{347}$ (here for Alice): `discrete_log(mod(49, 347), mod(11, 347))`. The returned value is 67.

Such number theoretic tasks can also be solved using other tools like PariGP, LiDIA, BC or Mathematica (see the list of web sites in the appendix at the end of this chapter):

- Pari-GP: `znlog(Mod(49,347),Mod(11,347))`.

- LiDIA: `dl(11,49,347)`.

- Mathematica: The general Solve function delivers the message “The equations appear to involve the variables to be solved for in an essentially non-algebraic way”.

- Mathematica: `MultiplicativeOrder[11, 347, 49]`.

All deliver the result 67.

⁷⁴Why have the functions delivered the value 67 for the discrete logarithm of Alice rather than 240 which Alice

To get the discrete logarithms, here we need to calculate:

For Alice: $11^x \equiv 49 \pmod{347}$, that means $\log_{11}(49) \pmod{347}$.

For Bob: $11^y \equiv 285 \pmod{347}$, that means $\log_{11}(285) \pmod{347}$.

selected as exponent a ?

The discrete logarithm is the smallest natural exponent that solves the equation $11^x \equiv 49 \pmod{347}$. Both $x = 67$ and $x = 240$ (the number selected in the example) satisfy the equation and can therefore be used to calculate the session key: $285^{240} \equiv 285^{67} \equiv 268 \pmod{347}$. If Alice and Bob had selected a primitive root modulo p as base g , then for every remainder from the set $\{1, 2, \dots, p-1\}$ there is exactly one exponent from the set $\{0, 1, \dots, p-2\}$.

As an aside, there are 172 different primitive roots modulo 347, 32 of which are prime (not necessary). Since the number 11 selected for g in the example is not a primitive root of 347, the remainders do not take all values from the set $\{1, 2, \dots, 346\}$. Thus, for a particular remainder there may be more than one exponent or even no exponent at all in the set $\{0, 1, \dots, 345\}$ that satisfies the equation.

With the relevant Sage commands you find:

`is_prime(347)=True, euler_phi(347)=346, gcd(11,347)=1` and `multiplicative_order(mod(11, 347))=173`.

i	$11^i \pmod{347}$	
0	1	
1	11	
2	121	
3	290	
67	49	searched exponent
172	284	
173	1	= multiplicative order of $11^i \pmod{347}$
174	11	
175	121	
176	290	
240	49	searched exponent

Further information can be found in chapter 4.18.4 “Primitive roots”.

4.13 The RSA procedure with actual numbers⁷⁵

Having described above how the RSA procedure works, we will now work through the steps using actual, but small, numbers.

4.13.1 RSA with small prime numbers and with a number as message

Before applying the RSA procedure to a text, we will first demonstrate it directly using a single number as message⁷⁶.

1. Let the selected prime numbers be $p = 5$ and $q = 11$.
Thus, $n = 55$ and $J(n) = (p - 1) * (q - 1) = 40$.
2. $e = 7$ (should⁷⁷ lie between 11 and 39 and must be relatively prime to 40).
3. $d = 23$ (since $23 * 7 \equiv 161 \equiv 1 \pmod{40}$),
→ Public key of the recipient: $(55, 7)$,
→ Private key of the recipient: $(55, 23)$.
4. Let the message be the number $M = 2$ (so no division into blocks is required).
5. Encryption: $C \equiv 2^7 \equiv 18 \pmod{55}$.
6. The ciphertext is simply the number $C = 18$ (we therefore do not need to divide it into blocks).
7. Decryption: $M \equiv 18^{23} \equiv 18^{(1+2+4+16)} \equiv 18 * 49 * 36 * 26 \equiv 2 \pmod{55}$.

We will now apply the RSA procedure to a text, first using the upper case alphabet (26 characters), then using the entire ASCII character set as the basis for the messages.

4.13.2 RSA with slightly larger primes and a text of upper case letters

We have the text “ATTACK AT DAWN” and the characters are coded according to table 4.11.⁷⁸

Key generation (steps 1 to 3):

1. $p = 47, q = 79$ ($n = 3713$; $J(n) = (p - 1) * (q - 1) = 3588$).
2. $e = 37$ (should⁷⁹ lie between 79 and 3587 and must be relatively prime to 3588).
3. $d = 97$ (since $e * d = 1 \pmod{J(n)}$; $37 * 97 \equiv 3589 \equiv 1 \pmod{3588}$)⁸⁰.

⁷⁵Additional material: Minh Van Nguyen, “Number Theory and the RSA Public Key Cryptosystem”, 2009. An introductory tutorial on using Sage to study elementary number theory and public key cryptography. A didactically very clear article about some basic number theory and Sage usage. <http://nguyenminh2.googlepages.com/numtheory-crypto-sage.pdf>.

⁷⁶Using CrypTool you can solve this with the menu **Indiv. Procedures \ RSA Cryptosystem \ RSA Demonstration**.

⁷⁷See footnote 47 on page 112.

⁷⁸Using CrypTool you can solve this with the menu **Indiv. Procedures \ RSA Cryptosystem \ RSA Demonstration**. This is also described in the tutorial/scenario in CrypTool’s online help [Options, specify alphabet, number system, block length 2 and decimal representation].

⁷⁹See footnote 47 on page 112.

⁸⁰How to compute $d = 97$ using the *extended gcd* algorithm is shown in appendix 4.14

Character	Numerical value	Character	Numerical value
Blank	0	M	13
A	1	N	14
B	2	O	15
C	3	P	16
D	4	Q	17
E	5	R	18
F	6	S	19
G	7	T	20
H	8	U	21
I	9	V	22
J	10	W	23
K	11	X	24
L	12	Y	25
		Z	26

Table 4.11: Capital letters alphabet

4. Encryption:

Text: A T T A C K A T D A W N
Number: 01 20 20 01 03 11 00 01 20 00 04 01 23 14

This 28-digit number is divided into 4-digit parts (because 2626 is still smaller than $n = 3713$):
0120 2001 0311 0001 2000 0401 2314

All 7 parts are encrypted using: $C \equiv M^{37} \pmod{3713}$ ⁸¹:
1404 2932 3536 0001 3284 2280 2235

5. Decryption:

Ciphertext: 1404 2932 3536 0001 3284 2280 2235

This 28-digit number is divided into 4-digit parts.

All 7 parts are decrypted using: $M \equiv C^{97} \pmod{3713}$:
0120 2001 0311 0001 2000 0401 2314

The 2-digit numbers are transformed into capital letters and blanks.

Using the selected values it is easy for a cryptanalyst to derive the secret values from the public parameters $n = 3713$ and $e = 37$ by revealing that $3713 = 47 * 79$.

If n is a 768-bit number, there is, according to present knowledge, little chance of this.

4.13.3 RSA with even larger primes and a text made up of ASCII characters

In real life, the ASCII alphabet is used to code the individual characters of the message as 8-bit numbers.

The idea for this task⁸² is taken from the example in [Eckert2003, p. 271].

⁸¹See chapter 4.18.5 “RSA examples with Sage” for source code to do RSA encryption using Sage.

You can also encrypt the message with CrypTool via the menu path **Indiv. Procedures \ RSA Cryptosystem \ RSA Demonstration**.

⁸²Using CrypTool you can solve this via the menu path **Indiv. Procedures \ RSA Cryptosystem \ RSA Demonstration**.

Coded in decimal notation, the text “RSA works!” is as follows:

Text:	R	S	A		w	o	r	k	s	!
Number:	82	83	65	32	119	111	114	107	115	33

We will work through the example in 2 variants. The steps 1 to 3 are common for both.

Key generation (steps 1 to 3):

1. $p = 503$, $q = 509$ ($n = 256,027$; $J(n) = (p-1)(q-1) = 255,016 = 2^3 * 127 * 251$)⁸³.
2. $e = 65,537$
(should⁸⁴ lie between 509 and 255,015 and must⁸⁵ be relatively prime to 255,016).
3. $d = 231,953$
(since $e \equiv d^{-1} \pmod{J(n)}$: $65,537 * 231,953 \equiv 15,201,503,761 \equiv 1 \pmod{255,016}$)⁸⁶.

Variant 1: All ASCII characters are en-/decrypted separately (no blocks are formed).

4. Encryption:

Text:	R	S	A		w	o	r	k	s	!
Number:	82	83	65	32	119	111	114	107	115	33

The letters are not combined⁸⁷!

Each character is encrypted using: $C = M^{65,537} \pmod{256,027}$ ⁸⁸:

212984	025546	104529	031692	248407
100412	054196	100184	058179	227433

5. Decryption:

Ciphertext:

212984	025546	104529	031692	248407
100412	054196	100184	058179	227433

Each character is decrypted using: $M \equiv C^{231,953} \pmod{256,027}$:

82 83 65 32 119 111 114 107 115 33

Variant 2: The ASCII characters are en-/decrypted two at a time as blocks.

In variant 2 the block formation is done in two different sub-variants: (4./5. and 4'./5').

⁸³See chapter 4.18.5 “RSA examples with Sage” for the source code to factorize the number $J(n)$ using Sage. Using CrypTool you can solve this with the **Indiv. Procedures \ RSA Cryptosystem \ Factorization of a Number**.

⁸⁴See footnote 47 on page 112.

⁸⁵ e cannot, therefore, be 2, 127 or 251 ($65,537 = 2^{16} + 1$) ($255,016 = 2^3 * 127 * 251$).

In real life, $J(n)$ is not factorized but rather the Euclidean algorithm is used for the selected e to guarantee that $\gcd(e, J(n)) = 1$.

⁸⁶Other possible combinations of (e, d) include: (3, 170, 011), (5, 204, 013), (7, 36, 431).

⁸⁷For secure procedures we need large numbers that assume – as far as possible – all values up to $n - 1$. If the possible value set for the numbers in the message is too small, even large prime numbers cannot make the procedure secure. An ASCII character is represented by 8 bits. If we want larger values we must combine several numbers. Two characters need 16 bits, whereby the maximum value that can be represented is 65536. The modulus n must then be greater than $2^{16} = 65536$. This is applied in variant 2. When the numbers are combined, the leading zeros are kept in binary notation (just as if we were to write all numbers with 3 digits in decimal notation above and were then to obtain the sequence 082 083, 065 032, 119 111, 114 107, 115 033).

⁸⁸See chapter 4.18.5 “RSA examples with Sage” for the source code for RSA exponentiation using Sage.

Text: R S A w o r k s !
 Number: 82 83 65 32 119 111 114 107 115 33

4. Encryption:

Blocks are formed⁸⁹ (each ASCII character is encoded into a 8 digit binary number below):
 21075 16672 30575 29291 29473⁹⁰

Each block is encrypted using: $C \equiv M^{65,537} \pmod{256,027}$ ⁹¹:
 158721 137346 37358 240130 112898

5. Decryption:

Ciphertext:
 158721 137346 37358 240130 112898

Each block is decrypted using: $M \equiv C^{231,953} \pmod{256,027}$:
 21075 16672 30575 29291 29473

4'. Encryption:

Blocks are formed: (each ASCII character is encoded into a 3 digit decimal number below):
 82083 65032 119111 114107 115033⁹²

Each block is encrypted using: $C \equiv M^{65,537} \pmod{256,027}$ ⁹³:
 198967 051405 254571 115318 014251

5'. Decryption:

Ciphertext:
 198967 051405 254571 115318 014251

Each block is decrypted using: $M \equiv C^{2473} \pmod{67,519}$:
 82083 65032 119111 114107 115033

4.13.4 A small RSA cipher challenge (1)

The task is taken from [Stinson1995, Exercise 4.6]: The pure solution has been published by Prof. Stinson at <http://www.cacr.math.uwaterloo.ca/~dstinson/solns.html>.⁹⁴

However, it is not the result that is important here but rather the individual steps of the solution, that is, the explanation of the cryptanalysis⁹⁵:

⁸⁹

single character	binary representation	decimal representation
01010010, 82	01010010 01010011	= 21075
01010011, 83		
01000001, 65	01000001 00100000	= 16672
00100000, 32		
01110111, 119	01110111 01101111	= 30575
01101111, 111		
01110010, 114	01110010 01101011	= 29291
01101011, 107		
01110011, 115	01110011 00100001	= 29473
00100001, 33:		

⁹⁰Using CrypTool you can solve this with the menu **Indiv. Procedures \ RSA Cryptosystem \ RSA Demonstration** with the following options: all 256 ASCII characters, b-adic, block length 2 and decimal representation.

⁹¹See chapter 4.18.5 “RSA examples with Sage” for the source code for RSA exponentiation using Sage.

⁹²The RSA encryption works correctly with the modulus $n = 256.027$ because each ASCII block of two characters will be encoded into a number that is smaller or equal than the number 255,255.

⁹³See chapter 4.18.5 “RSA examples with Sage” for the source code for RSA exponentiation using Sage.

⁹⁴or <http://bibd.unl/~stinson/solns.html>.

⁹⁵The method of solving the problem is outlined in the scenario of the online help to CrypTool and in the pre-

Two samples of RSA ciphertext are presented in Tables 4.12⁹⁶ and 4.13⁹⁷. Your task is to decrypt them. The public parameters of the system are

$n = 18,923$ and $e = 1261$ (for Table 4.12) and
 $n = 31,313$ and $e = 4913$ (for Table 4.13).

This can be accomplished as follows. First, factor n (which is easy because it is so small). Then compute the exponent d from $J(n)$, and, finally, decrypt the ciphertext. Use the square-and-multiply algorithm to exponentiate modulo n .

In order to translate the plaintext back into ordinary English text, you need to know how alphabetic characters are “encoded” as elements in \mathbb{Z}_n . Each element of \mathbb{Z}_n represents three alphabetic characters as in the following examples:

DOG $\mapsto 3 * 26^2 + 14 * 26 + 6 = 2398$
CAT $\mapsto 2 * 26^2 + 0 * 26 + 19 = 1371$
ZZZ $\mapsto 25 * 26^2 + 25 * 26 + 25 = 17,575$.

You will have to invert this process as the final step in your program.

The first plaintext was taken from “The Diary of Samuel Marchbanks”, by Robertson Davies, 1947, and the second was taken from “Lake Wobegon Days”, by Garrison Keillor, 1985.

4.13.5 A small RSA cipher challenge (2)

The following task is a corrected version from the book written by Prof. Yan [Yan2000, Example 3.3.7, p. 318]. However, it is not the result that is important here but rather the individual steps of the solution, that is, the explanation of the cryptanalysis⁹⁸.

There are three tasks with completely different degrees of difficulty here. In each case we know the ciphertext and the public key (e, n) :

- (a) Known plaintext: find the secret key d using the additionally known original message.
- (b) Ciphertext-only: find d and the plaintext.
- (c) Calculate the RSA modulus, in other words factorization (with no knowledge of the message).

sentation on the website. If anyone sends us a well prepared exact method of solving the problem, we would be pleased to include it in the documentation.

⁹⁶The numbers of this table can be worked with via Copy and Paste.

⁹⁷The numbers of this table are in the online-help “Example illustrating the RSA demonstration” of CrypTool.

⁹⁸The method of solving the problem is outlined in the scenario of the online help to CrypTool and in the CrypTool presentation. If anyone sends us a well prepared exact method of solving the problem, we would be pleased to include it in the documentation.

12423	11524	7243	7459	14303	6127	10964	16399
9792	13629	14407	18817	18830	13556	3159	16647
5300	13951	81	8986	8007	13167	10022	17213
2264	961	17459	4101	2999	14569	17183	15827
12693	9553	18194	3830	2664	13998	12501	18873
12161	13071	16900	7233	8270	17086	9792	14266
13236	5300	13951	8850	12129	6091	18110	3332
15061	12347	7817	7946	11675	13924	13892	18031
2620	6276	8500	201	8850	11178	16477	10161
3533	13842	7537	12259	18110	44	2364	15570
3460	9886	8687	4481	11231	7547	11383	17910
12867	13203	5102	4742	5053	15407	2976	9330
12192	56	2471	15334	841	13995	17592	13297
2430	9741	11675	424	6686	738	13874	8168
7913	6246	14301	1144	9056	15967	7328	13203
796	195	9872	16979	15404	14130	9105	2001
9792	14251	1498	11296	1105	4502	16979	1105
56	4118	11302	5988	3363	15827	6928	4191
4277	10617	874	13211	11821	3090	18110	44
2364	15570	3460	9886	9988	3798	1158	9872
16979	15404	6127	9872	3652	14838	7437	2540
1367	2512	14407	5053	1521	297	10935	17137
2186	9433	13293	7555	13618	13000	6490	5310
18676	4782	11374	446	4165	11634	3846	14611
2364	6789	11634	4493	4063	4576	17955	7965
11748	14616	11453	17666	925	56	4118	18031
9522	14838	7437	3880	11476	8305	5102	2999
18628	14326	9175	9061	650	18110	8720	15404
2951	722	15334	841	15610	2443	11056	2186

Table 4.12: RSA ciphertext A

$n = 63978486879527143858831415041$, $e = 17579$

Message⁹⁹:

1401202118011200,
1421130205181900,
0118050013010405,
0002250007150400

Cipher:

45411667895024938209259253423,
16597091621432020076311552201,
46468979279750354732637631044,
32870167545903741339819671379

Comment:

The original message consisted of a sentence containing 31 characters (coded with the capital

⁹⁹The numbers of this table are in the online help “Example illustrating the RSA demonstration” of CrypTool.

6340	8309	14010	8936	27358	25023	16481	25809
23614	7135	24996	30590	27570	26486	30388	9395
27584	14999	4517	12146	29421	26439	1606	17881
25774	7647	23901	7372	25774	18436	12056	13547
7908	8635	2149	1908	22076	7372	8686	1304
4082	11803	5314	107	7359	22470	7372	22827
15698	30317	4685	14696	30388	8671	29956	15705
1417	26905	25809	28347	26277	7897	20240	21519
12437	1108	27106	18743	24144	10685	25234	30155
23005	8267	9917	7994	9694	2149	10042	27705
15930	29748	8635	23645	11738	24591	20240	27212
27486	9741	2149	29329	2149	5501	14015	30155
18154	22319	27705	20321	23254	13624	3249	5443
2149	16975	16087	14600	27705	19386	7325	26277
19554	23614	7553	4734	8091	23973	14015	107
3183	17347	25234	4595	21498	6360	19837	8463
6000	31280	29413	2066	369	23204	8425	7792
25973	4477	30989					

Table 4.13: RSA ciphertext B

letters alphabet from section 4.13.2). Each group of 16 decimal numbers is then combined to form one number (the last number is filled with zeros). These numbers are raised to the power of e .

When you decrypt the message you must fill the calculated numbers with leading zeros in order to obtain plaintext.

This needs to be stressed because the type of padding is extremely important during implementation and standardization for interoperable algorithms.

4.14 Appendix: The greatest common divisor (gcd) of whole numbers and the two algorithms of Euclid¹⁰⁰

The greatest common divisor of two natural numbers a and b is an important value that can be calculated very quickly. Here we make use of the fact that if a number c divides the numbers a and b (i.e. there exists an a' and a b' such that $a = a' * c$ and $b = b' * c$), then c also divides the remainder r of a/b . In short notation we can write: If c divides a and b it follows that c divides $r = a - \lfloor a/b \rfloor * b$ ¹⁰¹.

As the latter statement is valid for each common divisor c of a and b it follows that:

$$\gcd(a, b) = \gcd(a - \lfloor a/b \rfloor * b, b).$$

Using this information, the algorithm for calculating the gcd of two numbers can be written as follows (in pseudo code):

```
INPUT: a, b != 0
1. if ( a < b ) then  x = a; a = b; b = x; // Swap a and b (a > b)
2. a = a - int(a/b) * b           // a is smaller than b, the
                                   // gcd(a, b) is unchanged
3. if ( a != 0 ) then goto 1.     // a falls after each step and
                                   // the algorithm ends when a==0.
OUTPUT "gcd(a,b) = " b          // b is the gcd of the original a and b
```

Also further relationships can be derived from the gcd: For this, we need the set of equations for a and b :

$$\begin{aligned} a &= 1 * a + 0 * b \\ b &= 0 * a + 1 * b, \end{aligned}$$

or, in matrix notation:

$$\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} * \begin{pmatrix} a \\ b \end{pmatrix}.$$

We summarize this information in the extended matrix:

$$\left(\begin{array}{cc|cc} a & & 1 & 0 \\ b & & 0 & 1 \end{array} \right)$$

If we apply the above gcd algorithm to this matrix, we obtain the *extended Euclid algorithm* which can be used to calculate the multiplicative inverse:

¹⁰⁰With the educational tool for number theory **NT** you can see

a) how Euklid's algorithm calculates the gcd (learning unit 1.3, pages 14-19/21) and

b) how Euklid's enhanced algorithm finds the multiplicative inverse (learning unit 2.2, page 13/40).

NT can be called in CrypTool via the menu path **Indiv. Procedures \ Number Theory Interactive \ Learning Tool for Number Theory**. See appendix A.4.

¹⁰¹The Gauss bracket $\lfloor x \rfloor$ of a real number x is defined via: $\lfloor x \rfloor$ is the next integer less or equal x .

INPUT: $a, b \neq 0$

0. $x_{1,1} := 1, x_{1,2} := 0, x_{2,1} := 0, x_{2,2} := 1$

$$1. \left(\begin{array}{c|cc} a & x_{1,1} & x_{1,2} \\ b & x_{2,1} & x_{2,2} \end{array} \right) := \left(\begin{array}{cc} 0 & 1 \\ 1 & -\lfloor a/b \rfloor * b \end{array} \right) * \left(\begin{array}{c|cc} a & x_{1,1} & x_{1,2} \\ b & x_{2,1} & x_{2,2} \end{array} \right).$$

2. if $(b \neq 0)$ then goto 1.

OUTPUT: “gcd(a, b) = $a * x + b * y$: ”, “gcd(a, b) = ” b , “ x =” $x_{2,1}$, “ y =” $x_{2,2}$

Since this algorithm only performs linear transformations, the same equations always apply

$$\begin{aligned} a &= x_{1,1} * a + x_{1,2} * b \\ b &= x_{2,1} * a + x_{2,2} * b, \end{aligned}$$

We get the extended gcd equation at the end of the algorithm¹⁰²:

$$\gcd(a, b) = a * x_{2,1} + b * x_{2,2}.$$

Example:

Using the extended gcd we can determine for $e = 37$ the multiplicative inverse number d to modulo 3588 (i.e. $37 * d \equiv 1 \pmod{3588}$):

$$\begin{aligned} 0. & \left(\begin{array}{c|cc} 3588 & 1 & 0 \\ 37 & 0 & 1 \end{array} \right) \\ 1. & \left(\begin{array}{c|cc} 37 & 1 & 0 \\ 36 & 0 & -96 \end{array} \right) = \left(\begin{array}{cc} 0 & 1 \\ 1 & -(\lfloor 3588/36 \rfloor = 96) * 37 \end{array} \right) * \left(\begin{array}{c|cc} 3588 & 1 & 0 \\ 37 & 0 & 1 \end{array} \right). \\ 2. & \left(\begin{array}{c|cc} 36 & 1 & -96 \\ 1 & -1 & 97 \end{array} \right) = \left(\begin{array}{cc} 0 & 1 \\ 1 & -(\lfloor 37/36 \rfloor = 1) * 36 \end{array} \right) * \left(\begin{array}{c|cc} 37 & 1 & 0 \\ 36 & 0 & -96 \end{array} \right). \\ 3. & \left(\begin{array}{c|cc} 1 & -1 & 97 \\ 0 & 37 & -3588 \end{array} \right) = \left(\begin{array}{cc} 0 & 1 \\ 1 & -(\lfloor 36/1 \rfloor = 36) * 1 \end{array} \right) * \left(\begin{array}{c|cc} 36 & 1 & -96 \\ 1 & -1 & 97 \end{array} \right). \end{aligned}$$

OUTPUT:

gcd(37, 3588) = $a * x + b * y$:

gcd(37, 3588) = 1, $x = -1$, $y = 97$.

Thus

1. 37 and 3588 are relatively prime (37 has an inverse modulo 3588).
2. $37 * 97 = (1 * 3588) + 1$ in other words $37 * 97 \equiv 1 \pmod{3588}$.
and therefore the number 97 is the multiplicative inverse to 37 modulo 3588.

¹⁰²By termination of the gcd algorithm, the program variables a and b contain the values $a = 0$ and $b = \gcd(a, b)$. Please keep in mind, that the program variables are different to the numbers a and b and that they are only relevant for the scope of the algorithm.

4.15 Appendix: Forming closed sets

The property of closeness within a set is always defined in relation to an operation. The following shows how to construct the “closed set” G with respect to the operation $+$ (mod 8) for a given initial set G_0 :

$$\begin{aligned}
 G_0 &= \{2, 3\} \quad \text{--- addition of the numbers in } G_0 \text{ determines further numbers :} \\
 &\quad 2 + 3 \equiv 5 \pmod{8} = 5 \\
 &\quad 2 + 2 \equiv 4 \pmod{8} = 4 \\
 &\quad 3 + 3 \equiv 6 \pmod{8} = 6 \\
 G_1 &= \{2, 3, 4, 5, 6\} \quad \text{--- addition of the numbers in } G_1 \text{ determines :} \\
 &\quad 3 + 4 \equiv 7 \pmod{8} = 7 \\
 &\quad 3 + 5 \equiv 8 \pmod{8} = 0 \\
 &\quad 3 + 6 \equiv 9 \pmod{8} = 1 \\
 G_2 &= \{0, 1, 2, 3, 4, 5, 6, 7\} \quad \text{--- addition of the numbers in } G_2 \text{ does not extend the set!} \\
 G_3 &= G_2 \quad \text{--- we say : } G_2 \text{ is closed for addition (mod 8).}
 \end{aligned}$$

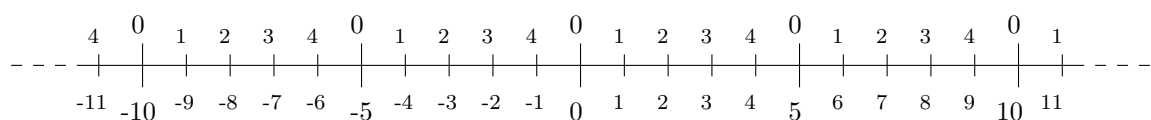
4.16 Appendix: Comments on modulo subtraction

Comment on subtraction modulo 5: $2 - 4 = -2 \equiv 3 \pmod{5}$.

It is therefore not true that $-2 = 2 \pmod{5}$!

People often make the mistake of equating this. You can show this clearly if you place the permutation $(0, 1, 2, 3, 4)$ in \mathbb{Z}_5 , for example from -11 to $+11$, over the range of numbers in \mathbb{Z} .

range of numbers modulo 5



range of numbers in \mathbb{Z}

4.17 Appendix: Base representation and base transformation of numbers, estimation of length of digits

For a given number z one may ask how to represent such a number. In general we use representations like $z = 2374$ or $z = \sqrt{2}$. The second number consists of an infinite number of digits and therefore it can never be described precisely by the first representation. You can get around this problem by writing the number symbolically. But if you have to write it in digits, the number must be rounded.

We represent numbers usually in the decimal system (base 10). Computers are working with the binary representation of numbers — only for the display numbers are represented in decimal or sometimes hexadecimal (base 16) form.

This appendix describes how to generate arbitrary base representations of any positive integer and how to determine the number of required digits via the logarithm function.

b -adic sum representation of positive integers

Given base b , each positive integer z can be represented as a b -adic sum

$$z = a_n b^n + a_{n-1} b^{n-1} + \cdots + a_1 b + a_0,$$

where $a_i \in \{0, 1, \dots, b-1\}$, $i = 0, 1, \dots, n$ are called *digits*.

For this sum, it follows that:

- 1) For arbitrary digits a_0, a_1, \dots, a_n it is: $b^{n+1} > a_n b^n + a_{n-1} b^{n-1} + \cdots + a_1 b + a_0$.
 - 2) There exist digits a_0, a_1, \dots, a_n (namely $a_i = b-1$ for $i = 0, \dots, n$), following that $b^{n+1} - 1 \leq a_n b^n + a_{n-1} b^{n-1} + \cdots + a_1 b + a_0$.
- (Using these inequalities it can be shown that each positive integer can be represented by a b -adic sum).

By writing the digits $a_n a_{n-1} \cdots a_1 a_0$ in a row directly after each other (without the b^i) the usual writing for numbers comes to hand.

Example:

Base $b = 10$: $10278 = 1 \cdot 10^4 + 0 \cdot 10^3 + 2 \cdot 10^2 + 7 \cdot 10^1 + 8$

Base $b = 16$: $FE70A = 15 \cdot 16^4 + 14 \cdot 16^3 + 7 \cdot 16^2 + 0 \cdot 16^1 + 10$.

Number of digits to represent a positive integer

For a positive integer z the length of the b -adic representation can be determined via the following steps. Starting from the inequality $b^{n+1} > z \geq b^n$ we have — after applying the logarithm function on basis b^{103} : $n+1 > \log_b z \geq n$. Therefore we have $n = \lfloor \log_b z \rfloor^{104}$. We call $l_b(z)$ the number of required digits to represent the number z on the base b . We have

$$l_b(z) := \lfloor \log_b z \rfloor + 1.$$

Example 1 (decimal→hex):

We compute for the decimal number $z = 234$ (EA in hex) the hexadecimal representation

¹⁰³Applying the logarithm formula on base b and b' we have $\log_b z = \log_{b'} z / \log_{b'}(b)$. It is therefore easy using e.g. logarithm tables for the base $b' = 10$ to compute the logarithm of base $b = 2$.

¹⁰⁴The function $\lfloor x \rfloor$ determines the next integer smaller than x (in case $x \geq 0$ the digits after the decimal point are truncated).

(number base $b = 16$)

$$l_{16}(z) = \lfloor \log_{16}(z) \rfloor + 1 = \lfloor \ln(z)/\ln(16) \rfloor + 1 = \lfloor 1.96... \rfloor + 1 = 1 + 1 = 2.$$

Example 2 (decimal→binary):

We compute for the decimal number $z = 234$ (11101010 in binary) the binary representation (number base $b = 2$)

$$l_2(z) = \lfloor \log_2(z) \rfloor + 1 = \lfloor \ln(z)/\ln(2) \rfloor + 1 = \lfloor 7.87... \rfloor + 1 = 7 + 1 = 8.$$

Example 3 (binary→decimal):

We compute for the decimal number $z = 11101010$ (234 decimal) the decimal representation (number base $b = 10$)

$$l_{10}(z) = \lfloor \log_{10}(z) \rfloor + 1 = \lfloor \ln(z)/\ln(10) \rfloor + 1 = \lfloor 2,36... \rfloor + 1 = 2 + 1 = 3.$$

Algorithm to compute the base representation

Given the number z one can compute the base b representation of z using the following algorithm:

input: z, b

$n := 0, z' := z$

while $z' > 0$ **do**

$a_n := z' \bmod b,$

$z' := \lfloor z'/b \rfloor$

$n := n + 1$

end do

output: $a_n a_{n-1} \cdots a_1 a_0$ in base b representation.

Example 1 (decimal→hex):

The integer $z = 234$ on the number base 10 will be transformed into the hex representation via $a_0 = 234 \bmod 16 = 10 = A$, $234/16 = 14 = E$,

$a_1 = 14 \bmod 16 = E$

and therefore we have EA .

Example 2 (binary→decimal):

The binary number $z = 1000100101110101$ is transformed into the decimal representation via the following steps:

$1000100101110101 = 1001 \pmod{1010} \implies a_0 = 9, \quad 1000100101110101/1010 = 110110111110$

$110110111110 = 1000 \pmod{1010} \implies a_1 = 8, \quad 110110111110/1010 = 101011111$

$101011111 = 1 \pmod{1010} \implies a_2 = 1, \quad 10101111/1010 = 100011$

$100011 = 101 \pmod{1010} \implies a_3 = 5, \quad 100011/1010 = 1$

$11 = 11 \pmod{1010} \implies a_4 = 3$

therefore $z = 35189$.

4.18 Appendix: Examples using Sage

Below is Sage source code related to contents of the chapter 4 (“Introduction to Elementary Number Theory with Examples”).

4.18.1 Multiplication table modulo m

The multiplication table 4.4 (from page 99) for $a \times i \pmod{m}$, where $m = 17$, $a = 5$ and $a = 6$, and i ranges over all integers from 0 to 16 can be computed using Sage as follows:

Sage sample 4.1 Multiplication tables for $a \times i \pmod{m}$ with $m = 17$, $a = 5$ and $a = 6$

```
sage: m = 17; a = 5; b = 6
sage: [mod(a * i, m).lift() for i in xrange(m)]
[0, 5, 10, 15, 3, 8, 13, 1, 6, 11, 16, 4, 9, 14, 2, 7, 12]
sage: [mod(b * i, m).lift() for i in xrange(m)]
[0, 6, 12, 1, 7, 13, 2, 8, 14, 3, 9, 15, 4, 10, 16, 5, 11]
```

The function `mod()` returns an object that represents integers modulo m (in our case $m = 17$). From the `Mod` object you can get its single components either with the function `component` or with the function `lift`. We use the method `lift()` to convert that object to an integer representation.

The other multiplication table examples modulo 13 (table 4.5) and modulo 12 (table 4.6) on page 99 can similarly be computed by replacing `m = 17` with `m = 13` and `m = 12` respectively.

