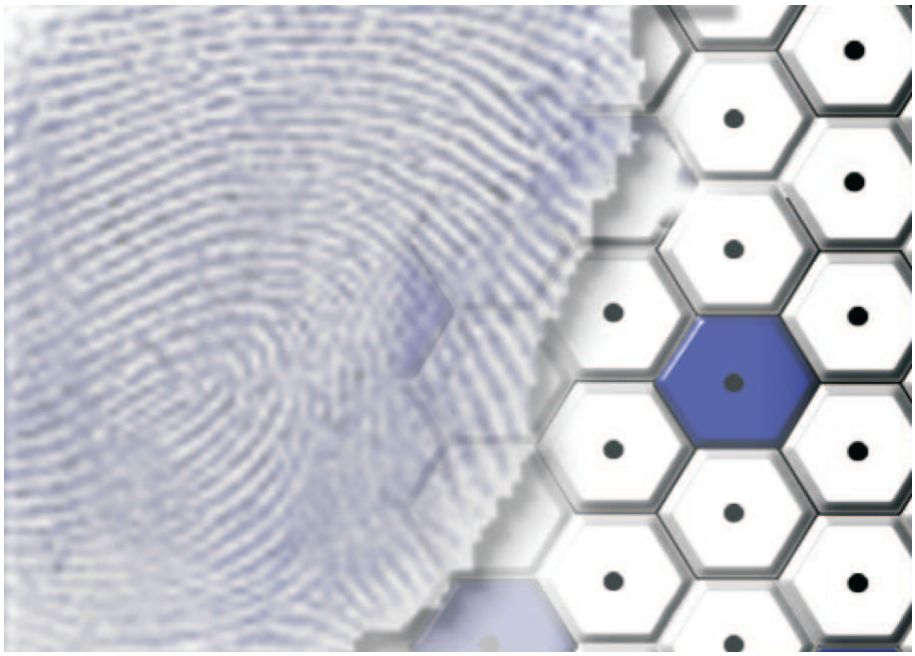
Prof. Dr.-Ing. Firoz Kaderali

Foundations and Applications of Cryptology

Symmetric and Asymmetric Encryption, Digital Signatures, Hash Functions, Key Management and PKI



Preface

Cryptology as a part of scientific research, especially in the public sector, is a relatively young discipline despite its thousands of years of history. Up until a few decades ago, it was only conducted seriously in the military sector and in the sector of state security. In the 1990s there was an increasing interest in cryptographic techniques due to the growth in the number of computer networks and the popularity of electronic commerce. Security relevant aspects have to be considered in the following Internet and network applications: data exchange via FTP or e-mail, information offered on the WWW, client/server relationships, remote access, virtual private networks, e-commerce, payments transactions, Internet banking, exchange of legally valid documents (digitally signed with a timestamp), signing of contracts over the net, virtual town halls (on-line application forms, issuing of certificates, passports, on-line enrollment), on-line elections, auctions etc.

Subject of this course is cryptology, the science of information secrecy and security. The Cryptology consist of cryptography, with data encryption as a main subject, and cryptoanalysis, which involves and analyses techniques for breaking the ciphers. The cryptological algorithms and protocols are the most important modules of the security architecture of any network. At the beginning of the course some mathematical background, important for the understanding of the cryptological algorithms, is treated. This includes subjects of number, group, and field theory, polynomials over finite fields, theory of complexity and probability. Before the modern cryptographic algorithms are discussed in detail, an overview of the classical encryption algorithms is given. Both symmetric (DES, IDEA, stream ciphers based on pseudorandom generators) and asymmetric (RSA, ElGamal) encryption schemes are presented and analysed. Further focuses are digital signatures, hash functions and authentication codes, entity authentication and general key management techniques. Public key infrastructure (PKI) and important PKI-Standards are also presented.

F. Kaderali

Summer 2007

Author

Prof. Dr.-Ing. Firoz Kaderali

1963 - 69	Studied theoretical electrical engineering at the Technische Hochschule Darmstadt
1969 - 74	Assistant at the Faculty of Electrical Engineering at the Technische Hochschule Darmstadt
1974	Doctorate (Promotion) at the Faculty of Electrical Engineering, Technische Hochschule Darmstadt, Subject: Network Theory
1974 - 76	Lecturer of Statistical Signal Theory at the Technische Universität Darmstadt
1976 - 81	Member of the Research Center at SEL (ITT)/Stuttgart. Projektleader of Bundespost study and Fieldtrial DIGON (Digital Local Area Network)
1981 - 86	Head of the Department System Development/Large Systems at Telefonbau und Normalzeit/Frankfurt Development of ISDN-PABXs
Since 1986	Professor for Communication Systems at the FernUniversität in Hagen, Main interests: Communication Systems, Networks and Protocols, Network Security
1989 - 94	Head of the regional telecommunications initiative TELETECH NRW (Consultant and supervisor of over 120 Telecommunications Projects)
1990 - 96	Member of the ISDN Research Commission NRW (Projects on ISDN Applicationsdesign and Technology Assessment)
Since 1992	Director of the Research Institute of Telecommunications (FTK) in Dortmund, Hagen and Wuppertal, Joint Institute of the Universities Hagen (Electrical Engineering) and Wuppertal (Economy)
1995 - 2001	Member of management of mediaNRW, an initiative to promote the development and spreading of multimedia-applications as a whole and interactive services in enterprises, private households, and the public sector
1999 - 2003	Project manager of the research alliance Data Security Northrhein-Westphalia
2000 - 2002	Chairman of the advisory board of GITS (Gesellschaft für IT-Sicherheit) in Bochum, Germany
Since 2002	Chairman of the open source initiative CampusSource

Table of Contents

1 Introduction

1.1 IT-Security

In the last decades the use of computers and communication networks has consistently grown and they have found their way into many areas of private and commercial life. Thus a new workspace, the cyberspace has evolved. Apart from the many new opportunities opened by cyberspace, there are also a number of dangers resulting from the patchy protection against attacks on computers and networks. This basically is because of two aspects: Firstly because of the prevailing computer architecture, the PC, whose initial hardware design lacks any security mechanism and secondly because security aspects have not been taken into account in the development of the Transport Control Protocol /Internet Protocol (TCP/IP), the prevailing communication protocol in computer networks. For example for TCP/IP in the widespread version 4 no encryption is in use (not even the encryption of passwords), sender addresses can be easily forged and even complete messages can be forged or redirected by the intermediate nodes of the network, the routers. When computers or communication networks are used for the processing and distribution of sensitive data or for commercial applications it is necessary to take additional security measures.

cyberspace

TCP/IP

If an unauthorized person gains access to a computer or communication network, data is in danger of being spied on, forged or deleted. Even the computer or network itself can be tampered with or crash. Depending on the application affected, an attack can have diverse and sometimes even disastrous consequences. Attacks range from industrial espionage and spying on public offices to the forgery and prevention of business and financial transactions.

Generally speaking all information, management, supply and transport systems can be affected. Attacks similar to the ones already described, can be launched against each of the following systems: air traffic control, highway police, toll collection, process control in enterprises, etc. The number and scale of these IT-Systems has been constantly growing for several years and as a result of that IT-Security has become more and more significant.

A particular aspect of IT-Security is data security. Data security can mean protecting data from eavesdropping, alteration (or forgery) and observation (of data exchange). It can also mean protection from incorrect personalisation during the data exchange and enforcement of copyrights linked to the data.

data security

Fig. 1.1-1 shows the four different measures which contribute to data security. These are legal measures (e.g. the German “Telekommunikations-Datenschutzverordnung”), organizational measures (e.g. the four-eye-principle), physical measures (e.g. protection of hardware against unauthorized access) and cryptological measures.

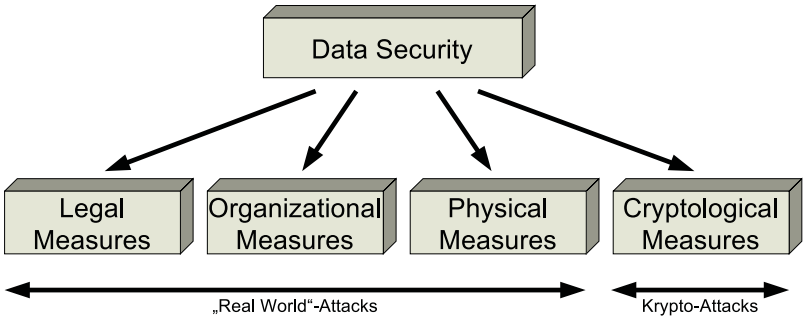


Fig. 1.1-1: The four different measures which contribute to data security.

The individual measures must be well coordinated to guarantee comprehensive data security. For example even the best cryptological methods are of no use when passwords are treated irresponsibly (missing or insufficient organizational measures) or everyone has free access to rooms with important hardware, servers or routers (missing or insufficient physical measures).

Obviously computers, networks and data have to be protected from the above mentioned threats in the “real world” as well as in cyberspace. Based on these thoughts this book “Foundations and Applications of Cryptology” deals with a “core technology of cyberspace” [Schneier04], the cryptography.

1.2 Cryptography

cryptography

Cryptography (from the Greek kryptós, "hidden" and gráphein, "to write") has been in existence almost as long as texts have been recorded in written form. It means "secret writing" or encryption. Today we consider cryptography to be a part of cryptology, the other part being cryptanalysis. Cryptanalysis is the science of finding weaknesses in cryptosystems.

cryptology

cryptanalysis

classical cryptography

transposition ciphers

substitution ciphers

In classical cryptography essentially two methods were used for encryption. The first method was the use of transposition ciphers (a cipher is a set of algorithms for encryption and decryption). When using this method the sequence of symbols in the message was altered. The second method was the substitution ciphers, which systematically replaced symbols or groups of symbols by other symbols or groups of symbols.

Example 1.2-1: Caesar Cipher
One of the most well-known and simplest examples of the substitution ciphers is the Caesar Cipher, in which each letter of the alphabet is replaced by another one, which is located a given number of places further back in the alphabet. Thus the letters of the alphabet are simply shifted. The number of places to be shifted serves as a "secret key" for the encoding and decoding of the message and is given in the form of a letter of the alphabet. For example if E (fifth letter in the alphabet) were the secret key, then the letter A would be substituted by the letter F and the letter B by the letter G. If one reaches the end of the alphabet

when counting the offset, then one has simply to continue counting from the beginning of the alphabet (modulo 26). Thus the letter Y is substituted by the letter E.

ciphertext

Encrypted texts, called ciphertexts, which were provided by one of the two methods mentioned above, can be decrypted relatively easily with the help of the frequency analysis (first known recordings were made in the 9.th century by the Arab Al-Kindi [Ibraham92]). In its simplest form the frequency of letters in unencrypted texts, called plaintexts and the ciphertext are compared. For example the letter E is most frequently represented in English-language texts. Thus if X appears with the highest frequency in the ciphertext it is highly probable that it must be replaced by the letter E for decryption. Frequency analysis is regarded as one of the first steps in cryptanalysis.

plaintext

modern cryptography

The change to modern cryptography was marked in 1949 by the paper “Communication Theory of Secrecy Systems” by C. E. Shannon [Shannon49a] and his other work on information and communication theory [Shannon49b], with which a solid theoretical basis for cryptography was established. Cryptography gained public interest in the seventies with the publication of the Data Encryption Standard (DES) as the official (non secret) encryption standard for the USA and with the paper "New Directions in Cryptography" by M. E. Hellmann and W. Diffie [Diffie76a]. In this paper one of the major problems of cryptography, the secure distribution of cryptographic keys was solved using the Diffie-Hellmann key exchange. Furthermore the development of asymmetric encryption methods was commenced.

Data Encryption Standard

Diffie-Hellmann key exchange

In the last decades the term cryptography has been expanded. Apart from encryption, the term now also covers authentication, digital signatures, access control, data integrity, data confidentiality and nonrepudiation.

Nowadays cryptography is, on the one hand, a mature technology, which is integrated in many applications and services and, on the other hand, an interdisciplinary research area where mathematicians, computer scientists and engineers participate. This interdisciplinary approach can be seen in the topic selection of this book, in particular in the chapters on the foundations of mathematics.

1.3 Overview of this Book

This book *Foundations and Applications of Cryptology* is structured as follows: Chapter 1 continues with an introduction to basic cryptographic primitives. Required basic mathematical principles from the number theory are explained comprehensively (also for non-mathematicians) in chapter 2. Chapter 3 (stream ciphers) and chapter 4 (block ciphers) deal with symmetric-key encryption techniques. Design and analysis of encryption systems as well as the most well-known encryption systems are introduced and the notions of stream and block ciphers further discussed. In particular binary sequences and linear feedback shift registers are looked at in chapter 3 and important algorithms which are based on block ciphers (IDEA,

DES) in chapter 4. The other form of encryption, public-key encryption is examined in chapter 5. Whereby, important algorithms (RSA, ElGamal, ECC) are looked at in detail. Chapter 6 is dedicated to digital signatures; various signature schemes like RSA, ElGamal, DSA and ECDSA are presented. In chapter 7 hash functions and authentication codes are introduced. Chapter 8 deals with entity authentication methods like password based, challenge-response, zero-knowledge, and biometrics. In chapter 9 key management techniques are handled. The last chapter is dedicated to basics of public key infrastructure (PKI) and important PKI-standards. The chapter closes with a list of references and recommended readings.

1.4 Symmetric-key Encryption

One of the most important goals of cryptography is confidentiality. It is a service used to keep secret the content of information from all but those authorized to have it. Here the objective of encryption is to alter the message in such a way that only authorized persons can decrypt and read the message. In order to decrypt the message one has to have a key. Depending on how the keys are used, we can subdivide encryption into two categories. These are:

1. symmetric-key encryption and
2. asymmetric-key encryption (which shall be handled in Section 1.5).

Characteristic of the symmetric-key encryption is the fact that both the sender and the receiver of an encrypted message have a common secret key k (see Fig. 1.4-1). In order to encrypt the message m, also referred to as plaintext, the sender uses the function E together with the key

$$c = E_k(m)$$

to obtain the ciphertext c. The ciphertext c is transmitted via a channel which is commonly accessible. The secret key k however has to reach the receiver via a secure channel. The receiver can now recover the original message m with the decryption function D from the ciphertext c with the aid of the secret key k:

$$m = D_k(c).$$

Thus the encryption function E has to be invertible.

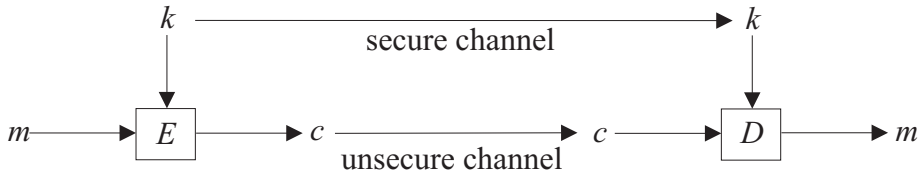


Fig. 1.4-1: Symmetric-key encryption system with encryption function E, decryption function D and secret key k (m = message, c = encrypted message).

When an encryption scheme is designed one should assume that E and D are known to the public, and obtaining the message m from ciphertext c merely depends on

principle of Kerckhoff

the secret key k (*principle of Kerckhoff*). In practice, the principle of Kerckhoff is not always used. That means that the encryption scheme is kept secret. There are two reasons for this: one can obtain an even higher security through this additional secrecy. This is of special importance when one wants to protect a system not only against cryptographic attacks but also against attacks on the hardware. Secondly, the use of a weak and inadequately examined algorithm is concealed through secrecy. When the technique is used in mass products it is better to assume that, in the long term, the algorithm cannot be kept secret.

Example 1.4-1: Simple encryption system
Let $M = (m_1, m_2, m_3)$ be the set of messages and $C = (c_1, c_2, c_3)$ the set of ciphertexts. There are precisely $3! = 6$ bijections from M to C. Each key k_i from the key space $K = (k_1, k_2, k_3, k_4, k_5, k_6)$ can at any time serve as a member of one of these bijections. Let, for example, the following bijection be assigned to key k_2:

$$E_{k_2}(m_1) = c_1, E_{k_2}(m_2) = c_3 \text{ and } E_{k_2}(m_3) = c_2 \ .$$

The decryption function D for key k_2 is then:

$$D_{k_2}(c_1) = m_1, D_{k_2}(c_2) = m_3 \text{ and } D_{k_2}(c_3) = m_2 \ .$$

When participant A wants to transmit a message $m = m_3$ to participant B, a key, for example k_2, is chosen and then interchanged in a secure and authentic way. Then A can encrypt the message $m = m_2$ with E and obtain the ciphertext

$$c_3 = E_{k_2}(m_2),$$

which is transmitted to B via an unsecured channel. Participant B who receives ciphertext c_3 and knows the secret key k_2 can compute the plaintext belonging to it by using D:

$$D_{k_2}(c_3) = m_2.$$

The way we defined the encryption function in the previous example is not very effective. For each key, one bijection was defined. It is more effective when the key is integrated in a function E through mathematical operations.

An encryption system is said to be secure when it is able to overcome the following attacks:

- Ciphertext-only attack:
  The attacker tries to deduce the decryption key or plaintext by only observing the ciphertext.

- Known-plaintext attack:
  The attacker has a quantity of plaintext and corresponding ciphertext and tries to determine the secret key k or decrypt further ciphertexts.

These two attacking scenarios can be expanded by variants like chosen-plaintext attacks or chosen-ciphertext attacks. Here the attacker can choose the plaintexts or ciphertexts he knows. Another variant is that the chosen texts can be picked out adaptively. These attacking scenarios are relevant for symmetric-key encryption as well as for asymmetric encryption.

block ciphers

The methods for symmetric-key encryption can be further divided into *block ciphers* and stream ciphers. Block ciphers divide the message m into blocks $m_1, m_2, \ldots$ of a fixed length. Typical values for the length of a block m_t are 64, 128 or 256 bits. Normally all blocks m_t are encrypted with the same key k and a ciphertext block c_t (of the same length) is built. A block cipher can be used in various working modes.

Feistel Ciphers

Many block ciphers like the DES belong to the group of *Feistel Ciphers*. The advantage of these ciphers is that one has a lot of freedom when designing the function E, and at the same time the guarantee that the decryption function D exists and that this function can be expressed explicitly.

DES
IDEA

Examples of well known and frequently used block ciphers are *DES* (Data Encryption Standard) and *IDEA* (International Data Encryption Standard). Both are algorithms with a block length of 64 bits. DES has a key length of 56 bits while IDEA has a key length of 128 bits. The days of the simple DES are numbered because the keys of length 56 bits used in DES are now vulnerable to exhaustive search attacks, which try out all possible keys. This is why the effectiveness of DES is increased through multiple application (for example Triple DES).

Example 1.4-2: Simple substitution ciphers
Let the plaintext symbols be the numbers from 0 to 9. Let the key k be a number between 1 and 9. The encryption function E transforms a plaintext number m and the key k as follows:

$$c = f(k, m) = m + k \bmod 10.$$

The decryption is carried out by subtracting the key k from the received ciphertext c:

$$m = f^*(k, c) = c - k \bmod 10.$$

Now let $k = 3$ and the message $m = 8$ is to be encrypted:

$$c = f(3, 8) = 11 \bmod 10 = 1.$$

Decryption:

$$m = f^*(3,1) = -2 \bmod 10 = 8.$$

Such a substitution cipher is not very secure because the key k can easily be detected through frequency analysis of symbols in the plaintext and in the ciphertext.

stream ciphers

Beside block ciphers there are *stream ciphers* in symmetric-key enryption. They do not encrypt the message block-wise, but symbol-wise and the currently used keystream alters from symbol to symbol. Formally, an additive stream cipher (see Fig. 1.4-2) is a function f which produces a keystream $z_t, 1 \leq t \leq l$ with the length $l > n$ from a secret key k with the length n. The message m, which is to be encrypted, has l message symbols. This is illustrated as the message symbol sequence

$$m = m_1, m_2, \ldots, m_l$$

and is combined with the keystream z_t to produce the ciphertext sequence c_t via a symbol-wise XOR operation:

$$c_t = z_t + m_t.$$

The decryption is carried out in such a way that the receiver of the encrypted message sequence c_t produces the same keystream z_t with the function f and the secret key k. The receiver recovers the message sequence m_t through a symbol-wise XOR of keystream z_t and ciphertext sequence c_t:

$$m_t = z_t + c_t.$$

The keystream sequence z_t is a pseudorandom bit sequence. It should, besides certain statistical features, correspond to further cryptographic standards.

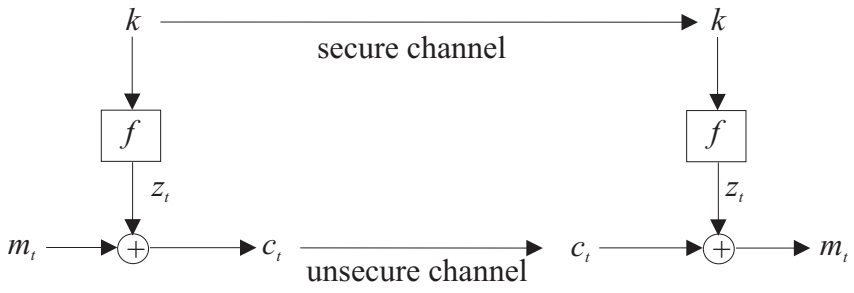


Fig. 1.4-2: Principle of an additive stream cipher.

One-time-pad

The prototype of all stream ciphers is the *One-time-pad*, where it is assumed that the keystream sequence z_t is a random sequence and the length n of the key k has to be at least l, where l is the length of the message. If the key is only used once then this technique is absolutely secure, and its security can even be proven theoretically. However, it has the disadvantage that the key has to be as long as the

plaintext. When the same key is used several times the system can be cracked with a known-plaintext attack. In practice it is unusual to use a One-time-pad. Instead a pseudorandom sequence generator is used for generating the keystream sequence z_t. Here the advantage is that only a short secret information is needed to initialize the pseudorandom sequence generator. As in the case of block ciphers, only a short secret information has to be transmitted from the sender to the receiver.

Example 1.4-3: Additive stream ciphers
A wants to transmit to B the encrypted message sequence

$$m = 0, 1, 0, 1, 1, 1, 0, 1$$

with an additive stream cipher. A and B choose a suitable keystream generator. Then they interchange via a secure and authentic channel a key k with which the keystream generator is initialized. Let the generator produce the keystream sequence

$$z = 1, 0, 1, 0, 1, 0, 1, 1$$

at A as well as at B as both have the common and secret key k. The ciphertext sequence c is obtained through a symbol-wise XOR with the message sequence m and the keystream sequence z:

$$c = m + z = 1, 1, 1, 1, 0, 1, 1, 0.$$

The ciphertext sequence c can now be transmitted via an unsecure channel and the participant B can recover the message sequence m through a symbol-wise XOR with the aid of the key stream sequence z:

$$m = z + c = 0, 1, 0, 1, 1, 1, 0, 1.$$

1.5 Asymmetric Encryption

A major problem which is inherent to the previously explained symmetric-key encryption techniques is the distribution and administration of the symmetric key k. When two participants A and B want to communicate, they must first exchange a secret key $k_{A,B}$. For this they must have a secure channel, so that the key can be transmitted secretly and with integrity. Integrity means, that the receiver can detect if the message was changed during the transmission. The higher the number of participants in a network that want to communicate, the more difficult the problem becomes. When there are N participants communicating with each other, each pair of participants must exchange a secret key. Hence

$$\frac{N(N-1)}{2}$$

keys must be transmitted and stored secretly. Another possibility is to carry out the whole communication via a central trusted third party. In this case only N key pairs have to be generated, distributed and stored. The key management is easier, when asymmetric techniques are used. Concepts and ideas of asymmetric cryptography are based on the research of W. Diffie and M. Hellmann in the mid 70s ([Diffie76a]). The first asymmetric encryption system was the RSA technique which was proposed in 1978 by R. Rivest, A. Shamir and L. Adleman ([Rivest78a]).

The principle of asymmetric encryption is as follows (see Fig. 1.5-1): each participant T of the system has a private key $k_d = k_{d,T}$ and a public key $k_e = k_{e,T}$ of T. k_d is kept secret and k_e is made public. Now, when a second participant wants to send a message m to participant T, he has to obtain the public key k_e, for example from an electronic directory similar to a telephone book. The encryption function E assigns the ciphertext to the message by using the key k_e:

$$c = E_{k_e}(m).$$

Participant T, to whom the ciphertext c is sent, uses his private key k_d on c with the decryption function D:

$$m' = D_{k_d}(c).$$

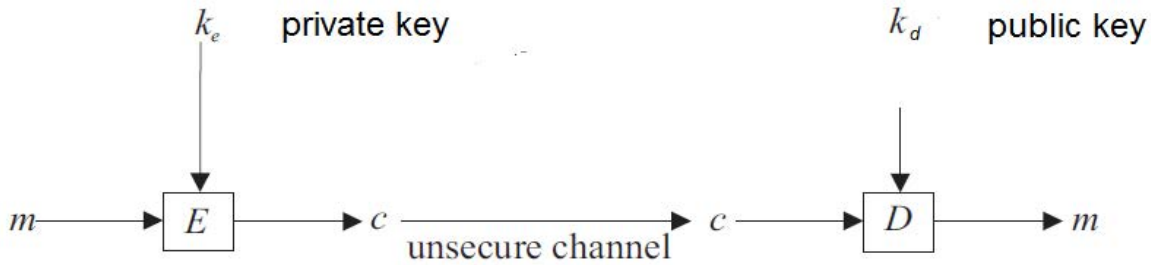


Fig. 1.5-1: Principle of asymmetric encryption.

The functions E and D must have the following properties:

1. correct decryption:
   the correct plaintext must be reproduced. This means that

   $$m' = D_{k_d}(c) = D_{k_d}(E_{k_e}(m)) = m$$

   for all plaintexts m,

2. asymmetric-key property:
   it is practically impossible to recover the private key k_d from the public key k_e. The same applies to the corresponding functions: it is practically impossible to deduce $D_{k_d}(.)$ from $E_{k_e}(.)$.

When the principle of asymmetric encryption was described, one could see that no keys had to be exchanged secretly for encrypted communication. But there is another problem: when someone wants to send a message to participant T, he must be sure that the public key of T, which he gets from a public key register, is the actual public key of T. If an attacker manages to replace the public key of T in the database with his own key or when he puts the key otherwise in circulation, he can decrypt all messages sent to T. Thus, the public keys must be authentic. This can be obtained by using secure registers or digital signatures and certificates.

Asymmetric encryption techniques not only facilitate key management and key exchange for users, but can also be used for digital signatures. Unfortunately the known asymmetric techniques are not as efficient as many symmetric-key encryption techniques. That is the reason for combining, in practice, asymmetric and symmetric-key techniques as a hybrid technique (see Fig. 1.5-2). When a message m is to be encrypted and transmitted, the sender initially produces a symmetric session key k. The session key is encrypted with the public key k_e of the receiver using an asymmetric technique E_{asy} and the message m is encrypted with the key k using a symmetric-key technique E_{sym}:

$$c_1 = E_{asy,k_e}(k)$$

and

$$c_2 = E_{sym,k}(m).$$

The receiver obtains the ciphertexts c_1 and c_2 and decrypts them in the following order:

$$k = D_{asy,k_d}(c_1)$$

and

$$m = D_{sym,k}(c_2),$$

where k_d is the secret key of the receiver for the asymmetric technique.