4.18.2 Fast exponentiation

The fast exponentiation modulo m can be computed using the Sage function `power_mod()`. The result of this function is an integer. We can compute the exponentiation in the example in chapter “Fast calculation of high powers” on page 101 as follows:

Sage sample 4.2 Fast exponentiation mod $m = 103$

```
sage: a = 87; m = 103
sage: exp = [2, 4, 8, 16, 32, 43]
sage: [power_mod(a, e, m) for e in exp]
[50, 28, 63, 55, 38, 85]
```

4.18.3 Multiplicative order

The order $\text{ord}_m(a)$ of a number a in the multiplicative group \mathbf{Z}_m^* is the smallest number $i \geq 1$ such that $a^i \equiv 1 \pmod{m}$ holds (see chapter 4.9, “Multiplicative order and primitive roots”). To create table 4.7 on page 108 we can print all exponentiation $a^i \pmod{11}$ as follows:

Sage sample 4.3 Table with all powers $a^i \pmod{m}$ for $m = 11$, $a = 1, \dots, 10$

```
sage: m = 11
sage: for a in xrange(1, m):
....:     print [power_mod(a, i, m) for i in xrange(1, m)]
....:
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[2, 4, 8, 5, 10, 9, 7, 3, 6, 1]
[3, 9, 5, 4, 1, 3, 9, 5, 4, 1]
[4, 5, 9, 3, 1, 4, 5, 9, 3, 1]
[5, 3, 4, 9, 1, 5, 3, 4, 9, 1]
[6, 3, 7, 9, 10, 5, 8, 4, 2, 1]
[7, 5, 2, 3, 10, 4, 6, 9, 8, 1]
[8, 9, 6, 4, 10, 3, 2, 5, 7, 1]
[9, 4, 3, 5, 1, 9, 4, 3, 5, 1]
[10, 1, 10, 1, 10, 1, 10, 1, 10, 1]
```

and including the last column with the order of each $a \pmod{11}$

```
sage: m = 11
sage: for a in xrange(1, m):
....:     lst= [power_mod(a, i, m) for i in xrange(1, m)]
....:     lst.append(multiplicative_order(mod(a,m)))
....:     print lst
....:
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[2, 4, 8, 5, 10, 9, 7, 3, 6, 1, 10]
[3, 9, 5, 4, 1, 3, 9, 5, 4, 1, 5]
[4, 5, 9, 3, 1, 4, 5, 9, 3, 1, 5]
[5, 3, 4, 9, 1, 5, 3, 4, 9, 1, 5]
[6, 3, 7, 9, 10, 5, 8, 4, 2, 1, 10]
[7, 5, 2, 3, 10, 4, 6, 9, 8, 1, 10]
[8, 9, 6, 4, 10, 3, 2, 5, 7, 1, 10]
[9, 4, 3, 5, 1, 9, 4, 3, 5, 1, 5]
[10, 1, 10, 1, 10, 1, 10, 1, 10, 1, 2]
```

Table 4.8 on page 109 gives examples for the order modulo 45 $\text{ord}_{45}(a)$ and the Euler number $J(45)$.

The following Sage code constructs a table similar to that on page 109.

Sage sample 4.4 Table with all powers $a^i \pmod{45}$ for $a = 1, \dots, 12$ plus the order of a

```
sage: m = 45
sage: for a in xrange(1, 13):
....:     lst = [power_mod(a, i, m) for i in xrange(1, 13)]
....:     try:
....:         lst.append(multiplicative_order(mod(a, m)))
....:     except:
....:         lst.append("None")
....:     lst.append(euler_phi(m))
....:     print lst
....:
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 24]
[2, 4, 8, 16, 32, 19, 38, 31, 17, 34, 23, 1, 12, 24]
[3, 9, 27, 36, 18, 9, 27, 36, 18, 9, 27, 36, 'None', 24]
[4, 16, 19, 31, 34, 1, 4, 16, 19, 31, 34, 1, 6, 24]
[5, 25, 35, 40, 20, 10, 5, 25, 35, 40, 20, 10, 'None', 24]
[6, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 36, 'None', 24]
[7, 4, 28, 16, 22, 19, 43, 31, 37, 34, 13, 1, 12, 24]
[8, 19, 17, 1, 8, 19, 17, 1, 8, 19, 17, 1, 4, 24]
[9, 36, 9, 36, 9, 36, 9, 36, 9, 36, 9, 36, 'None', 24]
[10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 'None', 24]
[11, 31, 26, 16, 41, 1, 11, 31, 26, 16, 41, 1, 6, 24]
[12, 9, 18, 36, 27, 9, 18, 36, 27, 9, 18, 36, 'None', 24]
```

The number $\text{ord}_m(a)$ only exists if a is relatively prime to m , which can be checked with $\text{gcd}(a, m)$.

In the above code example, we put the calculation of the multiplicative order within a `try-except` block. This allows Sage to catch any exceptions or errors raised by the function `multiplicative_order()`. If an exception or error is raised in the `try` block, then we know that $\text{ord}_m(a)$ does not exist for that particular value of a , hence in the `except` block we append the string "None" to the row as represented by the object `lst`.

Table 4.9 on page 110 displays exponentiation $a^i \pmod{46}$ as well as the order $\text{ord}_{46}(a)$.

Sage can create that table as follows:

Sage sample 4.5 Table with all powers $a^i \pmod{46}$ for $a = 1, \dots, 23$ plus the order of a

```
sage: m = 46
sage: euler_phi(m)
22
sage: for a in xrange(1, 24):
....:     lst = [power_mod(a, i, m) for i in xrange(1, 24)]
....:     try:
....:         lst.append(multiplicative_order(mod(a, m)))
....:     except:
....:         lst.append("None")
....:     print lst
....:
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[2, 4, 8, 16, 32, 18, 36, 26, 6, 12, 24, 2, 4, 8, 16, 32, 18, 36, 26, 6, 12, 24, 2, 'None']
[3, 9, 27, 35, 13, 39, 25, 29, 41, 31, 1, 3, 9, 27, 35, 13, 39, 25, 29, 41, 31, 1, 3, 11]
[4, 16, 18, 26, 12, 2, 8, 32, 36, 6, 24, 4, 16, 18, 26, 12, 2, 8, 32, 36, 6, 24, 4, 'None']
[5, 25, 33, 27, 43, 31, 17, 39, 11, 9, 45, 41, 21, 13, 19, 3, 15, 29, 7, 35, 37, 1, 5, 22]
[6, 36, 32, 8, 2, 12, 26, 18, 16, 4, 24, 6, 36, 32, 8, 2, 12, 26, 18, 16, 4, 24, 6, 'None']
[7, 3, 21, 9, 17, 27, 5, 35, 15, 13, 45, 39, 43, 25, 37, 29, 19, 41, 11, 31, 33, 1, 7, 22]
[8, 18, 6, 2, 16, 36, 12, 4, 32, 26, 24, 8, 18, 6, 2, 16, 36, 12, 4, 32, 26, 24, 8, 'None']
[9, 35, 39, 29, 31, 3, 27, 13, 25, 41, 1, 9, 35, 39, 29, 31, 3, 27, 13, 25, 41, 1, 9, 11]
[10, 8, 34, 18, 42, 6, 14, 2, 20, 16, 22, 36, 38, 12, 28, 4, 40, 32, 44, 26, 30, 24, 10, 'None']
[11, 29, 43, 13, 5, 9, 7, 31, 19, 25, 45, 35, 17, 3, 33, 41, 37, 39, 15, 27, 21, 1, 11, 22]
[12, 6, 26, 36, 18, 32, 16, 8, 4, 2, 24, 12, 6, 26, 36, 18, 32, 16, 8, 4, 2, 24, 12, 'None']
[13, 31, 35, 41, 27, 29, 9, 25, 3, 39, 1, 13, 31, 35, 41, 27, 29, 9, 25, 3, 39, 1, 13, 11]
[14, 12, 30, 6, 38, 26, 42, 36, 44, 18, 22, 32, 34, 16, 40, 8, 20, 4, 10, 2, 28, 24, 14, 'None']
[15, 41, 17, 25, 7, 13, 11, 27, 37, 3, 45, 31, 5, 29, 21, 39, 33, 35, 19, 9, 43, 1, 15, 22]
[16, 26, 2, 32, 6, 4, 18, 12, 8, 36, 24, 16, 26, 2, 32, 6, 4, 18, 12, 8, 36, 24, 16, 'None']
[17, 13, 37, 31, 21, 35, 43, 41, 7, 27, 45, 29, 33, 9, 15, 25, 11, 3, 5, 39, 19, 1, 17, 22]
[18, 2, 36, 4, 26, 8, 6, 16, 12, 32, 24, 18, 2, 36, 4, 26, 8, 6, 16, 12, 32, 24, 18, 'None']
[19, 39, 5, 3, 11, 25, 15, 9, 33, 29, 45, 27, 7, 41, 43, 35, 21, 31, 37, 13, 17, 1, 19, 22]
[20, 32, 42, 12, 10, 16, 44, 6, 28, 8, 22, 26, 14, 4, 34, 36, 30, 2, 40, 18, 38, 24, 20, 'None']
[21, 27, 15, 39, 37, 41, 33, 3, 17, 35, 45, 25, 19, 31, 7, 9, 5, 13, 43, 29, 11, 1, 21, 22]
[22, 24, 22, 24, 22, 24, 22, 24, 22, 24, 22, 24, 22, 24, 22, 24, 22, 24, 22, 24, 22, 24, 22, 'None']
[23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 'None']
```

4.18.4 Primitive roots

Computing a primitive root (see chapter 4.9, “Multiplicative order and primitive roots”) in Sage is very straightforward. If n is an integer, the command `primitive_root(n)` computes a primitive root of the multiplicative group $(\mathbf{Z}/n\mathbf{Z})^*$, if one exists. Where n is prime, then this is the same as calculating a primitive root of $\mathbf{Z}/n\mathbf{Z}$.

Here, we calculate some primitive roots of a few integers.

Sage sample 4.6 Calculating one primitive root for some primes

```
sage: primitive_root(4)
3
sage: primitive_root(22)
13
sage: for p in primes(1, 50):
....:     print p, primitive_root(p)
....:
2 1
3 2
5 2
7 3
11 2
13 2
17 3
19 2
23 5
29 2
31 3
37 2
41 6
43 3
47 5
```

If p is prime, then $\mathbf{Z}/p\mathbf{Z}$ has at least one primitive root.

Sometimes we want to compute all the primitive roots of $\mathbf{Z}/p\mathbf{Z}$, not just any primitive root of $\mathbf{Z}/p\mathbf{Z}$. The following function can do this¹⁰⁵.

Sage sample 4.7 Function “enum.PrimitiveRoots_of_an_Integer” to calculate all primitive roots for a given number

```
def enum_PrimitiveRoots_of_an_Integer(M):
    r"""
    Return all the primitive roots of the integer M (if possible).
    """
    try:
        g = primitive_root(M)
    except:
        return None
    targetOrder = euler_phi(M)
    L=[]
    # Stepping through all odd integers from 1 up to M, not including
    # M. So this loop only considers values of i where 1 <= i < M.
    for i in xrange(1,M,2):
        testGen = Mod(g^i,M)
        if testGen.multiplicative_order() == targetOrder:
            L.append(testGen.lift())
    # removing duplicates
    return Set(L)

# AA_Start -- Testcases for enum_PrimitiveRoots_of_an_Integer(M)
print "AA_Start -- Testcases for enum_PrimitiveRoots_of_an_Integer(M)"
M=10; print "1-----Testcase: M = %s" % M
LL = enum_PrimitiveRoots_of_an_Integer(M)
if LL==None:
    print M
else:
    print LL
M=8; print "2-----Testcase: M = %s" % M
# M=8 hat keine primitive root mod m. Checke, ob per try - except abgefangen.
LL = enum_PrimitiveRoots_of_an_Integer(M)
if LL==None:
    print M
else:
    print LL
M=17; print "3-----Testcase: M = %s" % M
LL = enum_PrimitiveRoots_of_an_Integer(M)
if LL==None:
    print M
else:
    print LL
# AA_End -- Testcases
```

```
OUTPUT:
AA_Start -- Testcases for enum_PrimitiveRoots_of_an_Integer(M)
1-----Testcase: M = 10
{3, 7}
2-----Testcase: M = 8
8
3-----Testcase: M = 17
{3, 5, 6, 7, 10, 11, 12, 14}
```

¹⁰⁵This code was developed in a Sage script file and executed non-interactively. That is why you don't see "sage:" and "...." at the beginning of the lines like in the Sage samples before.

For example, here is a list of all primitive roots of the prime 541.

Sage sample 4.8 Table with all primitive roots for the given prime 541

```
sage: L=enum_PrimitiveRoots_of_an_Integer(541); L
{2, 517, 10, 523, 13, 14, 527, 528, 18, 531, 24, 539, 30, 37, 40, 51,
54, 55, 59, 62, 65, 67, 68, 72, 73, 77, 83, 86, 87, 91, 94, 96, 98,
99, 107, 113, 114, 116, 117, 126, 127, 128, 131, 132, 138, 150, 152,
153, 156, 158, 163, 176, 181, 183, 184, 195, 197, 199, 206, 208,
210, 213, 218, 220, 223, 224, 244, 248, 250, 257, 258, 259, 260,
261, 263, 267, 269, 270, 271, 272, 274, 278, 280, 281, 282, 283,
284, 291, 293, 297, 317, 318, 321, 323, 328, 331, 333, 335, 342,
344, 346, 357, 358, 360, 365, 378, 383, 385, 388, 389, 391, 403,
409, 410, 413, 414, 415, 424, 425, 427, 428, 434, 442, 443, 445,
447, 450, 454, 455, 458, 464, 468, 469, 473, 474, 476, 479, 482,
486, 487, 490, 501, 504, 511}
sage: len(L)
144
```

With a little bit of programming, we can count how many primitive roots are in a given range of integers. We can check this for all numbers or only for the primes within this range.

Sage sample 4.9 Function “count_PrimitiveRoots_of_an_IntegerRange” to calculate all primitive roots for a given range of integers

```
def count_PrimitiveRoots_of_an_IntegerRange(start, end, bPrimesOnly=True):
    r"""
    Compute all primitive roots of all numbers between start and end,
    inclusive, and count them.
    If the flag bPrimesOnly is True, it performs primality tests, so it
    allows us to count the number of primes from start to end, inclusive.
    If the flag bPrimesOnly is false, it additionally counts these even
    numbers which have no primitive root.
    """
    nCheckedNumb = 0
    nCheckedNumb_WithoutPrimitivRoots = 0
    nPrimitiveRoots = 0
    for n in xrange(start, end+1):
        if bPrimesOnly:
            if is_prime(n):
                nCheckedNumb += 1
                L = enum_PrimitiveRoots_of_an_Integer(n)
                nPrimitiveRoots += len(L)
        else:
            nCheckedNumb += 1
            L = enum_PrimitiveRoots_of_an_Integer(n)
            if L==None:
                nCheckedNumb_WithoutPrimitivRoots += 1
            else:
                nPrimitiveRoots += len(L)

    if bPrimesOnly:
        print "Found all %s " % nPrimitiveRoots + \
            " primitive roots of %s primes." % nCheckedNumb
    else:
        if nCheckedNumb_WithoutPrimitivRoots == 0:
            print "Found all %s " % nPrimitiveRoots + \
                "primitive roots of %s numbers." % nCheckedNumb
        else:
            print "Found all %s " % nPrimitiveRoots + \
                "primitive roots of %s numbers." % \
                (nCheckedNumb - nCheckedNumb_WithoutPrimitivRoots)
            print "(Total of numbers checked: %s, " % nCheckedNumb + \
                "Amount of numbers without primitive roots: %s)" % \
                nCheckedNumb_WithoutPrimitivRoots
```

Using the Sage command `time`, we can also find out how long it takes on our computer.

Sage sample 4.10 Function “`count_PrimitiveRoots_of_an_IntegerRange`”: testcases and output

```
# BB_Start -- Testcases for count_PrimitiveRoots_of_an_IntegerRange(start, end, bPrimesOnly=True)
print "\n\nBB_Start -- Testcases for count_PrimitiveRoots_of_an_IntegerRange(start, end, True)"

print "\n1-----Testcase: (1, 500)"
time count_PrimitiveRoots_of_an_IntegerRange(1, 500)

print "\n2-----Testcase: (5, 6, False)"
time count_PrimitiveRoots_of_an_IntegerRange(5, 6, False)

print "\n3-----Testcase: (1, 500, False)"
time count_PrimitiveRoots_of_an_IntegerRange(1, 500, False)
# BB_End -- Testcases
```

OUTPUT:

```
BB_Start -- Testcases for count_PrimitiveRoots_of_an_IntegerRange(start, end, bPrimesOnly=True)
```

```
1-----Testcase: (1, 500)
```

```
Found all 8070 primitive roots of 95 primes.
```

```
Time: CPU 0.94 s, Wall: 0.97 s
```

```
2-----Testcase: (5, 6, False)
```

```
Found all 3 primitive roots of 2 numbers.
```

```
Time: CPU 0.00 s, Wall: 0.00 s
```

```
3-----Testcase: (1, 500, False)
```

```
Found all 11010 primitive roots of 170 numbers.
```

```
(Total of numbers checked: 500, Amount of numbers without primitive roots: 330)
```

```
Time: CPU 1.52 s, Wall: 1.59 s
```

Using our custom-defined function `enum_PrimitiveRoots_of_an_Integer`, we can find all primitive roots of one prime integer p .

The following function counts how many primes are in a given range and enumerate all their primitive roots.

From this list of primitive roots, we can determine the smallest and largest primitive root for $\mathbf{Z}/p\mathbf{Z}$, as well as count the number of primitive roots of $\mathbf{Z}/p\mathbf{Z}$.

Sage sample 4.11 Function “`count_PrimitiveRoots_of_a_PrimesRange`” to calculate the number of primitive roots for a given range of primes

```
def count_PrimitiveRoots_of_a_PrimesRange(start, end):
    r"""
    Compute all primitive roots of all primes between start and end,
    inclusive. This uses a primes iterator.
    """
    nPrimes = 0
    nPrimitiveRoots = 0
    for p in primes(start, end+1):
        L = enum_PrimitiveRoots_of_an_Integer(p)
        print p, len(L)
        nPrimes += 1
        nPrimitiveRoots += len(L)
    print "Found all %s" % nPrimitiveRoots + " primitive roots of %s primes." % nPrimes

# CC_Start -- Testcases for count_PrimitiveRoots_of_a_PrimesRange(start, end)
print "\n\nBB_Start -- Testcases for count_PrimitiveRoots_of_a_PrimesRange(start, end)"
print "-----Testcase: (1, 1500)"
time count_PrimitiveRoots_of_a_PrimesRange(1, 1500)
# CC_End -- Testcases
```

OUTPUT:

```
CC_Start -- Testcases for count_PrimitiveRoots_of_a_PrimesRange(start, end)
-----Testcase: (1, 1500)
2 1
3 1
5 2
7 2
11 4
13 4
17 8
19 6
23 10
29 12
31 8
37 12
...
1483 432
1487 742
1489 480
1493 744
1499 636
Found all 62044 primitive roots of 239 primes.
Time: CPU 7.55 s, Wall: 7.85 s
```

A slightly modified version of our function `count_PrimitiveRoots_of_a_PrimesRange`, is used to generate a database of all primitive roots of all primes between 1 and 100,000.

Sage sample 4.12 Code to generate the database with all primitive roots for all primes between 1 and 100,000

```
start = 1
end = 10^5
fileName = "/scratch/mvngu/primroots.dat"
file = open(fileName, "w")
for p in primes(start, end+1):
    L = enum_PrimitiveRoots_of_an_Integer(p)
    print p, len(L)
    # Output to a file. The format is:
    # (1) the prime number p under consideration
    # (2) the number of primitive roots of Z/pZ
    # (3) all the primitive roots of Z/pZ
    file.write(str(p) + " " + str(len(L)) + " " + str(L) + "\n")
    file.flush()
file.close()
```

It took about one day on the machine `sage.math` to generate the file “primroots.dat” (done in July 2009 by Minh Van Nguyen).

This code and the function `enum_PrimitiveRoots_of_an_Integer` was put in a Sage script file and executed non-interactively.

The file “primroots.dat” is a database of all primitive roots of all primes between 1 and 100,000 inclusive. It is a very large file (about 1 GB uncompressed, and 285 MB compressed with bzip2). You can find the file at <http://sage.math.washington.edu/home/mvngu/doc/primitive-roots/primroots.dat.bz2>.

This database file “primroots.dat” was used then to create three graphics using the following code.

Sage sample 4.13 Code to generate the graphics about the primitive roots

```
sage: # open a database file on primitive roots from 1 to 100,000
sage: file = open("/scratch/mvngu/primroots.dat", "r")
sage: plist = []      # list of all primes from 1 to 100,000
sage: nlist = []      # number of primitive roots modulo prime p
sage: minlist = []    # smallest primitive root modulo prime p
sage: maxlist = []    # largest primitive root modulo prime p
sage: for line in file:
....:     # get a line from the database file and tokenize it for processing
....:     line = line.strip().split(" ", 2)
....:     # extract the prime number p in question
....:     plist.append(Integer(line[0]))
....:     # extract the number of primitive roots modulo p
....:     nlist.append(Integer(line[1]))
....:     # extract the list of all primitive roots modulo p
....:     line = line[-1]
....:     line = line.replace("{", "")
....:     line = line.replace("}", "")
....:     line = line.split(", ")
....:     # sort the list in non-decreasing order
....:     line = [Integer(s) for s in line]
....:     line.sort()
....:     # get the smallest primitive root modulo p
....:     minlist.append(line[0])
....:     # get the largest primitive root modulo p
....:     maxlist.append(line[-1])
....:
sage: file.close() # close the database file
sage: # plot of number of primitive roots modulo p
sage: nplot = point2d(zip(plist, nlist), pointsize=1)
sage: nplot.axes_labels(["x", "y"])
sage: nplot
sage: # plot of smallest primitive root modulo prime p
sage: minplot = point2d(zip(plist, minlist), pointsize=1)
sage: minplot.axes_labels(["x", "y"])
sage: minplot
sage: # plot of largest primitive root modulo prime p
sage: maxplot = point2d(zip(plist, maxlist), pointsize=1)
sage: maxplot.axes_labels(["x", "y"])
sage: maxplot
```

Figure 4.2 graphs the number of primitive roots for each prime between 1 and 100,000. The x -axis represents primes between 1 and 100,000, while the y -axis counts the number of primitive roots for each prime within that interval.

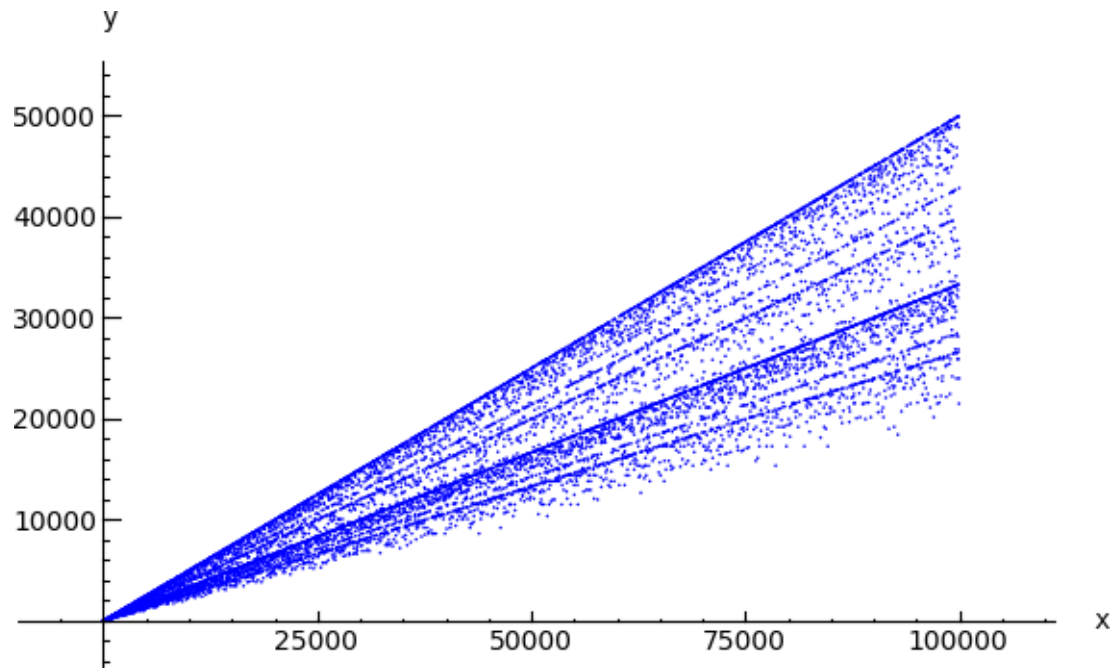


Figure 4.2: The number of primitive roots of all primes between 1 and 100,000.

Figure 4.3 graphs the smallest primitive roots of all primes between 1 and 100,000. The x -axis represents primes between 1 and 100,000. The y -axis represents the smallest primitive root of each prime within that interval.

Figure 4.4 shows a corresponding graph for the largest primitive root of each prime within the above interval.

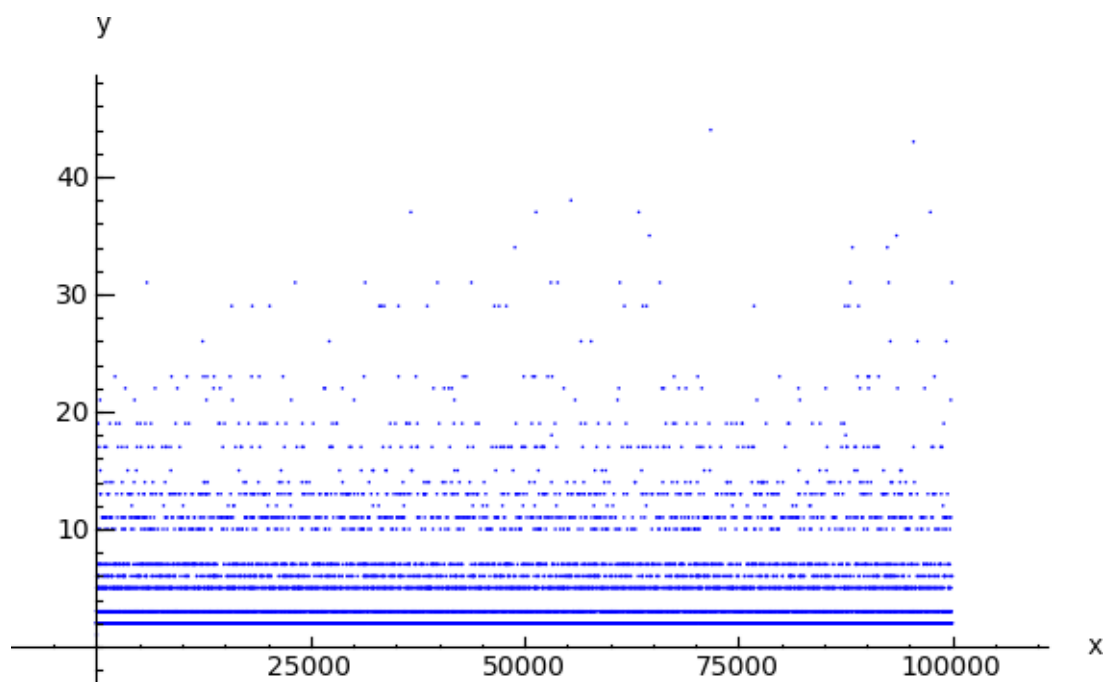


Figure 4.3: The smallest primitive roots of all primes between 1 and 100,000.

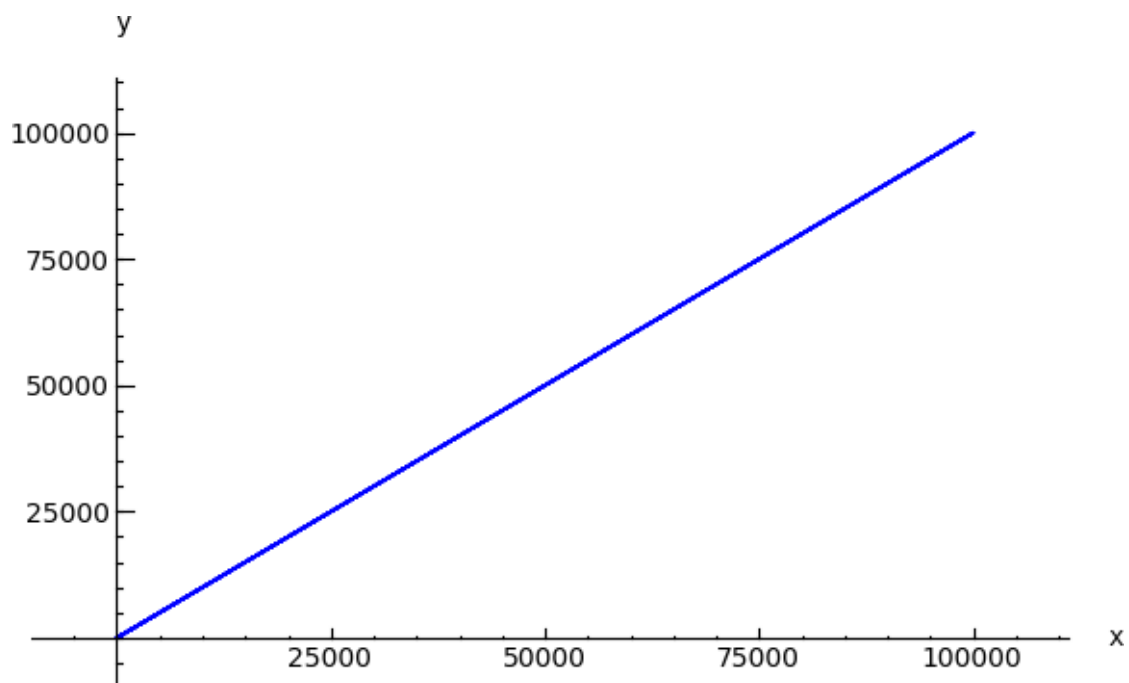


Figure 4.4: The largest primitive roots of all primes between 1 and 100,000.

4.18.5 RSA examples with Sage

Below is Sage source code for the simple RSA examples in section 4.13 (“The RSA procedure with actual numbers”).

Example on page 132:

The RSA exponentiation $M^{37} \pmod{3713}$ on message $M = 120$ can be calculated in Sage as follows:

```
sage: power_mod(120, 37, 3713)
1404
```

Example on page 133:

The factorization of $J(256027) = 255016 = 2^3 * 127 * 251$ can be calculated using Sage as follows:

Sage sample 4.14 Factoring a number

```
sage: factor(255016)
2^3 * 127 * 251
```

Example on page 133:

Sage can do RSA encryption as follows:

Sage sample 4.15 RSA encryption by modular exponentiation of a number (used as message)

```
sage: A = [82, 83, 65, 32, 119, 111, 114, 107, 115, 33]
sage: e = 65537; m = 256027
sage: [power_mod(a, e, m) for a in A]
[212984, 25546, 104529, 31692, 248407, 100412, 54196, 100184, 58179, 227433]
```

Example on page 134:

RSA encryption using Sage:

```
sage: A = [21075, 16672, 30575, 29291, 29473]
sage: e = 65537; m = 256027
sage: [power_mod(a, e, m) for a in A]
[158721, 137346, 37358, 240130, 112898]
```

Example on page 134:

RSA encryption using Sage:

```
sage: A = [82083, 65032, 119111, 114107, 115033]
sage: e = 65537; m = 256027
sage: [power_mod(a, e, m) for a in A]
[198967, 51405, 254571, 115318, 14251]
```

4.18.6 How many RSA keys exist within a given modulo range?

The RSA encryption procedure was described in section 4.10.2 (“How the RSA procedure works”). Steps 1 to 3 constitute key generation, steps 4 and 5 are the encryption:

1. Select two distinct random prime numbers p and q and calculate $n = p * q$.
The value n is called the RSA modulus.
2. Select an arbitrary $e \in \{2, \dots, n - 1\}$ such that:
 e is relatively prime to $J(n) = (p - 1) * (q - 1)$.
We can then “throw away” p and q .
3. Select $d \in \{1, \dots, n - 1\}$ with $e * d \equiv 1 \pmod{J(n)}$,
i.e. d is the multiplicative inverse of e modulo $J(n)$. We can then “throw away” $J(n)$.
 $\rightarrow (n, e)$ is the public key P .
 $\rightarrow (n, d)$ is the private key S (only d must be kept secret).
4. For encryption, the message represented as a (binary) number is divided into parts such that each part of the number represents a number less than n .
5. Encryption of the plaintext (or the parts of it) $M \in \{1, \dots, n - 1\}$:

$$C = E((n, e); M) := M^e \pmod{n}.$$

The default way to crack a given RSA ciphertext C would be to use the public key of the recipient and to try to factorize n . Then you can go through the steps 2 and 3 and generate the private key e , which is normally used to decrypt a ciphertext.

According to the “prime number theorem” (described in section 3.7.2 “The number of prime numbers $PI(x)$ ”) the number of prime numbers $PI(x)$ is asymptotic to $x/\ln(x)$. If you have a given n then there are about $n/\ln(n)$ different possible values for the prime p .

If you don’t want to use factorization but ask the question like in classic encryption, you may want to find out: How many possible private keys (n, d) are there for a given key size range $n \in [a, b]$?

Sage source code 4.16 below defining the function `count_Number_of_RSA_Keys` can answer this question concretely (if the modulus is not too big).¹⁰⁶

As there are many more private keys (n, d) within a bigger range of values for n , even brute-force factoring is much more efficient as brute-force trying all the keys.

¹⁰⁶ a) Calling `sage: count_Number_of_RSA_Keys(100, 1000)` means to consider the interval $[100, 1000]$ for n . n is defined by the two primes $p, q : n = p * q$. So the primes can have the maximal value 500 because $2 * 500 = 1000$. The number of possible combinations of primes is $comb = 258$.

The number of primes in the given range is 143.

The number of private keys is 34,816.

b) Calling `sage: count_Number_of_RSA_Keys(100, 100, True)` has the following output:

- Number of private keys for modulus in a given range: 0

- Number of primes in a given range: 0

The reason for that is, that now we only consider $n = 100$, and 100 is not semi-prime, this means 100 is not the product of only two primes.

Sage sample 4.16 How many RSA keys are there if you know a range for the public keys n?

```
def count_Number_of_RSA_Keys(start, end, Verbose=False):
    r"""
    How many private RSA keys (n,d) exist, if only modulus N is given, and start <= N <= end?
    (prime_range(u,o) delivers all primes >=u und < o).
    """
    a = start
    b = end
    s = 0
    comb = 0
    for p in prime_range(1, b/2+1):
        for q in prime_range(p + 1, b/2+1):
            if a <= p * q and p * q <= b:
                comb = comb + 1
                s = s + (euler_phi(euler_phi(p * q))-1)
                if Verbose:
                    print "p=%s, " % p + "q=%s, " % q + "s=%s" % s
    print "Number of private keys for modulus in a given range: %s" % s + " (comb=%s), " % comb

    # Just for comparison: How many primes are in this range?
    s = 0
    for p in prime_range(a, b+1):
        if Verbose:
            print "a=%s, " % a + "b=%s, " % b + "p=%s" % p
        s = s + 1
    print "Number of primes in a given range: %s" % s

print "\n\nDD_Start -- Testcases for count_Number_of_RSA_Keys(start, end)"
print "\n-----Testcase: (100, 1000) [Should deliver 34.816]"
time count_Number_of_RSA_Keys(100, 1000)
print "\n-----Testcase: (100, 107, True) [Should deliver 23]"
time count_Number_of_RSA_Keys(100, 107, True)
u = 10^3; o = 10^4;
print "\n-----Testcase: (%s, " % u + "%s) [Should deliver 3.260.044]" % o
time count_Number_of_RSA_Keys(u, o)
```

OUTPUT:

DD_Start -- Testcases for count_Number_of_RSA_Keys(start, end)

-----Testcase: (100, 1000) [Should deliver 34.816]

Number of private keys for modulus in a given range: 34816 (comb=258),

Number of primes in a given range: 143

Time: CPU 0.03 s, Wall: 0.04 s

-----Testcase: (100, 107, True) [Should deliver 23]

p=2, q=53, s=23

Number of private keys for modulus in a given range: 23 (comb=1),

a=100, b=107, p=101

a=100, b=107, p=103

a=100, b=107, p=107

Number of primes in a given range: 3

Time: CPU 0.00 s, Wall: 0.00 s

-----Testcase: (1000, 10000) [Should deliver 3.260.044]

Number of private keys for modulus in a given range: 3260044 (comb=2312),

Number of primes in a given range: 1061

Time: CPU 0.63 s, Wall: 0.66 s

4.19 Appendix: List of the definitions and theorems formulated in this chapter

	Short description	Page
Definition 4.3.1	prime numbers	90
Definition 4.3.2	composite numbers	90
Theorem 4.3.1	factors of composite numbers	91
Theorem 4.3.2	1st fundamental theorem of number theory	91
Definition 4.4.1	divisibility	92
Definition 4.4.2	remainder class r modulo m	92
Definition 4.4.3	congruent	93
Theorem 4.4.1	congruence with difference	93
Theorem 4.6.1	multiplicative inverse (existence)	97
Theorem 4.6.2	exhaustive permutation	98
Theorem 4.6.3	power mod m	101
Definition 4.7.1	\mathbb{Z}_n	103
Definition 4.7.2	\mathbb{Z}_n^*	104
Theorem 4.7.1	multiplicative inverse in \mathbb{Z}_n^*	104
Definition 4.8.1	Euler function $J(n)$	105
Theorem 4.8.1	$J(p)$	105
Theorem 4.8.2	$J(p * q)$	105
Theorem 4.8.3	$J(p_1 * \cdots * p_k)$	105
Theorem 4.8.4	$J(p_1^{e_1} * \cdots * p_k^{e_k})$	105
Theorem 4.8.5	little Fermat	106
Theorem 4.8.6	Euler-Fermat theorem	106
Definition 4.9.1	multiplicative order $\text{ord}_m(a)$	108
Definition 4.9.2	primitive root of m	109
Theorem 4.9.1	exhausting of all possible values	109

Bibliography

- [Agrawal2002] M. Agrawal, N. Kayal, N. Saxena,
PRIMES in P, August 2002, revised paper:
http://www.cse.iitk.ac.in/manindra/algebra/primality_v6.pdf
See also the website "The AKS "PRIMES in P" Algorithm Resource":
<http://fatphil.org/maths/AKS/>
- [Bartholome1996] A. Bartholome, J. Rung, H. Kern,
Zahlentheorie für Einsteiger, Vieweg 1995, 2nd edition 1996.
- [Bauer1995] Friedrich L. Bauer,
Entzifferte Geheimnisse, Springer, 1995.
- [Bauer2000] Friedrich L. Bauer,
Decrypted Secrets, Springer 1997, 2nd edition 2000.
- [Bernstein2001] D. J. Bernstein,
Circuits for integer factorization: a proposal,
<http://cr.yp.to/papers/nfscircuit.ps>
<http://cr.yp.to/djb.html>
- [Beutelspacher1996] Albrecht Beutelspacher,
Kryptologie, Vieweg 1987, 5th edition 1996.
- [Bourseau2002] F. Bourseau, D. Fox, C. Thiel,
Vorzüge und Grenzen des RSA-Verfahrens,
In: Datenschutz und Datensicherheit (DuD) 26/2002, pp 84-89 (see www.dud.de),
<http://www.secorvo.de/publikationen/rsa-grenzen-fox-2002.pdf>
- [Brands2002] Gilbert Brands,
Verschlüsselungsalgorithmen – Angewandte Zahlentheorie rund um Sicherheitsprotokolle, Vieweg, 2002.
- [Buchmann2004] Johannes Buchmann,
Introduction to Cryptography, Springer, 2nd edition, 2004.
- [Buhler1993] J.P. Buhler, H.W. Lenstra, C. Pomerance,
Factoring integers with the number field sieve,
In: A.K. Lenstra, H.W. Lenstra (Hrsg.): The Development of the Number Field Sieve,
Lecture Notes in Mathematics, Vol. 1554, Springer, Heidelberg 1993, pp 50–94.
- [Eckert2003] Claudia Eckert,
IT-Sicherheit: Konzepte-Verfahren-Protokolle, Oldenbourg 2001, 2nd edition 2003.

- [Ertel2001] Wolfgang Ertel,
Angewandte Kryptographie, Fachbuchverlag Leipzig FV 2001.
- [GISA2002] GISA (German Information Security Agency),
Recommendation for key length selection, Bonn, Sep. 2002,
<http://www.bsi.bund.de/esig/basics/techbas/krypto/bund02v7.pdf>
A statement on these recommendations:
<http://www.secorvo.de/publikat/stellungnahme-algorithmenempfehlung-020307.pdf>
- [Graham1994] Graham, Knuth, Patashnik,
Concrete Mathematics, a Foundation of Computer Science,
Addison Wesley 1989, 6th printing 1994.
- [Kippenhahn1997] Rudolph Kippenhahn,
Verschlüsselte Botschaften – Geheimschrift, Enigma und Chipkarte, Rowohlt, 1997.
- [Kippenhahn1999] Rudolph Kippenhahn,
Code Breaking – A History and Exploration, Constable, 1999.
- [Knuth1998] Donald E. Knuth,
The Art of Computer Programming, vol 2: Seminumerical Algorithms,
Addison-Wesley, 2nd edition 1998.
- [Lenstra1993] A. Lenstra, H. Lenstra:
The development of the Number Field Sieve,
Lecture Notes in Mathematics 1554, Springer, New York 1993
- [Lenstra1999] Arjen K. Lenstra, Eric R. Verheul
Selecting Cryptographic Key Sizes (1999),
Journal of Cryptology: the journal of the International Association for Cryptologic Research
<http://www.cryptosavvy.com/cryptosizes.pdf>
- [Lenstra2002] Arjen K. Lenstra, Adi Shamir, Jim Tomlinson, Eran Tromer,
Analysis of Bernstein's Factorization Circuit,
<http://www.cryptosavvy.com/mesh.pdf>
- [Menezes2001] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone
Handbook of Applied Cryptography, CRC Press 1997, 5th printing 2001.
- [Pfleeger1997] Charles P. Pfleeger,
Security in Computing, Prentice-Hall, 2nd edition 1997.
- [Pomerance1984] C. Pomerance,
The quadratic sieve factoring algorithm,
In: G.R. Blakley, D. Chaum (Hrsg.): *Proceedings of Crypto '84*, LNCS 196, Springer
Berlin 1995, pp 169-182.
- [RSA Security 2002] RSA Security,
Has the RSA algorithm been compromised as a result of Bernstein's Paper?,
April 8th, 2002,
<http://www.rsasecurity.com/>

- [SchneiderM2004] Matthias Schneider,
Analyse der Sicherheit des RSA-Algorithmus.
Mögliche Angriffe, deren Einfluss auf sichere Implementierungen und ökonomische Konsequenzen,
 Diploma thesis at the University of Siegen, Germany, 2004.
- [Schneier1996] Bruce Schneier,
Applied Cryptography, Protocols, Algorithms, and Source Code in C,
 Wiley and Sons, 2nd edition 1996.
- [Schwenk2002] Jörg Schwenk,
Sicherheit und Kryptographie im Internet, Vieweg 2002.
- [Sedgewick1990] Robert Sedgewick,
Algorithms in C, Addison-Wesley, 1990.
- [Shamir2003] Adi Shamir, Eran Tromer,
Factoring Large Numbers with the TWIRL Device, Januar 2003,
<http://www.wisdom.weizmann.ac.il/~tromer/>.
- [Shamir2003a] Adi Shamir, Eran Tromer,
On the Cost of Factoring RSA-1024, RSA Laboratories CryptoBytes Volume 6, No. 2,
 Summer 2003, p. 11-20
http://www.rsasecurity.com/rsalabs/cryptobytes/CryptoBytes_August_2003.pdf
- [Silverman2000] Robert D. Silverman:
A Cost-Based Security Analysis of Symmetric and Asymmetric Key Lengths
 In: RSA Laboratories Bulletin, No. 13, April 2000, p. 1-22
- [Stinson1995] Douglas R. Stinson,
Cryptography - Theory and Practice, CRC Press, 1995.
- [Weis2003] Rüdiger Weis, Stefan Lucks, Andreas Bogk,
Sicherheit von 1024 bit RSA-Schlüsseln gefährdet,
 In: Datenschutz und Datensicherheit (DuD) 6/2003, pp 360-362 (see www.dud.de)
 The article explains details about the TWIRL device.
- [Welschenbach2001] Welschenbach, Michael,
Kryptographie in C und C++, Springer 2001.
- [Wiles1994] Wiles, Andrew,
Modular elliptic curves and Fermat's Last Theorem,
 In: Annals of Mathematics 141 (1995).
- [Wolfenstetter1998] Albrecht Beutelspacher, Jörg Schwenk, Klaus-Dieter Wolfenstetter,
Moderne Verfahren in der Kryptographie, Vieweg 1995, 2nd edition 1998.
- [Yan2000] Song Y. Yan,
Number Theory for Computing, Springer, 2000.

Web links

1. Ron Knott's Fibonacci page,
Here, everything revolves around Fibonacci numbers.
<http://www.mcs.surrey.ac.uk/personal/R.Knott/Fibonacci/fib.html>
2. CrypTool,
Open source e-learning software to illustrate cryptography and cryptanalysis
<http://www.cryptool.de>,
<http://www.cryptool.org>,
<http://www.cryptool.com>
3. Mathematica,
Commercial mathematics package
<http://www.wolfram.com>
4. LiDIA,
Extensive library containing number-theory functions and the LC interpreter
<http://www.informatik.tu-darmstadt.de/TI/LiDIA>
5. BC,
Interpreter with number-theory functions
<http://www.gnu.org/software/bc/bc.html>
6. Pari-GP,
Excellent, fast, free interpreter with number theoretical functions.
<http://pari.math.u-bordeaux.fr/>
<http://en.wikipedia.org/wiki/PARI/GP>
Resources for PARI/GP at Karim Belabas's website:
<http://www.math.u-bordeaux.fr/~belabas/pari/>
7. Only after I had completed this article, did I come across the website of Mr. Münchenbach, which interactively and didactically uses elementary number theory to provide a sophisticated description of the fundamental mathematical thought processes. It was created for a teaching project in the 11th grade of the technical grammar school (unfortunately only available in German):
<http://www.hydrargyrum.de/kryptographie>
8. Web site of Mr. Wagner, who is responsible for the development of the curriculum of computer science in one of the German federal states (Länder). Here you can get hold of a collection of texts and (Java-)programs (available only in German):
<http://www.saar.de/~awa/kryptolo.htm>
9. GISA,
German Information Security Agency
<http://www.bsi.bund.de>
10. Factorization records and challenges,
<http://www.crypto-world.com/>
<http://www.crypto-world.com/FactorWorld.html>, page by Scott Contini
<http://www.loria.fr/~zimmerma/records/factor.html>
http://www.tutorgig.com/ed/RSA_number

<http://www.uni-bonn.de/Aktuelles/Pressemitteilungen/pm02/pm035-02.html>
http://www.ercim.org/publication/Ercim_News/enw49/franke.html, 2002-01
<http://www.loria.fr/~zimmerma/records/rsa160>

<http://www.rsa.com/rsalabs/node.asp?id=2092>

11. The Cunningham Project,
<http://www.cerias.purdue.edu/homes/ssw/cun/>
12. Sage,
Excellent, open source computer algebra system with Python as script language, used to build the code samples in this chapter. See the introduction in chapter A.5.
<http://www.sagemath.org/>
http://en.wikipedia.org/wiki/Sage_%28mathematics_software%29

Acknowledgments

I would like to take this opportunity to thank

- Henrik Koy for making many very useful suggestions, for the very constructive proof-reading of the first version of this article, and for helping with TeX.
Mr. Koy designed and developed in his leisure time the functions and the complex dialog box of the RSA cryptosystem within CrypTool, which enables you to execute the RSA samples of this article.
- Jörg Cornelius Schneider for his enthusiastic TeX support and for the many cases where he helped when facing programming or design problems.
- Dr. Georg Illies for pointing me to Pari-GP.
- Lars Fischer for his help with fast Pari-GP code for primitive roots.
- Minh Van Nguyen from Australia for his always fast, professional and exhaustive help with the Sage code samples in this chapter.

Chapter 5

The Mathematical Ideas behind Modern Cryptography¹

(Oyono R. / Esslinger B./ Schneider J., Sep. 2000; Updates Nov. 2000, Feb. 2003, Apr. 2007)

Georg Christoph Lichtenberg²:

I don't know if its getting better, if we change it,
but I know, that we have to change it, if it should become better.

5.1 One way functions with trapdoor and complexity classes

A **one way function** is a function that can be calculated efficiently, but whose inverse is extremely complicated and practically impossible to calculate.

To put it more precisely: A one way function is a mapping f from a set X to a set Y , such that $f(x)$ can be calculated easily for each element x of X , whereas for (almost) every y from Y it is practically impossible to find an inverse image x (i.e. an x where $f(x) = y$).

An everyday example of a one way function is a telephone book: the function to be performed is to assign a name to the corresponding telephone number. This can be done easily due to the fact that the names are sorted alphabetically. However, the inverse function - assigning a name to a given number - is obviously difficult if you only have a telephone book available.

One way functions play a decisive role in cryptography. Almost all cryptographic terms can be rephrased using the term one way function. Let's take for example public key encryption (asymmetric cryptography):

Each subscriber T to the system is assigned a private key d_T and what is known as a public key e_T . These keys must have the following property (public key property):
For an opponent who knows the public key e_T , it is practically impossible to determine the private key d_T .

¹With the educational tool for number theory **NT** you can apply some of the methods introduced here (RSA, Rabin, DH, ElGamal) (see learning unit 4.2 and 4.3, pages 9-17/17).

NT can be called in CrypTool via the menu path **Indiv. Procedures \ Number Theory Interactive \ Learning Tool for Number Theory**. See appendix A.4.

²Georg Christoph Lichtenberg, German writer and physicist (1742-1799),
(also see: http://en.wikipedia.org/wiki/Georg_Christoph_Lichtenberg)

In order to construct useful public key procedures, therefore, we look for a one way function that is “easy” to calculate in one direction, but is “difficult” (practically impossible) to calculate in the other direction, provided that a particular piece of additional information (trapdoor) is not available. This additional piece of information allows the inverse to be found efficiently. Such functions are called **trapdoor one way functions**. In the above case, d_T is the trapdoor information.

In this process, we describe a problem as “easy” if it can be solved in polynomial time as a function of the length of the input. If the length of the input is n bits, then the time for calculating the function is proportional to n^a , where a is a constant. We say that the complexity of such problems is $O(n^a)$ [Landau- or Big-O notation].

If you compare two functions 2^n and n^a , where a is a constant, then there always exists a value for n , from which for all further n applies: $n^a < 2^n$. The function n^a has a lower complexity. Sample: for $a = 5$ the following applies: from the length $n = 23$, 2^n is greater than n^5 ; for further n 2^n clearly increases more quickly [($2^{22} = 4,194,304$, $22^5 = 5,153,632$), ($2^{23} = 8,388,608$, $23^5 = 6,436,343$), ($2^{24} = 16,777,216$, $24^5 = 7,962,624$)].

The term “practically impossible” is slightly less precise. In general, we can say that a problem cannot be solved efficiently, if the time required to solve it increases more quickly than the polynomial time as a function of the size of the input. If, for example, the length of the input is n bits and the time required for calculating the function is proportional to 2^n , then the following currently applies: the function practically cannot be calculated for $n > 80$.

In order to develop a public key procedure that can be implemented in practice, it is therefore necessary to discover a suitable trapdoor one way function.

In order to tidy things up among this confusing multitude of possible problems and their complexities, we group problems with similar complexities into classes.

The most important complexity classes are the classes **P** and **NP**:

- The class **P**: This class contains those problems that can be solved in a polynomial amount of time.
- The class **NP**: The definition of this class doesn’t look at the time required to solve a problem, but rather at the time required to verify a given solution. The class **NP** consists of those problems for which a given solution can be verified in a polynomial amount of time. Hereby, the term **NP** “non-deterministic” means polynomial and is based on a calculation model, i.e. on a computer that only exists in theory and can “guess” correct solutions non-deterministically then verify them in polynomial time.

The class **P** is contained in the class **NP**. A well-known unsolved problem is the question whether or not $\mathbf{P} \neq \mathbf{NP}$ is true, i.e. whether or not **P** is a true subset. An important property of the class **NP** is that it also contains what are known as “**NP-complete**” problems. These are problems that represent the class **NP** as follows: If a “good” algorithm for such a problem exists, then “good” algorithms exist for all problems from **NP**. In particular: if **P** only contained one complete problem, i.e. if a polynomial solution algorithm existed for this problem, then **P** would be equal to **NP**. In this sense, the **NP-complete** problems are the most difficult problems in **NP**.

Many cryptographic protocols are formed in such a way that the “good” subscribers only have to solve problems from **P**, whereas a perpetrator is faced with problems from **NP**.

Unfortunately, we do not yet know whether one way functions actually exist. However, we

can prove that one way functions exist if and only if $\mathbf{P} \neq \mathbf{NP}$ [Balcazar1988, S.63].

Mathematicians have again and again claimed to have proven the equivalence, e.g.

http://www.geocities.com/st_busygin/clipat.html),

but so far the claims have always turned out to be false.

A number of algorithms have been suggested for public key procedures. In many cases - although they at first appeared promising - it was discovered that they could be solved polynomially. The most famous failed applicant is the knapsack with trapdoor, suggested by Ralph Merkle [Merkle1978].

5.2 Knapsack problem as a basis for public key procedures

5.2.1 Knapsack problem

You are given n objects G_1, \dots, G_n with the weights g_1, \dots, g_n and the values w_1, \dots, w_n . The aim is to carry away as much as possible in terms of value while restricted to an upper weight limit g . You therefore need to find a subset of $\{G_1, \dots, G_n\}$, i.e. $\{G_{i_1}, \dots, G_{i_k}\}$, so that $w_{i_1} + \dots + w_{i_k}$ is maximised under the condition $g_{i_1} + \dots + g_{i_k} \leq g$.

Such questions are called **NP**-complete problems (not deterministically polynomial) that are difficult to calculate.

A special case of the knapsack problem is:

Given the natural numbers a_1, \dots, a_n and g , find $x_1, \dots, x_n \in \{0, 1\}$ where $g = \sum_{i=1}^n x_i a_i$ (i.e. where $g_i = a_i = w_i$ is selected). This problem is also called a **0-1 knapsack problem** and is identified with $K(a_1, \dots, a_n; g)$.

Two 0-1 knapsack problems $K(a_1, \dots, a_n; g)$ and $K(a'_1, \dots, a'_n; g')$ are called congruent if two co-prime numbers w and m exist in such a way that

1. $m > \max\{\sum_{i=1}^n a_i, \sum_{i=1}^n a'_i\}$,
2. $g \equiv wg' \pmod{m}$,
3. $a_i \equiv wa'_i \pmod{m}$ for all $i = 1, \dots, n$.

Comment:

Congruent 0-1 knapsack problems have the same solutions. No quick algorithm is known for clarifying the question as to whether two 0-1 knapsack problems are congruent.

A 0-1 knapsack problem can be solved by testing the 2^n possibilities for x_1, \dots, x_n . The best method requires $O(2^{n/2})$ operations, which for $n = 100$ with $2^{100} \approx 1.27 \cdot 10^{30}$ and $2^{n/2} \approx 1.13 \cdot 10^{15}$ represents an insurmountable hurdle for computers. However, for special a_1, \dots, a_n the solution is quite easy to find, e.g. for $a_i = 2^{i-1}$. The binary representation of g immediately delivers x_1, \dots, x_n . In general, the a 0-1 knapsack problem can be solved easily if a permutation³ π of $1, \dots, n$ exists with $a_{\pi(j)} > \sum_{i=1}^{j-1} a_{\pi(i)}$. If, in addition, π is the identity, i.e. $\pi(i) = i$ for $i = 1, 2, \dots, n$, then the sequence a_1, \dots, a_n is said to be super-increasing. Crypto procedure 5.1 solves the knapsack problem with a super-increasing sequence in the time of $O(n)$.

³A permutation π of the numbers $1, \dots, n$ is a change in the order in which these numbers are listed. For example, a permutation π of $(1, 2, 3)$ is $(3, 1, 2)$, i.e. $\pi(1) = 3$, $\pi(2) = 1$ and $\pi(3) = 2$.

Crypto Procedure 5.1 Solving knapsack problems with super-increasing weights

```
for  $i = n$  to 1 do
  if  $T \geq a_i$  then
     $T := T - s_i$ 
     $x_i := 1$ 
  else
     $x_i := 0$ 
if  $T = 0$  then
   $X := (x_1, \dots, x_n)$  is the solution.
else
  No solution exists.
```

5.2.2 Merkle-Hellman knapsack encryption

In 1978, Merkle and Hellman [Merkle1978] specified a public key encryption procedure that is based on “defamiliarizing” the easy 0-1 knapsack problem with a super-increasing sequence into a congruent one with a super-increasing sequence. It is a block ciphering that ciphers an n -bit plaintext each time it runs, see crypto procedure 5.2 for the details.

Crypto Procedure 5.2 Merkle-Hellman (based on knapsack problems)

Let (a_1, \dots, a_n) be super-increasing. Let m and w be two co-prime numbers with $m > \sum_{i=1}^n a_i$ and $1 \leq w \leq m - 1$. Select \bar{w} with $w\bar{w} \equiv 1 \pmod{m}$ the modular inverse of w and set $b_i := wa_i \pmod{m}$, $0 \leq b_i < m$ for $i = 1, \dots, n$, and verify whether the sequence b_1, \dots, b_n is not super-increasing. A permutation $b_{\pi(1)}, \dots, b_{\pi(n)}$ of b_1, \dots, b_n is then published and the inverse permutation μ to π is defined secretly. A sender writes his/her message in blocks $(x_1^{(j)}, \dots, x_n^{(j)})$ of binary numbers n in length, calculates

$$g^{(j)} := \sum_{i=1}^n x_i^{(j)} b_{\pi(i)}$$

and sends $g^{(j)}$, ($j = 1, 2, \dots$).

The owner of the key calculates

$$G^{(j)} := \bar{w} g^{(j)} \pmod{m}, \quad 0 \leq G^{(j)} < m$$

and obtains the $x_{\mu(i)}^{(j)} \in \{0, 1\}$ (and thus also the $x_i^{(j)}$) from

$$\begin{aligned} G^{(j)} &\equiv \bar{w} g^{(j)} = \sum_{i=1}^n x_i^{(j)} b_{\pi(i)} \bar{w} \equiv \sum_{i=1}^n x_i^{(j)} a_{\pi(i)} \pmod{m} \\ &= \sum_{i=1}^n x_{\mu(i)}^{(j)} a_{\pi(\mu(i))} = \sum_{i=1}^n x_{\mu(i)}^{(j)} a_i \pmod{m}, \end{aligned}$$

by solving the easier 0-1 knapsack problems $K(a_1, \dots, a_n; G^{(j)})$ with super-increasing sequence a_1, \dots, a_n .

In 1982, Shamir [Shamir1982] specified an algorithm for breaking the system in polynomial time without solving the general knapsack problem. Len Adleman [Adleman1982] and Jeff

Lagarias [Lagarias1983] specified an algorithm for breaking the twice iterated Merkle-Hellman knapsack encryption procedure in polynomial time. Ernst Brickell [Brickell1985] then specified an algorithm for breaking multiply iterated Merkle-Hellman knapsack encryption procedures in polynomial time. This made this procedure unsuitable as an encryption procedure. It therefore delivers a one way function whose trapdoor information (defamiliarization of the 0-1 knapsack problem) could be discovered by an eavesdropper.

5.3 Decomposition into prime factors as a basis for public key procedures

5.3.1 The RSA procedure^{4,5}

As early as 1978, R. Rivest, A. Shamir, L. Adleman [RSA1978] introduced the most important asymmetric cryptography procedure to date.

Crypto Procedure 5.3 RSA (based on the factorization problem)

Key generation:

Let p and q be two different prime numbers and $N = pq$. Let e be any prime number relative to $\phi(N)$, i.e. $\gcd(e, \phi(N)) = 1$. Using the Euclidean algorithm, we calculate the natural number $d < \phi(N)$, such that

$$ed \equiv 1 \pmod{\phi(N)}.$$

whereby ϕ is the **Euler phi Function**.

The output text is divided into blocks and encrypted, whereby each block has a binary value $x^{(j)} \leq N$.

Public key:

$$N, e.$$

Private key:

$$d.$$

Encryption:

$$y = e_T(x) = x^e \pmod{N}.$$

Decryption:

$$d_T(y) = y^d \pmod{N}.$$

Comment:

The Euler phi function is defined as: $\phi(N)$ is the number of natural numbers that do not have a common factor with N $x \leq N$. Two natural numbers a and b are co-prime if $\gcd(a, b) = 1$.

For the Euler phi function it holds that:

$$\phi(1) = 1, \phi(2) = 1, \phi(3) = 2, \phi(4) = 2, \phi(6) = 2, \phi(10) = 4, \phi(15) = 8.$$

For example, $\phi(24) = 8$, because

$$|\{x < 24 : \gcd(x, 24) = 1\}| = |\{1, 5, 7, 11, 13, 17, 19, 23\}|.$$

⁴Please compare chapters 4.10, ff.

⁵Using CrypTool you can gain practical experience with the RSA procedure via the menu **Indiv.Procedures \ RSA Cryptosystem \ RSA Demonstration**.

If p is a prime number, then $\phi(p) = p - 1$.

If we know the various prime factors p_1, \dots, p_k of N , then

$$\phi(N) = N \cdot \left(1 - \frac{1}{p_1}\right) \cdots \left(1 - \frac{1}{p_k}\right).^6$$

Table 5.1 gives the values up to 15. In the case of $N = pq$,

$$\phi(N) = pq(1 - 1/p)(1 - 1/q) = p(1 - 1/p)q(1 - 1/q) = (p - 1)(q - 1).$$

n	$\phi(n)$	The natural numbers that are co-prime to n and less than n .
1	1	1
2	1	1
3	2	1, 2
4	2	1, 3
5	4	1, 2, 3, 4
6	2	1, 5
7	6	1, 2, 3, 4, 5, 6
8	4	1, 3, 5, 7
9	6	1, 2, 4, 5, 7, 8
10	4	1, 3, 7, 9
15	8	1, 2, 4, 7, 8, 11, 13, 14

Table 5.1: Euler phi function

The function e_T is a one way function whose trapdoor information is the decomposition into primes of N .

At the moment, no algorithm is known that can factorize two prime numbers sufficiently quickly for extremely large values (e.g. for several hundred decimal places). The quickest algorithms known today [Stinson1995] factorize a compound whole number N in a time period proportional to $L(N) = e^{\sqrt{\ln(N) \ln(\ln(N))}}$. Some example values can be found in table 5.2.

N	10^{50}	10^{100}	10^{150}	10^{200}	10^{250}	10^{300}
$L(N)$	$1.42 \cdot 10^{10}$	$2.34 \cdot 10^{15}$	$3.26 \cdot 10^{19}$	$1.20 \cdot 10^{23}$	$1.86 \cdot 10^{26}$	$1.53 \cdot 10^{29}$

Table 5.2: $L(N)$ value table

To this date, it has not been proved that the problem of breaking RSA is equivalent to the factorization problem. Nevertheless, it is clear that the RSA procedure will no longer be safe if the factorization problem is “solved”.⁷

⁶Further formulas for the Euler phi function are in the article “Introduction to Elementary Number Theory with Examples”, chapter 4.8.1.

⁷In 2000 the authors assumed that values of the order magnitude 100 to 200 decimal places are currently safe. They estimates that the current computer technology indicates that a number with 100 decimal places could be factorized in approximately two weeks at justifiable costs, and using an expensive configuration (e.g. of around 10 million US dollars), a number with 150 decimal places could be factorized in about a year, and a 200-digit number should remain impossible to factorize for a long time to come, unless there is a mathematical breakthrough. However, you can never be sure that there won’t be a mathematical breakthrough tomorrow. How easy it is to guess the future wrong is shown by the Factorization of RSA-200 (see chapter 4.11.4) – completely without a “mathematical breakthrough”.

5.3.2 Rabin public key procedure (1979)

The Rabin public key procedure (crypto procedure 5.4) has been shown to be equivalent to breaking the factorization problem. Unfortunately, this procedure is susceptible to chosen-ciphertext attacks.

Crypto Procedure 5.4 Rabin (based on the factorization problem)

Let p and q be two different prime numbers with $p, q \equiv 3 \pmod{4}$ and $n = pq$. Let $0 \leq B \leq n-1$.

Public key:

$$e = (n, B).$$

Private key:

$$d = (p, q).$$

Encryption:

$$y = e_T(x) = x(x + B) \pmod{n}.$$

Decryption:

$$d_T(y) = \sqrt{y + B^2/4} - B/2 \pmod{n}.$$

Caution: Because $p, q \equiv 3 \pmod{4}$ the encryption is easy to calculate (if the key is known). This is not the case for $p \equiv 1 \pmod{4}$. In addition, the encryption function is not injective: There are precisely four different source codes that have $e_T(x)$ as inverse image: $x, -x-B, \omega(x+B/2)-B/2, -\omega(x+B/2)-B/2$, where ω is one of the four roots of unity. The source codes therefore must be redundant for the encryption to remain unique!

Backdoor information is the decomposition into prime numbers of $n = pq$.

5.4 The discrete logarithm as basis for public key procedures⁸

Discrete logarithms form the basis for a large number of algorithms for public-key procedures.

5.4.1 The discrete logarithm in \mathbb{Z}_p

Let p be a prime number and let $g \in \mathbb{Z}_p^* = \{0, 1, \dots, p-1\}$. Then the discrete exponential function base g is defined as

$$e_g : k \longrightarrow y := g^k \pmod{p}, \quad 1 \leq k \leq p-1.$$

The inverse function is called a discrete logarithm function \log_g ; the following holds:

$$\log_g(g^k) = k.$$

The problem of the discrete logarithm (in \mathbb{Z}_p^*) is understood to be as follows:

Given p, g and y , determine k such that $y = g^k \pmod{p}$.

⁸With the educational tool for number theory **NT** you can play with the distribution of the discrete logarithm values and apply Shank's baby-step-giant-step method: See learning units 6.1-6.3, pages 1-6/6.

NT can be called in CrypTool via the menu path **Indiv. Procedures \ Number Theory Interactive \ Learning Tool for Number Theory**. See appendix A.4.

It is much more difficult to calculate the discrete logarithm than to evaluate the discrete exponential function (see chapter 4.9). Table 5.3 lists several procedures for calculating the discrete logarithm and their complexity [Stinson1995].

Name	Complexity
Baby-step-giant-step	$O(\sqrt{p})$
Silver-Pohlig-Hellman	polynomial in q , the greatest prime factor of $p - 1$.
Index-Calculus	$O(e^{(1+o(1))\sqrt{\ln(p) \ln(\ln(p))}})$

Table 5.3: Procedures for calculating the discrete logarithm over \mathbb{Z}_p^*

The current record (as of April 2007) for calculating discrete logarithms was established in February 2007 by the group Kleinjung, Franke and Bahr at University of Bonn.⁹ Kleinjung calculated the discrete logarithm modulo a 160 digit prime number p and generator g :

$$\begin{aligned}
p &= \lfloor 10^{159} \pi \rfloor + 119849 \\
&= 314159265358979323846264338327950288419716939937510582097494 \\
&\quad 459230781640628620899862803482534211706798214808651328230664 \\
&\quad 7093844609550582231725359408128481237299 \\
g &= 2
\end{aligned}$$

The discrete logarithms k of the following integer y was determined:¹⁰

$$\begin{aligned}
y &= \lfloor 10^{159} e \rfloor \\
&= 271828182845904523536028747135266249775724709369995957496696 \\
&\quad 762772407663035354759457138217852516642742746639193200305992 \\
&\quad 1817413596629043572900334295260595630738 \\
k &= \log_g(y) \mod p \\
&= 829897164650348970518646802640757844024961469323126472198531 \\
&\quad 845186895984026448342666252850466126881437617381653942624307 \\
&\quad 537679319636711561053526082423513665596
\end{aligned}$$

The search was performed with GNFS method (General Number Field Sieve, Index-Calculus) and took about 17 CPU years on 3.2 GHz Xeon machines.

5.4.2 Diffie-Hellman key agreement¹¹

The mechanisms and algorithms of classical cryptography only take effect when the subscribers have already exchanged the secret key. In classical cryptography you cannot avoid exchanging secrets without encrypting them. Transmission safety here must be achieved using non-cryptographic methods. We say that we need a secret channel for exchanging secrets. This channel can be realised either physically or organisationally.

What is revolutionary about modern cryptography is, amongst other things, that you no longer need secret channels: You can agree secret keys using non-secret, i.e. public channels.

One protocol that solves this problem is that of Diffie and Hellman (crypto procedure 5.5).

⁹[http://www.nabble.com/Discrete-logarithms-in-GF\(p\)-----160-digits-t3175622.html](http://www.nabble.com/Discrete-logarithms-in-GF(p)-----160-digits-t3175622.html)

¹⁰The integer y was chosen as the first 159 digits of the Euler number e .

¹¹With CrypTool this exchange protocol has been visualized: you can execute the single steps with concrete numbers using menu **Indiv. Procedures \ Protocols \ Diffie-Hellman Demonstration**.

Crypto Procedure 5.5 Diffie-Hellman key agreement

Two subscribers A and B want to agree on a joint secret key.