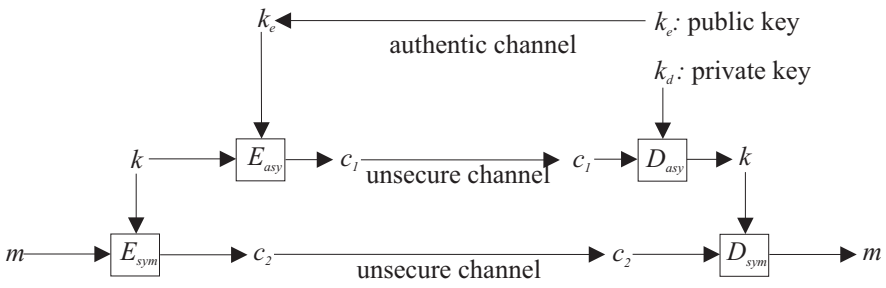


Fig. 1.5-2: Principle of a hybrid encryption system.

1.6 One-way Functions

One-way functions are a basic building block of cryptography. Many primitives like asymmetric cryptography, hash functions, digital signatures or pseudorandom bit generators are based on them. A one-way function $f : X \to Y$ is a function whose value $y = f(x)$ is easily computable, but the preimage x for essentially all y is computationally infeasible. The term *for essentially all* means that, for instance, there can be a table for a small number of preimages, which contains an x for a given y. The term *easily computable* means that y can be determined in polynomial time, and *computationally infeasible* means that x for a given y cannot be, on average, determined in polynomial time (see Fig. 1.6-1).

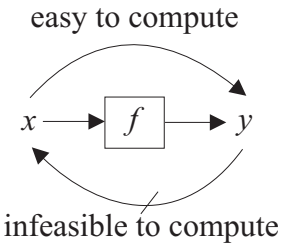


Fig. 1.6-1: Principle of a one-way function.

If f is a bijection ($f : X \to X$) it is also called a one-way permutation. A one-way function is called *collision-free* when it is practically impossible to find two different values x and x' in the preimage set X with $f(x) = f(x')$.

collision-free

The one-way functions introduced so far can be used by all participants equally. Modern cryptography requires a further concept, namely the preimage x can only be computed easily from y when a secret value is known. This concept is achieved by a trapdoor one-way function. A *trapdoor one-way function* $f : X \to Y$ is a one-way function for which there is a secret information so that the function is easily invertible. One example for this is to square modulo n:

trapdoor one-way function

$$f(x) = x^2 \bmod n,$$

with $n = pq$, where p, q are prime numbers. The computation of $y = f(x)$ can be carried out in polynomial time. Without knowing p and q, x from a given y cannot be determined in adequate time. When the factorization of n is known, effective algorithms exist to determine x. In this case the trapdoor information is that p and q are known.

1.7 Cryptographic Hash Functions

Cryptographic hash functions belong to the group of one-way functions (see Section 1.6). Hash functions are important elementary security mechanisms which are especially used to protect authentication and integrity of messages. A further example is the computing of electronic signatures where, instead of signing a message m, the cryptographic hash value $h(m)$ is signed.

A cryptographic hash function h is an algorithm which maps any message m on a hash value (test value) $h(m)$ of a fixed length. Without loss of generalization we only consider the binary alphabet with the symbols $\{0, 1\}$. Formally a hash function is then defined as

$$h : \{0,1\}^* \rightarrow \{0,1\}^n$$

where n gives the fixed length of the hash values. The value $h(m)$ should be efficiently determinable and it should be computationally infeasible to determine m from $h(m)$ (see Fig. 1.7-1).

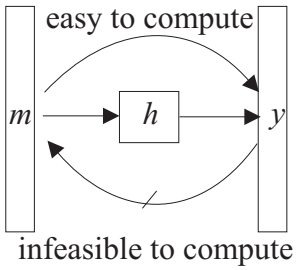


Fig. 1.7-1: Principle of a hash function.

collision
collision resistant

A *collision*(m, m') of h is a pair of messages for which $m \neq m'$ and $h(m) = h(m')$ is valid. A hash function h is called (weakly) *collision resistant* when it is difficult to find a collision (m, m') for a given m. Sometimes it is sufficient that the hash function used is collision resistant. Others, like ones used in electronic signatures for instance, need stronger properties. A hash function h is called strongly collision resistant (or collision-free) when it is practically impossible to find any collision (m, m').

Hash functions are usually applied on a block basis, i.e. a message m is divided into blocks of a fixed length (e.g. 64 bits) and each block is compressed using the hash function. The compressed values are concatenated to give the hash function $h(m)$. In this process the last block to be compressed might have to be filled up by zeros (i.e. padded) to give the full length (e.g. 64 bits).

The hash functions MD4 and MD5 were designed by R. L. Rivest and S. Dusse in 1990 and 1992 respectively. The abbreviation MD stands for message digest. MD4 provides hash values with a length of 128 bits. MD5 is a strengthened version of MD4. It is more complex than MD4, but similar in design. The developers explain that 2^{64} operations are required to find two distinct messages with the same hash value, and about 2^{128} operations to find a message yielding a pre-specified hash value. According to recently published attacking methods MD4 is no longer regarded as secure.

In 1992 the American National Institute of Standards and Technology (NIST) proposed to standardize a dedicated hash function, whose design was similar to the one of MD4 and MD5. This technique generates hash values with a length of 160 bit. In 1993 it was published as Federal Information Processing Standard (FIPS 180) and is now referred to as Secure Hash Standard (SHS).

When messages are transmitted, intentional or unintentional faults can occur. They can be caused by technical defects or failures of the communication technology, but also by aimed manipulations of attackers. In order to detect and correct unintentional faults, methods of coding theory are used. To each message word m of the length l a valid code word m' of the length $l', l' > l$ is assigned. With the information added to m, fault detection or even fault correction can be carried out. The code word m' is sent to the receiver, is received as code word m'' and is checked if it is a valid code word.

The methods of coding theory are not sufficient to protect messages from attackers because the attacker can send, instead of the valid code word m', another valid code word to the receiver. The receiver cannot detect the manipulation as it becomes a valid code word.

In order to protect oneself against manipulations, digital signatures or hash functions in combination with a symmetric key can be used. Digital signatures solve the problem with an asymmetric approach and have further useful properties. We will introduce a solution based on hash functions.

When a message m is to be transmitted from a sender to a receiver, a secret key k must be chosen and exchanged first, like it is in symmetric encryption systems. The sender has first to compute the hash value $h(m)$ of the message m and then encrypt it using the secret key k as $E_{sym,k}(h(m))$. Finally, the tupel $m, E_{sym,k}(h(m))$ is sent to the receiver. The receiver can check the correctness of the received message m'. For this purpose the receiver computes the hash function $h(m')$ and then uses the secret key k on $h(m')$ to calculate $E_{sym,k}(h(m'))$. The authentity of the message ($m = m'$) is given in case $E_{sym,k}(h(m)) = E_{sym,k}(h(m'))$[1].

1 In fact, the functions $E_{sym,k}$ and h must fulfill some requirements to provide message authentity. A description of these requirements is beyond the scope of this course. For more details, see [Menezes96a] on page 366.

1.8 Entity Authentication

Authentication is one of the most important of all information security objectives. Until the mid 1970s there was the general belief that secrecy and authentication were intrinsically connected. With the discovery of hash functions and digital signatures, it was realized that authentication and secrecy were truly independent and separate information security objects [Menezes96a].

Authentication attempts to solve following problems with cryptographic methods:

- How can I identify myself to another person, beyond all doubt?
- How can an information system check the access authorization of a user?
- How can I be sure that a message is originated from the indicated sender?

A typical example for entity authentication is to prove one´s identity to a computer, for instance an ATM (automated teller machine). The ATM must be sure that the person who has the credit card is actually the owner of that credit card. In this case a PIN (personal identity number) has to be keyed in.

In the real world we recognize people by their appearance, behavior or their voice. We perceive personal characteristics of persons to authenticate them. When a computer checks the personal characteristics of a person, this is called biometric authentication. Examples for this can be:

- Iris scan
- Fingerprint
- Face recognition
- Voice recognition
- Recognition by measuring typing speed or other behavioral biometrics.

In electronic systems these techniques usually do not work without problems or the costs of a biometric system might be too high. That is the reason, why in cryptography methods are examined, in which, for authentication, a person has to submit a secret information or has to prove that he is in possession of a certain information. Examples of secret information are:

- Password
- PIN
- Symmetric key
- Asymmetric key pair
- The solution of a problem which cannot be solved in polynomial time.

The different techniques which prove that one has a secret information can be classified according to the following criteria:

- Whether the secret must be transmitted directly to the verifier or if a value is sent to the verifier which is computed with the secret.
- If the secret is needed for verification.

Besides entity authentication discussed here, there is a need for message authentication, with which the origin and integrity of a document is proved. For this, digital signature or symmetric techniques are suitable. They are based on hash or encryption functions.

1.9 Key Management Techniques

cryptographic keys

Security services based on cryptographic mechanisms often assume cryptographic keys to be distributed to the parties which are involved in communication before securing the communication. The secure management of these keys is one of the most critical elements when integrating cryptographic functions into a system. Even the most ingenious security concept will be ineffective if the key management is weak.

Key management includes the

- generation,
- certification and authentication,
- establishment and distribution,
- escrow / recovery,
- storage, update and destruction,

of keying material. These topics will be discussed in detail in chapter 9. Key management techniques depend on the underlying cryptographic techniques, the intended use of the keys and the implied security policy. The appropriate protection of keys is subject to a number of factors, such as the type of application for which the keys are used, the threats they face, or the different states the keys may assume. Primarily, depending upon their type, keys have to be protected against disclosure, modification, destruction and replay.

1.10 Public Key Infrastructure

A major advantage of asymmetric key cryptography over symmetric key cryptography is that the key distribution problem is easier to solve. Symmetric key distribution systems are expensive and hard to manage. In high-security applications with imminent man-in-the-middle attacks, symmetric systems require expensive and cumbersome secure communication lines, face-to-face meetings or courier services. In asymmetric cryptosystems the public key can be distributed without the fear of compromising the secret private key. Nevertheless, key management in public key cryptography is still a difficult and complex issue.

Many currently emerging applications in the field of information technology rely on the principles of asymmetric key cryptography. The basic security related features that public-key systems can supply are confidentiality, data integrity, authentication, and non-repudiation. Typical real-world examples are:

Secure E-mail The need for a secure messaging environment for the Internet is of great importance. Although in the past the public awareness for the problems regarding insecure e-mail was very low, the spread of details about global surveillance systems immediately produced great concern about this issue.

Secure electronic payment At the moment, many payments in Internet-based e-commerce transactions are based on credit cards. Security of electronic credit card payments can be increased by applying asymmetric key cryptography. For example, a mechanism for authentication of involved parties (customers, merchants, banks) can be provided. Furthermore, the credit card and payment information should be encrypted during the transaction.

Access control The most common used prevailing method of access control in corporate and open networks is to employ weak authentication with passwords. Passwords that can be remembered (and thus be used) by human users, even if they have a reasonable length, normally have such a low entropy[2] that dictionary attacks are readily successful. Even though sophisticated methods for useful password selection do exist, these methods are often too cumbersome for casual users or users simply do not bother to use them. Hence, it can be advisable to replace low entropy passwords with large entropy asymmetric keys.

Authorization Allowing a user to access a computer system is a special form of authorization. Other forms of authorizations are, e.g. the authorization to provide medical advice over the Internet, the authorization to view the content of a video on demand stream, the authorization to spend money in the name of a company, etc. Such authorizations can be realized with so called authorization certificates, which bind a special form of authorization to a public key. The holder of the corresponding private key is then able to prove that he or she is allowed to carry out the certified action.

Electronic Signature The recent evolution of the Internet into an open and global communication platform has greatly stimulated electronic commerce and Internet-based business-to-business transactions. An increasing number of transactions are carried out online which leads to a demand for an electronic equivalent of traditional contracts. Especially politicians from the leading industry nations were under pressure from businesses and providers of e-commerce solutions to quickly adopt legislation of electronic signatures. Emerging electronic signature acts include the use of digital signatures as a legal replacement of hand-written signatures. In February 2001, the German 'Bundestag' approved the adoption of the European electronic signature directive. The directive uses the term *electronic signature* instead of *digital signature* and defines different types of electronic signatures.

2 Only random passwords have a maximal entropy.

> "Electronic signature" means data in electronic form which are attached to or logically associated with other electronic data and which serve as a method of authentication.
>
> Furthermore, the directive also defines an advanced electronic signature:
>
> "'Advanced electronic signature' means an electronic signature which meets the following requirements:
>
> a. it is uniquely linked to the signatory,
> b. it is capable of identifying the signatory,
> c. it is created using means that the signatory can maintain under his sole control and
> d. it is linked to the data to which it relates in such a manner that any subsequent change of the data is detectable."
>
> In practice, this will most often be achieved by digital signatures with explicit support of non-repudiation.

It can be seen from the examples above, that asymmetric key cryptography is applied in diverse disguises. For all these applications to effectively work, sophisticated key management and distribution systems have to be constructed. The key management system for applications of asymmetric key cryptography is called *public-key infrastructure (PKI)*. A typical PKI consists of hardware, software, the people working to administer and maintain the infrastructure, as well as policies regarding security, privacy and liability. In chapter 10 we will have a look at the basics of public-key infrastructures. We introduce identity certificates, explain certification structures and give an overview on important PKI standards, like X.509, PKIX or OpenPGP.

1.11 Recommended Literature

There are many good reference books on cryptography. Here we mention some books which can be useful for further study.

[Menezes96a]: An extensive reference book for cryptography. Basic mathematical principles as well as cryptographic primitives and protocols are illustrated in detail.

[Buchmann99a]: A detailed illustration of basic mathematical principles for cryptography, especially of the number theory. All important primitives of cryptography are illustrated in detail. It also offers separate chapters for the factorization of integer numbers, prime number tests and solutions to the discrete logarithm problem.

[Schneier96a]: like [Menezes96a] it is very extensive, but less formal and systematic.

[Stinson95b]: A book that treats the most interesting topics of cryptography. Many examples and excercises.

2 Mathematical Background

The algebraic part of this chapter is intended as an introduction to some fundamental algebraic systems such as groups, rings and fields. For further reading we recommend the following books: [Herstein86] for abstract algebra, [Lidl94] for fundamental algebra and finite fields, [Koblitz94], [Jackson87] for an introduction in number theory, [Buchmann99], [Menezes96], [Koblitz98] for algorithms in number theory and finite fields.

2.1 Sets and Mappings

An algebraic system can be described as a set of objects together with some operations for combining them. In our short introduction we first begin with the notion of

set the *set* S as a collection of objects, called *elements* of S. The elements of the set S can be combined, in one or several ways, for obtaining once more, elements of this

operations set S. These ways for combining elements of S are called *operations* on S. Then we try to condition or regulate the nature of S by imposing certain rules or axioms and

axioms define the particular structure on S. These *axioms* act as a license to reach certain mathematical objectives.

2.1.1 Set Theory

Here we only give a short overview of the basic notions and operations from set theory. As mentioned above, a set S can be described as a collection of distinct

elements objects called *elements* of S.

NOTE

The following notation will be used throughout:

1. $\mathbb{N}$ denotes the set of natural numbers; that is, the set $\{0,\ 1,\ 2, \ldots\}$.
2. $\mathbb{Z}$ denotes the set of integers; that is, the set $\{\ldots,\ -2,\ -1,\ 0,\ 1,\ 2, \ldots\}$.
3. $\mathbb{Q}$ denotes the set of rational numbers; that is, the set $\{\frac{a}{b} \mid a, b \in \mathbb{Z},\ b \neq 0\}$.
4. $\mathbb{R}$ denotes the set of real numbers.

To denote that a given element a is an element of S, we write $a \in S$. The set T

subset will be said to be a *subset* of the set S, if every element of T is an element of S. In this case, if $a \in T$ implies that $a \in S$, we write $T \subset S$. In terms of the basic

equal sets terminology of sets, two sets S and T are *equal*, (written $S = T$) if they consist of the same set of elements, i.e. $T \subset S$ and $S \subset T$. Thus, the standard method for demonstrating the equality of two sets is to prove that these two relations hold for them. In contrast to the previous case, if at least one of the opposite containing relations is not verified, the two sets S and T are said to be *not equal* and this will

be denoted by $T \neq S$. A subset T forms a *proper subset* of S if $T \subset S$ and $T \neq S$, i.e. if S contains more elements than T. As a particular set, the *empty* set is a set having no elements. It is a subset of every set and is denoted as $\emptyset$.

empty set

Supposing that the sets A and B are subsets of a given set S, we now deal with methods for constructing other subsets of S from A and B. In this regard we introduce the *union* of A and B denoted as $A \cup B$, as the subset of S containing elements of S that are elements of A or elements of B. The "or" we have just used means that the elements of $A \cup B$ can be contained in A, in B or in both sets. By *intersection* of A and B, written $A \cap B$, we mean the subset of S consisting of those elements that are both in A and in B. Two sets A and B are said to be *disjoint*, if their intersection is empty. Although the union and intersection are defined for two sets, they can be generalized to an arbitrary number of them.

union

intersection

The third operation we can perform on sets is the *difference* of two sets A and B. The difference is denoted by $A - B$ and consists of the set of elements that are in A but not in B. Under the consideration that A is a subset of the set S, the difference $S - A$ defines the complement of A in S and is denoted by A'. Another construction we can realise on the sets A and B is the *cartesian product*, written $A \times B$. The set $A \times B$ defines the set of all ordered pairs (a, b) where $a \in A$ and $b \in B$. We write

difference

cartesian product

$$A \times B = \{(a, b) \mid a \in A,\ b \in B\}.$$

Two pairs (a_1, b_1) and (a_2, b_2) are said to be equal if and only if $a_1 = a_2$ and $b_1 = b_2$. Generally, the cartesian product can be defined on an arbitrary number of sets $A_1, A_2, \ldots, A_n$. We write

$$A_1 \times A_2 \times \ldots A_n = \{(a_1, a_2, \ldots, a_n) \mid a_i \in A_i\}.$$

We now consider the cartesian product of a set A with itself, $A \times A$. If the set A is finite with n elements, then the set $A \times A$ is also finite and contains n^2 elements. Instead of referring to subsets of the cartesian product, we can refer to *relations*.

A *binary relation* R on A itself is a subset of the cartesian product $A \times A$, i.e. $R \subset A \times A$. The elements $a, b \in A$ are said to be related (a is related to b), if $(a, b) \in R$. We further denote a binary relation on the set A as $\sim$. The relation $\sim$ can have one or more of the following properties:

binary relation

1. Reflexivity: For all $a \in A$, $a \sim a$ (i.e. $(a, a) \in R$).
2. Symmetry: For all a, $b \in A$, $a \sim b$ implies $b \sim a$ (i.e if $(a, b) \in R$, then $(b, a) \in R$).
3. Transitivity: For all a, b, $c \in A$, $a \sim b$ and $b \sim c$ implies $a \sim c$ (i.e. $(a, b) \in R$ and $(b, c) \in R$ implies $(a, c) \in R$).

The binary relations which possess all three features mentioned above are very important. Such relations are called *equivalence relations*. A formal definition follows:

equivalence relations

Definition 2.1-1: *A binary relation $\sim$ on the set A is said to be an equivalence relation on A, if it is reflexive, symmetric and transitive.*

Example 2.1-1:
Let S be the set of all integers. Given $a, b \in S$, define $a \sim b$ if $a - b$ is an even integer. We can show that this is an equivalence relation on S.

1. Since $0 = a - a$ is even, $a \sim a$ (reflexivity).
2. If $a \sim b$, that is, if $a - b$ is even, then $b - a = -(a - b)$ is also even, thus $b \sim a$ (symmetry).
3. If $a \sim b$ and $b \sim c$, then both $a - b$ and $b - c$ are even, thus $a - c = (a - b) + (b - c)$ is also even, proving that $a \sim c$ (transitivity).

equivalence class

After the definitions of binary relation and equivalence relation, we now introduce the concept of an *equivalence class* which plays an extremely important role in mathematics.

Definition 2.1-2: *If A is a set and $\sim$ is an equivalence relation on A, then the equivalence class of $a \in A$ is the set $\{x \in A \mid a \sim x\}$. It is denoted by $cl(a)$.*

In Example 2.1-1, the equivalence class of a consists of all the integers of the form $a + 2m$, where $m = 0, \pm 1, \pm 2, \ldots$. In this example there are only two distinct equivalence classes, namely, $cl(0)$ and $cl(1)$.

2.1.2 Mappings

function, mapping

One basic concept in mathematics is that of a *function* or *mapping* from one set to another. The definition of a function from one set to another can be given in a formal way in terms of a subset of a cartesian product of these sets:

Definition 2.1-3: *If S and T are nonempty sets, then a mapping from S to T is a subset M of $S \times T$ so that for every $s \in S$ there is a unique $t \in T$ so that the ordered pair (s, t) is in M.*

Informally defined, a mapping (or function) f from one set S to another set T is a rule that assigns to each element $s \in S$ a unique element $t \in T$, i.e. given an element s of the set S, there is only one element t in T that is associated to s by the mapping f. The mapping f from S to T will be denoted by $f : S \to T$. The *image* of $s \in S$ is the element $t \in T$ which mapping f associates with s. It is denoted as $t = f(s)$.

image

We now consider the inverse case: given a mapping $f : S \to T$ and a subset $A \subset T$, the set $B = \{s \in S \mid f(s) \in A\}$ is called the *inverse image* of A under f and

inverse image

is denoted by $f^{-1}(A)$. $f^{-1}(t)$, the inverse image of the subset $\{t\} \subset T$ consisting only of the element $t \in T$ is of particular interest.

Now that we have briefly looked at mapping we shall single out some of them.

1. Let S be an arbitrary nonempty set. A mapping of $S \times S \to S$ is called a *binary operation* on S. **binary operation**
2. Let T be a nonempty set, and let $S = T \times T$ be the cartesian product of T with itself. The function $f : T \times T \to T$, defined by $f(t_1, t_2) = t_1$, is called the *projection* of $T \times T$ onto its first component. **projection**
3. Let S and T be nonempty sets, and let t_0 be a fixed element of T. The function $f : S \to T$ defined by $f(s) = t_0$ for every $s \in S$ is called a *constant function* from S to T. **constant function**
4. Let S be a nonempty set. The function $f : S \to S$, defined by $f(s) = s$ for every $s \in S$, is called the *identity function* on S. **identity function**
5. Two mappings f and g from one set S to another are declared as *equal*, if and only if $f(s) = g(s)$ for every $s \in S$. **equal mappings**

Now we introduce some types of mappings by the way they behave.

Definition 2.1-4: *The mapping $f : S \to T$ is onto or surjective if every $t \in T$ is the image under f of some $s \in S$; that is, if and only if, given any $t \in T$, there is an $s \in S$ so that $t = f(s)$.* **surjective mapping**

Definition 2.1-5: *A mapping $f : S \to T$ is one to one (written 1-1) or injective if for $s_1 \neq s_2$ in S, $f(s_1) \neq f(s_2)$ in T. In other words, f is 1-1 if $f(s_1) = f(s_2)$ implies that $s_1 = s_2$ (distinct objects have distinct images).* **injective mapping**

Definition 2.1-6: *The mapping $f : S \to T$ is said to be a 1-1 correspondence or bijection if f is both 1-1 and onto.* **bijective mapping**

If we have a mapping $f : S \to T$ and a mapping $g : T \to U$, we can introduce an operation of combining mappings under certain circumstances. The most obvious combination of f and g is to form their product defined by the mapping $g \circ f$ from S into U, so that $(g \circ f)(s) = g(f(s))$ for every $s \in S$. The mapping $g \circ f : f \to g$ is denoted as *composition* or *product* of the mappings f and g. **composition of mappings**

For three mappings $f : S \to T$, $g : T \to U$ and $h : U \to V$, the general *associative law* holds, i.e $h \circ (g \circ f) = (h \circ g) \circ f$.

2.2 Groups, Rings and Fields

In this section we introduce some topics of elementary algebra. Especially the basic definitions and properties of groups, rings and fields will be considered.

2.2.1 Groups

The theory of groups is one of the oldest parts of abstract algebra as well as one particularly rich in applications. In order to give a formal definition of a group, we first introduce the notion of a *semigroup*.

semigroup

Definition 2.2-1: *A semigroup is a pair $(G, *)$ of a set G and an operation $*$ with the following properties:*

1. *$*$ is a binary operation on the set G, i.e.$* : G \times G \to G$.*
2. *$*$ is an associative operation i.e. for all a, b, $c \in G : (a * b) * c = a * (b * c)$.*

A semigroup $(G, *)$ is called *commutative*, if the binary operation $*$ on G is commutative, i. e. for all a, $b \in G$, $a * b = b * a$.

Definition 2.2-2: *A pair $(G, *)$ is called a group, if it satisfies the following axioms:*

1. *$(G, *)$ is a semigroup.*
2. *There is an element $e \in G$, called identity element so that for all $a \in G : a * e = e * a = a$.*

identity element

3. *For each $a \in G$, there is an element $a^{-1} \in G$, called inverse of a, so that: $a * a^{-1} = a^{-1} * a = e$.*

inverse element

The group $(G, *)$ is an abelian (or commutative) group, if the group operation $*$ is commutative.

The associative law guarantees that expressions such as $a_1 * a_2 * * * a_n$ with $a_j \in G, 1 \leq j \leq n$, are unambiguous, since no matter how we insert parentheses, the expression will always represent the same element of G. To indicate the n-fold composite of an element $a \in G$ with itself, where $n \in \mathbb{N}$, we shall write $a^n = aa \cdots a \, (n \, factors \, a)$ if we use multiplicative notation. We call a^n the nth power of a. If we use additive notation for the operation $*$ on G, we write $na = a + a + \cdots + a$ (n summands a).

Following the customary notation, we have the following rules:

Multiplicative Notation	Additive Notation
$a^{-n} = (a^{-1})^n$	$(-n)a = n(-a)$
$a^n a^m = a^{n+m}$	$na + ma = (n + m)a$
$(a^n)^m = a^{nm}$	$m(na) = (mn)a$

For $n = 0 \in \mathbb{Z}$, one adopts the convention $a^0 = e$ in the multiplicative notation and $0a = e$ in the additive notation, where the last "zero" represents the identity element e of G.

Example 2.2-1:

1. Let G be the set of integers with the usual addition + as the group operation. The ordinary sum of two integers is a unique integer and the associativity is a familiar fact. The identity element is $e = 0$ (zero), and the inverse of an integer a is the integer $-a$, because $a + (-a) = (-a) + a = e = 0$. We denote this group by $\mathbb{Z}$.

2. The set consisting of a single element {e}, with the operation $*$ defined by $e = e * e$, forms a group. The sole element of the group {e} is the identity element and its own inverse element.