Let p be a prime number and g a natural number. These two numbers do not need to be secret. The two subscribers then select a secret number a and b from which they calculate the values $\alpha = g^a \bmod p$ and $\beta = g^b \bmod p$. They then exchange the numbers α and β . To end with, the two subscribers calculate the received value to the power of their secret value to get $\beta^a \bmod p$ and $\alpha^b \bmod p$.

Thus

$$\beta^a \equiv (g^b)^a \equiv g^{ba} \equiv g^{ab} \equiv (g^a)^b \equiv \alpha^b \bmod p$$

The safety of the **Diffie-Hellman protocol** is closely connected to calculating the discrete logarithm mod p . It is even thought that these problems are equivalent.

5.4.3 ElGamal public key encryption procedure in \mathbb{Z}_p^*

By varying the Diffie-Hellman key agreement protocol slightly, you can obtain an asymmetric encryption algorithm, crypto procedure 5.6. This observation was made by Taher ElGamal.

Crypto Procedure 5.6 ElGamal (based on the discrete logarithm problem)

Let p be a prime number such that the discrete logarithm in \mathbb{Z}_p is difficult to compute. Let $\alpha \in \mathbb{Z}_p^*$ be a primitive element. Let $a \in \mathbb{N}$ and $\beta = \alpha^a \bmod p$.

Public key:

$$p, \alpha, \beta.$$

Private key:

$$a.$$

Let $k \in \mathbb{Z}_{p-1}$ be a random number and $x \in \mathbb{Z}_p^*$ the plaintext.

Encryption:

$$e_T(x, k) = (y_1, y_2),$$

where

$$y_1 = \alpha^k \bmod p$$

and

$$y_2 = x\beta^k \bmod p.$$

Decryption:

$$d_T(y_1, y_2) = y_2(y_1^a)^{-1} \bmod p$$

5.4.4 Generalised ElGamal public key encryption procedure

The discrete logarithm can be generalised in any number of finite groups (G, \circ) . The following provides several properties of G , that make the discrete logarithm problem difficult.

Calculating the discrete exponential function Let G be a group with the operation \circ and $g \in G$. The (discrete) exponential function base g is defined as

$$e_g : k \mapsto g^k, \quad \text{for all } k \in \mathbb{N}.$$

where

$$g^k := \underbrace{g \circ \dots \circ g}_{k \text{ times}}.$$

The exponential function is easy to calculate:

Lemma. *The power g^k can be calculated in at most $2 \log_2 k$ group operations.*

Proof

Let $k = 2^n + k_{n-1}2^{n-1} + \dots + k_12 + k_0$ be the binary representation of k . Then $n \leq \log_2(k)$, because $2^n \leq k < 2^{n+1}$. k can be written in the form $k = 2k' + k_0$ with $k' = 2^{n-1} + k_{n-1}2^{n-2} + \dots + k_1$. Thus

$$g^k = g^{2k' + k_0} = (g^{k'})^2 g^{k_0}.$$

We therefore obtain g^k from $g^{k'}$ by squaring and then multiplying by g . The claim is thus proved by induction to n . \square

Problem of the discrete logarithm

Let G be a finite group with the operation \circ . Let $\alpha \in G$ and $\beta \in H = \{\alpha^i : i \geq 0\}$. We need to find a unique $a \in \mathbb{N}$ with $0 \leq a \leq |H| - 1$ and $\beta = \alpha^a$. We define a as $\log_\alpha(\beta)$.

Calculating the discrete logarithm A simple procedure for calculating the discrete logarithm of a group element, that is considerably more efficient than simply trying all possible values for k , is the baby-step-giant-step algorithm.

Theorem 5.4.1. *[baby-step-giant-step algorithm] Let G be a group and $g \in G$. Let n be the smallest natural number with $|G| \leq n^2$. Then the discrete logarithm of an element $h \in G$ can be calculated base g by generating two lists each containing n elements and comparing these lists. In order to calculate these lists, we need $2n$ group operations.*

Proof

First create the two lists

Giant-step list: $\{1, g^n, g^{2n}, \dots, g^{n \cdot n}\},$

Baby-step list: $\{hg^{-1}, hg^{-2}, \dots, hg^{-n}\}.$

If $g^{jn} = hg^{-i}$, i.e. $h = g^{i+jn}$, then the problem is solved. If the lists are disjoint, then h cannot be represented as g^{i+jn} , $i, j \leq n$. As all powers of g are thus recorded, the logarithm problem does not have a solution. \square

You can use the baby-step-giant-step algorithm to demonstrate that it is much more difficult to calculate the discrete logarithm than to calculate the discrete exponential function. If the numbers that occur have approximately 1000 bits in length, then you only need around 2000 multiplications to calculate g^k but around $2^{500} \approx 10^{150}$ operations to calculate the discrete logarithm using the baby-step-giant-step algorithm.

In addition to the baby-step-giant-step algorithm, there are also numerous other procedures for calculating the discrete logarithm [Stinson1995].

The theorem from Silver-Pohlig-Hellman In finite Abelian groups, the discrete logarithm problem can be reduced to groups of a lower order.

Theorem 5.4.2. [Silver-Pohlig-Hellman] Let G be a finite Abelian group with $|G| = p_1^{a_1} p_2^{a_2} \cdot \dots \cdot p_s^{a_s}$. The discrete logarithm in G can then be reduced to solving logarithm problems in groups of the order p_1, \dots, p_s .

If $|G|$ contains a “dominant” prime factor p , then the complexity of the logarithm problem is approximately

$$O(\sqrt{p}).$$

Therefore, if the logarithm problem is to be made difficult, the order of the group used G should have a large prime factor. In particular, if the discrete exponential function in the group \mathbb{Z}_p^* is to be a one way function, then $p - 1$ must be a large prime factor. In this case a generalized ElGamal procedure can be defined (crypto procedure 5.7).

Crypto Procedure 5.7 Generalized ElGamal (based on the factorization problem)

Let G be a finite group with operation \circ , and let $\alpha \in G$, so that the discrete logarithm in $H = \{\alpha^i : i \geq 0\}$ is difficult, Let a with $0 \leq a \leq |H| - 1$ and let $\beta = \alpha^a$.

Public key:

$$\alpha, \beta.$$

Private key:

$$a.$$

Let $k \in \mathbb{Z}_{|H|}$ be a random number and $x \in G$ be a plaintext.

Encryption:

$$e_T(x, k) = (y_1, y_2),$$

where

$$y_1 = \alpha^k$$

and

$$y_2 = x \circ \beta^k.$$

Decryption:

$$d_T(y_1, y_2) = y_2 \circ (y_1^a)^{-1}$$

Elliptic curves provide useful groups for public key encryption procedures.

Bibliography

- [Adleman1982] Adleman L.:
On breaking the iterated Merkle-Hellman public key Cryptosystem.
Advances in Cryptology, Proceedings of Crypto 82, Plenum Press 1983, 303-308.
- [Balcazar1988] Balcazar J.L., Daaz J., Gabarr J.:
Structural Complexity I.
Springer Verlag, pp 63.
- [Brickell1985] Brickell E.F.:
Breaking Iterated Knapsacks.
Advances in Cryptology: Proc. CRYPTO'84, Lecture Notes in Computer Science, vol. 196,
Springer-Verlag, New York, 1985, pp. 342-358.
- [Lagarias1983] Lagarias J.C.:
Knapsack public key Cryptosystems and diophantine Approximation.
Advances in Cryptology, Proceedings of Crypto 83, Plenum Press.
- [Merkle1978] Merkle R. and Hellman M.:
Hiding information and signatures in trapdoor knapsacks.
IEEE Trans. Information Theory, IT-24, 1978.
- [RSA1978] Rivest R.L., Shamir A. and Adleman L.:
A Method for Obtaining Digital Signatures and Public Key Cryptosystems.
Commun. ACM, vol 21, April 1978, pp. 120-126.
- [Shamir1982] Shamir A.:
A polynomial time algorithm for breaking the basic Merkle-Hellman Cryptosystem.
Symposium on Foundations of Computer Science (1982), 145-152.
- [Stinson1995] Stinson D.R.:
Cryptography.
CRC Press, Boca Raton, London, Tokyo, 1995.

Web links

1. http://www.geocities.com/st_busygin/clipat.html

Chapter 6

Hash Functions and Digital Signatures

(Schneider J. / Esslinger B. / Koy H., June 2002; Updates: Feb. 2003, June 2005, July 2009)

The aim of digital signatures is to guarantee the following two points:

- User authenticity:
It can be checked whether a message really does come from a particular person.
- Message integrity:
It can be checked whether the message has been changed (on route).

An asymmetric technique is used again (see encryption procedures). Participants who wish to generate a digital signature for a document must possess a pair of keys. They use their secret key to generate signatures and the recipient uses the sender's public key to verify whether the signature is correct. As before, it must be impossible to use the public key to derive the secret key¹.

In detail, a *Signature procedure* looks like this:

Senders use their message and secret key to calculate the digital signature for the message. Compared to hand-written signatures, digital signatures therefore have the advantage that they also depend on the document to be signed. Signatures from one and the same participant are different unless the signed documents are completely identical. Even inserting a blank in the text would lead to a different signature. The recipient of the message would therefore detect any injury to the message integrity as this would mean that the signature no longer matches the document and is shown to be incorrect when verified.

The document is sent to the recipient together with the signature. The recipient can then use the sender's public key, the document and the signature to establish whether or not the signature is correct. The procedure we just described has in practice, however, a decisive disadvantage. The signature would be approximately as long as the document itself. To prevent an unnecessary increase in data traffic, and also for reasons of performance, we apply a cryptographic hash

¹With CrypTool you can also generate and check digital signatures: Using the submenus of the main menu **Digital Signatures / PKI** or using menu **Indiv. Procedures \ RSA Cryptosystem \ Signature Demonstration (Signature Generation)**.

function² to the document – before signing. The output of the hash function will then be signed.

*Stanislaw Lem*³:

We can make everything out of this world, but we cannot create a world, where humans in some ten thousand years can think: 'Ok, now it is enough. Everything should stay like it is. Let's do no changes any more, don't do inventions any more, because it cannot become better, and if, then we don't want this.'

6.1 Hash functions

A *hash function*⁴ maps a message of any length to a string of characters with a constant size, the hash value.

6.1.1 Requirements for hash functions

Cryptographically secure hash functions fulfill the following three requirements (the order is in a way that the requirements increase):

- Resistance against 1st Pre-Image attacks:
It should be practically impossible, for a given number, to find a message that has precisely this number as hash value.
Given (fix): hash value H' ,
Searched: message m , so that: $H(m) = H'$.
- Resistance against 2nd Pre-Image attacks:
It should be practically impossible, for a given message, to find another message, which has precisely the same hash value.
Given (fix): message m_1 [and so the hash value $H_1 = H(m_1)$],
Searched: message m_2 , so that: $H(m_2) = H_1$.

²Hash functions are implemented within CrypTool at several places.

Using menus **Individual Procedures** \ **Hash** and **Analysis** \ **Hash** you have the possibilities

- to apply one of 6 hash functions to the content of the current window,
- to calculate the hash value of a file,
- to test, how changes to a text change the according hash value,
- to calculate a key from a password according to the PKCS#5 standard,
- to calculate HMACs from a text and a secret key, and
- to perform a simulation, how digital signatures could be attacked by a targeted search for hash value collisions.

³This was the answer of Stanislaw Lem to heavy critics at his philosophical main book "Summa Technologiae", 1964, where he thought about the possibility of an evolution creating artificial intelligence.

⁴Hash algorithms compute a condensed representation of electronic data (message). When a message is input to a hash algorithm, the result is an output called a message digest. The message digests typically range in length from 128 to 512 bits, depending on the algorithm. Secure hash algorithms are typically used with other cryptographic algorithms, such as digital signature algorithms and keyed-hash message authentication codes, or in the generation of random numbers (bits).

- Collision resistance:

It should be practically impossible to find any two messages with the same hash value (it doesn't matter what hash value).

Searched: 2 messages m_1 and m_2 , so that: $H(m_1) = H(m_2)$.

6.1.2 Current attacks against hash functions like SHA-1

So far, no formal proof has been found that perfectly secure cryptographic hash functions exist.

During the past several years no new attacks against hash algorithms came up, and so the candidates that had not yet shown any weaknesses in their structure in practice (e.g. SHA-1⁵ or RIPEMD-160⁶) were trusted.

At Crypto 2004 (August 2004)⁷ this safety-feeling was disputed: Chinese researchers published collision attacks against MD4, SHA-0 and parts of SHA-1. This globally caused new motivation to engage in new hash attack methods.

The initially published result reduced the expected complexity for one SHA-1 collision search from 2^{80} (brute-force) to 2^{69} [Wang2005]. More recent announcements claim to further reduce the required effort to 2^{63} [Wang2005b] and 2^{52} [McDonald2009]. This would bring collision attacks into the practical realm, as similar efforts have been mastered in the past (s. 1.1.2).

According to our current knowledge there is no need to run scared. But in the future digital signatures should use longer hash values and/or other hash algorithms.

Already before Crypto 2004 the U.S. National Institute of Standards and Technology (NIST) announced, to discontinue SHA-1 in the next few years. So it is recommended not to use SHA-1 for new products generating digital signatures. The SHA-2 family [FIPS180-3] provides stronger algorithms. To address new findings in cryptanalysis, NIST has opened a competition to develop a new cryptographic hash algorithm "SHA-3" in 2008. The competition is expected to end in 2012.⁸

Further information about this topic can be found in the article "Hash cracked – The consequences of the successful attacks on SHA-1" by Reinhard Wobst and Jürgen Schmidt⁹ by

⁵SHA-1 is a 160 bit hash function specified in FIPS 180-1 (by NIST), ANSI X9.30 Part 2 and [FIPS186].

SHA means Secure Hash Algorithm, and is widely used, e.g. with DSA, RSA or ECDSA.

The current standard [FIPS180-3] defines four secure hash algorithms – SHA-1, SHA-256, SHA-384, and SHA-512. For these hash algorithms there are also validation tests defined in the test suite FIPS 140-2.

The output length of the SHA algorithms was enhanced because of the possibility of birthday attacks: these make n -bit AES and a $2n$ -bit hash roughly equivalent:

- 128-bit AES – SHA-256
- 192-bit AES – SHA-384
- 256-bit AES – SHA-512.

With CrypTool you can comprehend the birthday attack on digital signatures:

using the menu **Analysis \ Hash \ Attack on the Hash Value of the Digital Signature**.

⁶RIPEMD-160, RIPEMD-128 and the optional extension RIPEMD-256 have object identifiers defined by the ISO-identified organization TeleTrust, both as hash algorithm and in combination with RSA. RIPEMD-160 is also part of the ISO/IEC international standard ISO/IEC 10118-3:1998 on dedicated hash functions, together with RIPEMD-128 and SHA-1. Further details:

- <http://www.esat.kuleuven.ac.be/~bosselae/ripemd160.html>

- <http://www.ietf.org/rfc/rfc2857.txt> ("The Use of HMAC-RIPEMD-160-96 within ESP and AH").

⁷<http://www.iacr.org/conferences/crypto2004/>

⁸<http://csrc.nist.gov/groups/ST/hash/sha-3/>

⁹<http://www.heise.de/security/artikel/56634>.

Further references are e.g.:

<http://www.bsi.bund.de/esig/basics/techbas/krypto/index.htm>

<http://csrc.nist.gov/CryptoToolkit/tkhash.html>.

6.1.3 Signing with hash functions

The signature procedure with hash functions is as follows:¹⁰

Rather than signing the actual document, the sender now first calculates the hash value of the message and signs this. The recipient also calculates the hash value of the message (the algorithm used must be known), then verifies whether the signature sent with the message is a correct signature of the hash value. If this is the case, the signature is verified to be correct. This means that the message is authentic, because we have assumed that knowledge of the public key does not enable you to derive the secret key. However, you would need this secret key to sign messages in another name.

Some digital signature schemes are based on asymmetric *encryption* procedures, the most prominent example being the RSA system, which can be used for signing by performing the private key operation on the hash value of the document to be signed.

Other digital signature schemes were developed exclusively for this purpose, as the DSA (Digital Signature Algorithm), and are not directly connected with a corresponding encryption scheme.

Both, RSA and DSA signature are discussed in more detail in the following two sections. After that we go one step further and show how digital signatures can be used to create the digital equivalent of ID cards. This is called Public Key Certification.

6.2 RSA signatures

As mentioned in the comment at the end of section 4.10.3 it is possible to perform the RSA private and public key operation in reverse order, i. e. raising M to the power of d and then to the power of $e \pmod{N}$ yields M again. Based on this simple fact, RSA can be used as a signature scheme.

The RSA signature S for a message M is created by performing the private key operation:

$$S \equiv M^d \pmod{N}$$

In order to verify, the corresponding public key operation is performed on the signature S and the result is compared with message M :

$$S^e \equiv (M^d)^e \equiv (M^e)^d \equiv M \pmod{N}$$

If the result matches the message M , then the signature is accepted by the verifier, otherwise the message has been tampered with, or was never signed by the holder of d .

As explained above, signatures are not performed on the message itself, but on a cryptographic hash value of the message. To prevent certain attacks on the signature procedure (alone or in combination with encryption) it is necessary to format the hash value before doing the exponentiation, as described in the PKCS#1 (Public Key Cryptography Standard #1 [PKCS1]). The fact that this standard had to be revised recently, after being in use for several years, can serve as an example of how difficult it is to get the details of cryptography right.

¹⁰Compare: http://en.wikipedia.org/wiki/Digital_signature.

6.3 DSA signatures

In August of 1991, the U.S. National Institute of Standards and Technology (NIST) proposed a digital signature algorithm (DSA), which was subsequently adopted as a U.S. Federal Information Processing Standard (FIPS 186 [FIPS186]).

The algorithm is a variant of the ElGamal scheme. Its security is based on the Discrete Logarithm Problem. The DSA public and private key and its procedures for signature and verification are summarised in crypto procedure 6.1.

Crypto Procedure 6.1 DSA signature

Public Key

p prime

q 160-bit prime factor of $p - 1$

$g = h^{(p-1)/q} \bmod p$, where $h < p - 1$ and $h^{(p-1)/q} > 1 \pmod{p}$

$y \equiv g^x \bmod p$

Remark: Parameters p, q and g can be shared among a group of users.

Private Key

$x < q$ (a 160-bit number)

Signing

m the message to be signed

k choose at random, less than q

$r = (g^k \bmod p) \bmod q$

$s = (k^{-1}(\text{SHA-1}(m) + xr)) \bmod q$

Remark:

- (s, r) is the signature.
- The security of the signature depends not only on the mathematical properties, but also on using a good random source for k .
- SHA-1 is a 160-bit hash function.

Verifying

$w = s^{-1} \bmod q$

$u_1 = (\text{SHA-1}(m)w) \bmod q$

$u_2 = (rw) \bmod q$

$v = (g^{u_1}y^{u_2}) \bmod p \bmod q$

Remark: If $v = r$, then the signature is verified.

While DSA was specifically designed, so that it can be exported from countries regulating export of encryption soft and hardware (like the U.S. at the time when it was specified), it has been noted [Schneier1996, p. 490], that the operations involved in DSA can be used to emulate RSA and ElGamal encryption.

6.4 Public key certification

The aim of public key certification is to guarantee the connection between a public key and a user and to make it traceable for external parties. In cases in which it is impossible to ensure that a public key really belongs to a particular person, many protocols are no longer secure,

even if the individual cryptographic modules cannot be broken.

6.4.1 Impersonation attacks

Assume Charlie has two pairs of keys (PK1, SK1) and (PK2, SK2), where SK denotes the secret key and PK the public key. Further assume that he manages to palm off PK1 on Alice as Bob's public key and PK2 on Bob as Alice's public key (by falsifying a public key directory).

Then he can attack as follows:

- Alice wants to send a message to Bob. She encrypts it using PK1 because she thinks that this is Bob's public key. She then signs the message using her secret key and sends it.
- Charlie intercepts the message, removes the signature and decrypts the message using SK1. If he wants to, he can then change the message in any way he likes. He then encrypts the message again, but this time using Bob's genuine public key, which he has taken from a public key directory, signs the message using SK2 and forwards it to Bob.
- Bob verifies the signature using PK2 and will reach the conclusion that the signature is correct. He then decrypts the message using his secret key.

In this way Charlie can listen in on communication between Alice and Bob and change the exchanged messages without them noticing. The attack will also work if Charlie only has one pair of keys.

Another name for this type of attack is "man-in-the-middle attack". Users are promised protection against this type of attack by public-key certification, which is intended to guarantee the authenticity of public keys. The most common certification method is the X.509 standard.

6.4.2 X.509 certificate

Each participant who wants to have an X.509 certificate ([X.509]) verifying that his public key belongs to a real person consults what is known as a certification authority (CA)¹¹. He proves his identity to this CA (for example by showing his ID). The CA then issues him an electronic document (certificate) which essentially contains the name of the certificate-holder and the name of the CA, the certificate-holder's public key and the validity period of the certificate. The CA then signs the certificate using its secret key.

Anyone can now use the CA's public key to verify whether a certificate is falsified. The CA therefore guarantees that a public key belongs to a particular user.

This procedure is only secure as long as it can be guaranteed that the CA's public key is correct. For this reason, each CA has its public key certified by another CA that is superior in the hierarchy. In the upper hierarchy level there is usually only one CA, which can of course then have its key certified by another CA. It must therefore transfer its key securely in another way. In the case of many software products that work with certificates (such as the Microsoft and Netscape Web browsers), the certificates of these root CAs are permanently embedded in the program right from the start and cannot be changed by users at a later stage. However, (public) CA keys, in particularly those of the root entity, can also be secured by means of making them available publicly.

¹¹Often called trust center, if the certificates are not only offered to a closed user group.

Bibliography

- [FIPS180-3] U.S. Department of Commerce/N.I.S.T. ,
Secure Hash Standard (SHS),
October 2008.
(FIPS 180-3 supersedes FIPS 180-2.)
- [FIPS186] U.S. Department of Commerce/N.I.S.T. ,
Entity authentication using public key cryptography,
February 18, 1997.
No more valid.
- [FIPS186-2] U.S. Department of Commerce/N.I.S.T. ,
Digital Signature Standard (DSS),
January 27, 2000. Change Note: October 5, 2001.
<http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf>
- [McDonald2009] Cameron McDonald, Philip Hawkes, Josef Pieprzyk,
Differential Path for SHA-1 with complexity $O(2^{52})$,
<http://eprint.iacr.org/2009/259>
- [PKCS1] RSA Laboratories,
PKCS #1 v2.1 Draft 3: RSA Cryptography Standard,
April 19, 2002.
- [Schneier1996] Bruce Schneier,
Applied Cryptography, Protocols, Algorithms, and Source Code in C,
Wiley, 2nd edition, 1996.
- [Wang2005] Xiaoyun Wang, Yiqun Yin, Hongbo Yu,
Finding Collisions in the Full SHA-1,
Advances in Cryptology-Crypto 2005, LNCS 3621: 17-36, 2005.
- [Wang2005b] Xiaoyun Wang, Andrew Yao and Frances Yao,
New Collision Search for SHA-1,
Crypto 2005 Rump Session
<http://www.iacr.org/conferences/crypto2005/rumpSchedule.html>
- [Wobst2005] Reinhard Wobst,
New Attacks Against Hash Functions,
Information Security Bulletin, April 2005.
- [X.509] ITU-T,
ITU-T Recommendation X.509 (1997 E): Information Technology – Open Systems In-

terconnection – The Directory: Authentication Framework,
June 1997.

[X.509v3] ITU-T,
X.509 (1993) Amendment 1: Certificate Extensions, The Directory Authentication Framework,
International Telecommunication Union, Geneva, Switzerland, July 1995
(equivalent to amendment 1 to ISO/IEC 9594-8).

Chapter 7

Elliptic Curves

(Filipovics B. / Büger M. / Esslinger B. / Oyono R., April 2000, Updates: Dec. 2001, June 2002, Mar. 2003, November 2009)

7.1 Elliptic curve cryptography – a high-performance substitute for RSA?

In many business sectors secure and efficient data transfer is essential. In particular, the RSA algorithm is used in many applications. Although the security of RSA is beyond doubt, the evolution in computing power has caused a growth in the necessary key length. Today, 1024-bit RSA keys are standard, but the GISA (German Information Security Agency) recommends the usage of 2048-bit keys from 2006 on (compare section 4.11). The fact that most chips on smart cards cannot process keys extending 1024 bit shows that there is a need for alternatives. Elliptic curve cryptography (ECC) can be such an alternative in the area of asymmetric cryptography.

The efficiency of a cryptographic algorithm depends on the key length and the calculation effort that is necessary to provide a prescribed level of security. The major advantage of ECC compared to RSA is that it requires much shorter key lengths. If we assume that the computing power increases by Moore's law (i. e. it doubles every 18 months)¹, then the evolution of the key lengths for secure communication will be as figure 7.1 [Lenstra1999] (source: Arjen Lenstra and Eric Verheul: <http://cryptosavvy.com/table.htm>).

In addition, a digital signature can be processed 10-times faster with ECC than with RSA. However, verification of a given signature is still more efficient with RSA than with ECC. Refer to figure 7.2 (source: Dr. J. Merkle, Elliptic Curve Cryptography Workshop, 2001) for a comparison. The reason is that RSA public keys can be chosen relatively small as long as the secret key is long enough.

Nevertheless, thin clients like smart cards usually have to store the (long) secret key and have to process a digital signature rather than verify one. Therefore, there is a clear advantage in using ECC in terms of efficiency.

Nowadays, the major problem with ECC implementations is the lack of standardization. There is only one way to implement RSA, but there are many ways for ECC: One can work with different sets of numbers, different (elliptic) curves — described by parameters² — , and

¹empirical knowledge by Gordon Moore, co-founder of Intel, 1965

²see chapter 7.4

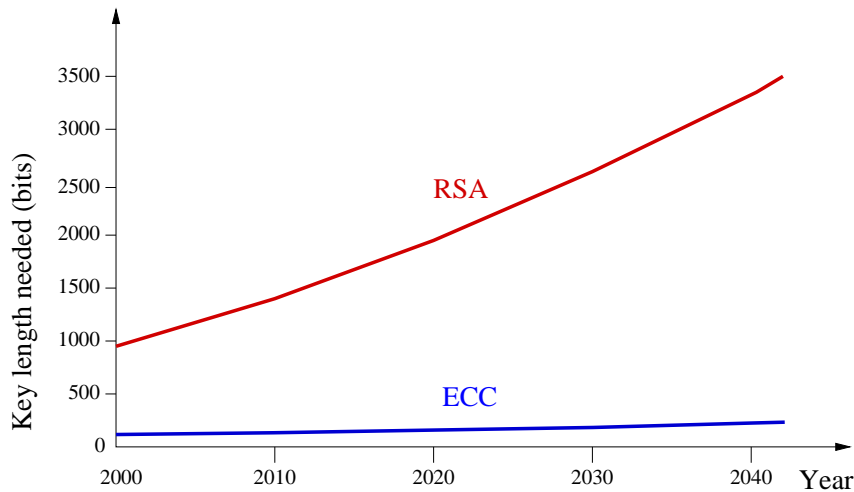


Figure 7.1: Prognosis of the key lengths to be regarded safe for RSA and Elliptic Curves

a variety of representations of the elements on the curve. Each choice has its advantages and disadvantages, and one can certainly construct the most efficient for each application. However, this causes problems in interoperability. But if all ECC-tools should be able to communicate with each other, they will have to support all different algorithms, which might put the advantage of efficient computation and the need of less storage capacity to the contrary.

Therefore, international standardization organizations like IEEE (P1363), ASC (ANSI X9.62, X9.63), ISO/IEC as well as major players like RSA labs or Certicom have recently started standardization initiatives. While the IEEE only describes the different implementations, the ASC has explicitly stated 10 elliptic curves and recommends their usage. The advantage of the ASC approach is that one needs only a single byte to indicate which curve is meant. However, it is not yet clear whether the ASC curves will become a de facto standard.

Although we see no need to replace RSA in any application today³, one should take the usage of ECC-based tools into consideration whenever a new system is set up — in particular, when the tool should be available beyond 2005⁴.

7.2 Elliptic curves – history

Mathematicians have been researching elliptic curves for over 100 years. Over the course of time, many lengthy and mathematically complex results have been found and published which are connected to elliptic curves. A mathematician would say that elliptic curves (or the mathematics behind them) are widely understood. This research was originally purely mathematical. That is to say, elliptic curves were investigated, for example, in the mathematical areas of number theory and algebraic geometry, which are generally highly abstract. Even in the recent past, elliptic curves played an important role in pure mathematics. In 1993 and 1994, Andrew Wiles published mathematical works that triggered enthusiasm far beyond the specialist audience. In these works, he proved a conjecture put forward in the 1960's. To put it short, this conjecture was concerned with the connection between elliptic curves and what are called module forms.

³Current information about the security of the RSA algorithm can be found in chapter 4.11.

⁴Compare the recommendation of GISA: “Fitting Crypto Algorithms” from October 24th, 2002.

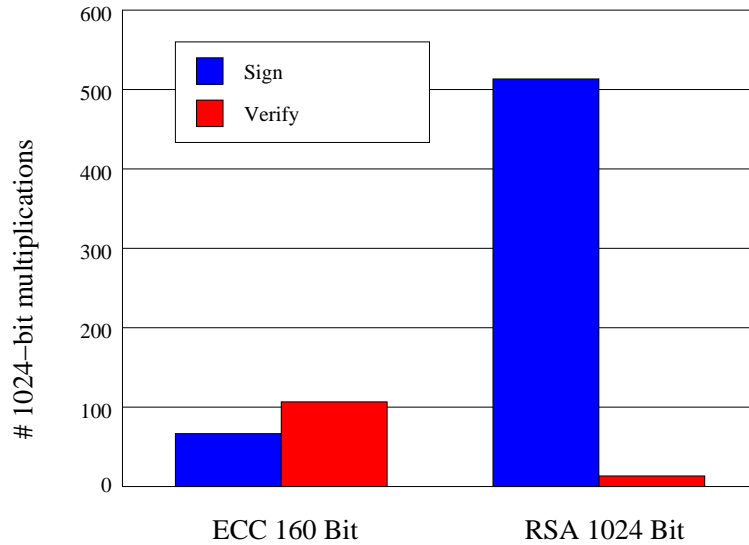


Figure 7.2: Comparison of signing and verification time for RSA and Elliptic Curves

What is particularly interesting for most people is that the works of Wiles also proved the famous second theorem of Fermat. Mathematicians had spent centuries (Fermat lived from 1601 to 1665) trying to find a strict proof of this theorem. Understandably, therefore, Wiles' proof got a good response. Fermat formulated his theorem as follows (written in the border of a book):

Cubum autem in duos cubos, aut quadratoquadratum in duos quadratoquadratos, et generaliter nullam in infinitum ultra quadratum potestatem in duos ejusdem nominis fas est dividere: cujus rei demonstrationem mirabilem sane detexi. Hanc marginis exiguitas non caperet.

With a free translation, using the denotation of modern mathematics, this means: No positive whole numbers x, y and z greater than zero exist such that $x^n + y^n = z^n$ for $n > 2$. I have found an amazing proof of this fact, but there is too little space within the confines of this book to include it.

This is truly amazing: A statement that is relatively simple to understand (we are referring to Fermat's second theorem here) could only be proved after such a long period of time, although Fermat himself claimed to have found a proof. What's more, the proof found by Wiles is extremely extensive (all of Wiles publications connected with the proof made up a book in themselves). This should therefore make it obvious that elliptic curves are generally based on highly complex mathematics.

Anyway that's enough about the role of elliptic curves in pure mathematics. In 1985 Neal Koblitz and Victor Miller independently suggested using elliptic curves in cryptography. Elliptic curves have thus also found a concrete practical application. Another interesting area of application for elliptic curves is for factorizing whole numbers (the RSA cryptographic system is based on the difficulty/complexity of finding prime factors of an extremely large number;

compare section 4.11.). In this area, procedures based on elliptic curves have been investigated and used since 1987 (compare section 7.8).

There are also prime number tests based on elliptic curves.

Elliptic curves are used differently in the various areas. Encryption procedures based on elliptic curves are based on the difficulty of a problem known as elliptic curve discrete logarithm. The factorization of whole numbers uses the fact that a large number of elliptic curves can be generated for a natural composite number n with several prime factors; however, these curves are not then groups for composite n . More information about this can be found under the chapter 7.8.

7.3 Elliptic curves – mathematical basics

This section provides information about *groups* and *fields*.

7.3.1 Groups

Because the term *group* is used differently in everyday language than in mathematics, we will, for reasons of completeness, begin by introducing the essential statement of the formal definition of a group:

- A group is a non-empty set G on which an operation “ \cdot ”. The set G is closed under this operation, which means that for any two elements a, b taken from G , performing the operation on them gives an element in G , i.e. $ab = a \cdot b$ lies in G .
- For all elements a, b and c in G : $(ab)c = a(bc)$ (associative law).
- There exists an element e in G that behaves neutrally with respect to the operation \cdot . That means that for all a in the set G : $ae = ea = a$.
- For each element a in G there exists a so-called inverse⁵ element a^{-1} in G such that: $aa^{-1} = a^{-1}a = e$.

If also $ab = ba$ (commutative law) for all a, b in G , then we call the group an *Abelian* group.

Since we may define different operations on the same set, we distinguish them by giving them different names (e.g. $+$ addition or \cdot multiplication).

The simplest example of an (Abelian) group is the group of whole numbers under the standard operation of addition. The set of whole numbers is denoted as \mathbb{Z} . \mathbb{Z} has an infinite number of elements, because $\mathbb{Z} = \{\dots, -4, -3, -2, -1, 0, 1, 2, 3, 4, \dots\}$. For example, the operation of $1 + 2$ lies in \mathbb{Z} , for $1 + 2 = 3$ and 3 lies in \mathbb{Z} . The neutral element in the group \mathbb{Z} is 0. The inverse element of 3 is -3 , for $3 + (-3) = 0$.

For our purpose, so-called *finite* groups play an important role. This means that there exists a set \mathcal{M} with a fixed number of elements and an operation $+$ such that the above conditions are fulfilled. One example of this is any set \mathbb{Z}_n where $\mathbb{Z}_n = \{0, 1, 2, 3, \dots, n-1\}$, n is a positive whole number and the operation is addition mod n , i.e. a and b in \mathbb{Z}_n are subject to the operation $a + b \bmod n$.

⁵The inverse is uniquely determined because if $x, y \in G$ are each inverse to a , i.e. $ax = xa = e$ and $ay = ya = e$, then $x = xe = x(ay) = (xa)y = ey = y$.