3. Let G be the set of remainders of all the integers after division by 6 – that is, $G = \{0, 1, 2, 3, 4, 5\}$ – and let $a * b$ be the remainder of division by 6 of the ordinary sum of a and b. The existence of an identity element and of inverses is again obvious. We can see that 0 is the identity element of this group G, since for each element a of the set $G = \{0, 1, 2, 3, 4, 5\}$, the remainder of division by 6 of the ordinary sum of a and 0 delivers a, i.e. for $a \in G$, $a * 0 = 0 * a = a$. Each element $a \in G$ possesses an inverse in G, e.g. the element 1 of G has 5 of G as inverse, since the remainder of division by 6 of the ordinary addition of 1 and 5 delivers the identity element 0. In the same manner we find out that the element 0, 2, 3, 4 and 5 respc. have the inverses 0, 4, 3, 2 and 1. It requires some computation to establish the associativity of $*$. This group can be readily generalized by replacing the integer 6 by any positive integer n.

Another natural characteristic of a group G is the number of elements it contains. If this number is finite, the group G is said to be a *finite group*. The number of elements in a finite group is called the *order* of G and is denoted by $|G|$.

finite group

order of the group

It might be desirable in some interesting cases to deal with appropriate parts of the group G, which are smaller, over which we have some control, and are so that the information gathered about them can be used to get relevant information and insight about G itself. Note here that the behaviour of these parts of G depends on the group operation $*$. The parts of G are called *subgroups* and will be defined in what follows.

subgroup

Definition 2.2-3: *A nonempty subset M of a group G is said to be a subgroup of G if, under the group operation $*$, M itself forms a group.*

Example 2.2-2:
Let M be a set of even integers, i.e $M \subset \mathbb{Z}$. The set M closed under the ordinary addition + forms a group, since

1. if $a, b \in M$, then $a + b \in M$. That is, if a and b are even, then $a + b$ is even, and
2. the set M possesses the element 0 as identity element, and
3. every element a of M possesses an inverse $(-a)$ so that $a + (-a) = (-a) + a = 0$.

Subsequently M is a subgroup of the group $(\mathbb{Z}, +)$, because $M \subset \mathbb{Z}$ and $(M, +)$ is a group.

We now consider the order and some further properties of the group elements.

order of group element

Definition 2.2-4: *Let g be an element of the group G. The order of g is the least positive integer δ so that $g^\delta = e$, provided that such an integer exists. The order of g is defined to be infinite, if δ does not exist.*

We shall introduce the set

$$\langle g \rangle = \{g^k \mid k \in \mathbb{Z}\}.$$

generator

Definition 2.2-5: *If $g \in G$ exists so that $\langle g \rangle = G$, G is said to be cyclic and g is a generator of G.*

Example 2.2-3:
We consider again the group G of item 3 of Example 2.2-1. We remember that $G = \{0, 1, 2, 3, 4, 5\}$ and that the group operation $*$ is defined to be the remainder on division by 6 of the ordinary sum of a and b. We now want to determine the set $\langle 1 \rangle$. It must be clear that the expression g^k, introduced in the definition of $\langle g \rangle$ with $g \in G$, is relative to the used group operation. In our example, we have:

$$g^k = g * g * \ldots * g \, (k \, multiple).$$

So, we get:

$$\begin{aligned} 1 * 0 &= 1 \\ 1 * 1 &= 2 \\ 1 * 1 * 1 &= 3 \\ 1 * 1 * 1 * 1 &= 4 \\ 1 * 1 * 1 * 1 * 1 &= 5 \\ 1 * 1 * 1 * 1 * 1 * 1 &= 0 \\ 1 * 1 * 1 * 1 * 1 * 1 * 1 &= 1 \\ &\vdots \end{aligned}$$

If we continue this computation, we always obtain an element of the group G as a result. Thus, we can write:

$$\langle 1 \rangle = \{0, 1, 2, 3, 4, 5\} = G.$$

We conclude that 1 is a generator of the group G and that G is a cyclic group.

2.2.2 Rings

In most number systems used in elementary arithmetics there are two distinct binary operations: addition and multiplication. Examples are provided by the integers, the rational numbers, and the real numbers. We now define a type of algebraic structure known as a *ring* that shares some of the basic properties of these number systems.

ring

Definition 2.2-6: *The triple $(R, *, \circ)$ is called a ring, if the following properties are verified:*

1. *$(R, *)$ is an abelian group with identity element 0_R. 0_R is called the zero element of the ring.*

   zero element

2. *The group operation $\circ$ is associative, that is, $(a \circ b) \circ c = a \circ (b \circ c)$ for all $a, b, c \in R$.*

3. *There is an element $1_R \in R$ so that for each element $a \in R : a \circ 1_R = 1_R \circ a = a$. 1_R is the multiplicative identity of the ring R.*

4. *For all $a, b, c \in R, a \circ (b * c) = (a \circ b) * (a \circ c), (a * b) \circ c = (a \circ c) * (b \circ c)$, i.e. the operation $\circ$ is distributive over $*$.*

The ring $(R, *, \circ)$ is a commutative ring, if the operation o is commutative. An element $a \in R$ of the ring $(R, *, \circ)$ is *invertible or unit,* if it is invertible in $(R, \circ)$. The set of unit elements in a ring $(R, *, \circ)$ forms a group under the operation o which is called the group of the units of R and is denoted by $(R^*, \circ)$.

Example 2.2-4:
The set of integers $\mathbb{Z}$ with the usual operations of addition and multiplication is a commutative ring, since

1. $(\mathbb{Z}, +)$ is an abelian group with zero element 0, and
2. the group operation $\cdot$ is associative, that is, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in \mathbb{Z}$, and
3. 1 is the multiplicative identity element of the ring $\mathbb{Z}$, since for each element $a \in \mathbb{Z} : a \cdot 1 = 1 \cdot a = a$, and
4. for all $a, b, c \in \mathbb{Z}$, $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$, $(a + b) \cdot c = (a \cdot c) + (b \cdot c)$, i.e. the operation $\cdot$ is distributive over $+$, and
5. the operation $\cdot$ is commutative.

2.2.3 Fields

field The next algebraic system we will consider is a *field*. It plays an important role in geometry, in the theory of equations, and in certain parts of number theory.

Definition 2.2-7: *A commutative ring $(K, *, \cdot)$ is a field, if for all $a \in K$, with $a \neq 0_K$ (0_K is the zero element of the set K), a is invertible or a unit.*

Example 2.2-5:

1. $(\mathbb{Z}, +, \cdot)$ is not a field, since the only non-zero integers, which possess inverse related to the multiplication $\cdot$ are 1 and -1.
2. $(\mathbb{R}, +, \cdot)$ is a field with $\mathbb{R}$ being the set of real numbers, $+$ and $\cdot$ are the ordinary operations: addition and multiplication.

2.3 Number Theory

The present section is intended as an introduction to the basic concepts of number theory. Several central topics in this section such as prime numbers and congruences are very important not only in cryptography, but in many scientific fields.

2.3.1 Divisibility

First we introduce the notion of *divisibility* of integers. **divisibility**

Definition 2.3-1: *Let $a, b \in \mathbb{Z}$. We say that a divides b if there is an integer n so that:*

$$b = an.$$

In this case a is said to be a divisor of b and b is a multiple of a. The divisibility of b by a is denoted by $a \mid b$. To indicate that a is not a divisor of b, we write $a \nmid b$

Example 2.3-1:
$5 \mid 125$ and $-8 \mid 360$, since $125 = 5 \cdot 25$ and $360 = (-8) \cdot (-45)$.

$3 \nmid 5$.

Theorem 2.3-1: *If a and b are integers with $b > 0$, then there are integers q and r, with $0 \leq r < b$, so that $a = qb + r$.*

In the previous theorem, the integer q can be determined as:

$$q = \lfloor a/b \rfloor = \max\{\alpha \in \mathbb{Z}; \alpha \leq a/b\},$$

where q is the *quotient* of the division of a by b. The symbol $\lfloor a/b \rfloor$ denotes the largest integer less than or equal to a/b. The integer r is called the *remainder* of the division of a by b and is denoted by $a \mod b$. **quotient** **remainder**

Example 2.3-2:
If *a = 18* and *b = 4*, then *q = 4* and $r = 2$.

2.3.2 Representation of integers in different bases

Positive integers can be expressed in various ways. The ordinary representation is used in the decimal system. For example, the integer 234 can be written in base 10 as follows:

$$234 = 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0.$$

The same number can be written in base 6 as:

$$234 = 1 \cdot 6^3 + 0 \cdot 6^2 + 3 \cdot 6^1 + 0 \cdot 6^0 = (1030)_6,$$

or in base 2 as:

$$234 = 1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = (11101010)_2.$$

Generally, each integer $a \geq 0$ has a *unique representation in base*

representation of an integer in base b

$b\,(b \in \mathbb{Z}, b > 1)$. We write

$$a = a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + a_{n-2} \cdot b^{n-2} + \ldots + a_1 \cdot b^1 + a_0 \cdot b^0 = \sum_{i=0}^{n} a_i b^i,$$

where $0 \leq a_i < b$, for $0 \leq i < n$ and $a_n \neq 0$. We write

$$a = (a_n a_{n-1} a_{n-2} \ldots a_1 a_0)_b.$$

Example 2.3-3:

1. For machine computations, it is preferable to write the positive integers in the binary representation $(b = 2)$. For example the integer 47 will be represented as follows:

   $47 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = (101111)_2$.

2. The integer 47 can be expressed in base 8 as follows:

   $47 = 5 \cdot 8^1 + 7 \cdot 8^0 = (57)_8$.

   Since $8 = 2^3$, we can convert the integer 47 from packets of base 2 to base 8. For this purpose we group three digits in the binary representation of 47 from the right. We obtain:

   $47 = ((101)_2(111)_2)_8 = (57)_8$.

2.3.3 Greatest common divisor

common divisor

Having introduced the concept of a divisor of an integer, we will now deal with the concept of the *common divisor* of two (or more) integers.

greatest common divisor

Given two integers a and b, the integer c is a common divisor of a and b, if $c \mid a$ and $c \mid b$. A notion of *greatest common divisor* is a very important one. It is the largest possible common divisor of a and b and is formally defined as:

Definition 2.3-2: *Let a, b, c and $d \in \mathbb{Z}$ with $d > 0$. d is the greatest common divisor of a and b denoted $d = gcd(a, b)$ if:*

1. $d \mid a$ and $d \mid b$

2. whenever $c \mid a$ and $c \mid b$, then $c \mid d$.

Example 2.3-4:
The common divisors of 12 and 18 are ± 1, ± 2, ± 3, ± 6, and $\gcd(12, 18) = 6$.

$\gcd(20, 30) = 10$ and $\gcd(-12, 8) = 4$.

We notice that the greatest common divisor $\gcd(a, b)$ of two integers a and b always exists and is unique. Furthermore, it can be written as a linear combination of a and b. This combination is however, not unique. For instance,

$\gcd(24, 9) = 3 = 3 \cdot 9 + (-1) \cdot 24 = (-5) \cdot 9 + 2 \cdot 24.$

Theorem 2.3-2: *If $a, b \in \mathbb{Z}$, are not both 0, then their greatest common divisor $\gcd(a, b)$ exists, is unique, and moreover can be written as linear combination of a and b, i.e $\gcd(a, b) = xa + yb$ for some suitable integers x and y.*

How does one go about finding the greatest common divisor of two integers a and b? The simplest, but inefficient way, is to find all common divisors of a and b and then choose the greatest of them. Another way is to employ the *Euclidean algorithm* which is a nontrivial efficient old algorithm. It will be discussed in the next section.

2.3.4 Euclidean Algorithm

Euclidean algorithm

The *Euclidean algorithm* is an efficient algorithm for computing the greatest common divisor of two integers that does not require the factorization of the integers. It is based on the following simple theorem.

Theorem 2.3-3: *Let $a, b \in \mathbb{Z}$. Then,*

1. *if $b = 0$, then $\gcd(a, b) = |b|$,*
2. *if $b \neq 0$, then $\gcd(a, b) = \gcd(|b|, a \, mod \, |b|)$.*

The previous theorem enables the computation of the greatest common divisor $\gcd(a, b)$ as follows:

We suppose $r_0 = a, r_1 = b$ and $a > b > 0$, then we introduce the notation

$r_{i+1} = r_{i-1} \mod r_i$ for each integer $i \geq 1$ and $r_i \neq 0$.

Then, we compute $r_{i+1} = r_{i-1} \mod r_i$ for $i = 1, 2, 3 \ldots$ until we obtain for a fixed $i_0 \geq 1$:

$$r_{i_0+1} = 0.$$

Then, the greatest common divisor is r_{i_0}.

If $a = 0$ resp. $b = 0$, then $\gcd(a, b) = b$ resp. $\gcd(a, b) = a$.

Example 2.3-5:
We want to determine $\gcd(110, 40)$. Using the notation introduced above, we obtain the following table:

i	0	1	2	3	4
r_i	110	40	30	10	0

From the table we get $r_4 = 0$, i.e. $i_0 = 3$ and $r_3 = 10$ is the greatest common divisor of 110 and 40, i.e.

$\gcd(110, 40) = 10$.

2.3.5 Extended Euclidean Algorithm

The Euclidean algorithm can be extended so that it not only yields the greatest common divisor of two integers a and b, but also integers x and y satisfying the linear combination:

$$ax + by = \gcd(a, b).$$

Extended Euclidean algorithm

This algorithm is called the *Extended Euclidean algorithm* and is very important, since it can be used to compute a multiplicative inverse in groups (see Section 2.3.7).

Corollar 2.3-1: *For all $a, b, n \in \mathbb{N}$, the equation $ax + by = n$ has two integers x and y as solution if $\gcd(a, b)$ divides n.*

This corollar means that the equation $ax + by = \gcd(a, b)$ is always solvable.

Given two integers a and b as input, with the Extended Euclidean algorithm the two unknown integers x and y as well as the greatest common divisor of a and b can be determined so that

$$ax + by = \gcd(a, b).$$

This will be illustrated in what follows:

Let $r_0 = a, r_1 = b, r_2 = a \mod b$ and $q_1 = \lfloor \frac{a}{b} \rfloor$.

If $r_2 \neq 0$, then:

$$r_3 = r_1 \mod r_2 \text{ and } q_2 = \lfloor \frac{r_1}{r_2} \rfloor.$$

Generally, we continue with this notation until $r_i = 0$:

$r_{i+1} = r_{i-1} \mod r_i$ and $q_i = \lfloor \frac{r_{i-1}}{r_i} \rfloor, 1 \leq i \leq n.$

We start with $x_0 = 1, y_0 = 0, x_1 = 0, y_1 = 1$ and compute in every further iteration x_{i+1} and y_{i+1} as:

$$x_{i+1} = q_i x_i + x_{i-1},$$
$$y_{i+1} = q_i y_i + y_{i-1}$$

for $1 \leq i \leq n$. Let n be defined so that $r_{n+1} = 0$. Then, the integers x and y (the solutions of the equation) can be computed as:

$$x = (-1)^n x_n$$
$$y = (-1)^{n+1} y_n.$$

Example 2.3-6:
Given $a = 146$ and $b = 20$, we want to solve the equation $146 \cdot x + 20 \cdot y = \gcd(146, 20)$ with the three unknowns x, y and $\gcd(146, 29)$.

The following table shows the steps of the Extended Euclidean algorithm to solve the equation:

i	0	1	2	3	4
r_i	146	20	6	2	0
q_i		7	3	3	
x_i	1	0	1	3	
y_i	0	1	7	22	

From the table we obtain $n = 3$, since $r_4 = 0$. Then we get:

$x = (-1)^3 x_3$ and
$y = (-1)^{3+1} y_3$.

Regarding the table, we have $x_3 = 3$ and $y_3 = 22$. So $x = -3$ and $y = 22$. Thus,

$$\gcd(146, 20) = 146 \cdot (-3) + 20 \cdot 22 = 2.$$

2.3.6 Prime numbers

We now focus our attention on an ultra-important class of positive integers, namely the *prime numbers* or simply the primes. These numbers play a major role in cryptography, since they are essential for the realization of many asymmetric algorithms and other cryptographic protocols. For example the RSA public-key system uses prime numbers (see chapter 5). In this section, we introduce prime numbers and give an overview of their properties which are relevant for cryptographic applications.

prime numbers

Prime numbers are numbers that have no divisors other than 1 and themselves, such as 2, 3, 5, 7, 11, 13, 17, etc. All primes are odd, except 2 – the "oddest" prime. The number 1 is considered neither prime nor *composite*. A formal definition of primes will be given in what follows.

composite

Definition 2.3-3: *An integer $p > 1$ is said to be prime if its only positive divisors are 1 and p. Otherweise p is called composite.*

Example 2.3-7:
The numbers 2, 11, 13 and 2269 are prime. The numbers 4, 6 and 2268 are composite.

Can we build integers from primes in a very precise and well-defined manner? This question is made clear by one important theorem called the *fundamental theorem of arithmetic*.

Theorem 2.3-4: *Every integer $n \geq 2$ can be written as product of prime powers:*

$$n = \prod_{i=1}^{k} p_i^{e_i},$$

with $p_i \neq p_j$ for $1 \leq i < j \leq k$. This decomposition is unique up to rearrangement of factors.

Example 2.3-8:
The number 392 has the factorization into the primes 2^3 and 7^2:

$$\begin{aligned} 392 &= 2^3 \cdot 7^2 \\ &= 7^2 \cdot 2^3. \end{aligned}$$

But, if we do not consider the rearrangement of the primes 2^3 and 7^2, we assert that the factorization of 392 is unique.

relatively prime numbers

Euler phi function

Before we describe some properties of prime numbers, we introduce the notion of *relatively prime numbers*. Integers a and b are said to be relatively prime, if their greatest common divisor is one, i.e. $\gcd(a, b) = 1$. We further introduce two important number-theoretic functions, namely the *Euler phi function* denoted $\varphi(n)$ and the function $\pi(x)$.

The Euler phi function $\varphi(n)$ gives the number of natural numbers, between 1 and n inclusive, which are relatively prime to n. For instance, to find $\varphi(6)$ we must consider each of the numbers 1, 2,. . ., 6. Here 2, 3, 4, 6 are not relatively prime to 6, since $\gcd(2, 6) = 2 \neq 1$, $\gcd(3, 6) = 3 \neq 1$, $\gcd(4, 6) = 2 \neq 1$ and $\gcd(6, 6) = 6 \neq 1$. However, the integers 1 and 5 are relatively prime to 6. Thus, $\varphi(6)$ = 2. For a formal definition of $\varphi(n)$ with $n \geq 1$, we write

$$\varphi(n) = |\{1 \leq m \leq n - 1 \mid \gcd(n, m) = 1\}|.$$

The $\pi(x)$-function $\pi(x)$ denotes the number of primes below x. For instance $\pi(2) = 1$, because we count 2 as the first prime, $\pi(10) = 4$, since there are four primes below 10, namely 2, 3, 5, and 7.

Property

We now specify some properties of prime numbers without prooving them.

1. If $n = \prod_{i=1}^{k} p_i^{e_i}$ and $m = \prod_{i=1}^{k} p_i^{f_i}$, then $\gcd(n, m) = \prod_{i=1}^{k} p_i^{min(e_i, f_i)}$.
2. If p is prime, then $\varphi(p) = p - 1$.
3. If $n, m \geq 1$ with $\gcd(n, m) = 1$, then $\varphi(nm) = \varphi(n)\varphi(m)$.
4. If $n = \prod_{i=1}^{k} p_i^{e_i}$, then $\varphi(n) = n \cdot \prod_{i=1}^{k}(1 - \frac{1}{p_i})$.
5. For $n \geq 5, \varphi(n) > \frac{n}{6 \ln \ln n}$.
6. $\lim_{x \to \infty} \frac{\pi(x)}{x/\ln x} = 1$. Consequently, for a large integer $\pi(x)$ can be approximated by $x/\ln x$.
7. For every $x \geq 17$, $\pi(x) > \frac{x}{\ln x}$.
8. For $x > 1, \pi(x) < 125506 \frac{x}{\ln x}$.

Example 2.3-9:

1. $\varphi(8) = 4$, since for every $x \in \{1, 3, 5, 7\}$ $\gcd(x, 8) = 1$.
2. $\varphi(2269) = 2268$, since 2269 is prime.
3. $\varphi(64) = \varphi(2^6) = 64(1 - \frac{1}{6}) = 25$.

2.3.7 Congruences

We are all familiar with examples of *modular arithmetic* even if we have never considered them to be different ways of calculating. One common example is given by counting hours. On a clock 3 hours added to 4 o'clock is 7 o'clock: 3 + 4 = 7. But 2 hours added to 11 o'clock is 1 o'clock! Similary 3 hours substracted from 5 o'clock is 2 o'clock but 3 hours substracted from 3 o'clock is 12 o'clock. In this arithmetic system 12 takes the place of zero and numbers which differ by 12 or a multiple of it are considered to be the same. With this approach the previous example can be written in mathematical notation as:

$$\begin{aligned} 3 + 4 &\equiv 7 \mod 12 \\ 2 + 11 &\equiv 1 \mod 12 \\ 5 - 2 &\equiv 3 \mod 12 \\ 3 - 3 &\equiv 0 \mod 12. \end{aligned}$$

For example the notation $2 + 11 \equiv 1 \mod 12$ will be read as "2+11 is

congruent to 1 modulo 12". We remember that 12 and all multiples of 12 are treated as zero and that, as a result, numbers which differ by a multiple of 12 are the

same, we can perform addition and substraction just as usual. We can also use the multiplication in the same manner:

$$\begin{aligned} 4 \cdot 5 = 20 &\equiv 8 \mod 12 \\ 5 \cdot (-1) = (-5) &\equiv 7 \mod 12 \\ 3 \cdot 8 = 24 &\equiv 0 \mod 12 \\ 3 \cdot (8-2) - 7 \cdot (4-13) = 81 &\equiv 9 \mod 12. \end{aligned}$$

However, when we begin using division the results appear even more strange. Let us find the following divisions modulo 12: 10 / 5, 27 / 5, 5 / 7 and 2 / 3. In the first case the result of dividing 10 by 5 is 2, since $5 \cdot 2 \equiv 10 \mod 12$. In the second case the numbers 27 and 15 differ by 12, so they are treated as being the same

$$3 \cdot 5 \equiv 27 \mod 12.$$

Thus $27/5 \equiv 3 \mod 12$. Similarly $5/7$ is 11 because

$$7 \cdot 11 \equiv 5 \mod 12.$$

The division $2/3 \mod 12$ does not exist. To explain this we firstly calculate the multiples of $3 \mod 12$:

Multipler	3 · Multipler
0	0
1	3
2	6
3	9
4	0
5	3
6	6
7	9
8	0
9	3
10	6
11	9

Certainly 2 does not appear as the value of any of these multiples $3 \cdot 0, \ldots, 3 \cdot 11$, but does that show that it could not be the value of any multiple of 3? It does, because any other number must be congruent modulo 12 to one of the numbers $0, 1, \ldots, 11$, so that its product with 3 would be congruent to one of the products $3 \cdot 0, \ldots, 3 \cdot 11$. For instance, $77 = 6 \cdot 12 + 5 \equiv 5 \mod 12$ so that $3 \cdot 77 = 18 \cdot 12 + 3 \cdot 5 \equiv 3 \cdot 5 \mod 12$ and similarly - $34 \equiv 2 \mod 12$ so that $3 \cdot (-34) \equiv 3 \cdot 2 \mod 12$. After giving these examples with modular arithmetic modulo 12, we will discuss the congruence topic in a general form.

Definition 2.3-4: *Let $a, b \in \mathbb{Z}$ and $n \in \mathbb{N}$. One says, a is congruent to b modulo n and writes $a \equiv b \mod n$, if n is a divisor of $(a - b)$.* **congruence**

Example 2.3-10:
$36 \equiv 12 \mod 4$, since $4 \mid (36 - 12)$, i.e $4 \mid 24$.

The relation congruence modulo n, i.e. $\equiv \mod n$, possesses the following properties:

1. Reflexivity: since for all $a \in \mathbb{Z}, a \equiv a \mod n$.
2. Symmetry: since for all $a, b \in \mathbb{Z}$, if $a \equiv b \mod n$, then $b \equiv a \mod n$.
3. Transitivity: since for all $a, b, c \in \mathbb{Z}$, if $a \equiv b \mod n$ and $b \equiv c \mod n$, then $a \equiv c \mod n$.

Hence, we conclude that the congruence modulo n is an *equivalence relation* on the set of all integers.

We now give some useful properties of congruences modulo n which are easy to prove. For all $a, b, c, d \in \mathbb{Z}$, if $a \equiv b$ mod n and $c \equiv d \mod n$, then the following holds:

1. $-a \equiv -b \mod n$.
2. $a + c \equiv b + d \mod n$.
3. $a \cdot c \equiv b \cdot d \mod n$.
4. $a/c \equiv b/d \mod n$.

In item 3 of the above listed properties, if we consider the special case ($a = c$ and $b = d$), we obtain $a^2 \equiv b^2 \mod n$. Using the item 3 again with this special case ($a = c$ and $b = d$) we get $a^3 \equiv b^3 \mod n$. After applying that for $k - 1$ steps we deduce

$$a^k \equiv b^k \mod n,$$

for any natural number k.

The *equivalence class of an integer* a is the set of all integers congruent to a modulo n. It will be denoted by $a + n\mathbb{Z}$. Formally defined, we write **equivalence class**

$$a + n\mathbb{Z} = \{b \in \mathbb{Z} \mid b \equiv a \mod n\}.$$

It works out that, in fact

$$a + n\mathbb{Z} = \{a + n\mathbb{Z} \mid z \in \mathbb{Z}\}.$$

Example 2.3-11:
The equivalence class of 2 mod 7 is the set:

$$\begin{aligned} 2 + 7\mathbb{Z} &= \{2 + (7 \cdot 0), 2 + (7 \cdot \pm 1), 2 + (7 \cdot \pm 2), 2 + (7 \cdot \pm 3), \ldots\} \\ &= \{\ldots, -19, -12, -5, 2, 9, 16, 23, \ldots\}. \end{aligned}$$

From Theorem 2.3-1, we now recall that each integer a can be written as:

$$a = kn + b \;\; with\, 0 \leq b < n,\; k \in \mathbb{Z}\, and\, n \in \mathbb{N}.$$

If we use the notation of congruences modulo n, we can write:

$a \equiv b \mod n$, with $0 \leq b < n, n \in \mathbb{N}$.

Hence, each integer is congruent modulo n to one and only one integer between 0 and $n - 1$, since $0 \leq b < n$.

residue classes

We define the set of all equivalence classes modulo n denoted by $\mathbb{Z}/n\mathbb{Z}$ and call the elements *residue classes*. Also the set $\mathbb{Z}/n\mathbb{Z}$ explicitly contains the elements $n\mathbb{Z}$, $1 + n\mathbb{Z}$, $2 + n\mathbb{Z}$, ..., $(n - 1) + n\mathbb{Z}$. If we choose an arbitrary element of each residue class $n\mathbb{Z}$, $1 + n\mathbb{Z}$, $2 + n\mathbb{Z}$, ..., $(n - 1)n\mathbb{Z}$, we form a set holding as a set of *representatives* for the residue classes $\mathbb{Z}/n\mathbb{Z}$. Any *set of representatives* for the residue classes is called a *complete set of residues modulo n*.

complete set of residues modulo n

Example 2.3-12:
The residue classes $\mathbb{Z}/3\mathbb{Z}$ consist of the elements (sets) $0 + 3\mathbb{Z}, 1 + 3\mathbb{Z}$ and $2 + 3\mathbb{Z}$, where

$$\begin{aligned} 3\mathbb{Z} &= \{\ldots, -9, -6, -3, 0, 3, 6, 9, \ldots\} \\ 1 + 3\mathbb{Z} &= \{\ldots, -8, -5, -2, 1, 4, 7, 10, \ldots\} \\ 2 + 3\mathbb{Z} &= \{\ldots, -7, -4, -1, 2, 5, 8, 11, \ldots\}. \end{aligned}$$

If we choose an arbitrary element of each residue class $3\mathbb{Z}, 1 + n\mathbb{Z}$ and $2 + n\mathbb{Z}$, we get a set of representatives for the residue classes $\mathbb{Z}/3\mathbb{Z}$ (resp. a complete set residues modulo 3), i.e {0, 1, 2}, {0, 7, -4} ...

There is an infinite number of possible complete sets of residues but a computer always uses $\{0, 1, 2, \ldots, n - 1\}$ as a standard set.

Definition 2.3-5: *The integers modulo n, denoted $\mathbb{Z}_n$, is the set of (equivalence classes of) integers $\{0, 1, 2, \ldots, n - 1\}$. Addition, substraction and multiplication in $\mathbb{Z}_n$ are performed modulo n.*

Since the set $\mathbb{Z}/n\mathbb{Z}$ and the set $\mathbb{Z}_n$ have the same mathematical behaviour, we only use the set $\mathbb{Z}_n$.

Example 2.3-13:

1. $\mathbb{Z}_6 = \{0, 1, 2, 3, 4, 5\}$ is a complete set of residues modulo 6.
2. In $\mathbb{Z}_6$:

$$5 + 4 = 3 \mod 6$$
$$5 \cdot 4 = 2 \mod 6.$$

definition of $\mathbb{Z}_n^*$

We introduce the set $\mathbb{Z}_n^*$ defined as:

$$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \gcd(a, n) = 1\}.$$

In particular, if n is a prime, then $\mathbb{Z}_n^* = \{a \in \mathbb{Z} \mid 1 \leq a \leq n - 1\}$.

Example 2.3-14:

$$\mathbb{Z}_{21}^* = \{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}$$
$$\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}.$$

We will now return to the operations in modular arithmetic, more precisely we will only look at division modulo n. So we can explain more adequately why the division of 3 by 2 modulo 12 does not exist. First we begin with the definition of *the multiplicative inverse* in $\mathbb{Z}_n$.

multiplicative inverse

Definition 2.3-6: *Let $a \in \mathbb{Z}_n$. The multiplicative inverse of a modulo n is an integer $x \in \mathbb{Z}_n$ so that:*

$$ax \equiv 1 \mod n.$$

If such an x exists, then it is unique and a is said to be invertible or unit. The inverse of a is denoted a^{-1}.

Claim

a is invertible in $\mathbb{Z}_n$, if and only if $\gcd(a, n) = 1$.

Example 2.3-15:

1. 2 is not invertible in $\mathbb{Z}_6$, since $\gcd(2,6) \neq 1$.
2. 2 is invertible in $\mathbb{Z}_7$ and has the inverse 4, since $2 \cdot 4 \equiv 1 \mod 7$.

division in $\mathbb{Z}_n$

Definition 2.3-7: *Let $a, b \in \mathbb{Z}_n$. Division of a by b in $\mathbb{Z}_n$ is the product of a and b^{-1} modulo n and is only defined if b is invertible.*

Example 2.3-16:

1. Division modulo 12 of 2 by 3 does not exist, since 3 is not invertible in $\mathbb{Z}_{12}$, i.e. $\gcd(3,12) \neq 1$.
2. Division modulo 12 of 27 by 5 delivers 3, since

$$\begin{aligned} 27/5 \mod 12 &\equiv 27 \cdot 5^{-1} \mod 12 \\ &\equiv 27 \cdot 5 \mod 12, (5^{-1} \equiv 5 \mod 12) \\ &\equiv 135 \mod 12, (135 = 3 + 11 \cdot 12) \\ &\equiv 3 \mod 12. \end{aligned}$$

We look at how to find the multiplicative inverse in the set $\mathbb{Z}_n^*$.

computing of the multiplicative inverse

Given an element $a \in \mathbb{Z}_n^*$, we want to find the *multiplicative inverse* a^{-1} of $a \in \mathbb{Z}_n^*$, i.e. we must determine an element $x \in \mathbb{Z}_n^*$ so that $ax \equiv 1 \mod n$. In this case, the congruence $ax \equiv 1 \mod n$ implies that $n \mid (ax - 1)$, i.e. there is an integer k so that $ax = 1 + nk$. This can be written as:

$$ax - nk = 1. \qquad \text{2.3-1}$$

Since the integers $a, x \in \mathbb{Z}_n^*$, we have $\gcd(a,x) = 1$ and Eq. 2.3-1 will be written as:

$$ax + n(-k) = \gcd(a,n), \qquad \text{2.3-2}$$

and, if we put $y = (-k)$, we obtain:

$$ax + ny = \gcd(a,n).$$

Corollar 2.3-1 says that the equation $ax + by = \gcd(a,b)$ will always be solvable and its solution will be determined by the two integers x and y. We can now use the Extended Euclidean algorithm to find such integers x and y.

Example 2.3-17:
We want to find the multiplicative inverse of 16 in $\mathbb{Z}_{21}^*$. So we must solve the equation:

$$16x + 21y = \gcd(16, 21) = 1.$$

For this purpose, we use the Extended Euclidean algorithm and obtain the table by permuting the notation of x_i and x resp. by y_i and y:

i	0	1	2	3	4
r_i	21	16	5	1	0
q_i		1	3	5	
y_i	1	0	1	3	
x_i	0	1	1	4	

From the table we obtain $n = 3, y = (-1)^3 \cdot 3 = -3$ and $x = (-1)^{3+1} \cdot 4 = 4$, i.e.

$$16 \cdot 4 + 21 \cdot (-3) = 1$$

$$16 \cdot 4 = 1 + 21 \cdot 3$$

$$16 \cdot 4 \equiv 1 \mod 21.$$

The multiplicative inverse of 16 in $\mathbb{Z}_{21}^*$ is 4.

Chinese remainder theorem

One of the important results of modular arithmetic, which is used in public cryptography, is given in the next theorem called the *Chinese remainder theorem*. It enables resolving of simultaneous congruence equations.

Theorem 2.3-5: *Let $m_1, m_2, \ldots, m_n \in \mathbb{N}$, with $\gcd(m_i, m_j) = 1$ for each $1 \leq i < j \leq n$. The system of simultaneous congruences:*

$$\begin{aligned} x &\equiv a_1 \mod m_1 \\ x &\equiv a_2 \mod m_2 \\ &\vdots \\ x &\equiv a_n \mod m_n \end{aligned}$$

has a unique solution

$$x = \left(\sum_{i=1}^{n} a_i y_i M_i\right) \mod m,$$

where $m = \prod_{i=1}^{n} m_i, M_i = \frac{m}{m_i}$ and y_i is the solution of the congruence:

$$y_i M_i \equiv 1 \mod m_i, \text{ for each } 1 \leq i \leq n.$$

Using the Chinese remainder theorem, we can decompose an operation in a given group in operations in its subgroups.

Example 2.3-18:
We want to solve a system of congruences

$$x \equiv 2 \mod 3$$
$$x \equiv 4 \mod 5.$$

We have:

$$m_1 = 3, m_2 = 5.$$

We calculate:

$$m = m_1 \cdot m_2 = 15,$$
$$M_1 = \frac{m}{m_1} = 5,$$
$$M_2 = \frac{m}{m_2} = 3.$$

The solution of the congruences (e.g. with the use of the Extended Euclidean algorithm) $y_1 5 \equiv 1 \mod 3$ and $y_2 3 \equiv 1 \mod 5$ is 2 and 2 resp. Finally

$$x \equiv (\sum_{i=1}^{2} a_i y_i M_i) \mod m$$
$$x \equiv a_1 y_1 M_1 + a_2 y_2 M_2 \mod m$$
$$x = 2 \cdot 2 \cdot 5 + 4 \cdot 2 \cdot 3$$
$$x \equiv 44 \mod 15$$
$$= 14.$$

We verify that:

$$14 \equiv 2 \mod 3$$
$$14 \equiv 4 \mod 5.$$

We now introduce the notion of quadratic residue modulo n and that of the square root of an integer modulo n.

quadratic residue modulo n

Definition 2.3-8: *Let $a \in \mathbb{Z}_n^*$, a is called quadratic residue modulo n, or square modulo n, if there is $ax \in \mathbb{Z}_n^*$ so that $x^2 \equiv a \mod n$. If such a x does not exist, then a is called quadratic non residue modulo n.*

$Q_n\,(resp.\ \overline{Q}_n)$ denotes the set of all quadratic residues (resp. non residues) modulo n.

Definition 2.3-9: *Let $a \in \mathbb{Z}_n^*$. If $x \in \mathbb{Z}_n^*$ and $x^2 \equiv a \mod n$, then x is called a square root of a modulo n.*

square root modulo n

Example 2.3-19:
If we calculate the values of an integer a from the congruence $x^2 \equiv a \mod n$ in $\mathbb{Z}_7^*$ we obtain:

$$1^2 = 1 \equiv 1 \mod 7$$
$$2^2 = 4 \equiv 4 \mod 7$$
$$3^2 = 9 \equiv 2 \mod 7$$
$$4^2 = 16 \equiv 2 \mod 7$$
$$5^2 = 25 \equiv 4 \mod 7$$
$$6^2 = 36 \equiv 1 \mod 7.$$

We conclude that the quadratic residues modulo 7 are 1, 2, and 4 while the quadratic non residues modulo 7 are 3, 5, and 6. Thus,

$Q_7 = \{1, 2, 4\}$ and $\overline{Q}_7 = \{3, 5, 6\}$.

2.3.8 Some algebraic systems formed by the Set $\mathbb{Z}_n$

This section summarizes some important results which are needed throughout this course. We remember that the set $\mathbb{Z}_n$ contains the elements $0, 1, 2, \ldots, n-1$, where n is a positive integer.

1. The pair $(\mathbb{Z}_n, +)$ consisting of the set $\mathbb{Z}_n$ and the operation addition modulo n forms an abelian group and is called *additive group*. $(\mathbb{Z}_n, +)$ has 0 as identity element. Each element a of the set $\mathbb{Z}_n$ has an inverse $-a$. The order of the group $(\mathbb{Z}_n, +)$ is $|\mathbb{Z}_n| = n$.

   additive group

2. The pair $(\mathbb{Z}_n, \cdot)$ consisting of the set $\mathbb{Z}_n$ and the operation multiplication modulo n is not a group, since with reference to multiplication, not all elements of $\mathbb{Z}_n$ have an inverse. For example, 2 has no inverse in $\mathbb{Z}_6$.

3. The pair $(\mathbb{Z}_n^*, \cdot)$, where $\cdot$ is a multiplication modulo n, forms a group with identity element 1. Such a group is called the *multiplicative group* of $\mathbb{Z}_n$. In contrast to $\mathbb{Z}_n$, each element of $\mathbb{Z}_n^*$, closed under the operation $\cdot$, is invertible. Note that per definition each element a of $\mathbb{Z}_n^*$ satisfies $\gcd(a, n) = 1$. The order of $\mathbb{Z}_n^*$ is defined to be the number of elements in $\mathbb{Z}_n^*$, namely $|\mathbb{Z}_n^*|$. It follows from the definition of the Euler phi function that $|\mathbb{Z}_n^*| = \varphi(n)$. If an element $\alpha \in \mathbb{Z}_n^*$ is of order $\varphi(n)$, then α is a generator or a *primitive element* of $\mathbb{Z}_n^*$. If $\mathbb{Z}_n^*$ has a generator, then $\mathbb{Z}_n^*$ is cyclic. Generally if $\mathbb{Z}_n^*$ is cyclic, then the number of generators of $\mathbb{Z}_n^*$ is $\varphi(\varphi(n))$. As an example we consider the multiplicative group $\mathbb{Z}_5^*$. We will determine the order of each element of $\mathbb{Z}_5^*$,

   multiplicative group

   primitive element

therefore, we compute the power of each element of $\mathbb{Z}_5^*$ and list the results in the following table:

δ	0	1	2	3	4
$1^\delta \mod 5$	1	1	1	1	1
$2^\delta \mod 5$	1	2	4	3	1
$3^\delta \mod 5$	1	3	4	2	1
$4^\delta \mod 5$	1	4	1	4	1

The table summarizes some properties of the elements of the group $\mathbb{Z}_5^*$:

a. The order of the elements 1, 2, 3 and 4 resp. is 1, 4, 4, and 2.

b. $\langle 2 \rangle$ and $\langle 3 \rangle$ are identical to $\mathbb{Z}_5^*$, i.e. they are generators of $\mathbb{Z}_5^*$.

   Hence $\mathbb{Z}_5^*$ is cyclic.

c. $\mathbb{Z}_5^*$ has $\varphi(|\mathbb{Z}_5^*|) = \varphi(4) = 2$ generators which have the order $|\mathbb{Z}_5^*| = 4$, namely 2 and 3.

4. The tuple $(\mathbb{Z}_n, +, \cdot)$ consisting of the set $\mathbb{Z}_n$ and the two binary operations addition modulo $n\,(+)$ and multiplication modulo $n\,(\cdot)$ forms a commutative ring. But it does not form a field, since not all non-zero elements have multiplicative inverses. However, $(\mathbb{Z}_n+, \cdot)$ is a field if and only if n is prime.

2.4 Finite Fields and Polynomials

polynomial

The topics that we consider in this section involve the concept of *polynomial* and that of *finite fields over polynomials*. We note that the most important application of polynomials is the construction of finite fields which are called *Galois fields*. These fields play a dominant role in some scientific areas such as cryptography, digital communications, etc.

finite fields
Galois fields

2.4.1 Polynomial over a Ring

Let $(R, +, \cdot)$ be an arbitrary ring. A polynomial over $\mathbb{R}$ is an expression of the form

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0,$$

where n is a non-negative integer and the elements $a_i \in R$, $i \in \{0, \ldots, n\}$ are called *coefficients* of the polynomial $f(x)$. The largest integer m for which $a_m \neq 0_R$ (zero-element of the ring R) is the *degree* of f and is denoted by $\deg(f)$. a_m is also called the *leading coefficient of* f. If $f(x) = a_0\, and\, a_0 \neq 0_R$, then the polynomial f has degree 0. The polynomial is said to be *monic* if its leading coefficient is equal to 1. The set of all polynomials over the ring $\mathbb{R}$ in the *indeterminente* x is denoted by $R[x]$.

coefficients
degree of polynomials
monic polynomial

Example 2.4-1:
The expression $f(x) = 4x^4 + 2x^2 + 1$ is a polynomial over the ring $(\mathbb{Z}_5, +, \cdot)$, i.e $f \in \mathbb{Z}_5$ (the operation + and $\cdot$ are closed under the set $\mathbb{Z}_5$ and are performed modulo 5). The degree of f is 4, since 4 is the highest power of x that occurs in the expression of $f(x)$ with a non-zero coefficient.

Arithmetic operations and equality in $R[x]$:

For$f(x), g(x) \in R[x]$ with $deg(f) = n, deg(g) = m, n \geq m$, let

$f(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$, and
$g(x) = b_m x^m + b_{m-1}x^{m-1} + \ldots + b_1 x + b_0$,

one gets:

1. The polynomials f and g are *equal* if and only if their corresponding coefficients are equal, that is $a_i = b_i \, for \, i \geq 0$.
2. The addition of f and g is defined as:
$$(f + g)(x) = (a_n + b_n)x^n + (a_{n-1} + b_{n-1})x^{n-1} + \ldots + (a_0 + b_0).$$
3. The multiplication of f and g is defined as:
$$(fg)(x) = c_{n+m}x^{n+m} + \ldots + c_0,$$
whereby $c_k = \sum_{i=0}^{k} a_i b_{k-i}$ for $0 \leq k \leq n + m$.

Theorem 2.4-1: *$(R[x], +, \cdot)$ with the above introduced operations $+$ and $\cdot$ forms a commutative ring called the polynomial ring over R.*

The only arithmetic operation in $R[x]$, which we have not dealt with yet, is the division of two polynomials f and g, where each of them is in $R[x]$. This will be shown in what follows.

Definition 2.4-1: *If $f, g \in R[x]$ with $g(x) \neq 0$, then ordinary polynomial division of g by f yields polynomials q and $r \in R[x]$ so that*

$g(x) = q(x)f(x) + r(x)$, *where* $deg(r) < deg(f)$.

Moreover, the polynomials q and r are unique. The polynomial q is called the *quotient*, while the polynomial r is called the *remainder* and is denoted by $r = g \mod f$. For this notation, we say r *is equal to* g *modulo* f.

Example 2.4-2:
We consider two polynomials g and f of $\mathbb{Z}_5[x]$ with $g(x) = 2x^5 + x^4 + 4x + 3$ and $f(x) = 3x^2 + 1$. The division of g by f in $\mathbb{Z}_5[x]$ yields the quotient $q(x) = 4x^3 + 2x^2 + 2x + 1$ and the remainder $r(x) = 2x + 2$. Note that $deg(r) < deg(f)$.

2.4.2 Finite Fields

In this section we will briefly introduce the concept of finite fields. Thereafter, we will describe their structure.

finite field

Definition 2.4-2: *A finite field K_q is a field with a finite number of elements. The order of the field K_q is the number q of its elements.*

irreducible polynomial

Now, we will describe the structure of finite fields. First we introduce the notion of an *irreducible polynomial*, since irreducible polynomials are indispensible for constructing finite fields. Moreover, each polynomial in $K_q[x]$ can be written as product of irreducible polynomials in an essentially unique manner.

Definition 2.4-3: *Let $f \in K_q[x]$ with $deg(f) \geq 1$. f is irreducible in $K_q[x]$ (or prime in $K_q[x]$), if it cannot be expressed as a product of two polynomials in $K_q[x]$, each of positive degree.*

Theorem 2.4-2: *Any polynomial $f \in K_q[x]$ of positive degree can be written as*

$$f = cf_1^{e_1} \dots f_k^{e_k},$$

where $c \in K_q, f_1^{e_1} \dots f_k^{e_k}$ are distinct monic irreducible polynomials in $K_q[x], and\, e_1, \dots, e_1$ are positive integers. Moreover, this factorization is unique apart from the order in which the factors occur.

factorization of polynomials

We remember from Definition 2.4-1 that by dividing the polynomial $g \in K_q[x]$ by $f \in K_q[x]$, we obtain a unique quotient q and remainder r, where

$$g = qf + r \text{ and } deg(r) < deg(f).$$

This formula can also be expressed as:

$$g \equiv r \mod f \text{ with } deg(r) < deg(f),$$

and we say "g *is congruent to* r *modulo* f". Before we continue to discuss, what the equation

$$g \equiv r \mod f \text{ with } deg(r) < deg(f)$$

means, we will see some properties of congruences.

Let g, h, g_1, h_1, s be elements of $K_q[x]$. The following properties hold:

1. $g \equiv h \mod f$ if and only if g and h result in the same remainder upon division by f.
2. $g \equiv g \mod f$, i.e. the relation congruence modulo $f \in K_q[x]$ is reflexive.
3. If $g \equiv h \mod f$, then $h \equiv g \mod f$, i.e. the relation congruence modulo $f \in K_q[x]$ is symmetric.

4. If $g \equiv h \mod f$ and $h \equiv s \mod f$, then $g \equiv s \mod f$, i.e. the relation congruence modulo $f \in K_q[x]$ is transitiv.
5. If $g \equiv g_1 \mod f$and $h \equiv h_1 \mod f$, then $g + g_1 \equiv h + h_1 \mod f$ and $g \cdot g_1 \equiv h \cdot h_1 \mod f$.

From items 2, 3, and 4, we conclude that the relation congruence modulo $f \in K_q[x]$ is an equivalence relation.

Any polynomial in $K_q[x]$ is congruent modulo f to a unique polynomial of degree at most $deg(f)-1$, since the polynomial r (resp. q) is unique and $deg(r) < deg(f)$. Moreover, the equivalence class of $r \in K_q[x]$ is defined as the set of all polynomials g which are congruent to r modulo $f \in K_q[x]$, i.e the equivalence class of $r \in K_q[x]$ consists of all polynomials $g \in K_q[x]$ so that

$$g \equiv r \mod f.$$

We now consider a case of great interest for cryptographic applications where $K_q[x] = \mathbb{Z}_p[x]$ and$f$ is an irreducible polynomial in $\mathbb{Z}_p[x]$. Note that the set $\mathbb{Z}_p$ consists of the elements $0, 1, \ldots, p-1$, where p is prime. Hence, we can define the set of all equivalence classes modulo f which will be denoted by $\mathbb{Z}_p[x]/f$.

The set $\mathbb{Z}_p[x]/f$ closed under addition and multiplication modulo f forms a field with the order p^n, where $deg(f) = n$. The fact that $f \in \mathbb{Z}_p[x]$ is an irreducible polynomial guarantees that all elements of the set $\mathbb{Z}_p[x]/f$ will be invertible.

Furthermore, all fields over various irreducible polynomials with a fixed degree n have the same structure and various representations of their field elements. One says that all those fields are isomorphic.

For this reason the notation of the above defined field will only refer to the prime p and the degree n. This field can be denoted by $GF(p^n)$, because there is an irreducible polynomial in $\mathbb{Z}_p[x]$ for each positive integer n.

$GF(p^n)$ is called Galois field. p is the *characteristic of the field* and $GF(p)$ is the *prime field*. **Galois field**

Example 2.4-3:

1. $f(x) = x^3 + x + 1$ with $deg(f) = 3$ is an irreducible polynomial in $\mathbb{Z}_2[x]$, since it cannot be expressed as a product of two polynomials in $\mathbb{Z}_2[x]$, each of positive degree.
2. We construct the finite field $GF(8)$:
   For the construction of $GF(8) = GF(2^3)$ we use the above irreducible polynomial $f(x) = x^3 + x + 1$. The elements of $GF(8)$ are the polynomials: $0, 1, x, x+1, x^2, x^2+1, x^2+x$, and x^2+x+1, i.e $GF(8)$ contains all polynomials in $Z_2[x]$ of degree at most 2. The arithmetic in $GF(8)$ is performed modulo $f(x) = x^3 + x + 1$. To compute the product of two elements in $GF(8)$, we multiply the two polynomials and we divide by $f(x) = x^3 + x + 1$ and then we get the remainder of this division as result. The addition in $GF(8)$ is the usual addition of polynomials in $Z_2[x]$.

For example, let $f_1(x) = x^2 + 1$ and $f_2(x) = x^2 + x + 1$ be two elements of $GF(8)$. We will compute the addition and the multiplication of f_1 and f_2:

a.

$$\begin{aligned} f_1(x) + f_2(x) &= (x^2 + 1) + (x^2 + x + 1) \\ &= (x^2 + x^2) + x + (1 + 1) \\ &= \ 0 \ + x + 0 \\ &= x. \end{aligned}$$

b. Now we will compute the product of f_1 and f_2 in $GF(8)$. First, we calculate $f_1 \cdot f_2 = x^4 + x^3 + x + 1$ and thereafter divide it by $f(x) = x^3 + x + 1$. We obtain:

$$\begin{aligned} x^4 + x^3 + x + 1 &= (x + 1)(x^3 + x + 1) + x^2 + x \\ &\equiv x^2 + x \mod x^3 + x + 1 \\ &\equiv x^2 + x \mod f. \end{aligned}$$

The product of f_1 and f_2 in $GF(8)$ delivers $x^2 + x$. The following table summarizes all possible multiplications of the non-zero elements of the field $GF(8)$. The notation $a_2a_1a_0$ substitutes that of the polynomial $a_2x^2 + a_1x + a_0$.

.	001	010	011	100	101	110	111
001	001	010	011	100	101	110	111
010	010	100	110	011	001	111	101
011	011	110	101	111	100	001	010
100	100	011	111	110	010	101	001
101	101	001	100	010	111	011	110
110	110	111	001	101	011	010	100
111	111	101	010	001	110	100	011

From the table, it can be seen, that all results of the multiplication are also elements of $GF(8)$.

Another operation in $GF(p^n)$ which we have not dealt with yet is the computation of the multiplicative inverse of a polynomial modulo f and the exponentiation modulo f. The reader is referred to [Menezes96].

We introduce the notion of a *primitive polynomial*, since it may be preferable in some applications to use a primitive polynomial for defining a finite field.

primitive polynomial

Definition 2.4-4: *Let $f(x)$ be an irreducible polynomial in $\mathbb{Z}_p[x]$ of degree n and p a prime. $f(x)$ is a primitive polynomial if x is a generator of the multiplicative group $GF(p^n)^*$ $(GF(p^n)^* = GF(p^n) - \{0\})$.*

Example 2.4-4:
In Example 2.4-3 the polynomial $f(x) = x^3 + x + 1$ is primitive, since x is a generator of the multiplicative group $(GF(2^3))^*$. All elements of $(GF(2^3))^*$ can be obtained as power of x modulo $f(x)$. This is shown in the next table.

i	$x^i \mod x^3 + x + 1$
0	1
1	x
2	x^2
3	$x + 1$
4	$x^2 + x$
5	$x^2 + x + 1$
6	$x^2 + 1$
7	1
...	...

Besides the degree, there is another important integer, which specifies a non-zero polynomial over a finite field, namely its *order*. The definition of the order of a polynomial is based on the following claim.

order of the polynomial

Claim

Let $f \in \mathbb{Z}_p[x]$ be a polynomial of degree $m \geq 1$ with $f(0) \neq 0$, then there is a positive integer $e \leq p^m - 1$ so that $f(x)$ divides $x^e - 1$.

Since a non-zero constant polynomial divides $x-1$, these polynomials can be included in the following definition.

Definition 2.4-5: *Let $f \in \mathbb{Z}_p[x]$ be a non-zero polynomial. If $f(0) \neq 0$, then the smallest positive integer e for which $f(x)$ divides $x^e - 1$ is called the order of f and is denoted by $ord(f) = ord(f(x))$. If $f(0) = 0$, then $f(x) = x^h g(x)$, where $h \in \mathbb{N}$ and $g \in \mathbb{Z}_p[x]$ with $g(0) \neq 0$ are uniquely determined, $ord(f)$ is defined to equal $ord(g)$.*

The order of the polynomial f is sometimes called the period of f or the exponent of f. The order of an irreducible polynomial f can be characterized in the following alternative fashion.

Corollar 2.4-1: *If $f \in \mathbb{Z}_p[x]$ is an irreducible polynomial over $\mathbb{Z}_p$ of degree m, then $ord(f)$ divides $p^m - 1$.*

As a method for determining the primitivity of a polynomial we can use the following theorem.

Theorem 2.4-3: *Let $f(x)$ be an irreducible polynomial with $f(x) \in \mathbb{Z}_p[x]$ of degree n and p prime. We suppose that the factorization of $p^n - 1$ is known. Let $p_1, p_2, \ldots, p_t$ be the distinct prime factors of $p^n - 1$. $f(x)$ is primitive if and only if for each $k, 1 \leq k \leq t$:*

$$x^{(p^n-1)/p_k} \neq 1 \mod f(x).$$

2.5 Complexity Theory

Generally, the word security is utilized in three contexts: information theoretical security, computational security and system theoretical security.