Cyclic groups Cyclic groups⁶ are those groups G' that possess an element g from which the group operation can be used to generate all other elements in the group. This means that for each element a in G' there exists a positive whole number i such that if g is subject to the operation i times (i.e. “ $g \cdot i$ ”), $g + g + \cdots + g = a$ (additive group) or $g^i = g \cdot g \cdots g = a$ (multiplicative group). The element g is the *generator* of the cyclic group — each element in G' can be generated using g and the operation.

Group order Now to the order of an element of the group: Let a be in G . The smallest positive whole number r for which a subject to the operation with itself r times is the neutral element of the group G' (i.e.: $r \cdot a = a + a + \cdots + a = e$ respectively $a^r = e$), is called the *order* of a .

The order of the group is the number of elements in the set G .

7.3.2 Fields

In mathematics, one is often interested in sets on which at least two (group) operations are defined — frequently called addition and multiplication. Most prominent are so called fields.

A field is understood to be a set K with two operations (denoted as $+$ and \cdot) which fulfils the following conditions:

- The set K forms an Abelian group together with the operation $+$ (addition), where 0 is the neutral element of the operation $+$.
- The set $K \setminus \{0\}$ also forms an Abelian group together with the operation \cdot (multiplication).
- For all elements a, b and c in K , we have $c \cdot (a + b) = c \cdot a + c \cdot b$ and $(a + b) \cdot c = a \cdot c + b \cdot c$ (distributive law).

Fields may contain an infinite number of elements (e.g. the field of real numbers). They are called *infinite* fields. In contrast we call a field *finite*, if it contains only a finite number of elements (e.g. $\mathbb{Z}_p = \{0, 1, 2, 3, \dots, p-1\}$, where p is a prime. \mathbb{Z}_p with addition mod p and multiplication mod p).

Characteristic of a field Let K be a field and 1 be the neutral element of K with respect to the multiplicative operation “ \cdot ”. Then the characteristic of K is said to be the order of 1 with respect to the additive operation. This means that the characteristic of K is the smallest positive integer n such that

$$\underbrace{1 + 1 + \cdots + 1}_{n \text{ times}} = 0.$$

If there is no such n , i.e. if $1 + 1 + \cdots + 1 \neq 0$ no matter how many 1 s we add, then we call K a field with characteristic 0 .

Thus, fields with characteristic 0 are infinite since they contain the (pairwise distinct) elements $1, 1 + 1, 1 + 1 + 1, \dots$. On the other hand, fields with finite characteristic may be finite or infinite.

If the characteristic is finite, it has to be prime. This fact can easily be proved: Assume $n = pq$, $p, q < n$, is the characteristic of a field K . By definition of n , the elements $\bar{p} =$

⁶Cyclic groups can be in general also endless like the additive group of the integer numbers. We consider here only finite cyclic groups.

$\underbrace{1 + 1 + \dots + 1}_{p \text{ times}}, \bar{q} = \underbrace{1 + 1 + \dots + 1}_{q \text{ times}}$ of K are not equal to 0. Thus, there exist inverse elements $\bar{p}^{-1}, \bar{q}^{-1}$ with respect to multiplication. It follows that $(\bar{p}\bar{q})(\bar{p}^{-1}\bar{q}^{-1}) = 1$, which contradicts the fact that $\bar{p}\bar{q} = \bar{n} = \underbrace{1 + 1 + \dots + 1}_{n \text{ times}} = 0$ and, hence, $\underbrace{(\bar{p}\bar{q})}_{=0}(\bar{p}^{-1}\bar{q}^{-1}) = 0$.

Comment:

The field of real numbers has the characteristic 0; the field \mathbb{Z}_p has the characteristic p . If p is not prime, \mathbb{Z}_p is not a field at all.

The most simple field is $\mathbb{Z}_2 = \{0, 1\}$. It contains only two elements, the neutral elements with respect to addition and multiplication. In particular, we have $0 + 0 = 0$, $0 + 1 = 1 + 0 = 1$, $1 + 1 = 0$, $1 \cdot 1 = 1$, $0 \cdot 0 = 0 \cdot 1 = 1 \cdot 0 = 0$.

Finite Fields As mentioned above, each finite field has a characteristic $p \neq 0$, where p is a prime. On the other hand, given a prime p there is a field which has exactly p elements, that is \mathbb{Z}_p .

However, the number of elements of a field need not be prime in general. For example, it is not hard to construct a field with 4 elements⁷.

One can show that the order of any field is a prime power (i.e. the power of a prime number). On the other hand, we can construct a field with p^n elements for any given prime p and positive integer n . Since two fields that have the same number of elements can not be distinguished⁸, we say that there is **the field with p^n elements** and denote it by $GF(p^n)$. Here GF stands for *Galois Field* to commemorate the French Mathematician Galois.

The fields $GF(p)$ of prime order play a prominent role. They are called prime fields and often denoted by \mathbb{Z}_p ⁹.

7.4 Elliptic curves in cryptography

In cryptography elliptic curve are a useful tool. Such curves are described by some equation. A detailed analysis has shown that curves of the form¹⁰

$$F(x_1, x_2, x_3) = -x_1^3 + x_2^2x_3 + a_1x_1x_2x_3 - a_2x_1^2x_3 + a_3x_2x_3^2 - a_4x_1x_3^2 - a_6x_3^3 = 0, \quad (7.1)$$

⁷The set $K = \{0, 1, a, b\}$ fitted with the operation defined in the tabular below is a field:

+	0	1	a	b
0	0	1	a	b
1	1	0	b	a
a	a	b	0	1
b	b	a	1	0

und

·	0	1	a	b
0	0	0	0	0
1	0	1	a	b
a	0	a	b	1
b	0	b	1	a

⁸If K, K' are fields with $k = p^n$ elements, then there is a one-to-one map $\varphi : K \rightarrow K'$, that respects the arithmetic of the field. Such a map is called an isomorphism. Isomorphic fields mathematically behave in the same way so that it makes no sense to distinguish between them. For example, \mathbb{Z}_2 und $K' = \{ZERO, ONE\}$ with zero-element $ZERO$ and one-element ONE are isomorphic. We note that mathematical objects are only defined by their mathematical properties.

⁹For prime fields additive as well as multiplicative group are cyclic. Furthermore, each field $GF(p^n)$ contains a subfield that is isomorphic to the prime field \mathbb{Z}_p .

¹⁰This curve is given by the zeros of a *polynomial* F of degree three in three variables. In general, expressions of the form $P = \sum_{i_1, \dots, i_n \in \mathbb{N}_0} a_{i_1 \dots i_n} x_1^{i_1} \dots x_n^{i_n}$ with coefficients $a_{i_1 \dots i_n} \in K$ are called polynomials in n variables x_1, \dots, x_n with underlying field K , if $\deg P := \max\{i_1 + \dots + i_n : a_{i_1 \dots i_n} \neq 0\}$ is finite, i.e. the sum has only finitely many non-zero terms (monomials). The sum of the exponents of the variables of each term of the sum is at most 3, at least one term of the sum has a single variable with 3 as value of the according exponent.

are especially useful. The variables x_1, x_2, x_3 and parameters a_1, \dots, a_4, a_6 are elements of a given field K , which has certain properties that make it useful from the cryptographic point of view. The underlying field K might be the well known field of real numbers or some finite field (see last section). In order to obtain a curve that is useful for cryptography, the parameters have to be chosen in a way that the following conditions hold

$$\frac{\partial F}{\partial x_1} \neq 0, \quad \frac{\partial F}{\partial x_2} \neq 0, \quad \frac{\partial F}{\partial x_3} \neq 0.$$

We identify points on the curve that can be derived from each other by multiplying each component with some scalar. This makes sense since (x_1, x_2, x_3) solves (7.1) if and only if $\alpha(x_1, x_2, x_3)$ ($\alpha \neq 0$) does. Formally, this means that we consider classes of equivalent points instead of single points, where points are called equivalent if one is the scalar multiple of the other one. If we put $x_3 = 0$ in the basic equation (7.1), then this equation collapses to $-x_1^3 = 0$, leading to $x_1 = 0$. Thus, the equivalence class which includes the element $(0, 1, 0)$ is the only one that contains a point with $x_3 = 0$. For all points on the curve that are not equivalent to $(0, 1, 0)$, we may apply the following transformation

$$K \times K \times (K \setminus \{0\}) \ni (x_1, x_2, x_3) \mapsto (x, y) := \left(\frac{x_1}{x_3}, \frac{x_2}{x_3} \right) \in K \times K,$$

which reduces the number of variables to two instead of three. We note that the basic equation (7.1) $F(x_1, x_2, x_3) = 0$ was chosen in a way that this transformation leads to the famous so-called Weierstrass-Equation¹¹ holds

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6. \quad (7.2)$$

Since all but one point (i.e. equivalence class) of the elliptic curve can be described using equation (7.2), this equation is often called the elliptic equation, and its solutions written as

$$\mathbf{E} = \{(x, y) \in K \times K \mid y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\} \cup \{\mathcal{O}\}.$$

Here, \mathcal{O} represents the point $(0, 1, 0)$ that is loosely speaking mapped to infinity by the transformation (division by x_3) that reduces the three variables to two.

In contrast to figure 7.3 only finite fields $K = GF(p^n)$ are used in elliptic curve cryptography. The reason is loosely speaking that in modern communication engineering data processing is always based on discrete data (simply because computers accept only discrete data).

For practical reasons, it turned out to be useful to take either $GF(p)$ with a large prime p or $GF(2^n)$ with a (large) positive integer n . Using $GF(p)$ has the advantage of providing a relatively simple arithmetic; on the other hand $GF(2^n)$ allows a binary representation of each element that supports the way computers work. Other fields like, for example, $GF(7^n)$ do not have any of these advantages and are, thus, not considered, although there is no mathematical reason why they should not.

A coordinate transformation can result in a simpler version¹² of the Weierstrass equation. Depending whether $p > 3$, different transformations are used, and we obtain

- in case of $GF(p)$, $p > 3$, the elliptic curve equation of the form

$$y^2 = x^3 + ax + b \quad (7.3)$$

with $4a^3 + 27b^2 \neq 0$

¹¹Karl Weierstrass, 31.10.1815–19.12.1897, German mathematician, famous for his rigorous formal approach to mathematics.

¹²Such a coordinate transformation is combination of a rotation and a dilatation of the coordinate system without changing the elliptic curve itself.

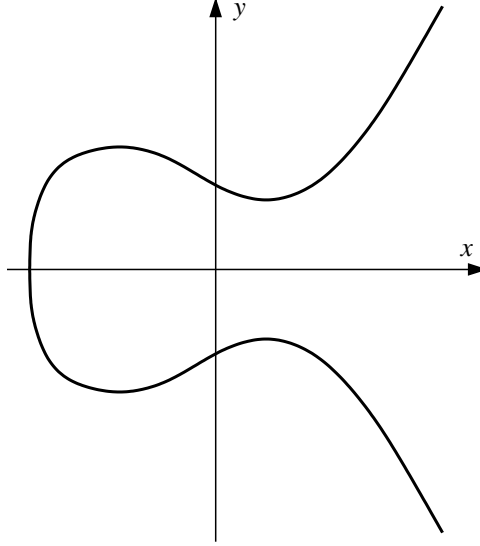


Figure 7.3: Example of an elliptic curve with the real numbers as underlying field.

- in case of $GF(2^n)$ the elliptic curve equation of the form

$$y^2 + xy = x^3 + ax^2 + b \quad (7.4)$$

with $b \neq 0$ ¹³.

This conditions on the parameters a, b ensure that the elliptic equation can be used in the context of cryptography¹⁴.

Let $|E|$ denote the number of elements of an elliptic curve E given an underlying field $GF(k)$ (for practical reasons either $k = p$ with p prim or $k = 2^n$). Then Hasse's theorem[Silverman1986] yields $||E| - k - 1| \leq 2 \cdot \sqrt{k}$. This Inequality is equivalent to $k + 1 - 2\sqrt{k} < |E| < k + 1 + 2\sqrt{k}$. In particular, this means that the number of elements of an elliptic curve is approximately k (for large k).

7.5 Operating on the elliptic curve

In order to work with elliptic curves in practice, we define an operation (often written in an additive way $+$) on the set of points on the curve. If we have a curve over the field $GF(p)$, we define the commutative operation $+$ by

1. $P + \mathcal{O} = \mathcal{O} + P = P$ for all $P \in E$,
2. for $P = (x, y)$ and $Q = (x, -y)$ we set $P + Q = \mathcal{O}$,
3. for $P_1 = (x_1, y_1), P_2 = (x_2, y_2) \in E$ with $P_1, P_2 \neq \mathcal{O}$ and $(x_2, y_2) \neq (x_1, -y_1)$ we set $P_3 := P_1 + P_2, P_3 = (x_3, y_3)$ defined by

$$x_3 := -x_1 - x_2 + \lambda^2, \quad y_3 := -y_1 + \lambda(x_1 - x_3)$$

¹³The form (7.3) is called the standard form of the Weierstrass-equation. If the characteristic of the field is 2 or 3, we obtain $4 = 0$ respectively $27 = 0$, which means that the condition on parameters a, b collapse. Loosely speaking, this is the reason why the transformation to the standard form does not work in these cases.

¹⁴Formally we call such curves non singular.

with the auxiliary quotient

$$\lambda := \begin{cases} \frac{y_1 - y_2}{x_1 - x_2} & \text{if } P_1 \neq P_2, \\ \frac{3x_1^2 + a}{2y_1} & \text{if } P_1 = P_2. \end{cases}$$

In particular, we obtain $-P = (x, -y)$ for $P = (x, y) \in E$.

If we deal with a curve over the field $GF(2^n)$, we define the operation $+$ in an analogous way by

1. $P + \mathcal{O} = \mathcal{O} + P = P$ for all $P \in E$,
2. for $P = (x, y)$ and $Q = (x, x + y)$ we set $P + Q = \mathcal{O}$,
3. for $P_1 = (x_1, x_2), P_2 = (x_2, y_2) \in E$ with $P_1, P_2 \neq \mathcal{O}$ and $(x_2, y_2) \neq (x_1, x_1 + y_1)$ we set $P_3 := P_1 + P_2, P_3 = (x_3, y_3)$ defined by

$$x_3 := -x_1 + x_2 + \lambda + \lambda^2 + a, \quad y_3 := y_1 + x_3 + \lambda(x_1 + x_3)$$

with auxiliary quotient

$$\lambda := \begin{cases} \frac{y_1 + y_2}{x_1 + x_2} & \text{if } P_1 \neq P_2, \\ x_1 + \frac{y_1}{x_1} & \text{if } P_1 = P_2. \end{cases}$$

In particular, we obtain $-P = (x, -y)$ for $P = (x, y) \in E$.

(Note that $-(-P) = (x, x + (x + y)) = (x, 2x + y) = (x, y)$, since the underlying field has characteristic 2.)¹⁵

One can verify that $+$ defines a group operation on the set $E \cap \{\mathcal{O}\}$. In particular this means that the sum of two points is again a point on the elliptic curve. How this operation works is geometrically visualized in the following section.

¹⁵ An animation of the addition of points on elliptic curves can be found on the Certicom homepage http://www.certicom.com/resources/ecc_tutorial/ecc_tutorial.html

How to add points on an elliptic curve

The following figures show how points on an elliptic curve over the field of real numbers are summed up using affine coordinates. We note that the point infinity \mathcal{O} cannot be shown in the affine plane.

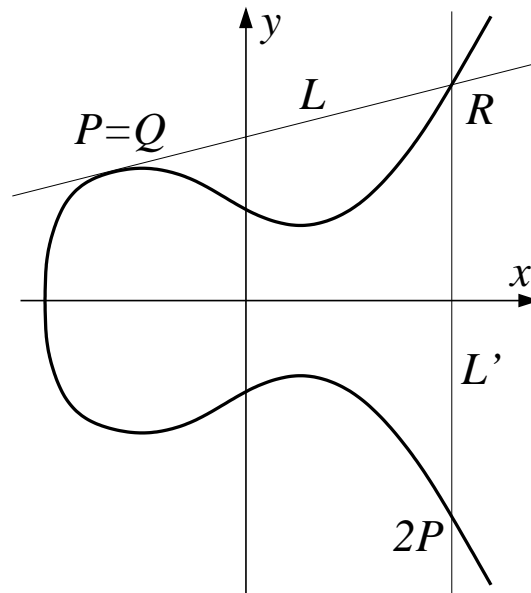


Figure 7.4: Doubling of a point

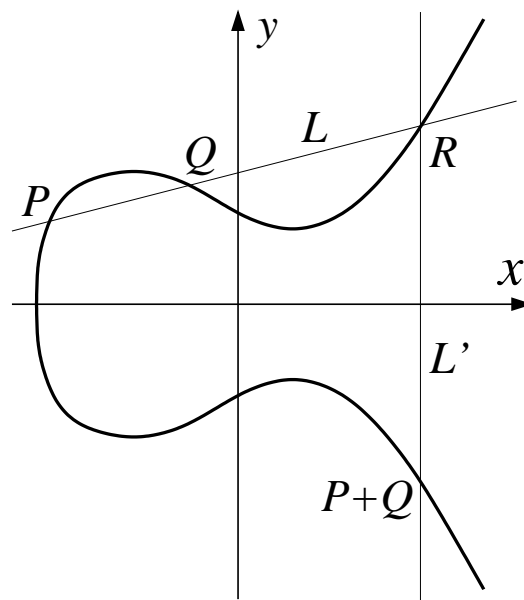


Figure 7.5: Summing up two different points over the real number field

7.6 Security of elliptic-curve-cryptography: The ECDLP

As mentioned above in section 7.4, we only consider elliptic curves over the finite¹⁶ fields $GF(2^n)$ or $GF(p)$ (for a large prime p). This means that all parameters that describe the curve are taken from this underlying field. If E is an elliptic curve over such a field and P is a point on the curve E , then we can derive for all positive integers m

$$mP := \underbrace{P + P + \dots + P}_{m \text{ times}}.$$

Looking on this operation from the cryptographic point of view, it turns out to be very interesting by the following reason: On the one hand one needs only $\log m$ operations to calculate mP — one simply has to calculate $P, 2P, 2^2P, 2^3P, \dots$, write m in a binary form and finally add all these multiples 2^kP of P with respect to the binary representation of m — on the other hand it seems to be very hard to find m given P and $Q = mP$ on E . Of course, we may simply calculate $P, 2P, 3P, 4P, 5P, \dots$ and compare each of them with Q . But this will take as much as m operations.

Yet there is no algorithm known that efficiently derives m given P and G . The best algorithms known so far need about \sqrt{q} operations where q is the (largest) prime factor of $p - 1$, in case the underlying field is $GF(p)$; here m should be between 1 and q liegen so that one needs at most $\log q$ operations to calculate mP . However, the quotient $\frac{\sqrt{q}}{\log q}$ tends to $+\infty$ very fast for large q .

If we choose the parameters sufficiently large (for example, let p be prime and at least 160 bits long), a computer will easily be able to calculate mP (in less than a second). The *inverse problem* however, to derive m from mP and P , can (still) not be solved in reasonable time.

This problem is known as the “Elliptic Curve Discrete Logarithm Problem” (for short ECDLP).

In elliptic curve cryptography we formally look at points on the elliptic curve as elements of a group with point addition $+$ as operation. Furthermore, we use only elliptic curves that have a sufficiently large number of points. However, in special cases curves may be weak and not useful due to other reasons. For such special cases the ECDLP can be much easier to solve than in the general case. This means that one has to look carefully at the parameters when choosing an elliptic curve for cryptographic applications.

Not useful for cryptography are *a-normal* (that are curves over \mathbb{Z}_p , for which the set \mathbf{E} consists of exactly p elements) and *supersingular* curves (that are curves, for which the ECDLP can be reduced to the “normal” discrete logarithms in another, smaller finite field). This means that there are cryptographically useful and non-useful elliptic curves. Given the parameters a and b , it is possible to determine whether a curve is useful or not. In many publications one can find parameters that turned out to be useful for cryptography. The open (scientific) discussion guarantees that these results take into account latest research.

Given a secure curve, the time that is needed to solve the ECDLP is strongly correlated with parameter p in case $GF(p)$ respectively n in case of $GF(2^n)$. The larger these parameters become, the more time an attacker needs to solve the ECDLP — at least with the best algorithms known so far. Experts recommend bit-lengths of 200 for p for secure curves. A comparison with RSA modulus length shows why elliptic curves are so interesting for applications. We note that the computation effort for signing and encryption is closely related to the bit-length of the

¹⁶Discrete in contrast to continuous.

parameters. In addition the initiation process, i.e. the generation of the private-public-key-pair, becomes more complicated the larger p is. Thus, one looks for the smallest parameters that still come along with the security required. It is remarkable that a length of 200 bits for p is sufficient to construct a *good* elliptic curve that is as secure as RSA with a 1024 bit RSA modulus (as far as we know today). For short, the reason for this advantage of ECC lies in the fact that the best algorithms known for solving the ECDLP need exponential time while the best algorithms for factorizing are sub-exponential (number field sieve, quadratic sieve or factorizing with elliptic curves). Hence, the parameters for a cryptosystem that is based on the problem of *factorizing large integers* have to be larger than the parameters for a system based on ECDLP.

7.7 Encryption and signing with elliptic curves

The *elliptic curve discrete logarithm problem* (ECDLP) is the basis for elliptic curve cryptography. Based on this problem, there are different signature schemes. In order to apply one of these, we need:

- An elliptic curve \mathbf{E} with an underlying field $GF(p^n)$.
- A prime $q \neq p$ and a point G on the elliptic curve \mathbf{E} with order q . This means that $qG = \mathcal{O}$ and $rG \neq \mathcal{O}$ for all $r \in \{1, 2, \dots, q-1\}$. Thus q is a factor of the group order (i.e. the number of elements) $\#\mathbf{E}$ of E . Since q is prime, G generates a cyclic sub-group of \mathbf{E} of order q .

The parameters mentioned are often called *Domain* parameter. They describe the elliptic curve \mathbf{E} and the cyclic sub-group of \mathbf{E} on which the signature scheme is based.

7.7.1 Encryption

Using elliptic curves one can construct a key exchange protocol based on the Diffie-Hellman protocol (see chapter 5.4.2). The key exchanged can be used for a subsequent symmetric encryption. We note that in contrast to RSA there is no pair of private and public key that can be used for encryption and decryption!

In the notation of elliptic curves, the Diffie-Hellman protocol reads as follows: First both partners (A und B) agree on a group G and an integer q . Then they choose $r_A, r_B \in \{1, 2, \dots, q-1\}$ at random, derive the points $R_A = r_A G$, $R_B = r_B G$ on the elliptic curve and exchange them (using an insecure channel). After that A easily obtains $R = r_A R_B$; B gets the same point ($R = r_A r_B G$) by calculating $r_B R_A = r_B r_A G = r_A r_B G = R$. We note that R_A, R_B are easy to derive as long as r_A respectively r_B are known G . However, the inverse operation, to get R_A respectively R_B from r_A respectively r_B is hard.

Using the best algorithms known so far, it is impossible for any attacker to obtain R without knowing either r_A or r_B — otherwise he would have to solve the ECDLP.

In order to prohibit a “Man-in-the-middle” attack, one may sign the values G, q, R_A, R_B as described in chapter 6.4.1.

7.7.2 Signing

Using the DSA signature scheme, one can proceed as follows: The signing party chooses a (non-trivial) number $s \in \mathbb{Z}_q$, which will be the private key, and publishes q , G and $R = sG$. We note that s cannot be obtained from G and R are not sufficient — a fact on which the security of the signature scheme is based.

Given the message m , which should be signed, one first constructs a digital finger print using a hash-algorithm h such that $h(m)$ has its values in $\{0, 1, 2, \dots, q-1\}$. Thus, $h(m)$ can be considered as an Element of \mathbb{Z}_q . Then the signing party chooses a random number $r \in \mathbb{Z}_q$ and derives $R = (r_1, r_2) = rG$. We note that the first component r_1 of R is an element of $GF(p^n)$. This component will then be projected onto \mathbb{Z}_q , i.e. in case of $n = 1$ it is interpreted as the remainder of an element of $\{0, 1, \dots, p-1\}$ divided by q . This projection of r_1 onto \mathbb{Z}_q is denoted by \bar{r}_1 . Then one determines $x \in \mathbb{Z}_q$ such that

$$rx - s\bar{r}_1 - h(m) = 0.$$

The triple (m, r_1, x) is then published as the digital signature of message m .

7.7.3 Signature verification

In order to verify a signature, one has to build $u_1 = h(m)/x$, $u_2 = \bar{r}_1/x$ (in \mathbb{Z}_q and derive

$$V = u_1G + u_2Q.$$

Since we have $Q = sG$, the point $V = (v_1, v_2)$ satisfies $v_1 = u_1 + u_2s$. We note that this operations take place in the field $GF(p^n)$. The projection of $GF(p^n)$ on \mathbb{Z}_q mentioned above should be chosen in such a way that $\bar{v}_1 = u_1 + u_2s$ is an element of \mathbb{Z}_q . Then it follows that

$$\bar{v}_1 = u_1 + u_2s = h(m)/x + \bar{r}_1s/x = (h(m) + \bar{r}_1s)/x = rx/x = r.$$

Since $R = rG$, we obtain $\bar{v}_1 = \bar{r}_1$, i.e. R and V coincide modulo the projection onto \mathbb{Z}_q .

7.8 Factorization using elliptic curves

There are factorization¹⁷ algorithms based on elliptic curves¹⁸. More precisely, these procedures exploit the fact that elliptic curves can be defined over \mathbb{Z}_n (n composite number). Elliptic curves over \mathbb{Z}_n do not form a group, because not every point on such an elliptic curve has an inverse point. This is connected with the fact that - if n is a composite number - there exist elements in \mathbb{Z}_n that do not have an inverse with respect to multiplication mod n . In order to add two points on an elliptic curve over \mathbb{Z}_n , we can calculate in the same way as on elliptic curves over \mathbb{Z}_p . Addition of two points (on an elliptic curve over \mathbb{Z}_n), however, fails if and only if a factor of n has been found. The reason for this is that the procedure for adding points on elliptic curves gives elements in \mathbb{Z}_n and calculates the inverse elements for these (with respect to multiplication

¹⁷Especially John M. Pollard was involved in the development of many different factorization algorithms; also at factorization with ECC he was one of the leading heads. As an employee of British Telekom he never published much. At the RSA data Security Conference in 1999 he was awarded for his “outstanding contributions in mathematics”: http://www.eff.org/Privacy/Crypto/misc/DESCracker/HTML/19990118_rsa_awards.html.

¹⁸In 1987 H.W. Lenstra published a factorization algorithm, based on elliptic curves (see [Lenstra1987]). The biggest compound number currently factorized with elliptic curves is the number $628^{59} - 1$, which has 55 decimal digits. It was found Oct. 6th, 2001 by M. Izumi (See ECMNET).

mod n) in \mathbb{Z}_n . The extended Euclidean algorithm is used here. If the addition of two points (that lie on an elliptic curve over \mathbb{Z}_n) gives an element in \mathbb{Z}_n that does not have an inverse element in \mathbb{Z}_n , then the extended Euclidean algorithm delivers a genuine factor of n .

Factorization using elliptic curves thus principally works as follows: Random curves over \mathbb{Z}_n are selected, as well as random points (that lie on this curve) and add them; you thus obtain points that also lie on the curve or find a factor of n . Factorization algorithms based on elliptic curves therefore work probabilistically. The opportunity of defining large number of elliptic curves over \mathbb{Z}_n allows you to increase the probability of finding two points which you can add to obtain a factor of n . These procedures are therefore highly suitable for parallelization.

7.9 Implementing elliptic curves for educational purposes

There are not many free programmes offering ECC under a graphical user interface. The following subsections explain which according functionality is available in CrypTool and in Sage.

7.9.1 CrypTool

CrypTool offers elliptic curves for the digital signature function¹⁹ and for ECC-AES hybrid encryption²⁰.

It implements the basic algorithms for group operations, for generating elliptic curves, for importing and exporting parameters for elliptic curves over finite fields with p elements (p prime). The algorithms have been implemented in ANSI C and comply with draft no. 8 of the IEEE P1363 work group *Standard Specifications for Public Key Cryptography*

<http://grouper.ieee.org/groups/1363>.

The procedure implements the cryptographic primitives for generating and verifying signatures for the variations of Nyberg-Rueppel signatures and DSA signatures based on elliptic curves.

7.9.2 Sage

In Sage elliptic curves are described at

http://www.sagemath.org/doc/constructions/elliptic_curves.html²¹.

Additionally there is an exhaustive, interactive ECC tutorial by Maike Massierer. This interactive introduction to Elliptic Curve Cryptography is built up as a Sage notebook.

Sage notebooks are running after a logon within a browser^{22,23}.

¹⁹The dialog box, which appears in CrypTool after clicking the menu **Digital Signatures/PKI \ Sign Message**, offers the EC methods ECSP-DSA and ECSP-NR.

²⁰Within CrypTool you can find this technique using the menu path **Crypt \ Hybrid**.

²¹According Sage samples can be found at the "Published Worksheets" at <http://www.sagenb.org/pub/>
- about Elliptic Curve: <http://www.sagenb.org/home/pub/606/>
- about Elliptic Curve El Gamal: <http://www.sagenb.org/home/pub/104/>, or at
- the "Elliptic Curve Cryptography (ECC) Tutorial"
<http://www.williamstein.org/simuw06/notes/notes/node12.html>

²²If you installed Sage on your own (Unix) server, you first have to enter at the command line the command `notebook()`.

²³The ECC notebook of Maike Massierer needs the KASH3 library: Therefore e.g. with Sage 4.2.1 the package "kash3-2008-07-31.spkg" has to be installed (command `sage -i`).

The ECC notebook of Massierer^{24,25} consists of 8 parts (title page plus 7 chapters) and aims to let even beginners understand what elliptic curves are:

0. ECC Notebook (title page and contents)
1. Introduction and Overview
2. Motivation for the use of Elliptic Curves in Cryptography
3. Elliptic Curves in Cryptography
4. Cryptographic Protocols in ECC
5. Domain Parameter Generation for ECC Systems
6. Conclusion and Further Topics
7. References

²⁴Instructions to use an interactive Sage notebook:

- Public Sage servers like <http://sage.mathematik.uni-siegen.de:8000> or <http://www.sagenb.org/> often offer running samples as "Published Worksheets", which you can run and download without login on. These worksheets are listed if you click on "Published" in the above right corner.
- Worksheets using the **interact** command currently need some additional todos for a user to work correctly: sign-in, make a copy and execute all commands again.
This works as follows (described for the sagenb server and for the ECC tutorial):
- Sign-up for a Sage notebook account at <http://sagenb.org/register> and sign in at <http://sagenb.org/>.
- Open the worksheet <http://sagenb.org/home/pub/1126/>. This contains the table of contents of the interactive ECC notebook. From here you can navigate via a click to the different chapters of the document.
- In the very top left corner, click **Edit a copy** in order to create your own copy of the worksheet.
- Sometimes it's necessary at the beginning to re-evaluate the worksheet. Click in the left upper corner on **Action -> Evaluate all**.
- Some of the applications still do not always work after opening a worksheet. Instead of nice output, they show lots of (blue) error messages. This normally can be solved quickly by clicking the gray "%hide" string: Then you get the code behind the graphics. Simply generate the graphics again with Shift-Enter.
Even after doing this, the graphics code does not always disappear. Instead, it sometimes turns gray. Should this happen, click on the gray text, then click somewhere outside of the text box. The code will then disappear and leave you with a nice layout of the worksheet.
- Some of the ECC tutorial's content uses a special math fonts that are not installed by default with most browsers. When you notice that formulas are not displayed correctly or even get an error message about missing fonts from your browser, you need to install the jsMath fonts for a better layout.
See <http://www.math.union.edu/~dpvc/jsMath/> and <http://pubpages.unh.edu/~jsh3/jsMath/>.
After installing these fonts you can see the jsMath symbol at the lower border of your browser. If you click this symbol you can find the download page for the TIF fonts. This fonts installation has to be done at every PC.
- According to the Sage-support newsgroup there is work underway to create a system for using **@interact** completely outside of the Sage notebook (JS code within a static html pages).

²⁵Since 2008 this ECC notebook can be found at <http://sage.mathematik.uni-siegen.de:8000/home/pub/45/> (#45 to #52). To logon in Siegen you have to allow port 8000 and Cookies.
Since 2009 an updated version of this ECC notebook can be found at <http://sagenb.org/home/pub/1126/> (#1126 to #1133).

7.10 Patent aspects

If the field $GF(2^n)$ is used instead of the prime field $GF(p)$, one has to make substantial changes in the implementation. The advantage of $GF(2^n)$ lies in the fact that calculations in $GF(2^n)$ can be implemented very efficiently using the binary representation. In particular, divisions are much easier to process compared to $GF(p)$ (this is particularly important in the signature scheme mentioned above where a division is needed for processing a signature as well as for the verification).

In order to achieve maximal gain in efficiency, one may choose a field that allows special basis like polynomial basis (useful for software implementations) or normal basis (best for hardware implementations). For special n (like, for example, $n = 163, 179, 181$) one may even combine both advantages. However, they are still non-standard.

Sometimes only the first component and one additional bit is used as representation of a point on the elliptic curve instead of the full two components. Since the first component together with the additional bit is sufficient to derive the full point, this representation minimizes the memory capacity needed. In particular, for normal basis this point compression can be implemented efficiently. In addition, the cryptographic protocols themselves become more effective. A disadvantage is, however, that *point compression* can be used for about half of all elliptic curves only and is protected under US patent (US Patent 6141420, Certicon), causing additional costs. In the general case $GF(p^n)$ (and also in case $n = 1$) often so called affine or projective co-ordinates are used. Depending on the application, these co-ordinates may result in a gain in efficiency as well.

A comprehensive description of all implementations and their advantages and disadvantages would go far beyond the scope of this paper. We only want to state that there is a variety of possible implementations for elliptic curve cryptography, much more than for RSA. Therefore, there are serious efforts to reduce this large to a small number of standard implementations. Some standardization committees even try to reduce the complexity by focusing on a small number of (prescribed) curves (ASC-approach).

Today it is still not clear whether these standardization initiatives will be successful or not. However, without agreed standards, ECC is not likely to become a real alternative for RSA. The committees might be forced to act fast if there was a break-through in factorization.

7.11 Elliptic curves in use

Today elliptic curve cryptography is already in use. A prominent example is the information network Bonn-Berlin²⁶, used for the exchange of strictly confidential documents between different German federal governmental institutions in Berlin and Bonn. With the help of ECC a high security solution could be realized. Interoperability, however, played only a minor role.

In Austria ECC has been massively launched: A bank card with digital signature function.

Both examples show the typical range of application for elliptic curve cryptography: For high security solutions and for implementations on smartcards in which the key length is crucial (because of physical memory available).

²⁶The Informationsverbund Bonn-Berlin (IVBB) connects governmental institutions in the old and new German capital.
http://www.cio.bund.de/cIn_094/sid_92C19118CBA5A021AFD1ABAEC15D2B77/DE/IT-Angebot/IT-Infrastrukturen/IVBB/ivbb_inhalt.html

Bibliography

- [Cassels1991] J. W. S. Cassels,
Lectures on elliptic curves,
Cambridge University Press, 1991, 143 pages.
- [Koblitz1984] N. Koblitz,
Introduction to elliptic curves and modular forms,
Graduate Texts in Mathematics, Springer-Verlag, 1984.
- [Koblitz1998] N. Koblitz,
Algebraic aspects of Cryptography. With an appendix on Hyperelliptic curves by Alfred J. Menezes, Yi Hong Wu and Robert J. Zuccherato,
Springer-Verlag, 1998, 206 pages.
- [Lenstra1987] H.W. Lenstra,
Factoring integers with elliptic curves,
Annals of Mathematics 126, pp. 649-673, 1987.
- [Lenstra1999] Arjen K. Lenstra, Eric R. Verheul
Selecting Cryptographic Key Sizes (1999),
Journal of Cryptology: the journal of the International Association for Cryptologic Research
<http://www.cryptosavvy.com/cryptosizes.pdf>
- [Menezes1993] A. J. Menezes,
Elliptic curve public key cryptosystems,
Kluwer Academic Publishers, 1993.
- [Silverman1986] J. Silverman,
The Arithmetic of Elliptic Curves,
Springer-Verlag, 1986.
- [Silverman1992] J. Silverman,
The arithmetic of elliptic curves,
Graduate Texts in Mathematics, Springer-Verlag, 1992.
- [SilvermanTate1992] J. Silverman, J. Tate,
Rational points on elliptic curves,
Springer-Verlag, 1992.

Web links

1. Interactive introduction to elliptic curves and elliptic curve cryptography with Sage by
Maike Massierer and the CrypTool team,
<http://sagenb.org/home/pub/1126/> (#1126 to #1133)
ECC Tutorial as Sage Notebook
Version 1.2, November 2009
2. Certicom Online Tutorial,
http://www.certicom.com/resources/ecc_tutorial/ecc_tutorial.html
3. Working group IEEE P1363,
<http://grouper.ieee.org/groups/1363>
4. An informative web page about factorization with elliptic curves,
<http://www.loria.fr/~zimmerma/records/ecmnet.html>
It contains literature related to the topic factorization with elliptic curves as well as links
to other web page.
5. Key length comparison by Arjen Lenstra and Eric Verheul,
<http://cryptosavvy.com/table.htm>

Chapter 8

Crypto 2020 — Perspectives for Long-Term Cryptographic Security

(Johannes Buchmann, Erik Dahmen, Alexander May and Ulrich Vollmer, TU Darmstadt, May 2007)

Cryptography is a basic building block of all IT security solutions. Yet, for how long are the cryptographic tools we use today going to remain secure? Is this time long enough to ensure the confidentiality of medical data, to name just one example? Even in the short-term, the potential for havoc is great if certain keys are broken. Just think of the digital signatures that protect the authenticity of automatic updates for the Windows operating system.

8.1 Widely used schemes

In 1978, Rivest, Shamir and Adleman suggested the RSA public key encryption and signature schemes [7]. RSA is still the most widely used public key scheme. The security of RSA depends on the difficulty of factoring so-called RSA moduli which are products of two large prime numbers. In their 1978 paper, the inventors of RSA suggested the use of RSA moduli with 200 decimal digits for long-term security. Later, the company RSA Security published a list of RSA moduli of increasing size, the RSA challenge numbers. RSA Security offered prizes totaling \$ 635,000 for the factorization of these numbers, cf. www.rsasecurity.com/rsalabs/.

In 2005, that is 27 years after the invention of RSA, Bahr, Boehm, Franke, and Kleinjung from Bochum University managed to factor a 200 digit RSA challenge number (www.mat.uniroma2.it/~eal/rsa640.txt). A key with size originally thought to be secure for a very long time was broken with a computation that took them just five months. This illustrates the tremendous progress factoring technology has made within the last 30 years. This progress is based on break-through mathematical ideas — e.g. the number field sieve proposed by John Pollard — as well as significant developments in computer hardware and software implementation technology.¹

In 2000, Lenstra and Verheul developed an extrapolation formula that is supposed to help us forecast the security one can achieve with RSA and other important cryptographic schemes in the long term (www.keylength.com). The formula suggests the use of 850 digit RSA moduli

¹Please compare chapter 4.11 Considerations regarding the security of the RSA algorithm, and especially chapters 4.11.4 and 4.11.5.

if one wishes to protect data for the next 30 years. This corresponds to a 3072 bit RSA key.

Yet, even a well thought out extrapolation formula is no security guarantee! At any time, a brilliant mathematical idea can allow us to factor large numbers easily, and destroy the security of RSA. In 1996, Peter Shor showed that a quantum computer — a new type of computer that leverages the laws of quantum mechanics to speed up certain types of computations — can in principle be used for the fast factorization of large numbers [8]. Despite intensive research in the area, it is still too early to judge whether we are ever going to be able to build quantum computers of sufficient capacity to apply Shor’s algorithm to numbers of relevant size.² Recent announcements of significant progress in this area made by the start-up company D-Wave (www.dwavesys.com) have been greeted with a lot of scepticism, even ridicule.

The development of attacks on another frequently used scheme called DSA (Digital Signature Algorithm) and the Elliptic Curve Cryptography (ECC) class of schemes moves in analogy to those on RSA. The security of these schemes depends on the difficulty of computing discrete logarithms. Even today, there is significant algorithmic progress. Quantum computers would render these schemes insecure.

What’s the state of affairs with the so-called secret-key encryption schemes? In 1977, DES was introduced as the Data Encryption Standard [9]. Twenty-one years later, the Electronic Frontier Foundation (EFF) built the special purpose machine Deep Crack which needed just 56 hours to break a DES key. The problem with DES was that it used keys which were too short. It seems that the inventors of DES did not foresee the speed of hardware development. The Advanced Encryption Standard AES [6], successor to DES, is deemed secure at the moment even though there are interesting, if still inefficient, methods to attack AES with algebraic methods.

8.2 Preparation for tomorrow

Is the security of today’s cryptography measuring up to its increasing importance? The experience shows: Carefully designed and implemented cryptographic schemes have a life time of five to twenty years. Whoever uses RSA, ECC or AES for short-term protection of data may feel safe. Moreover, it is also possible to achieve long-term authenticity, integrity and non-reputability of data, e.g., using the multiple signature scheme suggested by Sönke Maseberg [3].

However, current schemes cannot guarantee long-term confidentiality. And what is to be done in twenty years from now? What should we do if, quasi over-night, unexpected mathematical progress renders an important cryptographic scheme insecure? Three things are necessary to prepare us for this event:

- a pool of secure alternative cryptographic schemes,

²Required qbits for attacks on RSA, DSA and ECDSA using key with a bit length n :

RSA	$2n + 3$
DSA	$2n + 3$
ECDSA 2^n	$\sim 2n + 8 \log n$
ECDSA p	$\sim 4n$

Please compare chapter 5.3 in “SicAri – Eine Sicherheitsplattform und deren Werkzeuge für die ubiquitäre Internetnutzung, KB2.1 – Abschlussbericht, Übersicht über Angriffe auf relevante kryptographische Verfahren”, version 1.0, Mai 17, 2005, Prof. Dr. Johannes Buchmann et al., TUD-KryptC and cv cryptovision GmbH (http://www.cdc.informatik.tu-darmstadt.de/~schepers/kb_21_angriffe.pdf) and the dissertation of Axel Schmidt at the same faculty.

- infrastructures that enable us to exchange one cryptographic scheme for another, easily and quickly, and
- methods that ensure long-term confidentiality.

For many years, the cryptography group at the Technische Universität Darmstadt and its spin-off, the company FlexSecure (www.flexsecure.de), have worked to provide these tools. The trust center software FlexiTrust which is employed by the German National Root Certification Authority and the German Country Signing Authority offers an infrastructure within which cryptographic schemes can be easily exchanged. The open source library FlexiProvider implements a multitude of cryptographic schemes. Lately, we have intensified our research into “Post Quantum Cryptography” seeking cryptographic schemes which remain secure even in the event that powerful quantum computers are built.

The security of public key cryptography traditionally rests on the difficulty of the solution of certain mathematical problems. Today, the following alternatives to the factorization and discrete logarithm problems are discussed in depth: the decoding problem, the shortest and closest vector problem in lattices, and the problem of solving large systems of multivariate quadratic equations. It is conjectured that quantum computers offer little advantage if we try to solve these problems efficiently.

8.3 New mathematical problems

Let us look at these alternatives a little more closely. The first encryption scheme based on the decoding problem was proposed by McEliece [4]. The background: Error-correcting codes are used to transmit or store electronic data in such a way that they remain undistorted even if a small number of bits are changed in transit or on the storage media. This property is used in, e.g., compact discs (CDs). The data on a CD can be reconstructed even if the disc has been slightly scratched.

In a code-based encryption scheme a message is encrypted by adding a fixed number of errors to (i.e. flipping a fixed numbers of bits of) the encoded message. Decoding requires knowledge of a suitable decoding procedure which eliminates these errors efficiently. This method is the secret key. Code-based encryption is in general very efficient. At the moment, research focus on the question which codes lead to secure encryption schemes with keys which are as small as possible.

Encryption on the basis of lattice problems is very similar to that on the basis of error-correcting codes. Lattices are regular structures of points in space. For instance, the points where the lines on squared paper cross form a two-dimensional lattice. For cryptographic usage, the dimension of the lattices is chosen to be much larger. Encryption works as follows: The plain-text is used to construct a lattice point which is then slightly distorted in such a way that it is no longer a lattice point, but close to one. Whoever knows a secret about the lattice is able to find this lattice point in the vicinity of the given point in space. The lattice point in turn yields the plain text. A particularly efficient lattice based encryption scheme is NTRU Encrypt (www.ntru.com). However, because NTRU was introduced fairly recently (in 1998), and its specification underwent several changes due to a variety of attacks, more cryptanalytic scrutiny is required to achieve confidence in its security.

8.4 New signatures

In 1979, Ralph Merkle proposed a remarkable framework for new signature schemes in his PhD thesis [5]. Contrary to all other signature schemes, its security does not rest on the difficulty of a number-theoretic, algebraic or geometric problem. The only thing it requires is something which other signature schemes need anyway: a cryptographically secure hash function and a secure pseudo-random number generator. Each new hash function leads to a new signature algorithm. In consequence, the Merkle scheme has the potential to solve the problem of long-term availability of digital signature schemes.

Merkle uses in his construction so-called One-Time Signatures: Each new signature requires a new signing key and a new verification key. The idea Merkle had was to reduce the validity of many verification keys using a hash tree to the validity of a unique public hash value. When generating keys for the Merkle scheme one has to determine the number of signatures one can make with it in advance. For a long time this seemed a significant disadvantage. In [2], however, a variant of Merkle's scheme was proposed which allows to compute 2^{40} signatures with a single key pair.

8.5 Quantum cryptography – a way out of the impasse?

From the point of view of today's state of the art of cryptography, the problem of long-term confidentiality remains unsolved: There is *no* practical method to protect the confidentiality of an encrypted message over a very long period of time.

One way out of that dilemma may be to employ quantum cryptography: it allows for key agreement schemes (of very long keys for one-time pads) whose security is guaranteed by the laws of quantum mechanics, cf., e.g., [1]. At the moment, however, quantum cryptography is still rather inefficient, and it is unclear which cryptographic functionalities can be implemented on top of it.

8.6 Conclusion

What's on the balance sheet of today's cryptography? We have good tools to ensure short and medium term security. Software developers can employ these tools in their applications with good conscience as long as they make sure that components can quickly be exchanged when they become insecure.

In order to guarantee IT security for the future, too, we need to prepare a portfolio of secure cryptographic schemes. This portfolio needs to contain schemes which are suitable for the world of ubiquitous computing with many less powerful computers. It also needs to contain schemes which remain secure in the event that powerful quantum computers are built. Several promising candidates have been discussed in this article. They need to be studied carefully and prepared for use in everyday scenarios. The question how to ensure long-term confidentiality remains an important open research problem upon which cryptographic research should focus.

Bibliography

- [1] Charles H. Bennett and Gilles Brassard. An update on quantum cryptography. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO '84*, volume 196 of *Lecture Notes in Computer Science*, pages 475–480. Springer-Verlag, 1985.
- [2] Johannes Buchmann, Luis Carlos Coronado García, Erik Dahmen, Martin Döring, and Elena Klintsevich. CMSS – an improved Merkle signature scheme. In Rana Barua and Tanja Lange, editors, *7th International Conference on Cryptology in India - Indocrypt'06*, number 4392 in *Lecture Notes in Computer Science*, pages 349–363. Springer-Verlag, 2006.
- [3] Jan Sönke Maseberg. *Fail-Safe-Konzept für Public-Key-Infrastrukturen*. PhD thesis, TU Darmstadt, 2002.
- [4] Robert J. McEliece. A public key cryptosystem based on algebraic coding theory. *DSN progress report*, 42–44:114–116, 1978.
- [5] Ralph C. Merkle. *Secrecy, authentication, and public key systems*. PhD thesis, Department of Electrical Engineering, Stanford University, 1979.
- [6] National Institute of Standards and Technology (NIST). *Federal Information Processing Standards Publication 197: Advanced Encryption Standard*, 2002.
- [7] Ron Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
- [8] Peter W. Shor. Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.
- [9] U.S. Department of Commerce, National Bureau of Standards, National Technical Information Service, Springfield, Virginia. *Federal Information Processing Standards Publication 46: Data Encryption Standard*, 1977.

Appendix A

Appendix

- 1 CrypTool Menu Tree
- 2 Authors of the CrypTool Script
- 3 Bibliography of Movies and Fictional Literature with Relation to Cryptography, Books for Kids with Collections of Simple Ciphers
- 4 Learning Tool for Elementary Number Theory
- 5 Using Sage with this Script

A.1 CrypTool Menus

This appendix contains at the following page the complete menu tree of CrypTool version 1.4.30¹.

The main menu contains service functions in the menus

- File
- Edit
- View
- Options
- Window
- Help

and the actual crypto functions in the menus

- Crypt/Decrypt
- Digital Signature/PKI
- Individual Procedures
- Analysis.

Below **Individual Procedures** you find visualizations of single algorithms and of protocols. Some procedures are implemented both for a fast performance (mostly under the main menu **Crypt/Decrypt**) and for a step-by-step visualization.

Which menu items in CrypTool are active (that means not greyed), depends on the type of the currently active document window: The brute-force analysis for DES e. g. is only available, if the active window is opened in the hexadecimal view. On the other hand the menu item “Generate Random Numbers...” is always available (even if no document is opened).

¹In parallel to CrypTool 1.x the future versions CrypTool 2 and JCrypTool are currently developed in the CrypTool project.

- Webseite CT2: <http://www.cryptool2.vs.uni-due.de>

- Webseite JCT: <http://jcryptool.sourceforge.net>

These future versions are currently (December 2009) beta versions, but they are stable enough to be used by end-users already. If the according release versions are available, we will add the appropriate menus and menu trees.

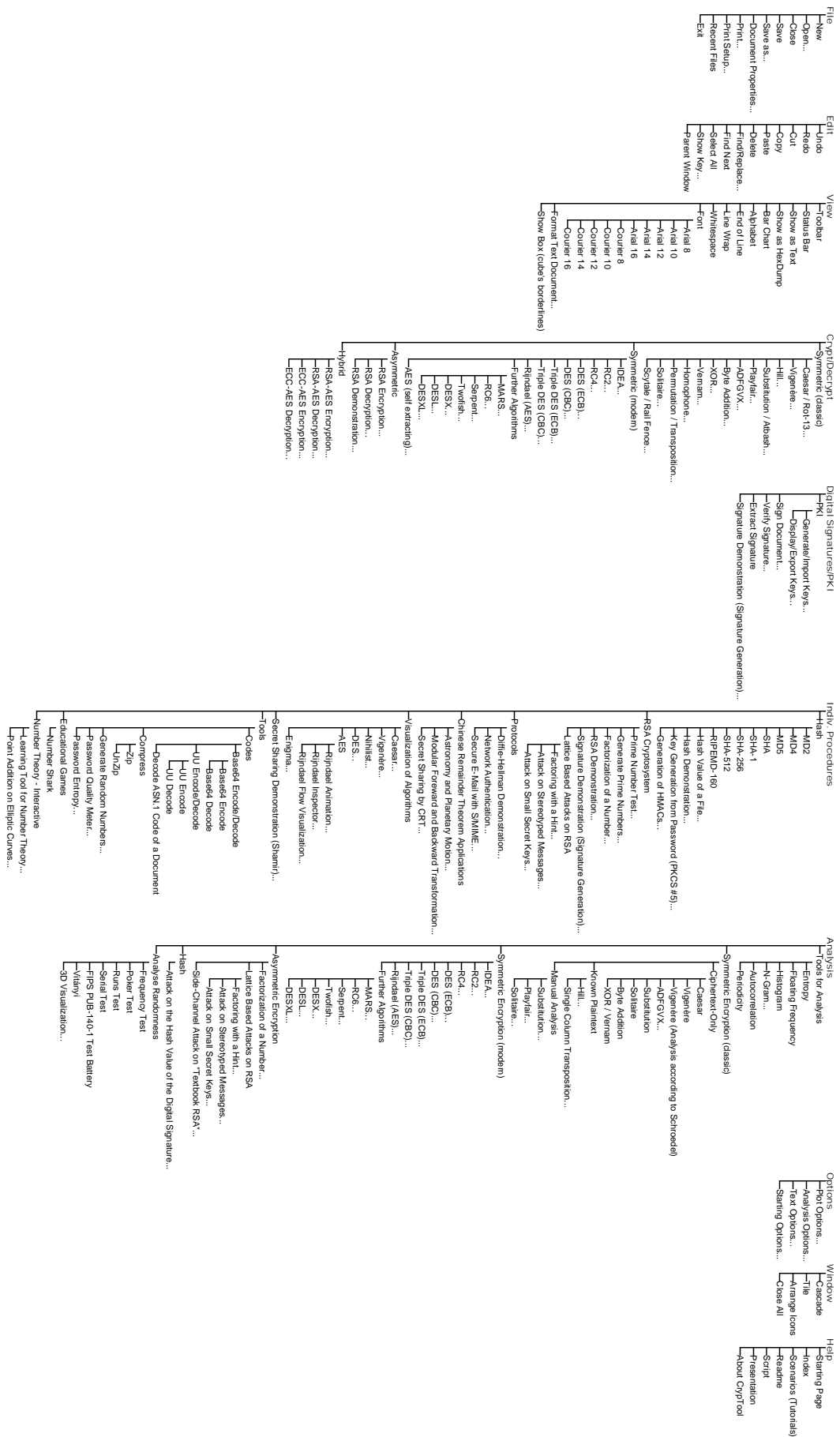


Figure A.1: Complete overview of the menu tree of CrypTool 1.4.30

A.2 Authors of the CrypTool Script

This appendix lists the authors of this document.

Please refer to the top of each individual chapter for their contribution.

Bernhard Esslinger,

initiator of the CrypTool project, main author of this script, head Information Security Management at Deutsche Bank and professor for IT security and cryptography at the University of Siegen. E-mail: besslinger@web.de, esslinger@fb5.uni-siegen.de.

Matthias Büger,

contributor to chapter 7 (“Elliptic Curves”), research analyst at Deutsche Bank.

Bartol Filipovic,

original author of the CrypTool elliptic curve implementation and the corresponding chapter in this script.

Henrik Koy,

main developer and co-ordinator of CrypTool development since version 1.3; script reviewer and T_EX guru; cryptographer and project leader IT at Deutsche Bank.

Roger Oyono,

implementer of the CrypTool factorization dialog and original author of chapter 5 (“The Mathematical Ideas behind Modern Cryptography”).

Jörg Cornelius Schneider,

design and support of CrypTool; crypto enthusiast and IT architect and senior project leader IT at Deutsche Bank.

Christine Stötzel,

Master of Business and Computer Science at the University of Siegen.

Johannes Buchmann,

Co-author of chapter 8 (“Crypto 2020 — Perspectives for Long-Term Cryptographic Security”). Johannes Buchmann holds the Chair for Theoretical Computer Science (Cryptography and Computer Algebra) at the department of Computer Science of the Technische Universität Darmstadt TUD). He is also a Professor at the department of Mathematics, and vice-president of the university.

Alexander May,

Co-author of chapter 8 (“Crypto 2020 — Perspectives for Long-Term Cryptographic Security”), Assistant Professor at the department of Computer Science of the Technische Universität Darmstadt, Germany.

Erik Dahmen,

Co-author of chapter 8 (“Crypto 2020 — Perspectives for Long-Term Cryptographic Security”), Researcher at the Chair for Theoretical Computer Science (Cryptography and Computer Algebra), department of Computer Science, Technische Universität Darmstadt, Germany.

Ulrich Vollmer,

Co-author of chapter 8 (“Crypto 2020 — Perspectives for Long-Term Cryptographic Security”), Researcher at the Chair for Theoretical Computer Science (Cryptography and Computer Algebra), department of Computer Science, Technische Universität Darmstadt, Germany.

Minh Van Nguyen,

Sage developer and documentation quality reviewer.

A.3 Movies and Fictional Literature with Relation to Cryptography, Books for Kids with Simple Ciphers

Cryptographic applications – classical as well as modern ones – have been used in literature and movies. In some media they are only mentioned and are a pure admixture; in others they play a primary role and are explained in detail; and sometimes the purpose of the story, which forms the framework, is primarily to transport this knowledge and achieve better motivation. Here is the beginning of an overview.

A.3.1 For Grownups

[Poe1843] Edgar Allan Poe,
The Gold Bug, 1843.

In this short story Poe tells as first-person narrator about his acquaintanceship with the curious Mr. Legrand. They detect a fabulous treasure via a gold bug and a vellum found at the coast of New England.

The cipher consists of 203 cryptic symbols and it proves to be a general monoalphabetic substitution cipher (see chapter 2.2.1). The story tells how they solve the riddle step by step using a combination of semantic and syntax analysis (frequency analysis of single letters in English texts).

In this novel the code breaker Legrand says the famous statement: “Yet it may be roundly asserted that human ingenuity cannot concoct a cipher which human ingenuity cannot resolve – given the according dedication.”

[Verne1885] Jules Verne,
Mathias Sandorf, 1885.

This is one of the most famous novels of the French author Jules Verne (1828-1905), who was called “Father of Science fiction”.

In “Mathias Sandorf” he tells the story of the freedom fighter Earl Sandorf, who is betrayed to the police, but finally he can escape.

The whistle-blowing worked, because his enemies captured and decrypted a secret message sent to him. For decryption they needed a special grille, which they stole from him. This turning grille was a quadratic piece of jig with 6x6 squares, of which 1/4 (nine) were holes (see the turning grille in chapter 2.1.1).

[Kipling1901] Rudyard Kipling,
Kim, 1901.

Rob Slade’s review² of this novel says: “Kipling packed a great deal of information and concept into his stories, and in “Kim” we find The Great Game: espionage and spying. Within the first twenty pages we have authentication by something you have, denial of service, impersonation, stealth, masquerade, role-based authorization (with ad hoc authentication by something you know), eavesdropping, and trust based on data integrity. Later on we get contingency planning against theft and cryptography with key changes.” The book is out of copyright³.

²See <http://catless.ncl.ac.uk/Risks/24.49.html#subj12>.

³You can read it at:

<http://whitewolf.newcastle.edu.au/words/authors/K/KiplingRudyard/prose/Kim/index.html>,

[Doyle1905] Arthur Conan Doyle,

The Adventure of the Dancing Men, 1905.

In this Sherlock Holmes short story (first published in 1903 in the “Strand Magazine”, and then in 1905 in the collection “The Return of Sherlock Holmes” the first time in book-form) Sherlock Holmes has to solve a cipher which at first glance looks like a harmless kid’s picture.

But it proves to be the monoalphabetic substitution cipher (see chapter 2.2.1) of the criminal Abe Slaney. Sherlock Holmes solves the riddle using frequency analysis.

[Sayers1932] Dorothy L. Sayers,

Have his carcase, Harper/Victor Gollancz Ltd., 1932.

In this novel the writer Harriet Vane finds a dead body at the beach. The police believe the death is suicide. Harriet Vane and the elegant amateur sleuth Lord Peter Wimsey together clear of the disgusting murder in this second of Sayers’s famous Harriet Vane mystery series.

This requires to solve a cryptogram. Surprisingly the novel not only describes the Playfair cipher in detail, but also the cryptanalysis of this cipher (see Playfair in chapter 2.2.3).

[Arthur196x] Robert Arthur,

The Three Investigators: The Secret Key (German version: *Der geheime Schlüssel nach Alfred Hitchcock* (volume 119), Kosmos-Verlag (from 1960)

The three detectives Justus, Peter and Bob have to decrypt covered and encrypted messages within this story to find out what is behind the toys of the Copperfield company.

[Simmel1970] Johannes Mario Simmel,

And Jimmy went to the Rainbow (original title: *Und Jimmy ging zum Regenbogen*), Knaur Verlag, 1970.

The novel plays between 1938 and 1967 in Vienna. The main character Manuel Aranda uncovers step by step the past of his murdered father. Important for the plot is an encrypted manuscript, which is decrypted in chapter 33. In the novel the cipher is called “25-fold Caesar cipher”. It is actually a Vigenère cipher with a 25 character key.

A movie of the novel appeared in 1971.

[Crichton1987] Michael Crichton,

Sphere, Pan Books, 1987.

A team of different scientists is sent to the ground of the ocean in order to investigate a highly developed 900 m long space ship. The human peculiarities and psychological problems of the researchers surface more and more, because of life threatening events and isolation. There are many mysteries: While the space ship lies on the ground for 300 years, it has English markings and a life of its own, and materializing of the researcher’s imaginations appear. On a computer screen a cipher text appears, which is completely printed in the book. The genius mathematician Harry deciphers the simple helical substitution code.

[Seed1990] Directed by Paul Seed,
House of Cards, 1990.

In this movie Ruth tries to solve the secret, which made her daughter fall silent. Here two young people suffering from autism communicate via 5- and 6-digit primes (see chapter 3). After more than 1 hour the movie contains the following encrypted two series of primes:

21,383; 176,081; 18,199; 113,933; 150,377; 304,523; 113,933
193,877; 737,683; 117,881; 193,877

[Robinson1992] Directed by Phil Alden Robinson,
Sneakers, Universal Pictures Film, 1992.

In this movie the “sneakers”, computer experts under their boss Martin Bishop, try to get back the deciphering box SETEC from the “bad guys”. SETEC, invented by a genius mathematician before he was killed, allows to decrypt all codes from any nation. The code is not described in any way.

[Baldacci1997] David Baldacci,
Total Control, Mass Market Paperback, 1997.

Jason Archer, executive with a technology company suddenly disappears. Sidney Archer tries to find out about her husband’s surprising death. She gets a clue how the global financial system is abused and that the real control belongs to those with the most money. Here even good passwords don’t help ...

[Natali1997] Directed by Vincenzo Natali,
Cube, Mehra Meh Film, 1997.

In this Canadian low-budget-movie 7 complete strangers of widely varying personality characteristics are involuntarily placed in an kafkaesque maze of cubical rooms containing deadly traps.

To get out the persons have to move through these rooms. To find out which rooms are dangerous, mathematics is crucial: Each cubic room has at its entrance a numerical marking consisting of three sets of three digits. First they deduce that all rooms marked at their entrance with at least one prime number are trapped. Later it comes out that a trapped room can also be marked by a number which is a power of a prime (so traps are p^n , e.g. $128 = 2^7$ or $101 = 101^1 = \text{prime}$, but not $517 = 11 * 47$).

[Becker1998] Directed by Harold Becker,
Mercury Rising, Universal Pictures Film, 1998.

The NSA developed a new cipher, which is pretended to be uncrackable by humans and computers. To test its reliability some programmers hide a message encrypted with this cipher in a puzzle magazine.

Simon, a nine year old autistic boy, cracks the code. Instead of fixing the code, a government agent sends a killer. FBI agent Art Jeffries (Bruce Willis) protects the boy and sets a snare for the killers.

The code is not described in any way.

[Brown1998] Dan Brown,

Digital Fortress, E-Book, 1998.

Dan Brown's first novel was published in 1998 as e-book, but it was largely unsuccessful then.

The National Security Agency (NSA) uses a huge computer, which enables it to decrypt all messages (needless to say only of criminals and terrorists) within minutes even if they use the most modern encryption methods.

An apostate employee invents an unbreakable code and his computer program Diabolus forces the super computer to do self destructing operations. The plot, where also the beautiful computer expert Susan Fletcher has a role, is rather predictable.

The idea, that the NSA or another secret service is able to decrypt any code, is currently popular on several authors: In "Digital Fortress" the super computer has 3 million processors – nevertheless from today's sight this is by no means sufficient to hack modern ciphers.

[Elsner1999] Dr. C. Elsner,

The Dialogue of the Sisters, c't, Heise, 1999.

In this short story, which is included in the CrypTool package as PDF file, the sisters confidentially communicate using a variant of RSA (see chapter 4.10 and the following). They are residents of a madhouse being under permanent surveillance.

[Stephenson1999] Neal Stephenson,

Cryptonomicon, Harper, 1999.

This very thick novel deals with cryptography both in WW2 and today. The two heroes from the 40ies are the excellent mathematician and cryptanalyst Lawrence Waterhouse, and the overeager and morphine addicted US marine Bobby Shaftoe. They both are members of the special allied unit 2702, which tries to hack the enemy's communication codes and at the same time to hide the own existence.

This secretiveness also happens in the present plot, where the grandchildren of the war heroes – the dedicated programmer Randy Waterhouse and the beautiful Amy Shaftoe – team up.

Cryptonomicon is notably heavy for non-technical readers in parts. Several pages are spent explaining in detail some of the concepts behind cryptography. Stephenson added a detailed description of the Solitaire cipher (see chapter 2.4), a paper and pencil encryption algorithm developed by Bruce Schneier which is called "Pontifex" in the book. Another, modern algorithm called "Arethusa" is not explained in detail.

[Elsner2001] Dr. C. Elsner,

The Chinese Labyrinth, c't, Heise, 2001.

In this short story, which is included in the CrypTool package as PDF file, Marco Polo has to solve problems from number theory within a competition to become a major consultant of the Great Khan. All solutions are included and explained.

[Colfer2001] Eoin Colfer,

Artemis Fowl, Viking, 2001.

In this book for young people the 12 year old Artemis, a genius thief, gets a copy of the top secret "Book of the Elves". After he decrypted it with his computer, he finds out things, men never should have known.

The used code is not described in detail or revealed.

[Howard2001] Directed Ron Howard,

A Beautiful Mind, 2001.

This is the film version of Sylvia Nasar's biography of the game theorist John Nash. After the brilliant but asocial mathematician accepts secret work in cryptography, his life takes a turn to the nightmarish. His irresistible urge to solve problems becomes a danger for himself and his family. Nash is – within his belief – a most important hacker working for the government.

Details of his way analysing code are not described in any way.

[Apted2001] Directed by Michael Apted,

Enigma, 2001.

This is the film version of Robert Harris' "historical fiction" *Enigma* (Hutchinson, London, 1995) about the World War II code-breaking work at Bletchley Park in early 1943, when the actual inventor of the analysis Alan Turing (after Polish pre-work) already was in the US. So the fictional mathematician Tom Jericho is the lead character in this spy-thriller. Details of his way analysing the code are not described.

[Isau2003] Ralf Isau,

The Museum of the stolen memories (original title: Das Museum der gestohlenen Erinnerungen), Thienemann-Verlag, 2003.

In this exciting novel the last part of the oracle can only be solved with the joined help of the computer community.

[Brown2003] Dan Brown,

The Da Vinci Code, Doubleday, 2003.

The director of the Louvre is found murdered in his museum in front of a picture of Leonardo da Vinci. And the symbol researcher Robert Langdon is involved in a conspiracy. The plot mentions different classic codes (substitution like Caesar or Vigenère, as well as transposition and number codes). Also there are hints about Schneier and the sunflower. The second part of the book contains a lot of theological considerations.

This book has become one of the most widely read books of all time.

[Hill2003] Tobias Hill,

The Cryptographer, Faber & Faber, 2003.

London 2021: The company SoftMark developed and establish an electronic currency, which guarantees highest security standards by an unbreakable code. The inventor and company founder, called the cryptographer because of his mathematical talent, has become the richest man in the world. But the code was hacked, and in a worldwide economic crisis his company goes bankrupt. Additionally the tax investigator Anna Moore is set on him.

[McBain2004] Scott McBain,

Final Solution, manuscript not published by Harper Collins, 2004 (German version has been published in 2005).

In a near future politicians, chiefs of military and secret services of many different countries take over all the power. With a giant computer network called “Mother” and complete surveillance they want to cement their power and commercialisation of life forever. Humans are only assessed according to their credit rating and globally acting companies elude of any democratic control. Within the thriller the obvious injustice, but also the realistic likelihood of this development are considered again and again.

With the help of a cryptographer a code to destroy was built into the super computer “Mother”: In a race several people try to start the deactivation (Lars Pedersen, Oswald Plevy, the female American president, the British prime minister and an unknown Finish named Pia, who wants to take revenge for the death of her brother). On the opposite side a killing group acts under the special guidance of the British foreign minister and the boss of the CIA.

[**Burger2006**] Wolfgang Burger,

Heidelberg Lies (original title: Heidelberger Lügen), Piper, 2006.

This detective story playing in the Rhein-Neckar area in Germany has several independent strands and local stories, but mainly it is about Kriminalrat Gerlach from Heidelberg. On page 207 f. the cryptographic reference for one strand is shortly explained: The soldier Hörle had copied circuit diagrams of a new digital NATO decryption device and the murdered man had tried to sell his perceptions to China.

[**Vidal2006**] Agustin Sanchez Vidal,

Kryptum, Dtv, 2006.