The information theoretical security, which is also called "unconditional security", is a measure which regards the security of cryptosystems without placing a bound on the amount of computation that an attacker can do. A cryptosystem is defined to be unconditionally secure if it can not be broken, even with infinite computational resources.

The computational security (security in sense of complexity theory) is a measure which regards the computational effort required to break a cryptosystem. We might define a cryptosystem to be computationally secure if the best algorithm for breaking it requires at least N operations, where N is some specified, very large number. Using this notion, many cryptosystems based on very difficult mathematical problems have been constructed.

By developing a cryptosystem which is system-theoretically secure, the designer tries to construct the system in a way that will resist all known cryptoanalytic attacks. In other words, the designer must consider all state of the art technology in the area of cryptoanalysis.

The Vernam cipher, called a one-time system or one-time pad, is a provable unconditionally secure cipher against a cipher-only attack[3]. However, this cipher has no practical importance. The security of all other cryptosystems is based on the computational effort required to break it. That is why the treatment of the complexity theory is very important in cryptography. With complexity theoretical analysis we can estimate the security of the cryptographical algorithm.

3 See [Menezes96] on page 192.

The main goal of complexity theory is to provide mechanisms for classifying computational problems according to the resources needed to solve them. Generally, the required resources are *time, storage space, random bits, number of processors,* etc., but the main focus is on time. By time we mean the *running time* expressed in number of *operations* that is needed for solving a problem. Here an *algorithm* can be seen as a tool to solve a well-specified computational problem [Menezes96].

running time
operation
algorithm

In the context of complexity theory, the *computational security* of a *cryptosystem* depends on the existence and exploitation of problems known as hard computational problems. But how can we characterise the computational difficulty of arbitrary problems. In most cases an estimation of the running time of an algorithm that enables us to solve the required problem, is a criterion to measure the computational difficulty of such a problem. However, an exact prediction of the running time of an algorithm is impossible. Hence, only an approximation can be achieved. For this purpose, we use the so called *asymptotic running time* i.e. we analyze how the running time increases as the *size* of the *input* increases without bound. In this approach the input size of an algorithm means the total number of bits needed to represent the input in ordinary binary notation using an appropriate encoding scheme [Menezes96].

asymptotic running time

This section deals with the *asymptotic notations,* namely the $O-, \Omega-, \Theta-, o-$, and ω-notation. Thereafter, we introduce some complexity classes: P, NP and $co-NP$.

asymptotic notation

2.5.1 Asymptotic Notation

By means of asymptotic notations, we can describe the running time of an algorithm. Before we give an abstract definition of the asymptotic notation, we will look at a simple algorithm, namely the Eucledian algorithm and then estimate its running time. Furthermore, we will see how the running time of the algorithm can be expressed in some asymptotic notations.

First we start with the running time of some arithmetic operations, namely the addition and the division. In what follows a and b are two integers that can be written in binary representation as:

$$a = (a_{k-1}a_{k-2}\ldots a_1a_0)_2$$
$$b = (b_{l-1}b_{l-2}\ldots b_1b_0)_2.$$

a is a k-bit integer and b is a l-bit integer. We want to illustrate the addition of a and b in the following example.

Example 2.5-1:
If we choose $a = 10010$ and $b = 110$, then the bit addition of a and b is computed as follows

$$\begin{array}{rr} & 10010 \\ + & 00110 \\ \text{carry} & \underline{11\quad\ } \\ & 11000 \end{array}$$

Since $(l = 3) < (k = 5)$ we need to add two 0 to the left of $b = 110$ so that a and b have the same bit length. The addition of two bits is called a bit operation. The addition of a and b required $k = 5$ bit operations.

Generally, the addition of a k-bit integer a and a l-bit integer b requires $max(k, l)$ bit operations. The running time for the addition of a and b can be estimated by

$$\text{Time}(k\text{-}bit + l\text{-}bit) = max(k, l).$$

We informally introduce the O-notation and denote the time required to execute one bit operation by $O(1)$. Considering this we can describe the running time by

$$\text{Time}(k\text{-}bit + l\text{-}bit) = O(max(k, l)).$$

The division of two integers a and b is illustrated by the following example:

Example 2.5-2:
We choose $a = 110101$ and $b = 101$. The division of a by b is performed as follows

```
1 1 0 1 0 1 = 1 0 1 · 1 0 1 0 + 0 1 1
1 0 1
  0 1 1
  0 0 0
    1 1 0
    1 0 1
      0 1 1
      0 0 0
        0 1 1.
```

To divide a by b we need 4 substractions of numbers having 3 bits each. Note that the bit length of the obtained quotient is 4 and the bit length of b is 3. That is, we need $O(4 \cdot 3)$ to compute the division.

Generally, if the quotient q of the division of a by b has m bits, then the time required to compute the division is at most $O(l \cdot m)$.

The estimated running time for all arithmetic operations is summarized in the following table [Buchmann99].

Operation	Bit complexity
Addition $a+b$	$O(max(k,l))$
Subtraction $a-b$	$O(max(k,l))$
Multiplication $a \cdot b$	$O(k \cdot l)$
Division $a = qb + r$	$O(l \cdot m)$

The Euclidean algorithm is needed to compute the greatest common divisor of two integers a and b with $a > b > 0$ (see Section 2.3.4). To estimate the running time of the Euclidean algorithm, we proceed as follows.

We put $r_0 = a$ and $r_1 = b$ and compute the divisions:

$$\begin{aligned}
r_0 &= q_1 r_1 + r_2 \qquad 0 < r_2 < r_1,\\
r_1 &= q_2 r_2 + r_3 \qquad 0 < r_3 < r_2,\\
r_2 &= q_3 r_3 + r_4 \qquad 0 < r_4 < r_3,\\
&\vdots\\
r_{i_0-2} &= q_{i_0-1} r_{i_0-1} + r_{i_0} \qquad 0 < r_{i_0} < r_{i_0-1},\\
r_{i_0-1} &= q_{i_0} r_{i_0}.
\end{aligned}$$

As mentioned in Section 2.3.4 the greatest common divisor of a and b is $r_{i_0}(i_0$ is a fixed indice for which $r_{i_0+1} = 0)$.

To estimate the running time of all steps of the algorithm, we consider that the number of bit operations for the division $a = q_0 b + r_1$ is at most $length(b) \cdot length(q_0)$. And so the time for the division $r_{i-1} = q_i r_i + r_{i+1}$ is limited by $length(r_i) \cdot length(q_i) \leq length(b) \cdot length(q_i)$. Thus, the total time for the algorithm can be written as

$$O(length(b)(length(q_1) + length(q_1)) + \ldots length(q_{i_0})).$$

Note that

$$a = q_1 b + r_2 \geq q_1 b = q_1(q_2 r_2 + r_3) \geq q_1 \cdot q_2 \cdot r_2 \geq \ldots \geq (q_1 \cdot q_2 \cdots q_{i_0}).$$

Thus, the running time of the Euclidean algorithm to compute the greatest common divisor $\gcd(a,b)$ is equal to $O(k \cdot l)$ although $O(k \cdot l)$ is an upper bound for the exact running time. The fact that this is only an upper bound must be considered when estimating the running time of an algorithm in the worst case.

Now we will introduce the asymptotic notation. The asymptotic notations are defined in terms of functions f and g, where f and g are two functions of the positive integers n which take positive real values n.

2.5.2 O-notation

We denote by $O(g(n))$ the set of functions

$$f(n) \in O(g(n))$$

for which there is a positive constant c and a positive integer n_0 so that

$$0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0. \qquad 2.5\text{-}1$$

This means that $f(n)$ does not grow asymptotically faster than $g(n)$ to a certain constant multiple and that f is bounded by g to the constant multiple. Instead of $f \in O(g(n))$, one usually writes

$$f(n) = O(g(n)).$$

If $f(n)$ is a sum of terms, then only the dominant term of f is considered. The dominant term is the term that grows the fastest as n gets larger. For example, if $f(n) = an^3 + bn^2 + c$ and a, b, c are positive integers, then only the term an^3 determines the behaviour of $f(n)$ as n gets larger.

Using the O-notation, we can describe the running time of an algorithm. If T denotes the time complexity of an algorithm and $T = O(n)$, then doubling the size of the input n doubles the running time of an algorithm. If $T = O(2^n)$ then adding 1 bit to the size of the input doubles the running time of the algorithm [Schneier96].

Hence, with the O-notation, we can classify algorithms according to their *time* or *space complexities*. An algorithm is *constant* if its complexity is independent of n. An algorithm is *linear* if its time complexity is $O(n)$. Algorithms can also be *quadratic*, *cubic*, and so on. An algorithm is called *exponential* if its complexity is $O(t^{f(n)})$ with t constant and $f(n)$ a polynomial. To see the difference between these complexities we give the following example.

Example 2.5-3:
The following table [Schneier96] shows the running times for different classes of algorithms with $n = 10^6$. We assume that the unit of time to compute an operation is 1 μs.

Class	Complexity	Time at 10^6 O/s
Constant	1	$1\mu\, s$
Linear	n	$1\, s$
Quadratic	n^2	11.6 days
Cubic	n^3	32 years
Exponential	2^n	$10^{301.006}$ times the age of the universe

The computer can complete a constant algorithm in a microsecond, a linear algorithm in a second and a quadratic algorithm in 11.6 days. It would take 32 years

to compute a cubic algorithm. This is not practical. Note that if the constant k in $O(n^k)$ grows by 1 from $O(n^2)$ to $O(n^3)$ the complexity $O(n^k)$ grows from 11.6 days to 32 years. Computing an exponential algorithm is hopeless [Schneier96].

2.5.3 Ω-Notation

In contrast to the O-notation, the Ω-notation provides a lower bound to the function f. This can be seen in the definition of $\Omega(g(n))$.

$$f(n) \in \Omega(g(n))$$

if there is a positive constant c and a positive integer n_0 so that

$$0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0.$$

This means that $f(n)$ grows at least as fast asymptotically as $g(n)$ to a constant multiple. The Ω-notation is used to bound the best-case running time of an algorithm, e.g. the shortest running time for an input size n. Instead of $f \in \Omega(g(n))$, one usually writes

$$f(n) = \Omega(g(n)).$$

We remember that the running time of the Euclidean algorithm in the worst case was expressed as $O(k \cdot l)$. Needless to say, the best running time of the algorithm can be achieved if only one step to the greatest common divisor of a and b is needed. In other words, under the consideration that $a > b > 0$, the greatest common divisor of a and b will be b itself. In this case, with $r_0 = a$ and $r_1 = b$ (see the notation introduced in Section 2.5.1) we have

$$r_0 = q_1 r_1 + 0.$$

That is $\gcd(a, b) = r_1 = b$.

Supposing that q_1 is a m-bit integer the best running time of the Euclidean algorithm can be expressed by Ω as $\Omega(l \cdot m)$, where l is always the bit length of b.

2.5.4 θ-notation

The θ-notation is related to the O- and Ω-notations. The expression $f(n) = \Theta(g(n))$ means that $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. The function f also has an upper and lower bound, i.e.

$$f(n) = \theta(g(n))$$

if there are positive constants c_1, c_2 and a positive integer n_0 so that

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0.$$

The definition of $\theta(g(n))$ implies that each element of the set $\theta(g(n))$ is asymptotically non-negative, e.g. $f(n)$ is non-negative whenever n is sufficiently large.

Related to the Euclidean algorithm, the running time expressed by θ shall describe the running time in the *average case* of the algorithm. That is the average running time over all inputs of a fixed size, expressed as a function of the input size [Menezes96]. The average running time of the Euclidean algorithm is also bounded between the lower bound $\Omega(l \cdot m)$ and the upper bound $O(k \cdot m)$. It can be expressed by $\theta(\frac{k}{2} \cdot m)$.

2.5.5 o-notation

The case where $g(n)$ is an upper bound for $f(n)$ that is not asymptotically tight is described by the o-notation. We formally define

$$f(n) \in o(g(n))$$

if for any positive constant $c > 0$ there is a constant $n_0 > 0$ so that

$$0 \leq f(n) < cg(n) \text{ for all } n \geq n_0.$$

Regarding the definition of $O(g(n))$ and $o(g(n))$ we conclude that the equation $0 \leq f(n) \leq cg(n)$ is the same for the two notations. The relation $f(n) = o(g(n))$ implies that

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0.$$

The expression $o(1)$ is often used to denote a function $f(n)$ whose limit is 0 as n approaches ∞.

2.5.6 ω-notation

We use the notation $f(n) = \omega(g(n))$ if $g(n)$ is a lower bound for $f(n)$ that is not asymptotically tight. We write

$$f(n) \in \omega(g(n))$$

if for any positive constant $c > 0$, there is a constant $n_0 > 0$ so that

$$0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0.$$

The relationship between the O-notation and the o-notation is similar to that of the Ω-notation and the ω-notation considering the constant c. The relation $f(n) = \omega(g(n))$ implies

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty.$$

2.5.7 Properties of the Complexity Notations

For any functions $f(n), g(n), h(n)$ and $l(n)$ the following relations are true [Menezes96][Cormen89]:

1. $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$.
2. $f(n) = \theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
3. Reflexivity:

$$f(n) = \theta(f(n))$$
$$f(n) = O(f(n))$$
$$f(n) = \Omega(f(n)).$$

4. Symmetry:
   $f(n) = \theta(g(n))$ if and only if $g(n) = \theta(f(n))$.
5. Transitivity:
   If $f(n) = O(g(n))$ and $g(n) = O(h(n))$, then $f(n) = O(h(n))$.
6. If $f(n) = O(h(n))$ and $g(n) = O(h(n))$, then $(f + g)(n) = O(h(n))$.
7. If $f(n) = O(h(n))$ and $g(n) = O(l(n))$, then $(f \cdot g)(n) = O(h(n)l(n))$.

2.5.8 Complexity classes

We define a *polynomial time algorithm* as an algorithm having the complexity $O(n^k)$, where k is a constant. Algorithms whose complexities are $O(t^{f(n)})$, with t a constant and $f(n)$ a polynomial of n are called *exponential time algorithms.* A *subexponential time algorithm* is an algorithm with the complexity $e^{o(n)}$.

polynomial time algorithm

exponential time algorithm

subexponential time algorithm

There is a given way to classify the running time of algorithms between polynomial time and exponential time. For this purpose, we give the following definition.

Definition 2.5-1: *Let n be a positive integer. Let u be a real number between 0 and 1 and let $v > 0$ be a constant. Then*

$$L_n[u, v] = O(e^{v(ln(n))^u (lnln(n))^{1-u}}).$$

In particular

$$L_n[1, v] = O(e^{v(ln(n))}) = O(n^v) \text{ and}$$
$$L_n[0, v] = O(e^{lnln(n)}) = O((ln(n))^v).$$

An $L[u]$-algorithm is an algorithm that, when applied to the integer n, has a running time of the form $L_n[u, v]$ for some constant v. In particular, a polynomial time algorithm is an $L[0]$-algorithm and the exponential time algorithm is an $L[1]$-algorithm. By a subexponential time algorithm we mean an $L[u]$-algorithm for some $u < 1$.

As mentioned above, for a fixed input an algorithm with complexity $O(n^k)$ has generally a polynomial running time. That is, the algorithm is *practically efficient* if the constant multiple c (we mean the constant c introduced in the definition of the O-notation) and the number k are small. The algorithm with the complexity $O(2^n)$ is *absolutely not efficient*.

complexity classes

We remember that a principal task of complexity theory is to classify problems into *complexity classes* which characterize something of their intrinsic computational properties. Problems that can be solved with polynomial time algorithms are called *tractable*. In contrast, problems that cannot be solved within polynomial time are called *intractable*.

decision problems

Using the concept of polynomial and exponential time algorithms, we can specify some complexity classes such as P, NP, and $co-NP$. We notice that the problems considered by the theory of computational complexity are *decision problems*, i.e. problems which have YES or NO as an answer. Not all problems are decision problems, as shown by the following problem:

Input: Given a graph G with m nodes and m edges. Let v_0 be a node of G.

Question: Find the shortest path that start from v_0, passes through all other nodes of G and returns to G.

Clearly, this is not a decision problem. There may be more than one answer to this problem, since many different paths, starting from v_0 and returning to v_0, can be given.

2.5.9 Complexity class P

complexity class P

The *complexity class* P is the set of all decision problems that are solvable in polynomial time. In this approach, we remember that the polynomial time algorithm with complexity $O(n^k)$ is considered as practically efficient only if the O-constants[4] and the number k are small. In other words the class P contains all problems whose solutions are feasible with regard to the computational resources. For example, problems as addition, multiplication and exponentiation of real numbers belong to the class P.

2.5.10 Complexity class NP

class NP
certificate

The *complexity class* NP is the set of all decision problems for which a YES answer can be verified in polynomial time given some extra information, called a *certificate* [Menezes96].

We can imagine that a decision problem is in NP if a person with unlimited computing power can give a positive answer to the question and prove this so that another

4 That are the constants c in equation (Eq. 2.5-1).

person can verify that the answer is correct in polynomial time. His proof that a YES answer is correct is called a certificate.

Example 2.5-4:

1. We consider the following decision problem:

   Input: Positive integers N and b.

   Question: Does N have a factor in the interval $[2, b]$?

   This problem is generally not in the class P, since it is known as a hard problem and cannot be solved in polynomial time[5].

   If the person with unlimited computing power asserted that the answer is YES, e.g. he can solve this problem which is unsolvable for us, then he must deliver a certificate for his answer, e.g. he must give an integer b' so that $2 \leq b' \leq k$ and $b' \mid N$. If such b' is given, we can verify that $b' \mid N$ (the YES answer) in polynomial time, that is, this problem is in NP.

2. With regard to cryptography, if a person looks at a ciphertext c and can guess a plaintext m and a key k, then he can verify in polynomial time whether the ciphertext c corresponds to a plaintext m encrypted with the key k. We notice that this attempt is not available in all classes of ciphers, i.e. one-time pads. This problem also belongs to the class NP.

2.5.11 Complexity class $co - NP$

class $co - NP$

The *complexity class* $co - NP$ is the class of all decision problems for which a NO answer can be verified in polynomial time using an appropriate certificate [Menezes96]. The definition of the class $co - NP$ is similar to the one of the class NP, but for $co - NP$ a certificate must be given with respect to the NO answer.

Example 2.5-5:
We treat the same problem of Example 2.5-4 item 1.

Supposing that a person with unlimited computing power gives NO as answer and gives as certificate the complete prime factorization of N, we can see in polynomial time one of the factors in the interval $[2, b]$, simply by dividing each number of the complete prime factorization of N by N. The division can be done in polynomial time. That is, the NO answer can be verified in polynomial time. Thus, this problem belongs the $co - NP$ class.

5 This is known as the factorization problem, see the next section.

2.5.12 Complexity class NPC

NP-**complete**

There are specific problems in NP which are at least as difficult to solve as any other problem in NP. Such problems are denoted as NP-*complete* problems. In other words, if one would have a polynomial time algorithm for an NP-complete problem, then one would also have polynomial time algorithms for all other NP problems. The class of NP-complete problems belongs to a class which is denoted by NPC.

Fig. 2.5-1 illustrates the relationship between the complexity classes $P, NP, co-NP$, and NPC. We can see that $P \subseteq NP$ and $P \subseteq co-NP$, since each problem solvable in polynomial time can be verified in polynomial time. $NPC \subset NP$ since NPC is defined as the hardest problems in NP.

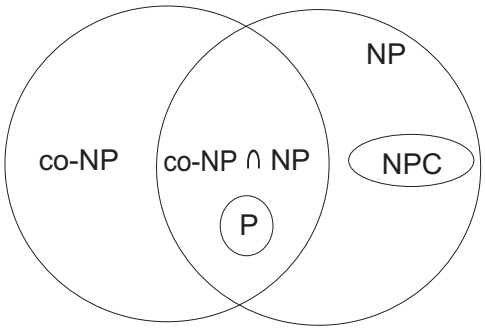


Fig. 2.5-1: Relationship between complexity classes.

2.6 Hard Problems in Number Theory

In this section we will consider some algorithms that solve two basic problems in number theory, namely:

1. The factorization problem:

   Given a composite n, factorize it into its prime powers, i.e. find the primes p_i and the exponents $e_i (1 \leq i < j \leq k$ and $p_i \neq p_j)$ so that

   $$n = \prod_{i=1}^{k} p_i^{e_i}.$$

2. The discrete logarithm problem:

   Given a cyclic group G with generator γ and an element α, find an integer x so that:

   $$\alpha = \gamma^x.$$

   The integer x is called the discrete logarithm of α to base γ.

These two problems are the basis of many cryptographic protocols and algorithms and enable to achieve the main cryptographic goals such as confidentiality, data integrity and authentication.

The factorization problem can be splitted into two parts: Firstly we must find out whether the integer n is prime or composite, this is called *primality test*. Secondly if the integer n is composite we try to find the factors.

primality test

2.6.1 Primality Tests

We mainly consider probabilistic primality tests with the following property: If n successes to pass a primality test, then it may be prime. If it passes a whole lot of primality tests, then it is very likely to be prime. If n fails one primality test, it is definitively composite. In the following we introduce some primality test algorithms.

Trial Division

First we introduce the following theorem.

Theorem 2.6-1: *If $n \in \mathbb{N}$ is composite, then n is divisible by one of the prime numbers $\leq \sqrt{n}$.*

By the trial division, if we want to test the primality of a number n, we divide it by each prime $\leq \sqrt{n}$. If n is divisible by one of these primes, then n is composite, otherwise n is prime.

Example 2.6-1:
Let $n = 1153$. The trial division of n by the primes $\leq \sqrt{1153} = 33$, namely 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 and 31 do not deliver any prime divisor. Therefore 1153 is prime.

Of course, for a large odd integer n, this method is an extremely time-consuming one. Other primality tests introduced in this section are faster.

Fermat's test

The basis of many efficient primality tests is Euler's theorem and Fermat's little theorem that will be introduced next:

Theorem 2.6-2: *(Euler's theorem)*

If $a \in \mathbb{Z}_n^*$, then $a^{\varphi(n)} \equiv 1 \mod n$.

Example 2.6-2:
If $n = 6$ and $a = 5 \in \mathbb{Z}_6^*$, we verify that $5^{\varphi(6)} = 5^2 = 25 = 1 + (6 \cdot 4) \equiv 1 \mod 6$.

Theorem 2.6-3: *(Fermat's little theorem)*

Let n be a prime. If $\gcd(a, n) = 1$ *then* $a^{n-1} \equiv 1 \mod n$.

Example 2.6-3:
Let $n = 7$ and $a = 2$, 7 is prime and $\gcd(7, 2) = 1$, we verify that

$$2^{7-1} = 64 = 1 + (7 \cdot 9) \equiv 1 \mod 7.$$

Fermat's theorem implies that if $a^{n-1} \not\equiv 1 \mod n$, then n is composite. In other words, if the integer n does not pass Fermat's test, then n is definitively composite. However, $a^{n-1} \equiv 1 \mod n$ does not automatically imply the primality of n. In this case, n can be prime or composite. This is illustrated by the following example.

Example 2.6-4:
Despite the fact that the number $n = 341$ satisfies the congruence

$$2^{341-1} \equiv 1 \mod 341 \text{ with } \gcd(2, 341) = 1,$$

341 is not prime, since $341 = 11 \cdot 31$. We say that 341 is a *pseudoprime* to base 2. The same number, namely 341, is not pseudoprime to base 3, since

$$\begin{aligned} 3^{341-1} &\equiv 56 \mod 341 \text{ with } \gcd(3, 341) = 1, \\ &\not\equiv 1 \mod 341. \end{aligned}$$

There are numbers n that are composite, even though they satisfy the equation

$$a^{n-1} \equiv 1 \mod n, \qquad 2.6\text{-}1$$

Carmichael number

for every a with $\gcd(n, a) = 1$. These numbers n are called *Carmichael numbers*.

Definition 2.6-1: *A Carmichael number is a composite integer n satisfying*

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

for every a with $\gcd(n, a) = 1$.

Example 2.6-5:
$561 = 3 \times 11 \times 17$ is the smallest Carmichael number, i.e. we get

$$a^{561-1} \equiv 1 \mod 561$$

for every integer a satisfying $\gcd(561, a) = 1$.

Using Fermat's test we cannot verify if an integer n is prime. But if (Eq. 2.6-1) holds for many numbers a, we conclude that n is probably prime, since pseudoprimes for a given base a are known to be rare. Hence Fermat's test is a probabilistic method.

Miller-Rabin Test

The Miller-Rabin test is more effective than the Fermat's test. For example: With the Fermat's test we cannot determine whether 561 is prime or not. We will see that using Miller-Rabin test, we get the information that 561 is not prime.

In order to explain the Miller-Rabin test, we define numbers s and d as follows:

$$s = max\{r \in \mathbb{N} : \ 2^r \text{ divides } n-1\}$$

with n an odd natural number. We put:

$$d = \frac{(n-1)}{2^s}.$$

With respect to the notation given above we introduce the following theorem:

Theorem 2.6-4: *If n is a prime and $\gcd(a,n) = 1$ with $a \geq 2$, then either*

$$a^d \equiv 1 \mod n$$

or there is an $r \in \{0,\ 1, \ldots,\ s-1\}$ so that

$$a^{2^r d} \equiv -1 \mod n.$$

The Miller-Rabin test is based on the previous theorem and enables us to find out whether an integer n is prime or not.

Example: [Buchmann99]
Let $n = 561$. The Fermat's test cannot determine whether 561 is prime or not, since 561 is a Carmichael number.

Using the Miller-Rabin test we obtain:

$n - 1 = 560$ and $s = 4$ is the maximal number so that $2^s \mid 560$.

$$d = \frac{(n-1)}{2^s} = \frac{560}{2^4} = 35.$$

Let $a = 2$ and $\gcd(2, 561) = 1$.

We have

$$\begin{aligned} 2^{35} &\equiv 263 \mod 561 \\ 2^{2\cdot 35} &\equiv 166 \mod 561 \\ 2^{4\cdot 35} &\equiv 67 \mod 561 \\ 2^{8\cdot 35} &\equiv 1 \mod 561. \end{aligned}$$

Thus, 561 is not prime, since neither

$$2^{35} \equiv 1 \mod 561$$

nor

$$2^{2^r \cdot 35} \equiv -1 \mod 561 \text{ for } r \in \{0, 1, 2, 3\}.$$

2.6.2 Factorization

The security of some cryptographic systems relies on the difficulty of the factorization problem. In fact, it is difficult to decompose a large odd integer n into its prime powers $p_i^{e_i}$, that is to write n as:

$$n = \prod_{i=1}^{k} p_i^{e_i}$$

with $p_i \neq p_j$ for $1 \leq i < j \leq k$.

This section deals with algorithms for the factorization problem, we introduce trial division and Pollard's rho algorithm.

Trial division

We want to find the prime powers of a composite number n. First we determine all primes p below a fixed bound B. Generally, a typical bound is 10^6. Having a list of primes p smaller than B we determine the largest integer $e(p)$ so that

$$p^{e(p)} \mid n.$$

Example 2.6-6:
We want to determine the prime powers which are factors of 525825. Let $B = 50$. The list of primes below 50 is 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47.

We start with computing the prime powers $p^{e(p)}$ so that $p^{e(p)} \mid 525825$:

$$2 \nmid 525825, \text{ that is } e(2) = 0$$

$$\frac{525825}{3} = 175275,$$

$$\frac{175275}{3} = 58425,$$

$$\frac{58425}{3} = 19475 \text{ but } 3 \nmid 19475.$$

Thus,

$$e(3) = 3 \text{ and } \frac{525825}{3^3} = 19475.$$

We now go to the next prime, namely 5:

$$\frac{19475}{5} = 3895,$$

$$\frac{3895}{5} = 779 \text{ and}$$

$$5 \nmid 779.$$

So $e(5) = 2$. We get

$$525825 = 3^3 \cdot 5^2 \cdot 779.$$

If we continue this way, we obtain $e(19) = 1$ and $e(41) = 1$ and then

$$525825 = 3^3 \cdot 5^2 \cdot 19 \cdot 41.$$

Note that the trial division is inefficient for large values of n. For example, the utilized input n in RSA is in the order of 700 bits ($lg(n) = 700$ bits, i.e. $n \approx 2^{700}$). We must perform $\sqrt{n} = 2^{350}$ divisions in the worst case to find a factor of n. Assuming that our computer executes 35 million operations in one second, we need approximately 10^{101} years to find a factor of n. Generally, the trial division is of no significance for cryptographic applications.

Pollard's rho method

Pollard's rho algorithm is a factoring algorithm for finding small factors of a composite integer. In the Pollard's rho method, we choose a function f from $\mathbb{Z}/\mathbb{Z}_n$ into itself, for example a polynomial function such as $f(x) = x^2 + 1$. Let x_0 be a randomly chosen element. We define the sequence $x_0, x_1, x_2, \ldots$ so that

$$x_{i+1} = f(x_i), \qquad \text{for } i = 1, 2, 3 \ldots$$

We hope to find two elements x_i and x_k which are in different residue classes modulo n, but in the same residue class modulo a divisor of n. In other words, we suppose

$x_i \equiv x_k \mod p$, where p is a prime.

In this case, $p \mid x_i - x_k$. But p is unknown. Therefore we proceed as follows:

We search for the terms x_i and x_k modulo n, we first verify whether $\gcd(x_i - x_k, n) > 1$. If this occurs and if $\gcd(x_i - x_k, n) < n$ then $\gcd(x_i - x_k, n)$ is a non-trivial factor of n.

Example 2.6-7:
Let $n = 91$. We would like to factorize n. By choosing $f(x) = x^2 + 1$ and $x_0 = 1$ we get:

$$\begin{aligned} x_1 &= 1^2 + 1 = 2 \\ x_2 &= x_1^2 + 1 \\ &= 2^2 + 1 \\ &= 5 \\ x_3 &= x_2^2 + 1 \\ &= 5^2 + 1 \\ &= 26 \\ &\vdots \end{aligned}$$

We have $\gcd(x_1 - x_0, 91) = \gcd(x_2 - x_1, 91) = \gcd(x_3 - x_1, 91) = 1$ and

$$\begin{aligned} \gcd(x_3 - x_2, 91) &= \gcd(26 - 5, 91) \\ &= \gcd(21, 91) \\ &= 7 < 91. \end{aligned}$$

Hence, the number $p = 7$ is a divisor of 91. Note that $91 = 7 \cdot 13$ and

$$\begin{aligned} x_3 &\equiv x_2 \mod 7 \\ 26 &\equiv 5 \mod 7. \end{aligned}$$

Assuming that the function $f(x) = x^2+1 \bmod p$ behaves like a random function, the expected time for Pollard's rho algorithm to find a factor p of n is $O(\sqrt{p})$ modular multiplications. This implies that the expected time to find a non-trivial factor of n is $O(n^{1/4})$ modular multiplications [Menezes96].

We need $O(\sqrt{p})$ pairs (x_i, x_k) to find a collision in p elements x_i (Birthday paradox). Since p can be found in $O(\sqrt{n})$ divisions (trial division), Pollard's rho algorithm has the complexity $O(\sqrt{\sqrt{n}}) = O(n^{1/4})$.

In our example of the previous section, the factorization of $n = 2^{700}$ with Pollard's rho algorithm would need 10^{44} years to find a factor of n.

2.6.3 Discrete logarithm

The security of many cryptographic techniques such as ElGamal encryption, ElGamal signature and the Diffie-Hellman key agreement depend upon the difficulty of the discrete logarithm problem. The description of this problem is given in the fol-

lowing: Let G be a cyclic group of order n. Let γ be a generator of G and α be an element of the group G. We want to determine the unique smallest integer x so that:

$$\alpha \equiv \gamma^x \qquad \text{2.6-2}$$

$x = \log_\gamma \alpha$ is called the discrete logarithm of α to base γ.

We keep the notations for α, γ, x, n, and their corresponding meanings throughout the next sections.

NOTE:

1. From the mathematical point of view, G does not have to be cyclic. In cryptographic applications it is preferable to choose G as a cyclic group and γ as a generator of G. This ensures the existence of x.
2. Generally, logarithm problems are not easily solvable. Otherwise cryptographic systems which are based on the difficulty of the DL problems would be considered insecure.

In this section we describe some algorithms for solving the DL problem in special cases.

Exhaustive search

We determine γ^0, γ^1, γ^2, ... until the result α is obtained.

Example 2.6-8:
The solution of $8 \equiv 26^x \mod 131$ with exhaustive search delivers

$$\begin{aligned}
26^0 &\equiv 1 \mod 131\\
26^1 &\equiv 26 \mod 131\\
26^2 &\equiv 21 \mod 131\\
26^3 &\equiv 22 \mod 131\\
26^4 &\equiv 46 \mod 131\\
26^5 &\equiv 69 \mod 131\\
26^6 &\equiv 91 \mod 131\\
26^7 &\equiv 8 \mod 131.
\end{aligned}$$

Hence $x = 7$.

Exhaustive search is impractical for a large value of n. In cryptographic applications the order of group is $\geq 2^{160}$. That is we need $2^{160} - 1$ group operations to find the discrete logarithm. The time needed to find discrete logarithm for $n = 2^{160}$ is in the order of 10^{33} years.

Baby Step Giant Step Algorithm

The baby step giant step method permits to solve the DL problem. It is faster than the exhaustive search but it consumes more storage space.

In the baby step giant step method, we proceed as follows:

Let $m = \lceil \sqrt{n} \rceil$, where

$$\lceil \sqrt{n} \rceil \quad = \min\{\alpha \in \mathbb{Z}; \alpha \geq \sqrt{n}\}.$$

Any number x can be expressed as follows (see Theorem 2.3-1):

$$x = qm + r, \qquad \text{with } 0 \leq r < m.$$

Eq. 2.6-2 can be written as:

$$\gamma^x = \gamma^{qm+r} = \alpha,$$

hence

$$(\gamma^m)^q = \alpha\gamma^{-r}.$$

We compute the set B called the "baby step" that is defined as:

$$B = \{(\alpha\gamma^{-r}, r) : 0 \leq r < m\}.$$

If the pair $(1, r_0)$ is contained in the set B for some integer r_0, then $x = r_0$. In this case, the discrete logarithm x has been found.

Otherwise we compute the elements $(\gamma^m)^q$ for $q = 1, 2, \ldots$ until we find an element $(\gamma^m)^q$ that is equal to a first component in the baby step set. Let q_0 be such an element. Then we get

$$(\gamma^m)^{q_0} = \alpha\gamma^{-r_0}.$$

This equation can be written as:

$$\gamma^{mq_0+r_0} = \alpha.$$

Thus, the discrete logarithm x has been determined:

$$x = q_0 m + r_0.$$

The computation of the elements $(\gamma^m)^q$ is called "giant step".

Example 2.6-9: [Buchmann99]
We want to solve the DL problem:

$$5^x = 3$$

in the group $\mathbb{Z}^*_{2017}$.

The order of the group is $n = 2017 - 1 = 2016$. We determine $m = \lceil\sqrt{2016}\rceil = 45$.

The baby step is

$$B = \{(3 \cdot 5^{-r}, r); 0 \leq r < 45\}.$$

For $r = 0$ we obtain the pair (3,0).

For $r = 1$ the first component of the pair is $3 \cdot 5^{-1} \mod 2017$. First we must determine the inverse of 5 in $\mathbb{Z}^*_{2017}$. We get $5^{-1} \mod 2017 = 807$. Then

$$3 \cdot 5^{-1} \mod 2017 \equiv 3 \cdot 807 \mod 2017 \equiv 404.$$

For $r = 2$ we get

$$\begin{aligned} 3 \cdot 5^{-2} \mod 2017 &\equiv 3 \cdot 25^{-1} \mod 2017 \\ &\equiv 3 \cdot 1775 \mod 2017 \\ &\equiv 1291. \end{aligned}$$

And so the computation of all pairs of B delivers

$$\begin{aligned} B = \{&(3,0),(404,1),(1291,2),(1065,3),\\ &(213,4),(446,5),(896,6),(986,7),\\ &(1004,8),(1411,9),(1089,10),(1428,11),\\ &(689,12),(1348,13),(673,14),(538,15)\\ &(511,16),(909,17),(1392,18),(1892,19),\\ &(1992,20),(2012,21),(2016,22),(1210,23),\\ &(242,24),(1662,25),(1946,26),(1196,27),\\ &(1046,28),(1016,29),(1010,30),(202,31),\\ &(1654,32),(1541,33),(1115,34),(223,35),\\ &(448,36),(493,37),(502,38),(1714,39),\\ &(1553,40),(714,41),(1353,42),\\ &(674,43),(1345,44)\}. \end{aligned}$$

The computation of the elements $(5^{45})^q \mod 2017$ for $q = 1, 2, \ldots$ gives

45, 8, 360, 64, 863, 512, 853, 512, 853, 62, 773, 496, 133, 1951

1064, 1489, 444, 1827, 1535, 497, 178, 1959, 1424, 1553.

The calculation is stopped when $q = q_0 = 22$, since 1553 is contained as first component in the pair (1553,40) of the baby step set. We get $r_0 = 40$ and subsequently

$$\begin{aligned} x &= q_0 m + r_0 \\ &= (22 \cdot 45) + 40 \\ &= 1030 \\ &\equiv 1030 \mod 2017. \end{aligned}$$

Thus, $5^{1030} = 3 \mod 2017$.

We need $O(\sqrt{n})$ modular multiplications to determine B, since $0 < r \leq \sqrt{n}$. To determine the giant step we need $O(\sqrt{n})$, since q can be equal to $\sqrt{n}$ in the worst case. $O(\sqrt{n} \lg n)$ is needed to make comparison between the elements of B and that of the set of giant step. Under the assumption that the group multiplications need more time than $\lg n$, the running time of baby step giant step algorithm is $O(\sqrt{n})$. The algorithm requires storage for $O(\sqrt{n})$ group elements [Menezes96], these group elements are the ($m = \sqrt{n}$) elements of the baby step.

3 Stream Ciphers

In Chapter 1 it was already explained that encryption systems can be subdivided in symmetric and asymmetric systems as well as in block and stream ciphers. In this chapter stream ciphers, which belong to the symmetric encryption techniques, are presented in more details. Design and analysis of stream cipher systems as well as the most well-kown encryption systems are introduced.

When a block cipher is used, a long message m is divided into blocks $m = m_0, m_1, \ldots, m_{N-1}$ of the same length. Here the blocks have usually a length of $n = 64$, 128 or 256 bits, depending on the processing length n of the block cipher. When stream ciphers are used, the message to be encrypted m is also divided into blocks. Here, however, only short blocks of length n occur. In this case we do not speak of a division into blocks, but into symbols. Usually, $n = 1$ or $n = 8$ bit. The encryption of the single symbols m_t is carried out through a state dependent unit.

3.1 Classification of Stream Ciphers

In the literature, the symmetric stream encryption systems are classified in

1. synchronous stream ciphers and
2. self-synchronizing stream ciphers.

In the following sections the two classes are introduced in more detail.

3.1.1 Synchronous Stream Ciphers

Fig. 3.1-1 depicts a symmetric, synchronous stream encryption system. The sender can be found on the left and the receiver on the right side. When a *synchronous stream cipher* is used, the sender and the receiver of an encrypted message have to compute the keystream z_t synchronously at any time $t \geq 0$ for encryption and decryption.

synchronous stream ciphers

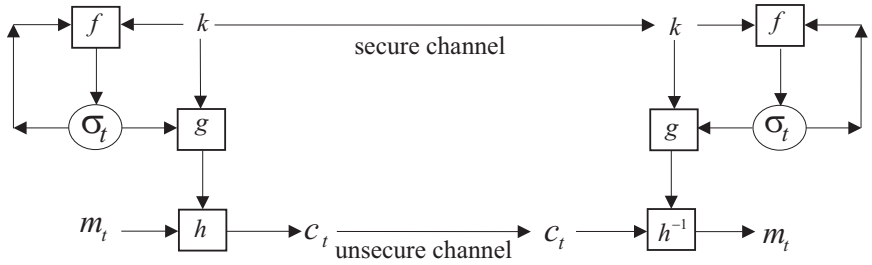


Fig. 3.1-1: Synchronous, symmetric stream cipher.

The keystream z_t is generated independently from the plaintext message and the ciphertext. The encryption of the message symbols $m_t, t \geq 0$, can be described by the following equations:

$$\begin{aligned} \sigma_{t+1} &= f(\sigma_t, k), \\ z_t &= g(\sigma_t, k), \\ c_t &= h(z_t, m_t), \end{aligned}$$

where $t \geq 0$ is valid. The system has a state variable σ_t whose initial state σ_0 can either be known publicly or be determined from the secret key k. In order to be able to carry out the encryption, the *encryption function* h must clearly be invertible. The function f is called the *next state function* and g is called the *output function*. The functions f, g and h are known publicly.

encryption function
next state function
output function

When the receiver of the ciphertext sequence $c_t, t \geq 0$, knows the secret key k and the initial state σ_0, he can decrypt the ciphertext c_t as follows:

$$\begin{aligned} \sigma_{t+1} &= f(\sigma_t, k), \\ z_t &= g(\sigma_t, k), \\ m_t &= h^{-1}(z_t, c_t). \end{aligned}$$

When the values of k and σ_t of sender and receiver correspond at any time $t \geq 0$ on both sides, the same keystream z_t is generated.

The functional unit which consists of the state σ_t and the functions f and g is called *keystream generator*. Its task is to generate the keystream sequence z_t, which is similar to a random sequence, from the key k and the initial state σ_0. In Section 3.2 we will introduce various methods for designing keystream generators.

keystream generator

A special type of synchronous stream ciphers is the *binary additive stream cipher*. Here the symbols are $m_t, c_t, z_t \in \mathrm{GF}(2)$ and the function h corresponds to the binary XOR operation.

binary additive stream cipher

We would like to mention here that each block cipher in the OFB mode (*output feedback mode*) can be used as a synchronous stream cipher. A summary of the single operation modes for block ciphers is given in Chapter 4.

A ciphertext symbol c_t that is modified does not cause error propagation, but it leads to a wrong decryption of the symbol c_t. Consequently, an attacker is able to make changes to symbols at selected positions t in the plaintext and see what this change causes in the ciphertext. Thus, it is absolutely necessary that additional mechanisms are employed in order to provide data origin authentication and data integrity.

However, when a symbol is inserted or deleted in the ciphertext sequence, the synchronization between sender and receiver is disturbed and the symbols of the ciphertext that follow can not be decrypted correctly. Thus, it is reasonable to use mechanisms such as inserting synchronization marks or a frequent reinitialization of the generator.

3.1.2 Self-synchronizing Stream Ciphers

self-synchronizing stream ciphers

Besides synchronous stream ciphers, there are also *self-synchronizing stream ciphers*, but they are hardly used in information and communication systems. In this case the keystream z_t depends on the key k and a fixed number l of previously generated ciphertext symbols.

encryption

The *encryption* of a sequence of plaintext symbols $m_t, t \geq 0$, can be described by the following equations:

$$\begin{aligned} \sigma_t &= (c_{t-l}, c_{t-l+1}, \ldots, c_{t-1}), \\ z_t &= g(\sigma_t, k), \\ c_t &= h(z_t, m_t). \end{aligned}$$

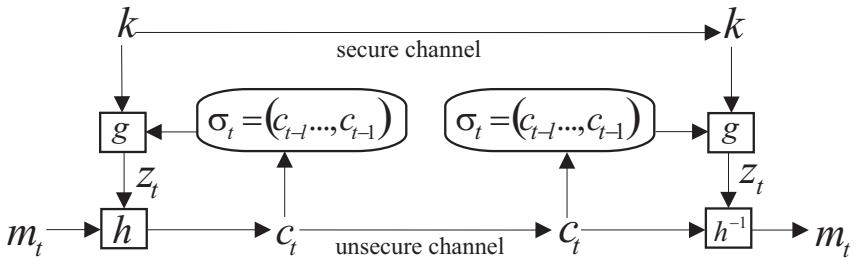


Fig. 3.1-2: Self-synchronizing, symmetric stream cipher.

The state of the encryption system is formed by the l previous ciphertext symbols and is described as σ_t. The function h has to be invertible so that the decryption process can be carried out correctly.

decryption

The *decryption* of a ciphertext sequence $c_t, t \geq 0$, can be carried out by using the following operations:

$$\begin{aligned} \sigma_t &= (c_{t-l}, c_{t-l+1}, \ldots, c_{t-1}), \\ z_t &= g(\sigma_t, k), \\ m_t &= h^{-1}(z_t, c_t). \end{aligned}$$

Encryption and decryption are depicted in Fig. 3.1-2.

When a symbol of the ciphertext sequence c_t is inserted or deleted, self-synchronizing stream ciphers are, unlike synchronous stream ciphers, capable of re-establishing proper decryption upon receiving l ciphertext symbols correctly. However, if a ciphertext symbol is modified, then decryption of up to l subsequent symbols may be incorrect and there is an error propagation.

It is hard to assess the cryptographic security of self-synchronizing stream ciphers because the keystream z_t depends on the ciphertext as well as on the plaintext. That is why they are hardly used in modern communication systems.

It has to be mentioned that every block cipher in the CFB mode (*cipher feedback mode*) can be operated as a self-synchronizing stream cipher.

3.2 Design of Keystream Generators

This section about stream ciphers continues describing the methods of designing keystream generators. Keystream generators are a major building block of synchronous stream encryption systems. A keystream generator computes the keystream sequence $z = z_0, z_1, \ldots$ from the key k. The sequence z is a pseudorandom sequence and it should be statistically similar to a random and uniformly distributed sequence. In the special case that keystream sequence symbols are from GF(2), a random sequence is characterized by the following properties:

properties of random sequences

- Balanceness of the single sequence elements: $P(z_t = 0) = P(z_t = 1) = 0.5$ for all $t \geq 0$.
- Statistical independence of the single sequence elements: $P(z_t|z_0, z_1, \ldots, z_{t-1}) = P(z_t)$.

one-time pad

The only stream encryption system which produces such a keystream is the *one-time pad*. Let $m = m_0, m_1, \ldots, m_{N-1}$ be the message sequence consisting of N sequence elements m_t of GF(2), which is to be encrypted and transmitted. Then a random keystream $z = z_0, z_1, \ldots, z_{N-1}$ which must have at least the same length N as the message sequence is necessary. The sequence z has to be available for both sender and receiver and it must not be accessible to unauthorized persons. The encryption is then carried out with

$$c_t = m_t + z_t$$

and the decryption with

$$m_t = c_t + z_t$$

for $t \geq 0$. In practice the disadvantage of the one-time pad is that the sender and the receiver must have the same keystream z which has to be transmitted via a secure channel. Here it has to be considered that the keystream z has to be just as long as the message m which is to be encrypted. C. Shannon could show that the one-time pad, from an information theoretical point of view, is to be regarded as perfectly secure.

information theoretical approach

When an *information theoretical approach* for assessing and designing an encryption system is used, it is assumed that the attacker has unlimited memory location and computing power for the attack. Although only ciphertext-only attacks are considered, it is allowed that the attacker knows a finite subsequence of the plaintext. An attack is assessed as successful when the plaintext or the secret key can be given with the probability of 1 after having observed the ciphertext c. Thus, an encryption system is regarded as perfectly secure when the attacker does not obtain information (transinformation $I(m, c) = 0$) about the plaintext m and the key k although he observed the ciphertext c.

As it turns out, this information theoretical approach is not practical. Therefore, it is attempted not to use a truly random keystream z for encryption. Instead, a deterministic algorithm is used for generating the keystream z. A *pseudorandom bit generator* (PRBG) generates a pseudo random keystream sequence $z = (z_0, \ldots, z_{N-1}) \in \text{GF}(2)^N$ of length N from a short key $k = (k_0, \ldots, k_{l-1}) \in \text{GF}(2)^l$ of length l:

PRBG

$$\text{PRBG: GF}(2)^l \rightarrow \text{GF}(2)^N, (k_0, \ldots, k_{l-1}) \rightarrow (z_0, \ldots, z_{N-1}).$$

The number of possible output sequences is at most a small fraction, namely $2^l/2^N$, of all possible sequences of length N. The intention is to take a small random sequence in form of the key k and to expand it with PRBG to a sequence of much larger length, in such a way that an attacker cannot efficiently distinguish between output sequences of the PRBG and truly random sequences.

The basic requirements on a PRBG can be checked by *statistical tests*, for example:

statistical tests

1. Frequency test (frequency of "0's" and "1's"),
2. Serial test (frequency of tuples),
3. Poker test (frequency of non-overlapping blocks of length i),
4. Runs test (frequency of runs),
5. Autocorrelation test (autocorrelation of output sequences are measured).

When a PRBG passes certain selected tests, its cryptographic suitability is still not shown. The output sequences of a linear feedback shift register (LSFR) with a primitive feedback polynomial of degree l pass several statistical tests, but when $2l$ output symbols z_t are observed, the initialization of the linear feedback shift register and hence the remaining sequence elements are predictable.

A basic requirement for using a PRBG in cryptography is that the key k is sufficiently long so that an attacker cannot perform an exhaustive search via all possible initial states. In applications with real time requirements and high throughput rates it is necessary that a keystream bit z_t can be generated in polynomial time or even in linear time.

In 1984 Blum and Micali ([Blum84]) defined a *cryptographic secure pseudo random bit generator* (CSPRBG) as an algorithm which has additionally the following property:

CSPRBG

- There is no polynomial time algorithm which can determine the next bit with a probability significantly greater than 0.5 from observing the subsequence $z_0, \ldots, z_t$ without knowing the key.

This definition for a CSPRBG is equivalent to a characterization of Yao from 1982 ([Yao82]):

- There is no polynomial time algorithm which can distinguish between the output of the generator and a truly random bit sequence with a probability significantly greater than 0.5 without knowing the key.

In [Blum84] examples for generators are given, whose security can be proven under the assumption that efficient algorithms of a well-known number theoretic problem (determination of the discrete logarithm) does not exist. This designing method which leads to a CSPRBG is also described as complexity-theoretical design method.

However, these examples do not have a practical meaning as it would require too much memory and computing time to generate the sequences, although it is polynomially restricted. Yao's results have made it possible to give the desired cryptographic security on the basis of all statistical tests for binary sequences which can be carried out in polynomial time.

Blum-Blum-Shup generator

As an example of a CSPRBG we would like to introduce the *Blum-Blum-Shub generator*. It is based on the assumption that integer factorization cannot be carried out in polynomial time. The method of generating the keystream $z = z_0, \ldots, z_{N-1}$ from the secret key k works as follows:

1. Setup:

   Determine two random and distinct primes p and q with a bit length of about $l/2$ and each congruent to $3 \bmod 4$.

   Let $n = pq$.

2. Initialization:

   Map the key k on the integer s. It is demanded that $1 \leq s \leq n-1$ and $\gcd(s, n) = 1$ is valid.

   Compute $x_0 = s^2 \bmod n$.

3. Generation of the keystream sequence $z = z_0, \ldots, z_{N-1}$:

   Carry out the following steps for $0 \leq t \leq N-1$:

   a. $x_{t+1} = x_t^2 \bmod n$.

   b. $z_t =$ least significant bit of x_{t+1}.

Example 3.2-1: Blum-Blum-Shup CSPRBG [Stinson95]

1. Setup:

   Suppose $p = 383$ and $q = 503$. p and q are primes and $p \equiv q \equiv 3 \mod 4$, since

   $$p = 383 = 3 + (95 \cdot 4), \text{ and}$$
   $$q = 503 = 3 + (125 \cdot 4).$$

   We compute $n = p \cdot q = 192649$.

2. Initialization:

   Let $s = 101355$. We have $1 \leq (s = 101355) \leq n - 1 = 192648$ and $\gcd(101355, 192649) = 1$. We compute

$$\begin{aligned} x_0 &\equiv s^2 \mod n \\ &\equiv 101355^2 \mod 192649 \\ &\equiv 20749. \end{aligned}$$

3. Generation of the keystream sequence $z = z_0, \ldots, z_{N-1}$ with $N = 20$:

   The least significant bits z_t of each $x_{t+1} (x_{t+1} = x_t^2 \mod n)$, which correspond to the output of the Blum-Blum-Shup generator, are given in the following table:

Tab. 3.2-1: *Bits produced by the Blum-Blum-Shup generator*

t	x_t	Binary representation of x_t	z_{t-1}
0	20749		
1	143135	100010111100011111	1
2	177671	101011011000000111	1
3	97048	10111101100011000	0
4	89992	10101111110001000	0
5	174051	101010011111100011	1
6	80649	10011101100001001	1
7	45663	1011001001011111	1
8	69442	10000111101000010	0
9	186894	101101101000001110	0
10	177046	101011001110010110	0
11	137922	100001101011000010	0
12	123175	11110000100100111	1
13	8630	10000110110110	0
14	114386	11011111011010010	0
15	14863	11101000001111	1
16	133015	100000011110010111	1
17	106065	11001111001010001	1
18	45870	1011001100101110	0
19	137171	100001011111010011	1
20	48060	1011101110111100	0

When, in practice, PRBG generators are designed for cryptographic applications the so-called *system-theoretical approach* is mostly used. Here, one tries to design a generator consisting of components with statistical properties which can be mathematically controlled. Basic components are, for example, linear or non-linear feed-

system-theoretical approach

fundamental attacks

back shift registers with regular or irregular clock control. One attempts to fulfill certain design objectives and requirements of keystream sequences and, at the same time, to design a generator which is resistant to all known attacks. *Fundamental attacks* are:

1. Exhaustive search through the entire key space
2. Linear substitutions
3. Divide-and-conquer attack
4. Statistical defects
5. Unconditional correlations
6. Conditional correlations.

In the next section we shall discuss the mathematical description of binary sequences and their generation by means of linear feedback shift registers. An important property of sequences is the linear complexity which can be effectively computed with the Berlekamp-Massey algorithm.