The first novel of the Spanish professor of art history has some similarities with Dan Brown’s “The Da Vinci Code” from 2003, but allegedly Vidal started his writing of the novel already in 1996. Vidal’s novel is a mixture between historic adventure and mystery thriller. It was a huge success in Spain and Germany. There is currently no English version available.

In the year 1582 Raimundo Randa is waiting to be condemned to death – he was all life long trying to solve a mystery. This mystery is about a parchment with cryptic characters, where a unique power is behind. Around 400 years later the American scientist Sara Toledano is fascinated by this power until she vanishes in Antigua. Her colleague, the cryptographer David Calderon, and her daughter Rachel are searching for her and simultaneously they try to solve the code. But also secret organizations like the NSA chase after the secret of the “last key”. They don’t hesitate to kill for it.

[**Larsson2006**] Stieg Larsson,

Perdition (original title: Flickan som lekte med elden), 2006.

The author was posthumously awarded in 2006 with the Scandinavian thriller award. The super hero Lisbeth Salander uses PGP and occupies herself with mathematical riddles like the Fermat theorem.

[**Schroeder2008**] Rainer M. Schröder,

The Judas Documents (original title: Die Judas-Papiere), Arena, 2008.

In the year 1899 Lord Pembroke has three men and one woman in his grip. So they have to follow his order to try to decipher the encrypted messages in the notebook of his dead brother Mortimer and to find the missing gospel according to Judas, which could shock

the whole of Christendom. The four people therefor have to solve riddles at many places in the world. The story explains some classic ciphers like Polybius and Freemason.

Remark 1: Further samples of cryptology in fictional literature can be found on the following German web page:

http://www.staff.uni-mainz.de/pommeren/Kryptologie99/Klassisch/1_Monoalph/Literat.html

For some older authors (e.g. Jules Verne, Karl May, Arthur Conan Doyle, Edgar Allen Poe) there are links to the original and relevant text pieces.

Remark 2: You can find title pages of some of these books on the web site of Tobias Schrödel, who collects classic books about cryptography:

http://tobiasschroedel.com/crypto_books.php

Remark 3: If you know of further books and movies, where cryptography has a major role then we would be very glad if you could send us the exact title and a short explanation about the movie/book's content. Thanks a lot.

A.3.2 For Kids

Kid books with collections of simpler cryptographic encryption methods, prepared in a didactic and exciting manner are in the following list (please send us similar English kid books, because at the moment our list contains only German kid books):

[**Mosesxxxx**] [no named author],

Top secret – The Book for Detectives and Spies (original title: Streng geheim – Das Buch für Detektive und Agenten), Edition moose, [no year named].

This is a thin book for small kids with Inspector Fox and Dr. Chicken.

[**Para1988**] Para,

Ciphers (original title: Geheimschriften), Ravensburger Taschenbuch Verlag, 1988 (1st edition 1977).

On 125 pages filled with a small font this mini format book explains many methods which young children can apply directly to encrypt or hide their messages. A little glossary and a short overview about the usage of encryption methods in history complete this little book.

Right at page 6 it summarizes for beginners in an old fashion style “The Important Things First” about paper&pencil encryption (compare chapter 2):

- “It must be possible to encrypt your messages at any place and at any location with the easiest measures and a small effort in a short time.
- Your cipher must be easy to remember and easy to read for your partners. But strangers should not be able to decrypt them.
Remember: Fastness before finesse, security before carelessness.
- Your message must always be as short and precise as a telegram. Shortness outranks grammar and spelling. Get rid of all needless like salutations or punctuation marks. Preferably use only small or only capital letters.”

[**Müller-Michaelis2002**] Matthias Müller-Michaelis,

The manual for detectives. Everything you need to know about ciphers, codes, reading tracks and the biggest detectives of the world (original title: Das Handbuch für Detektive. Alles über Geheimsprachen, Codes, Spurenlesen und die großen Detektive dieser Welt), Südwest, 2002.

[**Kippenhahn2002**] Rudolf Kippenhahn,

Top secret! – How to encrypt messages and to hack codes (original title: Streng geheim! – Wie man Botschaften verschlüsselt und Zahlencodes knackt), rororo, 2002.

In this novel a grandpa, an expert for secret writings teaches his four grandchildren and their friends, how to encrypt messages which nobody should read. Because there is someone who hacks their secrets, the grandpa has to teach them more and more complicated methods.

Within this story, which forms the framework, the most important classic encryption methods and its analysis are explained in a manner exciting and appropriate for children.

[**Harder2003**] Corinna Harder und Jens Schumacher,
Top secret. The big book for detectives (original title: Streng geheim. Das große Buch der Detektive), Moses, 2003.

[**Flessner2004**] Bernd Flessner,
The Three Investigators: Manual for Secret Messages (original title: Die 3 ????: Handbuch Geheimbotschaften), Kosmos, 2004.

On 127 pages you learn in an easy and exciting manner, structured by the method types, which secret languages (like the one of the Navajo Indians or dialects) and which secret writings (real encryption or hiding via technical or linguistic steganography) existed and how simple methods can be decrypted.

The author tells where in history the methods were used and in which novel authors used encryption methods [like in Edgar Allan Poe's "The Gold Bug", like with Jules Verne's hero Mathias Sandorf or like with Astrid Lindgren's master detective Blomquist who used the ROR language (similar inserting ciphers are the spoon or the B language)].

This is a didactically excellent introduction for younger teens.

[**Zübert2005**] Directed by Christian Zübert,
The Treasure of the White Hawks (original title: Der Schatz der weißen Falken), 2005.

This exciting adventure movie for kids ties in with the tradition of classics like "Tom Sawyer and Huckleberry Finn" or Enid Blytons "Five Friends". The plot happens in summer 1981. In an old half tumbledown villa three young kids find the treasure map of the "White Hawks", which they decrypt with the help of a computer. Traced by another gang they aim to go to an old castle.

Remark 1: You can find title pages of many of these kid books on the web site of Tobias Schrödel, who collects classic books about cryptography:
http://tobiasschroedel.com/crypto_books.php

Remark 2: If you know of further books, which address cryptography in a didactic and for children adequate way, then we would be very glad if you could send us the exact book title and a short explanation about the book's content. Thanks a lot.

A.4 Learning Tool for Elementary Number Theory

CrypTool contains an interactive educational tool for elementary number theory, called “NT”.⁴

The educational tool “NT” (number theory) by Martin Ramberger introduces number theory and visualizes many of the methods and concepts. Where necessary it show the according mathematical formulas. Often you can apply the mathematical methods dynamically with your own small numerical examples.

The content of this educational tool is mainly based on the books by J. Buchmann and H. Scheid [Buchmann2004, Scheid2003].

This visualized educational tool was build with Authorware 4.

Request for enhancement/upgrade: It would be desirable to update it to a new version Authorware or to use another development platform. If there are developers interested to do this, I'd be more than happy (please send an email at the author of this CrypTool script).

Figures: The figures A.2 till A.9 give you an impression of the educational tool “NT”:

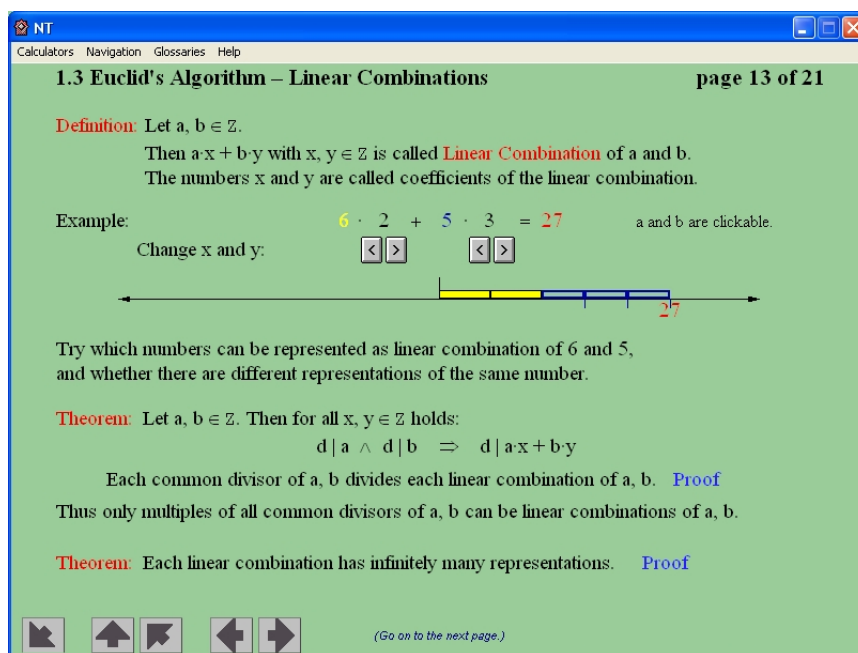


Figure A.2: Each common divisor of two integers also divides all its linear combinations

⁴NT can be called in CrypTool via the menu path **Indiv. Procedures \ Number Theory Interactive \ Learning Tool for Number Theory**.

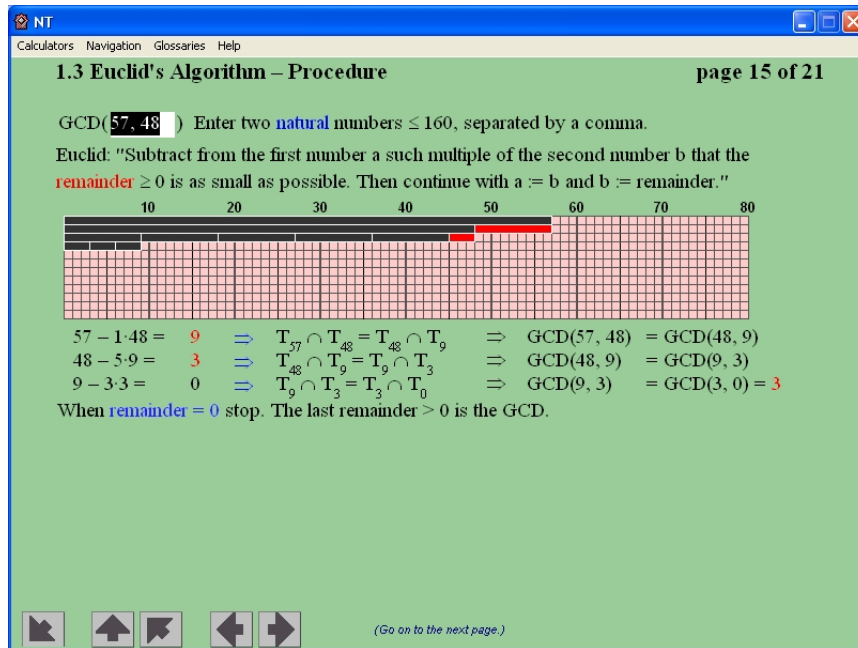


Figure A.3: Euklid's algorithm to determine gcd

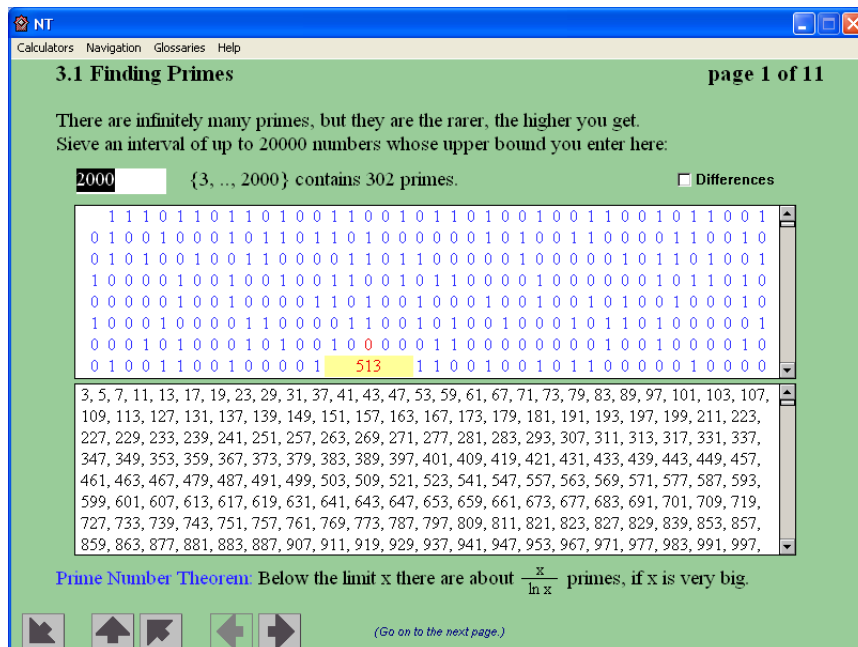


Figure A.4: Distribution of primes and its differences

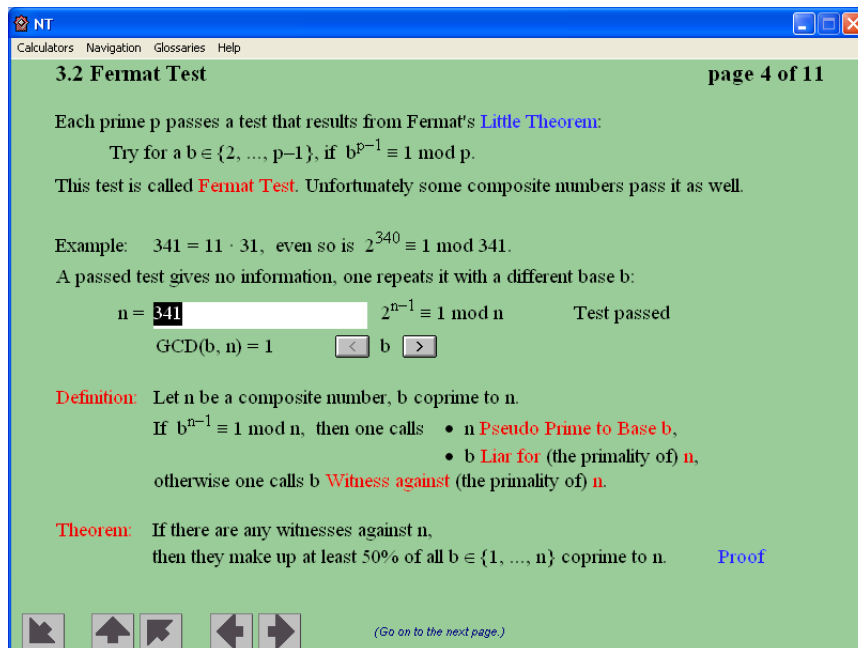


Figure A.5: Finding primes with the prime number test of Fermat

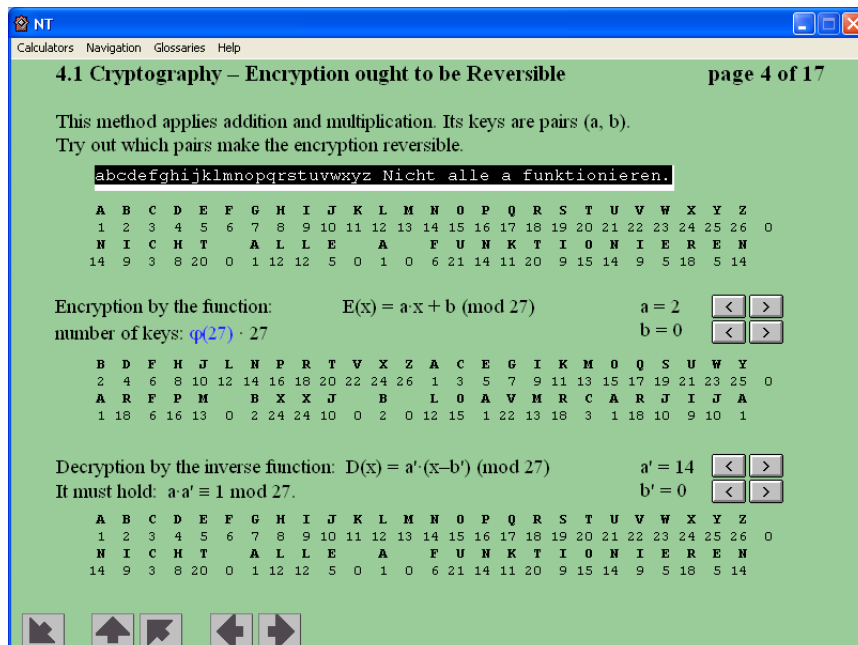


Figure A.6: Reversibility of encryption mechanisms exemplified with additive ciphers

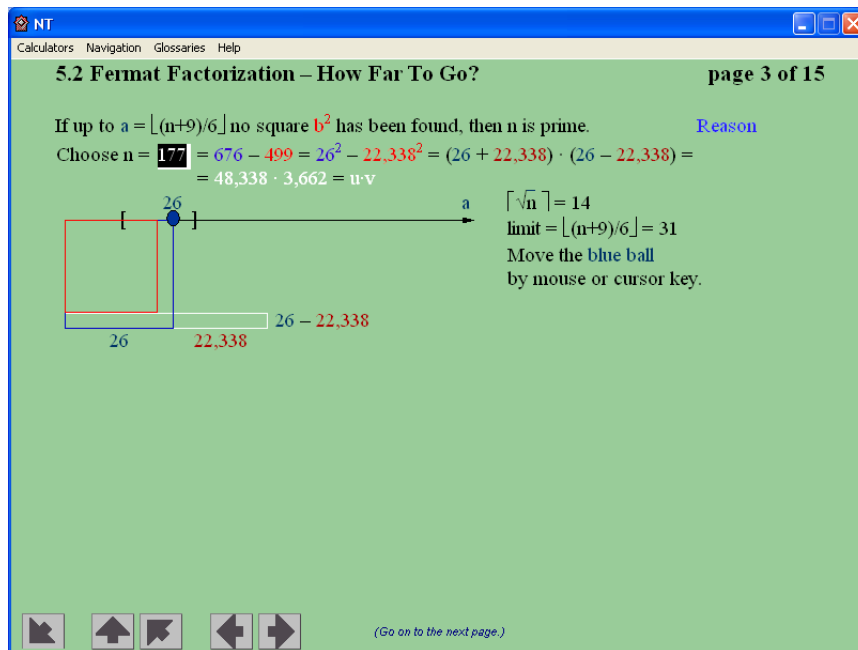


Figure A.7: Fermat factorization using the third binomial formula

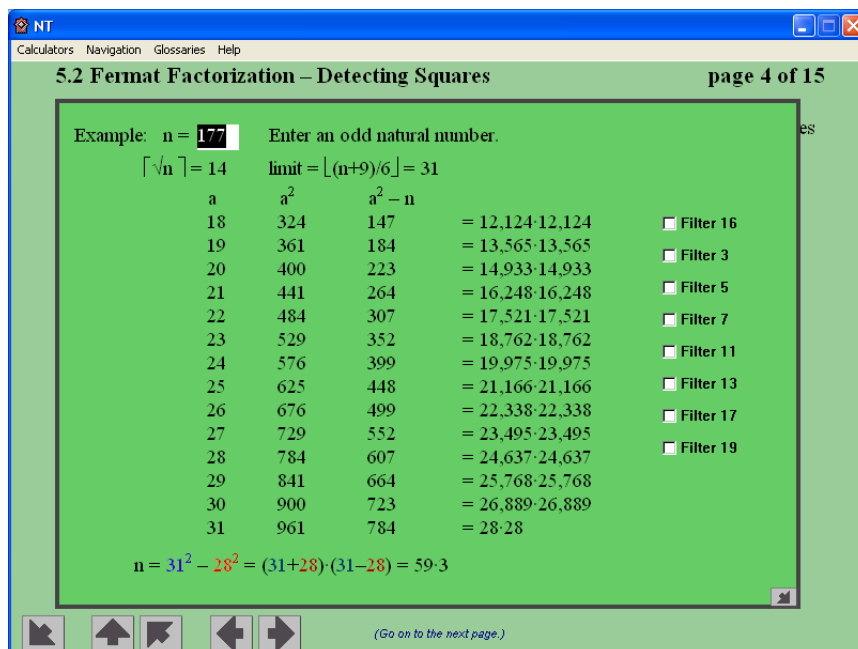


Figure A.8: Fermat factorization: Detecting squares

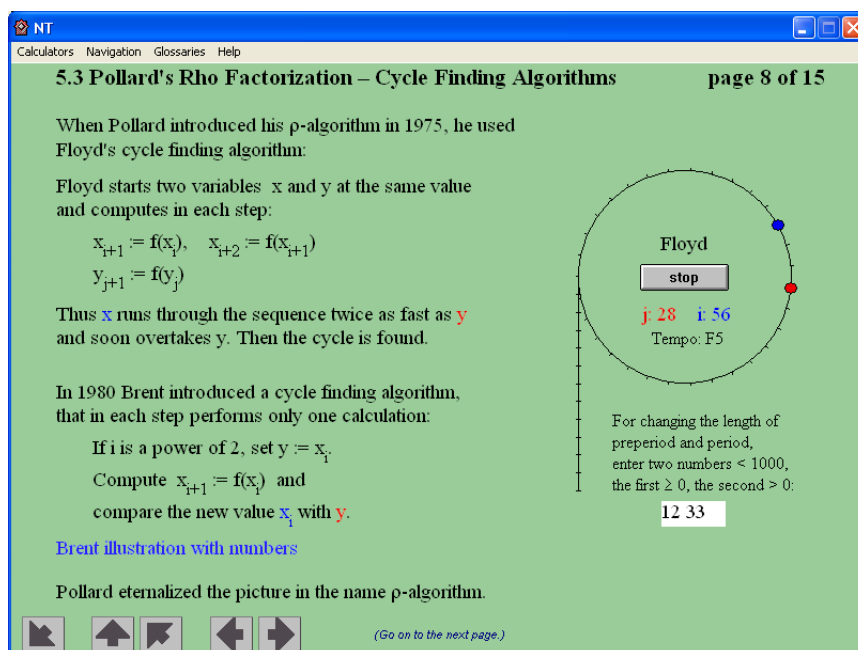


Figure A.9: Pollard's rho factorization: Floyd's cycle finding algorithm

Bibliography

- [Buchmann2004] Johannes Buchmann,
Introduction to Cryptography, Springer, 2nd edition, 2004.
- [Scheid2003] Harald Scheid,
Zahlentheorie, Spektrum Akademischer Verlag, 3rd edition, 2003.

A.5 Using Sage with this Script

This script includes numerous code samples using Sage. Sage is an open source computer algebra system (CAS) that supports teaching, study and research in mathematics. It combines many high-quality open source packages⁵ and provides access to their functionalities via a common interface, namely, a Python⁶ based programming language.

Sage can be used as a powerful desktop calculator, as a tool to help (undergraduate) students study mathematics, or as a programming environment for prototyping algorithms and research in algorithmic aspects of mathematics.

You can get a quick impression of Sage e.g. with David Joyner's "Invitation to Sage"⁷.

The official Sage online documentation⁸ is available at: <http://www.sagemath.org/doc>.

In the meantime there are lots of PDF and HTML documents about using Sage, so we name only a few of them as a good starting point⁹.

With respect to studying cryptography, Sage modules can be used to complement a first course in cryptography¹⁰.

Especially David Kohel's notes at

<http://www.sagemath.org/library/crypto.pdf> or the same eventually newer at

<http://sage.math.washington.edu/home/wdj/teaching/kohel-crypto.pdf>

can be used to teach such a course that incorporates Sage.

Sage user interfaces

Sage is available free of charge and can be downloaded from the following website:

<http://www.sagemath.org>

The default interface to Sage is command line based, as shown in figure A.10. However, there is a graphical user interface to the software as well in the form of the Sage notebook (see figure A.11). We can even use Sage notebooks¹¹ online at different servers, without having to install Sage locally, e.g:

⁵To get an impression of how big Sage is: After downloading the source of Sage 4.1, it took around 5 hours on an average Linux PC to compile the whole system including all libraries. The compiled version occupied 1.8 GB disk space.

⁶There is also an easy interface to the C language, called Cython, which can be used to substantially speed up functions in Sage.

See http://openwetware.org/wiki/Open_writing_projects/Sage_and_cython_a_brief_introduction.

⁷<http://sage.math.washington.edu/home/wdj/teaching/calci-sage/an-invitation-to-sage.pdf> (last update 2009).

⁸The corresponding official PDF documents can be downloaded at

<http://www.sagemath.org/help.html>

⁹- "Library": <http://www.sagemath.org/library/index.html>,

- "Documentation Project": <http://wiki.sagemath.org/DocumentationProject>,

- "Teaching": http://wiki.sagemath.org/Teaching_with_SAGE.

¹⁰- Module sources in the directory `SAGE_ROOT/devel/sage-main/sage/crypto`.

- Overview, what crypto currently is in Sage:

<http://www.sagemath.org/doc/reference/sage/crypto/>

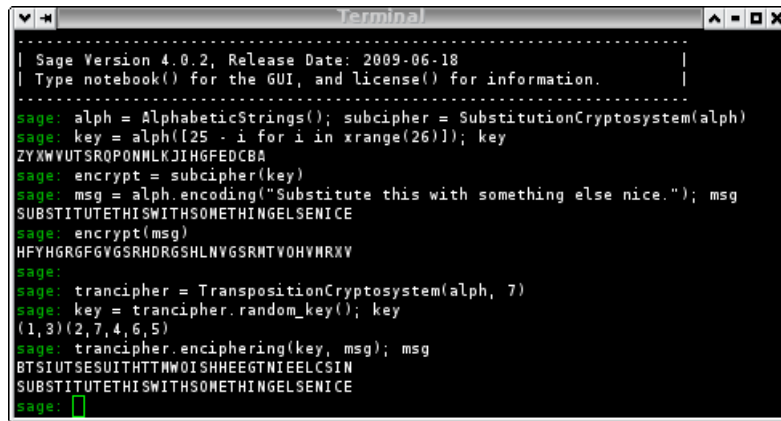
- Discussions about teaching related aspects of development crypto in Sage:

http://groups.google.com/group/sage-devel/browse_thread/thread/c5572c4d8d42d081

¹¹Further details about Sage notebooks can be found at chapter 7.9.2 ("Implementing elliptic curves for educational purposes" \Rightarrow "Sage")

<http://www.sagenb.org> or
<http://sage.mathematik.uni-siegen.de:8000>

Sage runs under many Linux distributions, Mac OS X, and Windows. For the Windows platform, a complete distribution of Sage currently only runs as a VMware image. However, a full native port of Sage to Windows is currently in progress.



```
Terminal
-----
| Sage Version 4.0.2, Release Date: 2009-06-18 |
| Type notebook() for the GUI, and license() for information. |
-----
sage: alph = AlphabeticStrings(); subcipher = SubstitutionCryptosystem(alph)
sage: key = alph([25 - i for i in xrange(26)]); key
ZYXWVUTSRQPONMLKJIHGFEDCBA
sage: encrypt = subcipher(key)
sage: msg = alph.encoding("Substitute this with something else nice."); msg
SUBSTITUTETHISWITHSOMETHINGELSENICE
sage: encrypt(msg)
HFYHGRGFGVGSRHDRGSHLNVGSRMTVOHVVRXV
sage:
sage: trancipher = TranspositionCryptosystem(alph, 7)
sage: key = trancipher.random_key(); key
(1,3)(2,7,4,6,5)
sage: trancipher.enciphering(key, msg); msg
BTSIUTSESUITHTHWOISHHEEGTNIEELCSIN
SUBSTITUTETHISWITHSOMETHINGELSENICE
sage: █
```

Figure A.10: Sage command line interface

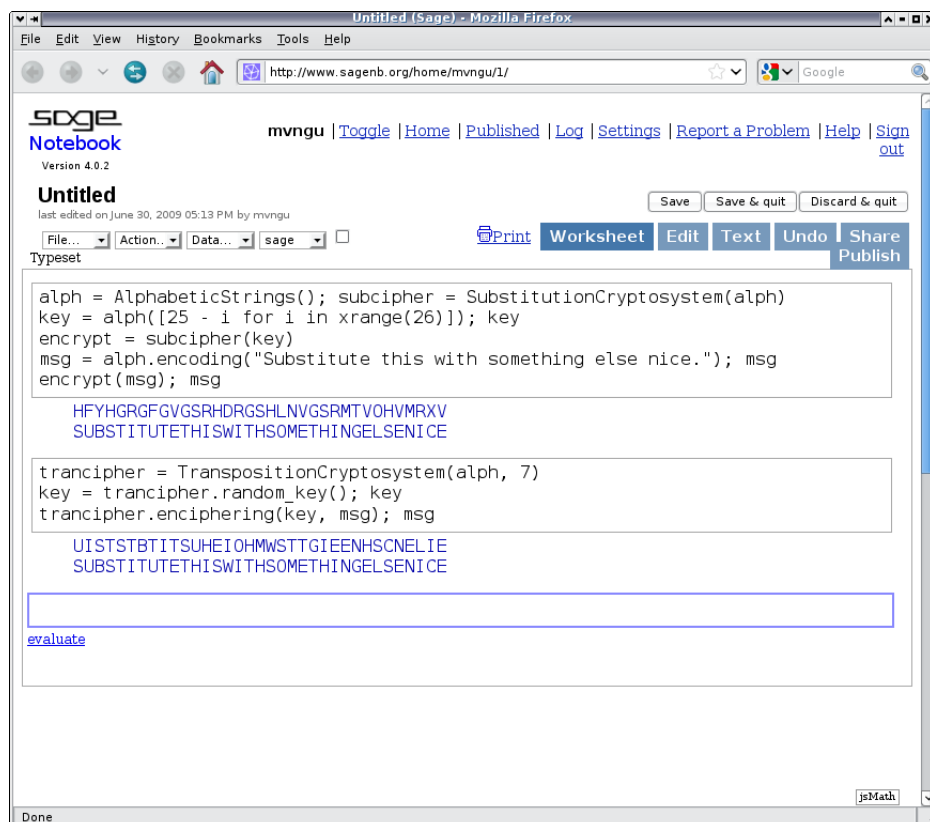


Figure A.11: Sage notebook interface¹²

¹²To start the Sage gui locally: Enter notebook() at the Sage prompt, and then your favorite browser (Iceweasel, Firefox, IE, ...) is started e.g. with the URL <http://localhost:8000>.

Getting help with using Sage

Upon loading Sage from the command line, we are presented with something similar to the following:

```
mnemonic:~$ sage
-----
| Sage Version 4.1, Release Date: 2009-07-09          |
| Type notebook() for the GUI, and license() for information. |
-----

sage: help
Type help() for interactive help, or help(object) for help about object.
sage:
sage:
sage: help()
```

Welcome to Python 2.6! This is the online help utility.

If this is your first time using Python, you should definitely check out the tutorial on the Internet at <http://docs.python.org/tutorial/>.

Enter the name of any module, keyword, or topic to get help on writing Python programs and using Python modules. To quit this help utility and return to the interpreter, just type "quit".

To get a list of available modules, keywords, or topics, type "modules", "keywords", or "topics". Each module also comes with a one-line summary of what it does; to list the modules whose summaries contain a given word such as "spam", type "modules spam".

Plenty of help is provided in the form of the official Sage documentation that is distributed with every release of Sage (see Figure A.12). The official Sage standard documentation includes the following documents:

- Tutorial — This tutorial is designed to help Sage beginners become familiar with Sage. It covers many features that beginners should be familiar with, and takes one to three hours to go through.
- Constructions — This document is in the style of a Sage “cookbook”. It is a collection of answers to questions about constructing various objects in Sage.
- Developers’ Guide — This guide is for developers who want to contribute to the development of Sage. Among other issues, it covers coding style and conventions, modifying the core Sage libraries, modifying the Sage standard documentation, and code review and distribution.
- Reference Manual — This manual provides complete documentation on the major features of Sage. The description of a class is accompanied by numerous code samples. All code samples in the reference manual are tested before each Sage release.
- Installation Guide — This guide explains how to install Sage under various platforms.
- A Tour of Sage — This is a tour of Sage that showcases various features of Sage that are useful for beginners.

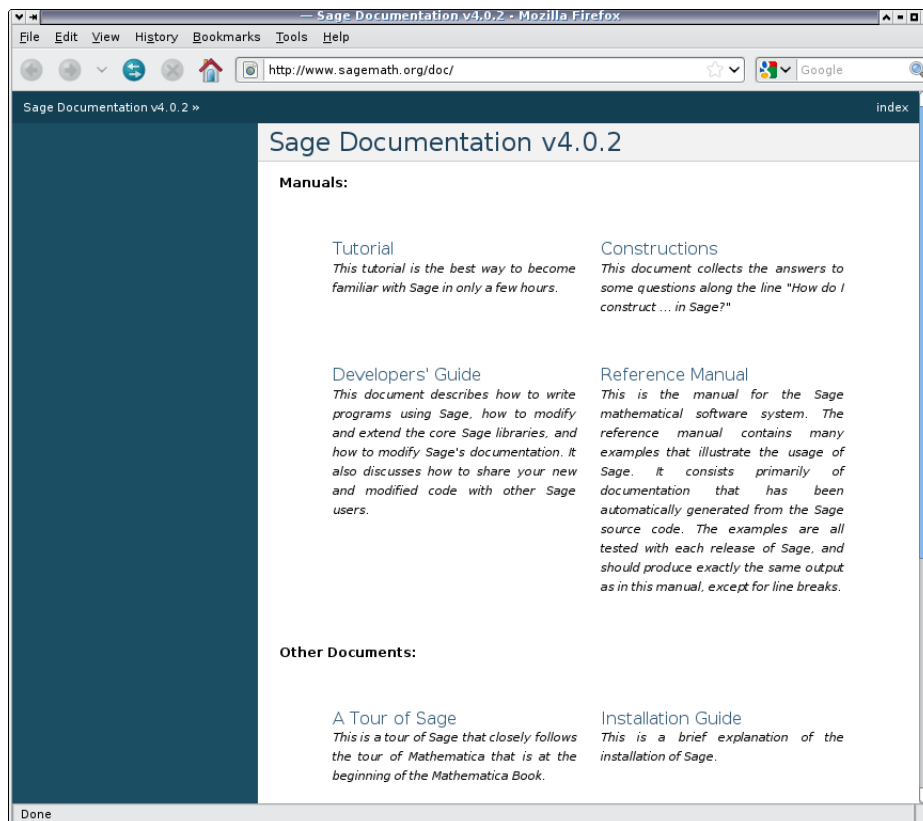


Figure A.12: The Sage standard documentation

- Numerical Sage — This document introduces tools available under Sage that are useful for numerical computation.
- Three Lectures about Explicit Methods in Number Theory Using Sage — This document is about using Sage to perform computations in advanced number theory.