3.3 Binary Sequences and Linear Feedback Shift Registers

LFSR

Many generators proposed in the literature have a shift register with *linear feedback* (LFSR) as a basic component. Such an LFSR of length l consists of l stages which store at time t l elements

$$s_t, s_{t+1}, \ldots, s_{t+l-1}$$

of the finite field GF(2), which consists of the elements $\{0, 1\}$. The l elements from the state of the LFSR at time t are summarized to a state vector

$$\underline{s}_t = (s_t, s_{t+1}, \ldots, s_{t+l-1})^T.$$

When the step t to $t + 1$ is carried out the elements of the stages are shifted one position to the left. Here the sequence element s_{t+l-1} is output and the sequence element which is stored on the left side of the stage is determined by the linear feedback of the storage elements of the LFSR as follows: Let $c_0, c_1, \ldots, c_{l-1} \in$ GF(2) be the feedback coefficients, then the sequence element s_{t+l} is determined as

$$s_{t+l} = \sum_{j=0}^{l-1} c_j s_{t+j}$$

by the l stored sequence elements of the linear feedback shift register. Consequently, a binary sequence

$$s = s_0, s_1, s_2, \ldots$$

can be generated with a given initialization of the stages with $s_0, s_1, \ \ldots, s_{l-1}$ and l feedback coefficients. The binary sequence is described as *linear shift register*

sequence. The feedback coefficients of the LFSR are summarized to the so-called *feedback polynomial* $c \in \mathrm{GF}(2)[x]$

LSFR sequence
feedback polynomial

$$c(x) = x^l - \sum_{j=0}^{l-1} c_j x^j.$$

The theory of LFSRs is mathematically completely developed. Unfortunately, this does not apply to shift registers with non-linear feedback so that LFSRs serve as major building blocks of keystream generators.

Now we want to explain the terms period and pre-period of a binary sequence $s = s_0, s_1, \ldots$. Every finite and autonomous state machine with output, for example an LSFR, generates an output sequence s which starts with a *pre-period* of length $t_0 \geq 0$ and is then followed by the *periodic* part with a cycle length of p, namely

$$s_{t+p} = s_t$$

for $t \geq t_0$. Such a sequence is called *ultimately periodic* and when $t_0 = 0$ is valid it is called *periodic*. The period of the sequence s is defined as the smallest cycle of length p for which

ultimatively periodic sequence
period of a sequence

$$s_{t+p} = s_t$$

is still valid for $t \geq t_0$.

A finite and autonomous state machine whose state is stored in a binary memory of length l can produce an output sequence with a maximum possible period 2^l. When we just consider a state machine with a linear next-state function, for example an LFSR, an output sequence with a maximum period $2^l - 1$ can be produced due to the fact that the zero state is mapped onto itself.

The existence of pre-periods with $t_0 > 0$ can easily be shown for an LFSR. When $c_0 = 0$ is valid, then $t_0 > 0$, and for $c_0 = 1, t_0 = 0$ is valid. In the later case all output sequences of the LFSR are periodic.

The value p of the period of the output sequences of an LFSR of length l can easily be expressed by the value e, the exponent of the feedback polynomial $c(x)$ of the LFSR. The exponent e of a polynomial c is the smallest integer for which

$$c(x) | x^e - 1$$

is valid. The maximum value of the exponent e to a polynomial of degree l is $2^l - 1$. Such a polynomial is also denoted as primitive.

Theorem 3.3-1: *Let the feedback polynomial $c(x) \in GF(2)[x]$ belonging to the LFSR be irreducible and e is the exponent of $c(x)$. Then every sequence produced by this LFSR, except for the zero sequence, has the period e.*

The definition mentioned above can be generalized for any polynomials (see [Lidl94]).

If p is the period of a sequence s, then this sequence can be produced by an LFSR with the feedback polynomial $c(x) = x^p - 1$.

Example 3.3-1: Computing of the exponent
We will write a program for the Crypto-Interpreter, which determines the exponents of all polynomials of degree 4. In addition, it should indicate whether the poloynomial is irreducible and/or primitive.

The Crypto-Interpreter programme:

```
PROCESS compute_exponent;

DCL c,expo integer;
DCL prim,irr boolean;

START;
TASK c:=16;

MACRO while (c<32);
  TASK expo:=poly\_order(c);
  TASK irr:=is\_irreducible(c);
  TASK prim:=is\_primitive(c);
  CALL writeln('c=',c);
  CALL writeln('Exponent=',expo);
  CALL writeln('is\_irreducible',irr);
  CALL writeln('is\_primitive',prim);
  TASK c := c+1;
MACRO wend(loop1);

STOP;
ENDPROCESS;
```

The result of the calculations are summarized in Tab. 3.3-1.

Tab. 3.3-1: *Determination of the exponent, irreducibility and primitivity of all polynomials of degree 4*

c	exponent	irreducible	primitive
x^4	1	no	no
x^4+1	4	no	no
x^4+x	3	no	no
x^4+x+1	15	yes	yes
x^4+x^2	2	no	no
x^4+x^2+1	6	no	no
x^4+x^2+x	7	no	no
x^4+x^2+x+1	7	no	no
x^4+x^3	1	no	no
x^4+x^3+1	15	yes	yes
x^4+x^3+x	7	no	no
x^4+x^3+x+1	6	no	no
$x^4+x^3+x^2$	3	no	no
$x^4+x^3+x^2+1$	7	no	no
$x^4+x^3+x^2+x$	4	no	no
$x^4+x^3+x^2+x+1$	5	yes	no

Example 3.3-2: Period of the output sequences of an LFSR
We consider the LFSR with the irreducible feedback polynomial

$$c(x) = x^4 + x^3 + x^2 + x + 1,$$

which has the exponent 5. This is valid, as $c(x)$ divides the polynomial x^5-1 and not x^4-1. Thus, the LFSR only produces, besides the zero sequence, sequences of period 5. The output sequences of the LFSR of period 5 can be subdivided into three categories, with each of them having five sequences:

1. $(1,0,0,0,1,1,\ldots),(0,0,0,1,1,0,\ldots),(0,0,1,1,0,0,\ldots),$
   $(0,1,1,0,0,0,\ldots),(1,1,0,0,0,1,\ldots)$
2. $(0,1,0,0,1,0,\ldots),(1,0,0,1,0,1,\ldots),(0,0,1,0,1,0,\ldots),$
   $(0,1,0,1,0,0,\ldots),(1,0,1,0,0,1,\ldots)$
3. $(1,0,1,1,1,1,\ldots),(0,1,1,1,1,0,\ldots),(1,1,1,1,0,1,\ldots),$
   $(1,1,1,0,1,1,\ldots),(1,1,0,1,1,1,\ldots).$

Theorem 3.3-2: *The period of a sequence of an LFSR with irreducible feedback polynomial $c(x)$ of degree l is a divisor of 2^l-1. The sequence has the maximum period 2^l-1 exactly, when $c(x)$ is primitive.*

Example 3.3-3: Determination of the state and the output of an LFSR
We want to determine the state and the output of an LFSR by using the Crypto-Interpreter. We use an LFSR with the feedback polynomial

$$c(x) = x^4 + x^3 + x^2 + x + 1,$$

and the initial state $\underline{s}_0 = s = (1, 0, 0, 0)^T$. The Crypto-Interpreter programme is as follows:

```
  PROCESS lfsr_output;

  DCL t integer;
  DCL c,state_t, lfsr, out_t integer;
  DCL state_s charstring;

  START;

  TASK c := 31;
  TASK state\_t := 1;

  CALL lfsr_init(lfsr,c,state_t);

  TASK t:=0;

  MACRO while (t<11);
    TASK state_t := lfsr_state(lfsr);
    TASK state_s := bin(state_t);
    CALL writeln('State of the LFSR: ',state_s);
    TASK out_t := lfsr_run(lfsr,1);
    CALL writeln('Output of the LFSR: ',out_t);
    TASK t := t+1;
    MACRO wend(loop1);

  CALL lfsr_exit(lfsr);

STOP;

ENDPROCESS;
```

The output of the program is summarized in Table 3.3-2.

Tab. 3.3-2: *Determination of output and state of an LFSR*

t	State	Output
0	(1,0,0,0)	1
1	(0,0,0,1)	0
2	(0,0,1,1)	0
3	(0,1,1,0)	0
4	(1,1,0,0)	1
5	(1,0,0,0)	1
6	(0,0,0,1)	0
7	(0,0,1,1)	0
8	(0,1,1,0)	0
9	(1,1,0,0)	1
10	(1,0,0,0)	1

Sequences generated by an LFSR with a primitive feedback polynomial are often referred to as maximum sequences or m-sequences. Maximum sequences are of special importance in cryptographic applications due to their large periods. The existence of primitive polynomials for any degree l is secure. The number of primitive polynomials of degree l over the field GF(2) is

$$\lambda(l) = \frac{\varphi(2^l - 1)}{l},$$

whereas φ depicts the Euler function.

Golomb's randomness postulates

Moreover, it has to be mentioned that the m-sequences fulfill *Golomb's randomness postulates*. These criteria were established by S.W. Golomb in 1967 for characterizing the suitability of a sequence s as a pseudorandom sequence. A subsequence of consecutive, identical symbols of maximum length are described as a run. A run of "0's" is called a gap, while a run of "1's" is called a block. Golomb's randomness postulates for a sequence s of period p are as follows:

G1: In each period of a sequence s, the number of "1's" differs from the number of "0's" by at most 1.

G2: In each period of a sequence s, $1/2^i$ of all runs have length i. For each of these lengths, there are equally many gaps and blocks. The application of G2 only makes sense for i where the sequence has 2^{i+1} runs of any length.

G3: The autocorrelation function $C(\tau)$ of the sequence s is constant for $1 \leq \tau \leq p-1$. $C(0) = C(p) = 1$ is valid.

The autocorrelation function $C(\tau)$ is defined as

$$C(\tau) = \frac{A(\tau) - D(\tau)}{p},$$

where $A(\tau)$ is the number of similar sequence elements between the sequence s and a shift of s by τ positions. The value of $D(\tau)$ is determined as $D(\tau) = n - A(\tau)$.

Example 3.3-4: Golomb's randomness postulates
Now we would like to check if the sequence

$$s = 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, \ldots$$

of period 15 fulfills Golomb's randomness postulates. The sequence s has eight 1's and seven 0's, hence it fulfills G1. The sequence has 8 runs. Four of length 1 and two of length 2, with which criterion G2 is fulfilled. As $C(\tau) = -1/15$ for $1 \leq \tau \leq 14$ is valid for the autocorrelation function, G3 is also fulfilled.

In the previous sections m-sequences have been introduced as pseudorandom sequences due to their large period and as they fulfilled Golomb's randomness postulates. However, we will see that LFSR sequences and m-sequences are easily predictable. First we want to show that you can set up a clearly solvable system of linear equations for determining the l feedback coefficients by using only $2l$ consecutive sequence elements. As a result we can identify the sequence elements of the entire period of the sequence s.

Let s be an LFSR sequence which was generated by an LSFR with an unknown feedback polynomial of degree l. Furthermore, the sequence elements are determined by:

$$\begin{aligned}
s_{t+l} &= \sum_{j=0}^{l-1} c_j s_{t+j} \\
s_{t+1+l} &= \sum_{j=0}^{l-1} c_j s_{t+1+j} \\
&\vdots \qquad\qquad \vdots \\
s_{t+2l-1} &= \sum_{j=0}^{l-1} c_j s_{t+l-1+j}.
\end{aligned}$$

We thus obtain a system of linear equations for the l unknowns $c_0, \ldots, c_{l-1}$:

$$\begin{pmatrix} s_t & s_{t+1} & \ldots & s_{t+l-1} \\ s_{t+1} & s_{t+2} & \ldots & s_{t+l} \\ \vdots & \vdots & & \vdots \\ s_{t+l-1} & s_{t+l} & \ldots & s_{t+2l-2} \end{pmatrix} \cdot \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{l-1} \end{pmatrix} = \begin{pmatrix} s_{t+l} \\ s_{t+l+1} \\ \vdots \\ s_{t+2l-1} \end{pmatrix}.$$

After these explanations it is clear that LFSRs cannot be considered for cryptographically relevant pseudorandom sequence generators. However, they are important building blocks for constructing suitable generators.

linear complexity

As a further quality criterion for cryptographically suitable sequences we can consider the *linear complexity* $L(s)$ of a sequence s. $L(s)$ denotes the length of the

smallest LFSR, with which the sequence s can be generated. A designer should be able to give the exact value or the lower bound for the linear complexity of the output sequences of its generator. The linear complexity of a given sequence s can effectively be determined by using the Berlekamp-Massey algorithm. This technique was established in 1969 ([Massey69]) and was originally used for decoding BCH Codes. A comprehensive description of this technique can be found in [Fumy94].

In the last section we found out that linear feedback shift registers used as keystream generators are not sufficiently secure. More secure generators based on linear shift registers can be designed by adding the output sequences of the shift registers to high nonlinear combinations. Furthermore, additional stages can be added or irregular clocking can be introduced. There are many suggestions in the literature, but here we would like to concentrate on three generators because their properties, such as period or linear complexity of the output sequence, can be analyzed easily: the nonlinear filter generator, the combiner generator and the combiner generator with memory.

3.3.1 Nonlinear Filter Generator (NLFG)

This generator (see Fig. 3.3-1) consists of an LFSR with a feedback polynomial $c \in \mathrm{GF}(2)[x]$, of which n stage positions (also called taps or phases) $\Gamma = (\gamma_1, \ldots, \gamma_n)$, with $0 \leq \gamma_1 < \gamma_2 < \ldots < \gamma_n \leq l - 1$, are added to a Boolean function $f : \mathrm{GF}(2)^n \rightarrow \mathrm{GF}(2)$ and the output

$$z_t = f(s_{t+\gamma_1}, \ldots, s_{t+\gamma_n})$$

for $t \geq 0$ is formed, where $s = s_0, s_1, \ldots$ is the output sequence of the LFSR. The combining function f is referred to as filtering function.

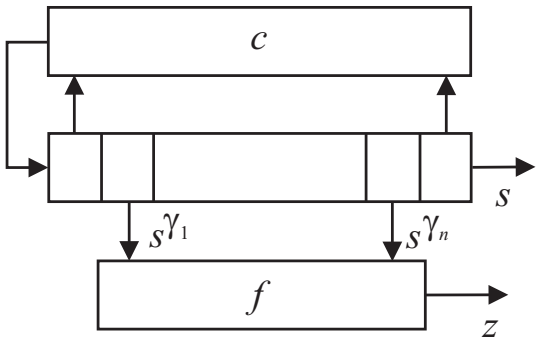


Fig. 3.3-1: A nonlinear filter generator.

3.3.2 Combiner Generator Without Memory

In the case of the combiner generator without memory (see Fig. 3.3-2) the sequences $s_j = s_{j,0}, s_{j,1}, \ldots$, are generated by LFSRs with feedback polynomials c_j of degree $l_j, 1 \leq j \leq n$, and combined with a Boolean function $f : \mathrm{GF}(2)^n \rightarrow \mathrm{GF}(2)$ in order to obtain the keystream sequence.

The keystream sequence z_t of the combiner generator is determined by

$$z_t = f(s_{1,t}, \ldots, s_{n,t})$$

for $t \geq 0$. In this case the function f is called a combining function.

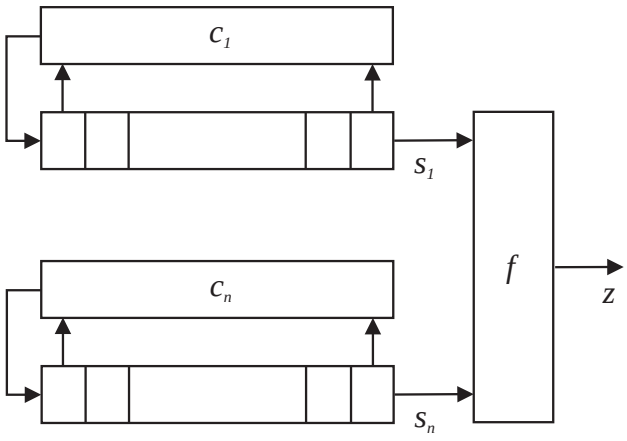


Fig. 3.3-2: A combiner generator without memory.

In order to avoid correlation and divide-and-conquer attacks on a combiner generator without memory, the function f should be balanced, should have a high degree of correlation immunity, should have a high algebraic degree and should have a high nonlinearity. Unfortunately, there is a tradeoff between a high order of correlation immunity and a high algebraic degree of nonlinearity in Boolean functions. This disadvantage can be overcome by using combiner generators with memory.

3.3.3 Combiner Generator With Memory

In contrast to a combiner generator without memory, the combiner generator with memory (see Fig. 3.3-3) has M bit memory. The state of the combining unit at time t is described as $\underline{v}_t \in \mathrm{GF}(2)^M$ and n input sequences $s_j = s_{j,0}, s_{j,1}, \ldots, 1 \leq j \leq n$, are added to it. The sequence s_j is generated by an LFSR with feedback polynomial c_j of degree $l_j, 1 \leq j \leq n$. The keystream sequence z is computed with the combining function $g : \mathrm{GF}(2)^n \times \mathrm{GF}(2)^M \rightarrow \mathrm{GF}(2)$, and the state at time $t+1$ is formed with the next state function $F : \mathrm{GF}(2)^n \times \mathrm{GF}(2)^M \rightarrow \mathrm{GF}(2)^M$. The next state transformation and the generation of the output are described by the following equations:

$$\underline{v}_{t+1} = F(s_{1,t}, \ldots, s_{n,t}, \underline{v}_t),$$
$$z_t = g(s_{1,t}, \ldots, s_{n,t}, \underline{v}_t).$$

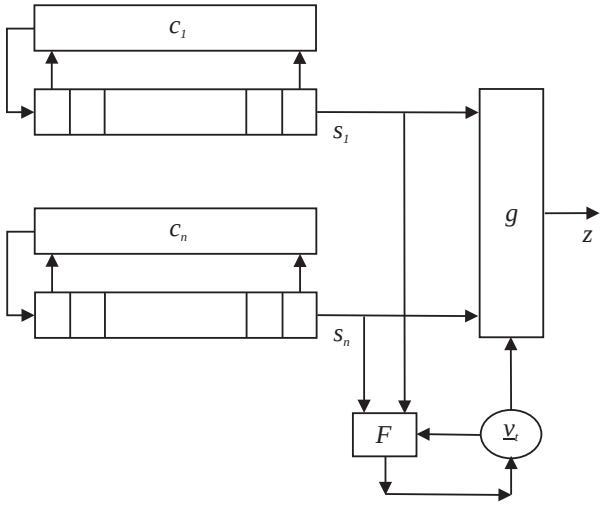


Fig. 3.3-3: A combiner generator with memory.

Examples of combiner generators with memory are the summation generator of Massey and Rueppel ([Rueppel86]) and the E_0-generator which is used as a keystream generator in the Bluetooth transmission technology where packet data are stream encrypted at the air interface.

3.4 Software-based Keystream Generators

Besides shift register-based keystream generators, a great number of software-based stream encryption systems can be found in the literature. The generators are called software-based because they can be implemented effectively with the instructions of common microprocessors or high level programming languages, such as C, C++ and Pascal. RC4 is probably the best-known software-based keystream generator. We would like to describe this software in more detail.

RC4 *RC4* (Rivest Cipher Nr. 4) is a software-based stream cipher which was designed in 1987 by Ron Rivest for the company RSA Data Security ([Rivest92]). This method works with a variable key length of l bytes, $0 < l \leq 256$, and outputs 1 byte per operation step. The sequence of these bytes can be used as a keystream to encrypt the plaintext.

The name RC4 is copyrighted by the company RSA Data Security. The algorithm could be kept secret until 1994, when the method was published in the Internet via an anonymous electronic mailing list and the Usenet News-Group sci.crypt. Currently, it is suggested to publish the algorithm under the name of Arcfour as a Request for Comments at the Internet Engineering Task Force (IETF) ([Kaukonen99]), so that the use of the RC4 algorithm is not impeded by trademark rights or licences.

RC4 is implemented in various commercial products, such as Lotus Notes, Apple Computers's AOCE and Oracle Secure SQL. The algorithm is also used in numerous security layers of network protocols for link or session connections. Examples of this are the link encryption of the packet service in the cellular and digital American mobile communication standard, in the SSL-Protocol (*Secure Socket Layer*) and the successor of the TLS-Protocol (*Transport Layer Secure*) used for encrypting message packets. It is also used to encrypt the transport protocol of the Internet application SSH (*Secure Shell*) and in the IPSec Internet Standard (*Security Architecture for the Internet Protocol*).

We would now like to describe RC4 in a more general version and describe it as RC4-(n, l), where n indicates the bit length of the memory words and variables and l the number of words of n bit length of the secret key k. The parameter l is in the range of $0 < l \leq 2^n$. Hence, the key has a length of $n \cdot l$ bit. The form of the RC4, described in [Rivest92] and [Kaukonen99], then corresponds to the RC-$(8, l)$ algorithm. It is remarkable that instead of 8 bit, the general version generates in parallel n bit as a keystream sequence in one operation step, and the required memory location for the table takes a value of $n2^n$ bit.

In its initialization phase the RC4-(n, l) algorithm puts up a table S by using the secret key $k = (k_0, ..., k_{l-1}), 0 \leq k_i \leq 2^n - 1$ and $0 \leq i \leq l-1$. In the second phase the table S is modified in each time step t and the keystream word $z_t, 0 \leq z_t \leq 2^n - 1$ and $t \geq 0$, is generated from a certain table entry. The table $S = (s_0, \ldots, s_{2^n-1})$ consists of 2^n word entries s_i, with $0 \leq s_i \leq 2^n - 1$ and $0 \leq i \leq 2^n - 1$.

1. Initialization phase:

   The table $S = (s_0, \ldots, s_{2^n-1})$ is initialized with the key $k = (k_0, ..., k_{l-1})$ as follows:

   a. Allocate the memory for table S and initialize it with $s_i := i$ for $0 \leq i \leq 2^n - 1$.

   b. The table S is now modified in a loop having the counter $i, 0 \leq i \leq 2^n - 1$ with the help of key k. At the beginning of the loop the variable j is set to zero.

   $$j := (j + s_i + k_{i \bmod l}) \bmod 2^n$$

   exchange the contents of s_i and s_j.

   c. Due to security reasons, the key k should now be set to zero, i.e. be removed from the memory. In the second phase the key k is no longer required.

   d. Initialize the variables i, j and t : $i := 0; j := 0; t := 0$.

2. Keystream generation phase:

   A keystream word $z_t, t \geq 0$, is generated with the help of the table S and the counters i and j:

   $$i := (i + 1) \bmod 2^n$$

   $$j := (j + s_i) \bmod 2^n.$$

   Exchange the contents of s_i and s_j

   $$h := (s_i + s_j) \bmod 2^n$$

   $$z_t := s_h$$

   $$t := t + 1.$$

   Despite the attacks ([Knudsen98], [Mister98]) on weakened or modified versions of the RC4-(n, l) which can be found in the literature, the RC4-$(8, 16)$ technique is regarded as very secure when a secret key with a length of 128 Byte is used. One disadvantage is the table size of 256 bit and the time needed to initialize the table.

Example 3.4-1:
To demonstrate the RC4 algorithm we will choose $n = 4$ and $l = 2$ (needless to say, these parameters are too small to be secure, but the demonstration of the algorithm is simpler). Furthermore, we suppose that the secret key k is set to $k = (k_0, k_1) = (3, 11)$.

1. In the initialization phase of RC-$(4, 2)$, we start with

   $$S = (0, 1, 2, \ldots, 15)$$

and obtain the Table 3.4-1.

Tab. 3.4-1: *Initialization phase of RC-*$(4, 2)$ *with* $k = (k_0, k_1) = (3, 11)$.

i	j	$(s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}\, s_{11}, s_{12}, s_{13}, s_{14}, s_{15})$
0	3	(3, 1, 2, 0, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)
1	15	(3, 15, 2, 0, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1)
2	4	(3, 15, 4, 0, 2, 5, 6, 7, 8, 9, 10, 11, 12,13, 14, 1)
3	15	(3, 15, 4, 1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 0)
4	4	(3, 15, 4, 1, 2, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 0)
5	4	(3, 15, 4, 1, 5, 2, 6, 7, 8, 9, 10, 11, 12, 13, 14, 0)
6	13	(3, 15, 4, 1, 5, 2, 13, 7, 8, 9, 10, 11, 12, 6, 14, 0)
7	15	(3, 15, 4, 1, 5, 2, 13, 0, 8, 9, 10, 11, 12, 6, 14, 7)
8	10	(3, 15, 4, 1, 5, 2, 13, 0, 10, 9, 8, 11, 12, 6, 14, 7)
9	14	(3, 15, 4, 1, 5, 2, 13, 0, 10, 14, 8, 11, 12, 6, 9, 7)
10	9	(3, 15, 4, 1, 5, 2, 13, 0, 10, 8, 14, 11, 12, 6, 9, 7)
11	15	(3, 15, 4, 1, 5, 2, 13, 0, 10, 8, 14, 7, 12, 6, 9, 11)
12	14	(3, 15, 4, 1, 5, 2, 13, 0, 10, 8, 14, 7, 9, 6, 12, 11)
13	15	(3, 15, 4, 1, 5, 2, 13, 0, 10, 8, 14, 7, 9, 11, 12, 6)
14	14	(3, 15, 4, 1, 5, 2, 13, 0, 10, 8, 14, 7, 9, 11, 12, 6)
15	15	(3, 15, 4, 1, 5, 2, 13, 0, 10, 8, 14, 7, 9, 11, 12, 6)

2. Keystream generation phase:

   As yet the table S has the value (3, 15, 4, 1, 5, 2, 13, 0, 10, 8, 14, 7, 9, 11, 12, 6). In Table 3.4-2, we give the iteration steps for generating the keystream z_t.