From within a Sage session, we can obtain a list of commands matching some pattern. To do so, we type the first few characters and then press the “Tab” key:

```
sage: Su[TAB]
Subsets                Subwords                SuzukiGroup
SubstitutionCryptosystem  SupersingularModule
```

If we know the exact name of a command, we can use the `help` function to obtain further information on that command, or append the question mark “?” to the command name. For example, the command `help(SubstitutionCryptosystem)` provides documentation on the built-in class `SubstitutionCryptosystem`. We can get documentation on this class with the question mark as follows:

```
sage: SubstitutionCryptosystem?
Type: type
Base Class: <type 'type'>
String Form: <class 'sage.crypto.classical.SubstitutionCryptosystem'>
Namespace: Interactive
File: /home/mvngu/usr/bin/sage-3.4.1/local/lib/python2.5/site-packages/sage/crypto/classical.py
Docstring:
```

Create a substitution cryptosystem.

INPUT:

- ‘‘S’’ - a string monoid over some alphabet

OUTPUT:

- A substitution cryptosystem over the alphabet ‘‘S’’.

EXAMPLES::

```
sage: M = AlphabeticStrings()
sage: E = SubstitutionCryptosystem(M)
sage: E
Substitution cryptosystem on Free alphabetic string monoid
on A-Z
sage: K = M([ 25-i for i in range(26) ])
sage: K
ZYXWVUTSRQPONMLKJIHGFEDCBA
sage: e = E(K)
sage: m = M(‘‘THECATINTHEHAT’’)
sage: e(m)
GSVXZGRMGSVSZG
```

TESTS::

```
sage: M = AlphabeticStrings()
sage: E = SubstitutionCryptosystem(M)
sage: E == loads(dumps(E))
True
```

For further assistance on specific problems, we can also search the archive of the **sage-support** mailing list at

<http://groups.google.com/group/sage-support>

Some examples using the built-in mathematical functions in Sage

Here are a few little examples¹³ (all in console mode, for ease) to see what you can do with Sage:

Sage sample A.1 Some small general samples in Sage from different areas in mathematics

```
# * Calculus:
sage: x=var('x')
sage: p=diff(exp(x^2),x,10)*exp(-x^2)
sage: p.simplify_exp()
1024 x^10 + 23040 x^8 + 161280 x^6 + 403200 x^4 + 302400 x^2 + 30240

# * Linear Algebra:
sage: M=matrix([[1,2,3],[4,5,6],[7,8,10]])
sage: c=random_matrix(ZZ,3,1);c
[ 7 ]
[-2 ]
[-2 ]
sage: b=M*c
sage: M^-1*b
[ 7 ]
[-2 ]
[-2 ]

# * Number theory:
sage: p=next_prime(randint(2^49,2^50));p
1022095718672689
sage: r=primitive_root(p);r
7
sage: pl=log(mod(10^15,p),r);pl
1004868498084144
sage: mod(r,p)^pl
1000000000000000

# * Finite Fields (\url{http://en.wikipedia.org/wiki/Finite_field}):
sage: F.<x>=GF(2) []
sage: G.<a>=GF(2^4,name='a',modulus=x^4+x+1)
sage: a^2/(a^2+1)
a^3 + a
sage: a^100
a^2 + a + 1
sage: log(a^2,a^3+1)
13
sage: (a^3+1)^13
a^2
```

¹³The examples are from the blog of Dr. Alasdair McAndrew, Victoria University,
<http://amca01.wordpress.com/2008/12/19/sage-an-open-source-mathematics-software-system>

Writing code samples with Sage

When you start using a CAS (computer algebra system) you normally type in the single commands on the command line as in the above example¹⁴.

But if you develop your own functions, modify them and call them, then it is much easier to do the development in your own editor, save it to a script file and execute the functions non-interactively on the command line manually. Both ways to develop code is done in the examples in chapter 2.5 (“Appendix: Examples using Sage”), chapter 3.13 (“Appendix: Examples using Sage”) and in chapter 4.18 (“Appendix: Examples using Sage”).

To program and test Sage code using an editor there are two useful commands: `load()` and `attach()`¹⁵.

Suppose you have a function definition like this:

```
def function(var1):
    r"""
    DocText.
    """
    ...
    return (L)
```

which has been saved to the file `primroots.sage`.

To load this function into Sage (and do a syntax check at once), use `load()` as follows:

```
sage: load primroots.sage
```

and you can then proceed to use on the command line any variable or function defined in that Sage script¹⁶.

Normally we also want to edit our own Sage script and reload the content of the changed script into Sage again. In that case, you can use the command `attach()` (you also can apply `attach()` directly after loading the script, even before having changed the script; and you can even omit `load()`, as this is contained in `attach()`):

```
sage: attach primroots.sage
```

¹⁴The standard way for presenting Sage code starts the lines with “sage:” and “...”.

```
sage: m = 11
sage: for a in xrange(1, m):
....:     print [power_mod(a, i, m) for i in xrange(1, m)]
....:
```

This script usually uses the above convention for presenting Sage code, if the code doesn’t come from a Sage script. When people copy and paste the Sage code from this script, in order to enter it at the command line, they should leave out “sage:” and “...” from the script (nevertheless in most cases the command prompt can deal with these prefixes correctly).

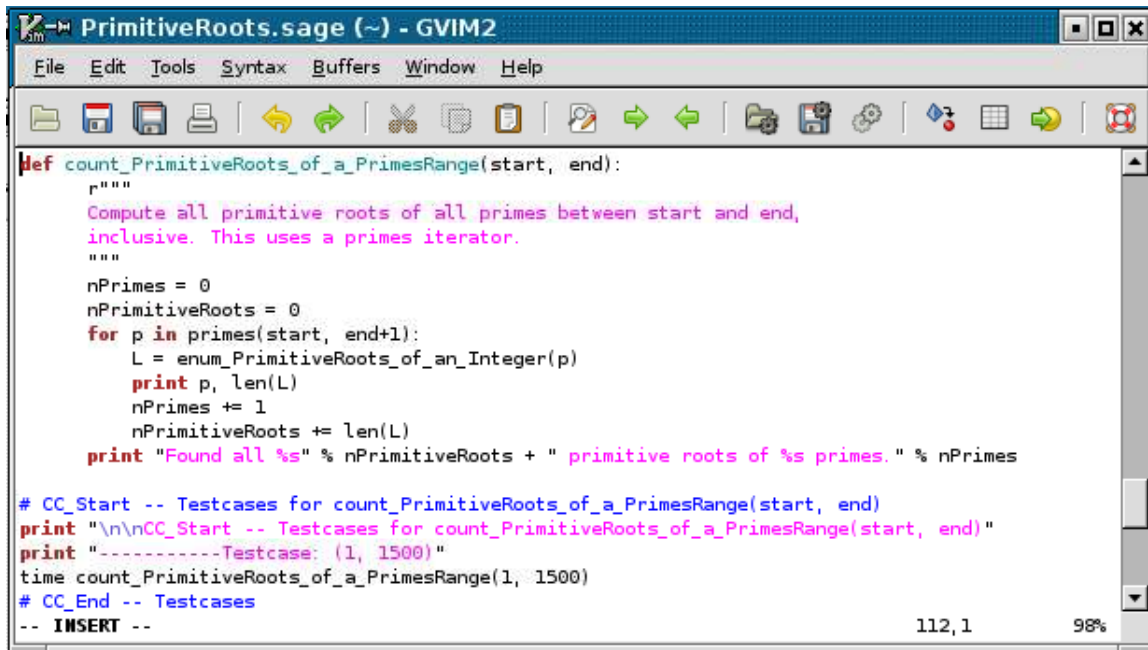
¹⁵See Sage tutorial about Programming, chapter “Loading and Attaching Sage files”, <http://www.sagemath.org/doc/tutorial/programming.html#loading-and-attaching-sage-files>.

¹⁶Notes:

- Don’t use white spaces in your file name.
- Its recommended that your Sage script has the file extension “.sage”, instead of “.py”. With a Sage script whose file name ends in “.sage”, when you load it into Sage then the default Sage environment is also loaded to make sure that it works as if you have defined your function from the Sage command line. This also applies if you run the script from a bash shell using `$ sage primroots.sage`.
- If you run your script as above, then Sage first parses your script, writes it to another file called “primroots.py” (note the “.py” extension), adds all necessary variables to “primroots.py” as well as writing any import statements to that file. That way, your Sage script is executed as if you had typed the definitions in your script to the Sage command line. An important difference is that all output needs a `print` statement.

Now edit the Sage script in a text editor, but don't exit Sage. After you saved it within your text editor, the changed function definition is reloaded into the running Sage session after the next typing of Enter (and a syntax check is done at once). This reloading is done automatically for you, provided that all changes to your script have been saved. You can think of the command `attach()` as a way of telling Sage to watch for all changes to a file, and reloading the file again once Sage notices that there have been changes. With this command, you don't have to copy and paste between your text editor and the Sage command line interface.

Here is a picture of Sage code in the editor GVIM with activated code highlighting (see figure A.13).



```

def count_PrimitiveRoots_of_a_PrimesRange(start, end):
    """
    Compute all primitive roots of all primes between start and end,
    inclusive. This uses a primes iterator.
    """
    nPrimes = 0
    nPrimitiveRoots = 0
    for p in primes(start, end+1):
        L = enum_PrimitiveRoots_of_an_Integer(p)
        print p, len(L)
        nPrimes += 1
        nPrimitiveRoots += len(L)
    print "Found all %s" % nPrimitiveRoots + " primitive roots of %s primes." % nPrimes

# CC_Start -- Testcases for count_PrimitiveRoots_of_a_PrimesRange(start, end)
print "\n\nCC_Start -- Testcases for count_PrimitiveRoots_of_a_PrimesRange(start, end)"
print "-----Testcase: (1, 1500)"
time count_PrimitiveRoots_of_a_PrimesRange(1, 1500)
# CC_End -- Testcases
-- INSERT --
  
```

Figure A.13: Sage sample shown in an editor with code highlighting

If you prefer to see the output of an attached file as if you would have typed in the commands on the commandline directly (not only what is shown via `print`) then you could use the command `iload()`: Each line is loaded one at a time. To load the next line, you have to press the Enter key. You have to repeatedly press the Enter key until all lines of the Sage script are loaded into the Sage session.

```
sage: iload primroots.sage
```

Some more hints:

- To get the version of your Sage environment: `version()`
- To move quickly to the Sage code examples in this script,
 - either look in the index at **Sage -> Code examples**,
 - or have a look at the appendix “List of Sage Code Examples”.
- The source code of the Sage samples in this script is delivered as Sage program files within the CrypTool setup program. After installing CrypTool 1.x you find them within the subdirectory **sage** within the CrypTool directory:
 - SAGE-Samples-in-Chap01.sage
 - SAGE-Samples-in-Chap02.sage
 - SAGE-Samples-in-Chap03.sage
 - SAGE-Samples-in-Chap04.sage

GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

<http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “**Document**”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “**you**”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “**Modified Version**” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “**Secondary Section**” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to

the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “**Invariant Sections**” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “**Cover Texts**” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “**Transparent**” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “**Opaque**”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “**Title Page**” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “**publisher**” means any person or entity that distributes copies of the Document to the public.

A section “**Entitled XYZ**” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “**Acknowledgements**”, “**Dedications**”, “**Endorsements**”, or “**History**”.) To “**Preserve the Title**” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or non-commercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with ... Texts.” line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

List of Figures

2.1	Structure and naming convention of the Sage cipher code examples	32
4.1	Forecast about future factorization records compared with current results (from Secorvo)	118
4.2	The number of primitive roots of all primes between 1 and 100,000.	155
4.3	The smallest primitive roots of all primes between 1 and 100,000.	156
4.4	The largest primitive roots of all primes between 1 and 100,000.	156
7.1	Prognosis of the key lengths to be regarded safe for RSA and Elliptic Curves . .	188
7.2	Comparison of signing and verification time for RSA and Elliptic Curves	189
7.3	Example of an elliptic curve with the real numbers as underlying field.	194
7.4	Doubling of a point	196
7.5	Summing up two different points over the real number field	196
A.1	Complete overview of the menu tree of CrypTool 1.4.30	213
A.2	Each common divisor of two integers also divides all its linear combinations . . .	225
A.3	Euklid's algorithm to determine gcd	226
A.4	Distribution of primes and its differences	226
A.5	Finding primes with the prime number test of Fermat	227
A.6	Reversibility of encryption mechanisms exemplified with additive ciphers	227
A.7	Fermat factorization using the third binomial formula	228
A.8	Fermat factorization: Detecting squares	228
A.9	Pollard's rho factorization: Floyd's cycle finding algorithm	229
A.10	Sage command line interface	232
A.11	Sage notebook interface	232
A.12	The Sage standard documentation	234
A.13	Sage sample shown in an editor with code highlighting	238

List of Tables

2.1	Rail Fence cipher	15
2.2	8x8 turning grille	16
2.3	Simple columnar transposition	16
2.4	Columnar transposition (General Luigi Sacco)	17
2.5	Nihilist transposition	18
2.6	Cadenus	18
2.7	Nihilist substitution	20
2.8	Straddling checkerboard with password “Keyword”	21
2.9	Variant of the straddling checkerboard	21
2.10	Baconian cipher	22
2.11	5x5 Playfair matrix	23
2.12	Four square cipher	24
2.13	Vigenère tableau	25
2.14	Autokey	25
2.15	Ragbaby	26
2.16	Bifid	27
2.17	Bazeries	28
2.18	Digrafid	28
2.19	Nicodemus	29
3.1	The 20+ largest known primes and its particular number types (as of July 2009)	54
3.2	The largest primes found by the GIMPS project (as of July 2009)	58
3.3	The longest arithmetic prime number sequences (as of May 2005)	71
3.4	How many primes exist within the first intervals of tens?	77
3.5	How many primes exist within the first intervals of dimensions?	77
3.6	List of particular n -th prime numbers	78
3.7	Likelihoods and dimensions from physics and everyday life	79
3.8	Special values of the binary and decimal systems	80
4.1	Addition table modulo 5	97
4.2	Multiplication table modulo 5	97
4.3	Multiplication table modulo 6	98

4.4	Multiplication table modulo 17 (for $a = 5$ and $a = 6$)	99
4.5	Multiplication table modulo 13 (for $a = 5$ and $a = 6$)	99
4.6	Multiplication table modulo 12 (for $a = 5$ and $a = 6$)	100
4.7	Values of $a^i \bmod 11, 1 \leq a, i < 11$ and according order of $a \bmod m$	108
4.8	Values of $a^i \bmod 45, 1 \leq a, i < 13$	109
4.9	Values of $a^i \bmod 46, 1 \leq a, i < 23$	110
4.10	The current factoring records (as of July 2009)	119
4.11	Capital letters alphabet	132
4.12	RSA ciphertext A	136
4.13	RSA ciphertext B	137
5.1	Euler phi function	171
5.2	$L(N)$ value table	171
5.3	Procedures for calculating the discrete logarithm over \mathbb{Z}_p^*	173

List of Crypto Procedures

5.1	Solving knapsack problems with super-increasing weights	169
5.2	Merkle-Hellman (based on knapsack problems)	169
5.3	RSA (based on the factorization problem)	170
5.4	Rabin (based on the factorization problem)	172
5.5	Diffie-Hellman key agreement	174
5.6	ElGamal (based on the discrete logarithm problem)	174
5.7	Generalized ElGamal (based on the factorization problem)	176
6.1	DSA signature	183

List of Sage Code Examples

1.1	Encryption and decryption with Mini-AES	9
2.1	Simple Transposition by shifting (key and inverse key explicitly given)	33
2.2	Simple Transposition by shifting (key and inverse key constructed with (“range”	34
2.3	Simple Column Transposition with randomly generated (permutation) key	35
2.4	Simple Column Transposition (showing the key_space)	36
2.5	Monoalphabetic Substitution with randomly generated key	37
2.6	Caesar (substitution by shifting the alphabet; key explicitly given, step-by-step approach)	38
2.7	Caesar (substitution by shifting the alphabet; substitution key generated)	39
2.8	A shift cipher over the upper-case letters of the English alphabet	40
2.9	Constructing the Caesar cipher using the shift cipher	40
2.10	An affine cipher with key (3, 13)	41
2.11	Constructing a shift cipher using the affine cipher	41
2.12	Constructing the Caesar cipher using the affine cipher	42
2.13	Monoalphabetic substitution with a binary alphabet	43
2.14	Monoalphabetic substitution with a hexadecimal alphabet (and decoding in Python)	44
2.15	Vigenère cipher	45
2.16	Hill cipher	47
3.1	Special values of the binary and decimal systems	80
3.2	Verify the primality of integers generated by a quadratic function	81
4.1	Multiplication tables for $a \times i \pmod{m}$ with $m = 17$, $a = 5$ and $a = 6$	143
4.2	Fast exponentiation mod $m = 103$	143
4.3	Table with all powers $a^i \pmod{m}$ for $m = 11$, $a = 1, \dots, 10$	144
4.4	Table with all powers $a^i \pmod{45}$ for $a = 1, \dots, 12$ plus the order of a	145
4.5	Table with all powers $a^i \pmod{46}$ for $a = 1, \dots, 23$ plus the order of a	146
4.6	Calculating one primitive root for some primes	147
4.7	Function “enum_PrimitiveRoots_of_an_Integer” to calculate all primitive roots for a given number	148
4.8	Table with all primitive roots for the given prime 541	149
4.9	Function “count_PrimitiveRoots_of_an_IntegerRange” to calculate all primitive roots for a given range of integers	150
4.10	Function “count_PrimitiveRoots_of_an_IntegerRange”: testcases and output	151
4.11	Function “count_PrimitiveRoots_of_a_PrimesRange” to calculate the number of primitive roots for a given range of primes	152
4.12	Code to generate the database with all primitive roots for all primes between 1 and 100,000	153
4.13	Code to generate the graphics about the primitive roots	154
4.14	Factoring a number	157

4.15	RSA encryption by modular exponentiation of a number (used as message)	. . .	157
4.16	How many RSA keys are there if you know a range for the public keys n?	. . .	159
A.1	Some small general samples in Sage from different areas in mathematics	236

- All samples have been tested with Sage v 4.2.1 (Release Date 2009-11-14).
- The source code of the Sage samples in this script is delivered as Sage program files within the CrypTool setup program. After installing CrypTool 1.x you find them within the subdirectory **sage** within the CrypTool directory:
 - SAGE-Samples-in-Chap01.sage
 - SAGE-Samples-in-Chap02.sage
 - SAGE-Samples-in-Chap03.sage
 - SAGE-Samples-in-Chap04.sage

Index

A

Aaronson 2003, 83
ACA 2002, 48
Addition, 96, 103
ADFGVX, 26
Adleman 1982, 177
Adleman, Leonard, 5, 169, 170
AES, 2, 6, 7
 Mini-AES, 8
Agrawal 2002, 161
AKS, 74, 126
Alice, 5, 128
AMSCO, 16
Apted 2001, 219
Arthur 196x, 216
Associative law, 94
Atbash, 19
Attack
 birthday, 181
 brute-force, 2, 4, 211
 chosen-ciphertext, 172
 ciphertext-only, 135
 known plaintext, 135
 man-in-the-middle, 184
Authenticity, 5, 184
 user, 179
Authors, 213

B

Baby-step-giant-step, 173, 175
Baconian Cipher, 22
Balcazar 1988, 177
Baldacci 1997, 217
Bartholome 1996, 83, 161
Bauer 1995, 48, 161
Bauer 2000, 48, 161
BC, 129, 164
Beale cipher, 22
Beaufort, 26
Becker 1998, 217
Berne, Eric, 105
Bernstein 2001, 161

Beutelspacher 1996, 161
Biryukov 2009, 10
Block length, 131, 133, 134
Blum 1999, 83
Bob, 5, 128
Bogk 2003, 163
Book cipher, 23
Bourseau 2002, 161
Brands 2002, 161
Brickell 1985, 177
Brickell, Ernst, 170
Brown 1998, 218
Brown 2003, 219
Buchmann 2004, 161, 229
Buhler 1993, 161
Bundschuh 1998, 83
Burger 2006, 220

C

C158, 120
C307, 122
Cadenus, 18
Caesar, 19
Caldwell Chris, 85
Capital letters alphabet, 132, 137
CAS, 80
Cascade cipher, 3, 26
Cassels 1991, 204
Catalan Eugene, 67
Certicom, 195, 205
Certification
 public key, 183
Certification authority (CA), 184
Ché Guevara, 21
Challenge, 5, 119
Cipher challenge, 5, 119
Closeness, 95, 103, 140
Cole, Frank Nelson, 56
Colfer 2001, 218
Collision, 180, 181
Collision resistance, 181
Commutative law, 94

- Complexity, 115, 167, 176, 189
- Congruence, 92, 93
- Coppersmith 2002, 10
- Courtois 2002, 10
- Crandall, Richard, 57
- Crandell 2001, 83
- Crichton 1987, 216
- Crowley 2000, 48
- Cryptanalysis, 2, 7, 132, 134, 135
- Crypto challenge, 5, 119
- Cryptography
 - modern, 50, 127, 166
 - Post Quantum, 208
 - public key, 50, 111, 168
- CrypTool, ii, x, xi, 2, 4–7, 15, 16, 19, 22, 23, 25, 26, 29, 56, 59, 69, 92, 108, 112, 115, 120, 122, 128, 131–136, 138, 164, 166, 170, 172, 173, 179–181, 201, 211, 213, 218, 224
- CrypTool 1.x, x, 211, 238, 252
- CrypTool 2.0, 211
- Cunningham project, 62, 85, 120, 122, 165
- D**
- DA 1999, 48
- Dedekind, Julius, 87
- DES, 2, 4, 7
 - SDES, 7
- Diffie, Whitfield, 5, 128, 173
- Diffie-Hellman, 87, 128, 173, 174, 198
- Discrete logarithm, 102, 129, 172
- Distributive law, 94
- Divisibility, 92
- Division modulo n , 94, 96
- Divisor, 92
- Domain parameter, 198
- Double column transposition, 17
- Doyle 1905, 216
- Doyle, Sir Arthur Conan, 216
- DSA, 5, 199, 201
 - signature, 183
- E**
- ECDLP, 197, 198
- Eckert 2003, 112, 132, 161
- ECMNET, 199
- Educational tool NT, 59, 69, 92, 108, 115, 138, 166, 172, 224
- EFF, 58
- ElGamal
 - public key, 174
- ElGamal, Tahir, 5
- Elliptic curves, 187
 - ECC notebook, 202
- Elsner 1999, 218
- Elsner 2001, 218
- Encryption, 1
 - asymmetric, 5, 86, 166
 - Cascade cipher, 3, 26
 - code-based, 208
 - ElGamal public key, 174
 - hybrid, 6
 - lattice problems, 208
 - NTR, 208
 - McEliece, 208
 - Merkle-Hellman, 169
 - Product algorithm, 3, 26
 - public key, 166
 - symmetric, 2, 14
- Eratosthenes
 - sieve, 59, 69
- Erdős, Paul, 70
- Ertel 2001, 162
- Euclid, 52
- Euclid's proof by contradiction, 53
- Euclidean algorithm, 200
 - extended, 98, 106, 138
- Euclidean number, 64
- Euler
 - (phi) function, 98, 102, 105, 170
- Euler, Leonhard, 105, 106
- Exponential function
 - calculation, 174
 - discrete, 172
- F**
- Factor, 92
- Factorization, 56, 107, 189
 - factoring records, 56, 74, 119, 164, 199
 - factorization problem, 107, 113, 124, 135, 170
 - forecast, 116
- Ferguson 2001, 10
- Fermat
 - last theorem, 163
 - little theorem, 59, 98, 106
 - number, 59
 - generalized, 53, 55, 63
 - prime number, 62
- Fermat, Pierre, 59, 106

Fibonacci, 87, 164

Field, 190

characteristic, 191

finite, 192

FIPS180-3, 185

FIPS186, 185

FIPS186-2, 185

Fixpoint, 105, 107

Flessner 2004, 223

FlexiProvider, 208

Fox 2002, 161

Fox, Dirk, 117

G

Gödel, Kurt, 72

Gallot, Yves, 61, 63

Gauss bracket, 138

Gauss, Carl Friedrich, 62, 68, 86, 87, 91, 111

gcd, 87, 98, 102, 138, 170

General Number Field Sieve (GNFS), 115, 116,
119–122, 124, 126, 173

GIMPS, 57, 85

GISA, 112, 117, 121, 164, 187

GISA 2002, 162

Goebel 2003, 48

Goldbach project, 85

Goldbach, Christian, 72

Google

Recruitment, 75

Graham 1989, 83

Graham 1994, 87, 162

Grid computing, 116

Group, 86, 102, 174, 190

cyclic, 191

H

Half prime, 71, 158

Harder 2003, 223

Hardy, Godfrey Harold, 70, 71

Hash function, 180

Hash value, 180

Hellman, Martin, 5, 128, 169, 173

Hill 1929, 48

Hill 1931, 48

Hill 2003, 219

Hoffman 2006, 10

Howard 2001, 219

Hybrid procedure, 6

I

IDEA, 2, 7

Identity, 94

IETF, 7

Impersonation attack, 184

Inverse

additive, 94

multiplicative, 95

Invertibility, 104

Isau 2003, 219

ISO/IEC 9594-8, 186

ITU-T, 186

IVBB, 203

J

JCrypTool 1.0, 211

K

Key

private, 166

public, 5, 166

secret, 5

Key agreement (key exchange)

Diffie-Hellman, 128, 173

Key management, 5, 6

Kipling 1901, 215

Kipling, Rudyard, 215

Kippenhahn 1997, 162

Kippenhahn 1999, 162

Kippenhahn 2002, 222

Klee 1997, 83

Knapsack, 168

Merkle-Hellman, 169

Knott, Ron, 87, 164

Knuth 1981, 83

Knuth 1998, 162

Koblitz 1984, 204

Koblitz 1998, 204

Koblitz, Neal, 189

Kronecker, Leopold, 87

L

Lagarias 1983, 177

Lagarias, Jeff, 170

Larsson 2006, 220

Lattice reduction, 116

Legendre, Adrien-Marie, 68, 111

Lem, Stanislaw, 180

Lenstra 1987, 204

Lenstra 1993, 162

Lenstra 2002, 162

Lenstra/Verheul, 206

Lenstra/Verheul 1999, 162, 204
 Lichtenberg, Georg Christoph, 166
 LiDIA, 129, 164
 Literature, 111, 215
 Logarithm, 102, 174
 Logarithm problem
 discrete, 102, 129, 172, 175, 183, 190
 record, 173
 Long integer, 101
 Lorenz 1993, 83
 Lucas, Edouard, 56, 58
 Lucks 2002, 10
 Lucks 2003, 163

M

M1039, 122
 Müller-Michaelis 2002, 222
 Mansoori, Bizaki 2007, 10
 Map cipher, 20
 Massierer, Maike, 202
 Mathematica, 129, 164
 McBain 2004, 219
 McDonald 2009, 185
 Menezes 1993, 204
 Menezes 2001, 162
 Merkle 1978, 177
 Merkle, Ralph, 169
 Mersenne
 number, 55, 56
 generalized, 53, 61–63
 prime number, 55, 56, 61, 74, 85
 M-37, 56
 M-38, 57
 M-39, 57, 61
 theorem, 55
 Mersenne, Marin, 55, 59
 Message integrity, 179
 Miller, Gary L., 60
 Miller, Victor, 189
 Modulus, 92
 Moore's law, 116
 Moore, Gordon E., 116
 Moses xxxx, 222
 Movies, 75, 111, 215
 Multiplication, 96, 103
 Münchenbach, Carsten, 164
 Musa, Schaefer, Wedig 2003, 10

N

Natali 1997, 217

Near prime, 71
 Nguyen 2009, 11, 48
 Nguyen, Minh Van , 131
 Nichols 1996, 11, 48
 Nihilist substitution, 19
 Nihilist transposition, 17
 NIST, 181, 183
 Noll, Landon Curt, 56
 Nomenclature, 20
 NSA, 2, 7
 NT, Learning Tool for Number Theory, 59, 69,
 92, 108, 115, 138, 166, 172, 224

Number

Carmichael, 60, 63
 Catalan, 67
 co-prime, 94, 98, 99, 106, 108, 109, 139,
 145, 168–171
 composite, 51, 90
 Fermat, 59
 Mersenne, 55
 natural, 50, 87
 prime, 50, 51
 Proth, 61
 pseudo prime, 60, 63
 semi prime, 71, 119
 Sierpinski, 55, 61
 strong pseudo prime, 60, 63

Number theory

elementary, 86, 90
 fundamental theorem, 51, 91, 102
 introduction, 87
 modern, 88

O

One Time Pad, 1
 One way function, 102, 127, 166
 with trapdoor , 167
 Open Source, 112
 Oppliger 2005, 83
 Order
 maximum, 108

P

P(n), 68
 Padberg 1996, 83
 Palladium, 126
 Paper and pencil methods, 14, 218
 Para 1988, 222
 Pari-GP, 129, 164, 165
 Patent, 112, 203

Performance, 50, 116, 179, 187
 Permutation, 15, 99, 110, 168
 Pfleeger 1997, 162
 Phan 2002, 11
 Phan 2003, 11
 PI(x), 68
 Pieper 1983, 84
 PKCS#1, 182, 185
 PKCS#5, 180
 PKI, 183
 Playfair, 23
 Poe 1843, 215
 Poe, Edgar Allan, 14, 215
 Pohlig, S. C., 173
 Pollard, John M., 199
 Polynomial, 66, 74, 115, 126, 167–169, 192
 Pomerance 1984, 162
 Pomerance 2001, 83
 Power, 101
 Pre-Image-Attack
 1st, 180
 2nd, 180
 Primality testing, 74, 123, 126
 Prime factor, 91
 decomposition, 91, 102, 105, 170
 Prime number, 50, 90
 density, 67
 Fermat, 62
 formula, 61
 gigantic, 56
 half prime, 71, 158
 Mersenne, 56, 61, 74
 near prime, 71
 number of, 111
 pseudo prime, 60, 63
 records, 53
 relative prime, 64, 98, 99, 107, 170
 semi-prime, 158
 strong pseudo prime, 60, 63
 test, 56, 59, 190
 theorem, 68
 titanic, 56
 Prime sequence
 arithmetic, 70
 Primitive root, 130, 147
 Problem of discrete logarithm, 198
 Product algorithm, 3, 26
 Proof by contradiction, 52, 53, 55
 Proof of existence, 71

Q

Quantum computer, 207–209
 Quantum cryptography, 209

R

Rabin
 public key procedure, 172
 Rabin, Michael O., 60, 172
 Rail Fence cipher, 15
 Raising to the power, 100
 Random, 6, 183
 RC5, 4
 Reducibility, 94
 Relative prime, 64, 98, 99, 107, 170
 Remainder class, 92
 Remainder set
 full, 104
 reduced, 104
 Richstein 1999, 72, 84
 Riemann, Bernhard, 72
 RIPEMD-160, 181
 Rivest, Ronald, 5, 170
 Robinson 1992, 217
 Robshaw 2002, 11
 Root, 102
 Rowling, Joanne, 90, 127
 RSA, 5, 50, 87, 101, 106, 107, 111, 112, 131, 170, 206
 cipher challenge, 134, 135
 modulus, 198
 RSA procedure, 111
 signature, 182
 RSA 1978, 177
 RSA Laboratories, 185
 RSA Security 2002, 162
 RSA-155, 120
 RSA-160, 121
 RSA-200, 121
 Runtime
 efficient, 167
 not polynomial NP, 168
 polynomial, 167

S

Sage, ii, 14, 32, 65, 80, 81, 129–131, 143, 165, 201, 205, 230
 Code examples, 8, 32, 81, 143, 201, 230
 Savard 1999, 49
 Sayers 1932, 216
 Schaefer 1996, 11

Scheid 1994, 84
 Scheid 2003, 229
 Schmeh 2003, 11
 Schmeh 2004, 49
 Schmeh 2007, 49
 SchneiderM 2004, 163
 Schneier 1996, 11, 84, 163, 185
 Schnorr, C.P., 5
 Schroeder 1999, 84
 Schroeder 2008, 220
 Schwenk 1996, 84
 Schwenk 2002, 163
 Scytale, 15
 SECUDE IT Security, 6
 Security
 forecast, 206
 long-term, 206
 Sedgewick 1990, 112, 163
 Seed 1990, 216
 Seneca, 96
 Session key, 6
 Seventeen or Bust SoB, 55
 SHA-1, 181, 183
 Shamir 1982, 177
 Shamir 2003, 163
 Shamir 2003a, 163
 Shamir, Adi, 5, 169, 170
 Short integer, 101
 Shoup 2005, 84
 Signature
 digital, 5, 50, 114, 179, 182, 183
 Merkle, 209
 Signature procedure, 179
 Silver, 173
 Silverman 1986, 204
 Silverman 1992, 204
 Silverman 2000, 163
 Silverman/Tate 1992, 204
 Simmel 1970, 216
 Singh 2001, 49
 Solitaire, 29
 Special Number Field Sieve (SNFS), 119, 122
 Square and multiply, 102, 135
 Stallings 2006, 11
 Stamp 2007, 12
 Steganography, 20
 Stephenson 1999, 218
 Stinson 1995, 134, 163, 171, 173, 175, 177
 Straddling Checkerboard, 20, 21
 Structure, 95, 103, 105, 108
 Substitution, 19
 homophonic, 22
 monoalphabetic, 19
 polyalphabetic, 25
 polygraphic, 23
 Superposition, 26
 Swenson 2008, 12

T
 ThinkQuest 1999, 49
 Tietze 1973, 84
 Transitivity, 95
 Transposition, 15
 Triple-DES, 4
 Turning grille, 15
 TWIRL device, 125, 163

V
 Verne 1885, 215
 Verne, Jules, 215
 Vidal 2006, 220
 Vigenère, 25

W
 Wang 2005, 185
 Weierstrass, Karl, 193, 194
 Weis 2003, 163
 Welschenbach 2001, 163
 Wiles, Andrew, 88, 163, 188
 Wobst 2002, 12
 Wobst 2005, 185
 Wolfenstetter 1998, 163
 Woltman, George, 57

X
 X.509, 184–186

Y
 Yan 2000, 135, 163
 Yates, Samuel, 56

Z
 \mathbb{Z}_n , 103
 \mathbb{Z}_n^* , 104
 Zübert 2005, 223
 Zemeckis 1997, 75