Tab. 3.4-2: *Keystream generation phase using RC-*$(4, 2)$ *with* $k = (k_0, k_1) = (3, 11)$

t	i	j	h	$(s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}, s_{11}, s_{12}, s_{13}, s_{14}, s_{15})$	z_t
0	1	15	5	(3, 6, 4, 1, 5, 2, 13, 0, 10, 8, 14, 7, 9, 11, 12, 15)	2
1	2	3	5	(3, 6, 1, 4, 5, 2, 13, 0, 10, 8, 14, 7, 9, 11, 12, 15)	2
2	3	7	4	(3, 6, 1, 0, 5, 2, 13, 4, 10, 8, 14, 7, 9, 11, 12, 15)	5
3	4	12	14	(3, 6, 1, 0, 9, 2, 13, 4, 10, 8, 14, 7, 5, 11, 12, 15)	12
4	5	14	14	(3, 6, 1, 0, 9, 12, 13, 4, 10, 8, 14, 7, 5, 11, 2, 15)	2
5	6	11	4	(3, 6, 1, 0, 9, 12, 7, 4, 10, 8, 14, 13, 5, 11, 2, 15)	9
6	7	15	3	(3, 6, 1, 0, 9, 12, 7, 15, 10, 8, 14, 13, 5, 11, 2, 4)	0
7	8	9	2	(3, 6, 1, 0, 9, 12, 7, 15, 8, 10, 14, 13, 5, 11, 2, 4)	1
8	9	3	10	(3, 6, 1, 10, 9, 12, 7, 15, 8, 0, 14, 13, 5, 11, 2, 4)	14
9	10	1	4	(3, 14, 1, 10, 9, 12, 7, 15, 8, 0, 6, 13, 5, 11, 2, 4)	9
10	11	14	15	(3, 14, 1, 10, 9, 12, 7, 15, 8, 0, 6, 2, 5, 11, 13, 4)	4
11	12	3	15	(3, 14, 1, 5, 9, 12, 7, 15, 8, 0, 6, 2, 10, 11, 13, 4)	4
12	13	14	8	(3, 14, 1, 5, 9, 12, 7, 15, 8, 0, 6, 2, 10, 13, 11, 4)	8
13	14	9	11	(3, 14, 1, 5, 9, 12, 7, 15, 8, 11, 6, 2, 10, 13, 0, 4)	2
14	15	13	1	(3, 14, 1, 5, 9, 12, 7, 15, 8, 11, 6, 2, 10, 4, 0, 13)	14
15	0	0	6	(3, 14, 1, 5, 9, 12, 7, 15, 8, 11, 6, 2, 10, 4, 0, 13)	7
16	1	14	14	(3, 0, 1, 5, 9, 12, 7, 15, 8, 11, 6, 2, 10, 4, 14, 13)	14
17	2	15	14	(3, 0, 13, 5, 9, 12, 7, 15, 8, 11, 6, 2, 10, 4, 14, 1)	14
18	3	4	14	(3, 0, 13, 9, 5, 12, 7, 15, 8, 11, 6, 2, 10, 4, 14, 1)	14
19	4	9	0	(3, 0, 13, 9, 11, 12, 7, 15, 8, 5, 6, 2, 10, 4, 14, 1)	3
20	5	5	8	(3, 0, 13, 9, 11, 12, 7, 15, 8, 5, 6, 2, 10, 4, 14, 1)	8
21	6	12	1	(3, 0, 13, 9, 11, 12, 10, 15, 8, 5, 6, 2, 7, 4, 14, 1)	0
22	7	11	1	(3, 0, 13, 9, 11, 12, 10, 2, 8, 5, 6, 15, 7, 4, 14, 1)	0
23	8	3	1	(3, 0, 13, 8, 11, 12, 10, 2, 9, 5, 6, 15, 7, 4, 14, 1)	0
24	9	8	14	(3, 0, 13, 8, 11, 12, 10, 2, 5, 9, 6, 15, 7, 4, 14, 1)	14
25	10	14	4	(3, 0, 13, 8, 11, 12, 10, 2, 5, 9, 14, 15, 7, 4, 6, 1)	11
26	11	13	3	(3, 0, 13, 8, 11, 12, 10, 2, 5, 9, 14, 4, 7, 15, 6, 1)	8
27	12	4	2	(3, 0, 13, 8, 7, 12, 10, 2, 5, 9, 14, 4, 11, 15, 6, 1)	13
28	13	3	7	(3, 0, 13, 15, 7, 12, 10, 2, 5, 9, 14, 4, 11, 8, 6, 1)	2
29	14	9	15	(3, 0, 13, 15, 7, 12, 10, 2, 5, 6, 14, 4, 11, 8, 9, 1)	1
⋮	⋮	⋮	⋮	⋮	⋮

4 Block Ciphers

In chapter 1 it was already explained that encryption systems can be subdivided in symmetric and asymmetric systems as well as in block and stream ciphers. When block ciphers are used, a long message m is divided into blocks $m = m_0, m_1, \ldots, m_{N-1}$ of the same length. Here the blocks have usually a length of $n = 64, 128$ or 256 bits, depending on the processing length n of the block cipher. Padding mechanisms are used to fill the last block when the message m is not long enough so that the last block m_{N-1} is also an n-bit block. Then the single blocks $m_t, 0 \leq t \leq N-1$ are assigned to a time-invariant encryption function f in order to obtain ciphertext $c_t = E_k(m_t)$, where k is the secret, symmetric key.

We speak of block ciphers when each of the message blocks m_t of the message $m = m_0, m_1, \ldots, m_{N-1}$ is encrypted independently. The bit size of the message blocks $m_t, t \geq 1$, is called n. As a fundamental building block, the versatility of the block ciphers allows construction of stream ciphers (see modes of operation in Section 4.2), pseudorandom number generators, message authentification codes (MACs) and cryptographic hash functions [6].

In this chapter techniques and design principles for block ciphers are introduced. Depending on the application, different requirements on the design of block ciphers are made. Therefore, we try to depict basic design steps for secure and practical block ciphers. After that, the most important modi of operation are explained, and some important block cipher algorithms (DES, IDEA, AES) are presented in detail.

4.1 Design Principles

In order to decrypt the ciphertexts $c = E_k(m)$ of a block cipher, the encryption function E for a fixed key k has to be an injection. When m and c are blocks of length n bit, then E for a fixed key has to be a bijection, namely a permutation of n-bit vectors. Ideally, each key k should have a different permutation. If a block cipher implements each possible permutation, the key k has to have a length of

$$\log_2(2^n!) \approx (n - 1.44)2^n \text{ bit}$$

to represent all $2^n!$ permutations of an n-bit vector. Thus, this enormous key length makes true random block ciphers impractical, even for small n. A block length which is too small may be vulnarable to code book attacks or correlation attacks. Nonetheless, one should consider the design objective that a randomly chosen key k yields a permutation chosen as randomly as possible. Consequently, for large n it is necessary to find an implementation, which at least ensures a certain pseudorandom selection of permutations.

6 Message authentification codes (MACs) and cryptographic hash functions are dealt with in chapter 7.

The parameters block length n and key length l should be chosen at least so large that a data complexity of 2^n as well as a processing complexity of 2^l is large enough not to allow an attacker to carry out an exhaustive key search in 10 or 20 years. Today, a block length of $n = 64, 128$ and 256 bits and an equally sized key length are used.

Feistel cipher

The iterated *Feistel cipher* has established itself as a common design principle for block ciphers. Here the basic components usually consist of a network of substitution boxes (S-boxes) and permutations of bit positions, which are referred to as product cipers. It is attempted to obtain a sufficiently complex encryption function through an iterated application of these components. Fig. 4.1-1 illustrates the principle of a product cipher consisting of various stages of S-boxes and permutations.

S-boxes

Now, let a (g, h)-bit *S-box* be a mapping of g-bit vectors on h-bit vectors and a g-bit permutation carries out exclusively a bijection on bit positions. Consequently, an S-box can be realized by using h Boolean functions with g inputs. Formally a (g, h) S-Box is defined by a Boolean mapping $F : \mathrm{GF}(2)^g \rightarrow \mathrm{GF}(2)^h$, with $F = (f_1, \ldots, f_h)$ and $f_i : \mathrm{GF}(2)^g \rightarrow \mathrm{GF}(2)$ for $1 \leq i \leq l$.

design criteria for S-boxes

Gordon and Retkin[Gordon82] formulated the following *design criteria* for cryptographically suitable S-boxes:

1. Each output bit should be truely independent from each input bit, that means that in the minimized Boolean expression, which denotes an output bit as a function of the input bits, all input variables should occur. A Boolean function with this property is called *complete*.

2. When an input bit is modified, it should, on average, change half of all output bits. This property of an S-box is called *Avalanche effect*.

3. There should be no linear dependency between an output bit and an input bit.

4. One should have no information about the output bits, as long as the input bits are unknown. This criterion is fulfilled when each of the possible output vectors is equally distributed in the set of all output vectors. In a bijective substitution this is always valid, as in this case each output vector exactly occurs one time.

In the meantime modified and increased requirements for the design of S-boxes have been developed in response of the new attack methods. The question about the existence of linear factors in the S-boxes, which can be, among other things, used for cryptoanalysis, plays a further important role. The S-boxes and permutations serve as an inner structure of encryption functions. Block ciphers are usually composed of the applications of S-boxes and permutations of various rounds after the so-called scheme of a Feistel cipher, which we would like to explain here in more detail.

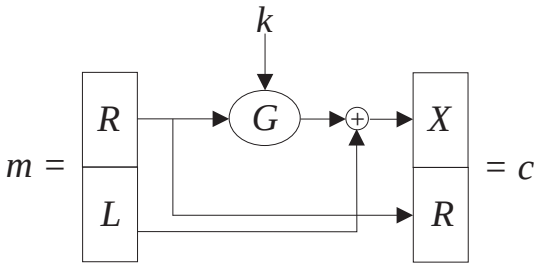


Fig. 4.1-1: Encryption principle of a Feistel cipher with one round.

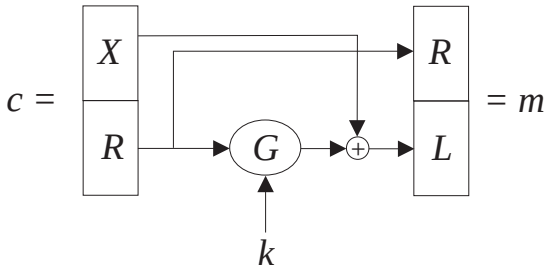


Fig. 4.1-2: Decryption principle of a Feistel cipher with one round.

The Feistel cipher is based on the idea of using the same function

$$G : \mathrm{GF}(2)^l \times \mathrm{GF}(2)^{n/2} \rightarrow \mathrm{GF}(2)^{n/2}$$

for encryption as well as for decryption. The function G, for example, consists of a product cipher. Here we assume that n is even and l is the length of the key k or a subkey derived from it. The plaintext block m of length n bit is split into two equally sized blocks L and R, each having a length of $n/2$ bit: $m = (L, R)$. Then the ciphertext block c is, as shown in Fig. 4.1-1, put together from the block R and the bitwise XOR operation of block L with the function value $G(k, R)$:

$$c = (R, L + G(k, R)) = (R, X) \ .$$

As the plaintext block R occurs unencrypted in the resulting ciphertext block c, the plaintext block m can be reconstructed out of c, as shown in Fig. 4.1-2, and the key k:

$$m = (G(k, R) + X, R) = (G(k, R) + L + G(k, R), R) = (L, R) \ .$$

It should be taken into consideration that no special requirements on the function G have been made and that the encryption scheme has been en- and decrypted with the same function.

This principle does guarantee the bijectivity of the resulting encryption function and the fixed key k, but it obviously represents a rather weak cipher as one half of each plaintext block remains completely unencrypted. A higher cryptographic security can be achieved by repeated iteration of this scheme. Here different functions G_i and subkeys k_i can be used in different encryption rounds E_i.

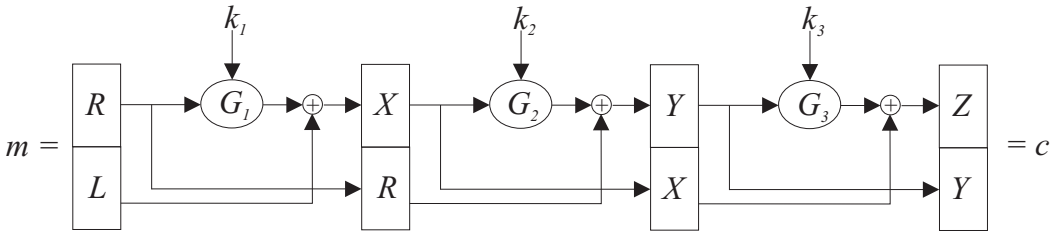


Fig. 4.1-3: Encryption principle of a Feistel cipher with three rounds.

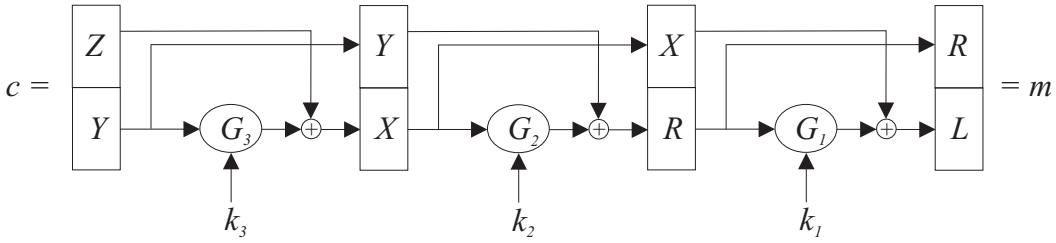


Fig. 4.1-4: Decryption principle of a Feistel cipher with three rounds.

encryption and decryption rounds

Such an *encryption round* E_i with subkey k_i is then structured as follows:

$$E_i(L_{i-1}, R_{i-1}) = (R_{i-1}, L_{i-1} + G_i(k_i, R_{i-1})) = (L_i, R_i),$$

wherein L_i and R_i are blocks of length $n/2$ bit. An encryption step E_i is selfinverse. When

$$V(L, R) = (R, L),$$

is defined, we obtain

$$\begin{aligned} &V \circ E_i \circ V \circ E_i(L_{i-1}, R_{i-1}) = \\ &= V \circ E_i(L_{i-1} + G_i(k_i, R_{i-1}), R_{i-1}) \\ &= V(R_{i-1}, L_{i-1} + G_i(k_i, R_{i-1}) + G_i(k_i, R_{i-1})) \\ &= (L_{i-1}, R_{i-1}). \end{aligned}$$

That means that the inverse D consisting of an r round cipher E

$$E = V \circ E_r \circ E_{r-1} \circ \ldots \circ E_2 \circ E_1$$

is the mapping

$$D = V \circ E_1 \circ E_2 \circ \ldots \circ E_{r-1} \circ E_r.$$

Ciphers which have this property are called Feistel ciphers. In Fig. 4.1-3 encryption and in Fig. 4.1-4 decryption of a Feistel cipher with three rounds are illustrated.

4.2 Modes of Operation

A block cipher usually encrypts a plaintext m_t having a fixed length of n bits. Longer messages are divided into blocks with a fixed block length of n bit: $m_0, m_1, \ldots, m_{N-1}$. The simplest approach is to encrypt each block separately by using the same key k. This mode of operation of a block cipher, also referred to as ECB mode, has various disadvantages so that other modes of operation have been developed and standardized for encrypting long messages. In what follows, we assume that E denotes the encryption function and D the decryption function of a symmetric block cipher which is able to process n-bit message blocks m_t. The symmetric, secret key is k. Here we describe the modes of operation as they are standardized in [ISO91].

You can test the modes of operation with several ciphers with the java-applet given as additional multimedia material on the book home page.

4.2.1 ECB mode

encryption and decryption in ECB

The *Electronic Codebook Mode* (ECB) of operation encrypts the message blocks $m_t, 0 \leq t \leq N-1$ by using

$$c_t = E_k(m_t),$$

and decrypts the ciphertext blocks c_t by using

$$m_t = D_k(c_t) \ .$$

The encryption and decryption process is illustrated in Fig. 4.2-1.

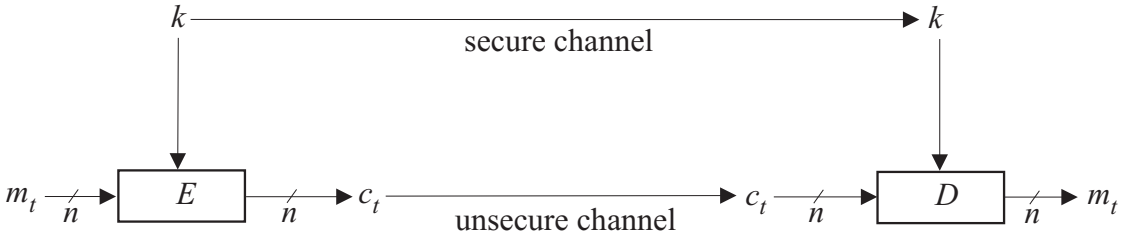


Fig. 4.2-1: ECB mode of operation for an n-bit block cipher.

properties of the ECB mode

The ECB mode of operation has the following properties:

1. Identical plaintext blocks are encrypted to identical ciphertext blocks under the same key k.

2. Message blocks are encrypted independently of each other. Re-ordering ciphertext blocks results in correspondingly re-ordered plaintext blocks. Thereby, an attacker can re-order or substitute message blocks.

3. Error propagation: One or more bit errors in a single ciphertext block affect the decryption of that block only.

For this reasons the ECB mode is not recommendable.

4.2.2 CBC mode

encryption and decryption in CBC

The *Cipher Block Chaining* (CBC) mode of operation carries out the *encryption* with the following step:

$$c_t = E_k(c_{t-1} + m_t),$$

where c_{-1} is initialized by IV (initial vector) and the addition of c_{t-1} and m_t corresponds to the bitwise addition. Decryption is carried out by:

$$m_t = c_{t-1} + D_k(c_t)$$

with $c_{-1} = \mathrm{IV}$. Encryption and decryption in CBC mode are illustrated in Fig. 4.2-2.

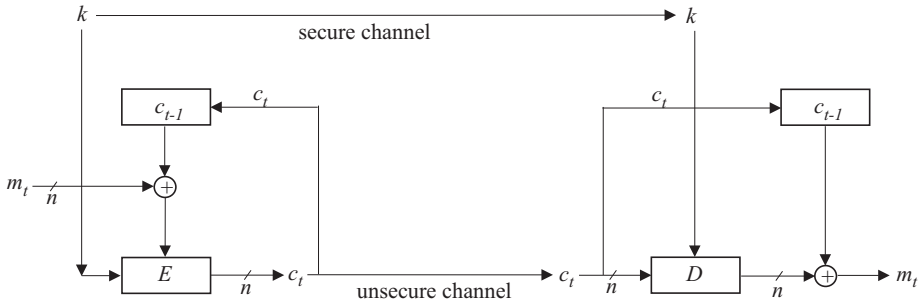


Fig. 4.2-2: CBC mode of operation for an n-bit block cipher.

properties of CBC mode

The CBC mode of operation has the following *properties*:

1. Identical plaintext and an identical value of the IV result in identical ciphertext. Changing the IV or m_0 results in different ciphertext.

2. Re-arranging the order of ciphertext blocks affects decryption directly.

3. Error propagation: A bit error in c_t affects the decryption of c_t and c_{t+1}, whereas the recovered plaintext m'_{t+1} has bit errors precisely where c_t did. Thus, an attacker can directly affect the decryption of message c_{t+1}. When c_{t+1} and c_{t+2} have been transmitted without errors, then c_{t+2} is decrypted correctly.

4. The value of IV does not have to be secret. Nonetheless, integrity and authentication of IV should be checked by the receiver.

4.2.3 OFB mode

Each symmetric block cipher can be operated in the *Output Feedback Mode* (OFB) of operation as a synchronous, additive stream cipher (see Fig. 4.2-3). In this case, instead of n-bit blocks, $r \leq n$-bit blocks are encrypted:

encryption and decryption in OFB

$$\begin{aligned} O_t &= E_k(I_t), \\ z_t &= \mathrm{Left}(O_t, r), \\ c_t &= m_t + z_t, \\ I_{t+1} &= O_t, \end{aligned}$$

where $t \geq 0$, I_t and O_t are n-bit blocks and the function Left selects the r leftmost bits of the n-bit block O_t.

The decryption of the r-bit ciphertext blocks $c_t, t \geq 0$, under the secret key k, is carried out as follows:

$$\begin{aligned} O_t &= E_k(I_t), \\ z_t &= \mathrm{Left}(O_t, r), \\ m_t &= c_t + z_t, \\ I_{t+1} &= O_t. \end{aligned}$$

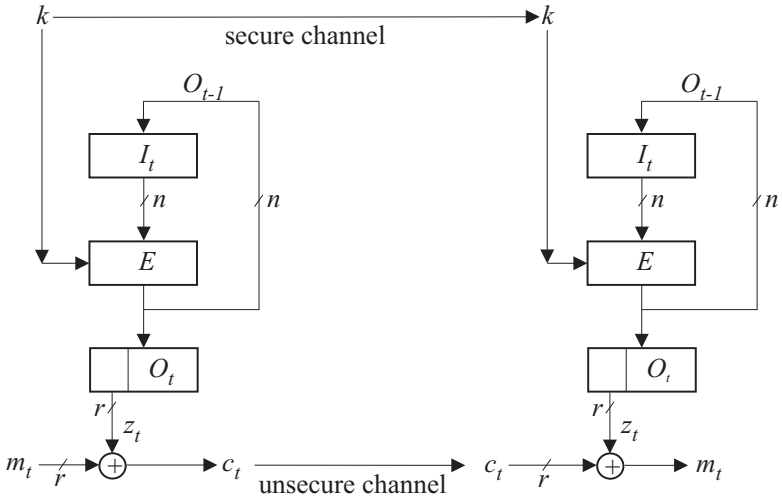


Fig. 4.2-3: OFB mode of operation for an n-bit block cipher with an r-bit keystream z_t.

properties of OFB mode

The OFB mode of operation has the following properties:

1. Changing the I_0 results in the same plaintext being encrypted to a different output.
2. The keystream z_t is not dependent on the plaintext.
3. Error propagation: a bit error in the ciphertext c_t exclusively affects the corresponding bit in the plaintext m_t.

4. For $r < n$ the throughput of a block cipher in the OFB mode of operation is decreased by the factor r/n, while the keystream z_t may be pre-computed, since the keystream is independent of the plaintext m_t.

5. The initialization vector I_0 does not need to be secret, but it should be changed when the key k is used again.

This version of the OFB mode of operation is described in [ISO91]. In [Fips81] you can also find an OFB mode of operation but it has a lower security level. Since in the OFB mode of operation the function E is used for encryption and decryption, it can not be used when E is an asymmetric cipher. The same applies to the CFB mode of operation which is introduced in the next section.

4.2.4 CFB mode

Each symmetric block cipher can be operated in the so-called *Cipher Feedback Mode* (CFB) of operation as a self-synchronizing stream cipher (see Fig. 4.2-4).

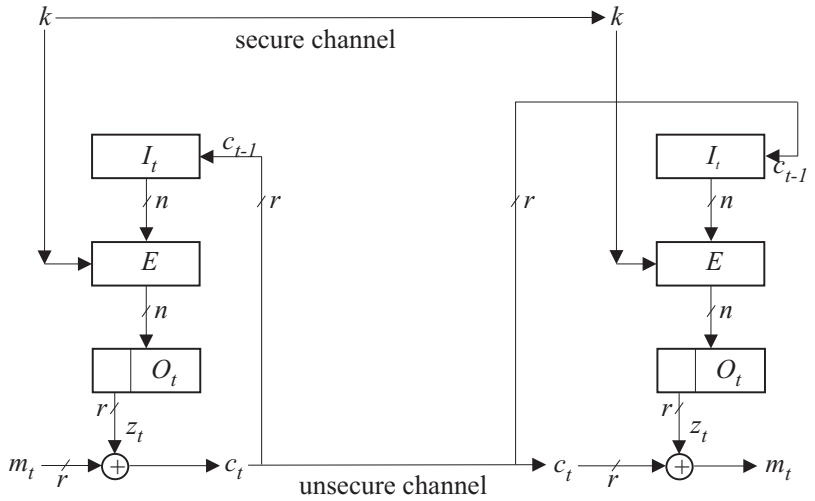


Fig. 4.2-4: CFB mode of operation for an n-bit block cipher with r-bit keystream z_t.

encryption and decryption in CFB

The *encryption* of r-bit plaintext blocks $m_t, t \geq 0$, under the secret key k, can be described as follows:

$$
\begin{aligned}
O_t &= E_k(I_t) \\
z_t &= \mathrm{Left}(O_t, r), \\
c_t &= m_t + z_t, \\
I_{t+1} &= \mathrm{ShiftLeft}(I_t, r) + c_t,
\end{aligned}
$$

while $t \geq 0$, I_t and O_t are n-bit blocks and the function $\mathrm{ShiftLeft}(I_t, r)$ shifts the n-bit block I_t to the left by r positions (this corresponds to a multiplication with 2^r), and in this process it removes all bits which are shifted over the n boundary.

The decryption of r-bit ciphertext blocks $c_t, t \geq 0$, under the secret key k, works as follows:

$$\begin{aligned} O_t &= E_k(I_t), \\ z_t &= \text{Left}(O_t, r), \\ m_t &= c_t + z_t, \\ I_{t+1} &= \text{ShiftLeft}(I_t, r) + c_t. \end{aligned}$$

properties of CFB mode

The CFB mode of operation has the following *properties*:

1. Different values of I_0 result in the same plaintext input being encrypted to a different output. The value of I_0 does not have to be secret. However, the receiver should be able to check its integrity and authentication.
2. Re-ordering ciphertext blocks affects the decryption.
3. Error propagation: Changing one bit in c_t affects the decryption of the next $\lceil n/r \rceil$ ciphertext blocks. Proper decryption of c_t requires the preceding $\lceil n/r \rceil$ blocks to be transmitted correctly. Consequently, the CFB mode of operation is self-synchronizing.
4. As far as the encryption function E is concerned, throughput is decreased by the factor r/n.

4.3 Data Encryption Standard (DES)

Data Encryption Standard

The *Data Encryption Standard* (DES) was standardized in 1977 by the American *National Bureau of Standards* (NBS), which is today called *National Institute of Standards and Technology NIST*, in the form of a *Federal Information Processing Standard* (FIPS) and was published in the FIPS Publication 46 ([Fips81]). Now, this technique has been adopted by other institutions as a standard, too.

The DES has always been criticized for the following reasons:

1. The short key length of 56 bits.
2. The secret design principles, especially the S-boxes used in the round function.

Nonetheless, DES was very common in many encryption products up until the mid 90s.

DES as a Feistel cipher

DES is a *Feistel cipher* which processes plaintext blocks of $n = 64$ bit, using a key k with an effective length of $l = 56$ bits. The Feistel cipher consists of $r = 16$ rounds. The DES gets a key k' of length 64 bits, of which 8 bits $(8, 16, 24, \ldots, 64)$ are used as parity bits and the remaining 56 bits are summed up to the actual key.

From the key k, 16 subkeys $k_i, 1 \leq i \leq 16$, each having 48 bits are generated for each round of the Feistel cipher. In the following, the plaintext block is called $m = (m_1, \ldots, m_{64}) \in \text{GF}(2)^{64}$ and the ciphertext block $c = (c_1, \ldots, c_{64}) \in \text{GF}(2)^{64}$.

First of all the plaintext m undergoes an initial bit permutation IP and is divided into 32 bit halves L_0 and R_0. The blocks are assigned to a Feistel cipher of $r = 16$ rounds:

$$\begin{aligned} L_i &= R_{i-1}, \\ R_i &= L_{i-1} + G(k_i, R_{i-1}) \end{aligned}$$

for $1 \leq i \leq 16$. In each of the 16 rounds the same function G is used. The blocks L_{16} and R_{16} are finally exchanged and undergo a bit permutation IP^{-1} in order to obtain the ciphertext block $c = (c_1, \ldots, c_{64})$. The internal round function G is composed of the following steps in round $i, 1 \leq i \leq 16$:

- Expanding the 32 bit-block R_{i-1} to 48 bits: $E : \mathrm{GF}(2)^{32} \rightarrow \mathrm{GF}(2)^{48}, T = E(R_{i-1})$.
- Bitwise XOR operation of T and the subkey k_i : $T' = T + k_i$.
- T' is divided into 8 blocks $B_1, \ldots, B_8$ with each having 6 bits: $T' = (B_1, \ldots, B_8)$.
- Block $B_j, 1 \leq j \leq 8$, undergoes a substitution S_j which transforms a 6-bit input to a 4-bit output: $S_j : \mathrm{GF}(2)^6 \rightarrow \mathrm{GF}(2)^4, B'_j = S_j(B_j)$. The eight substitutions are summed up to function S.
- The 4-bit blocks $B'_j, 1 \leq j \leq 8$, are summed up to a block T'' of length 32 bits:

$$T'' = (B'_1, \ldots, B'_8) = (S_1(B_1), \ldots, S_8(B_8)) \ .$$

- T'' is bit-permuted by P : $T''' = P(T'')$.

To sum up, the function G for round i can be described as follows:

$$\begin{aligned} T''' &= G(k_i, R_{i-1}) \\ &= P(S(E(R_{i-1}) + k_i)) \\ &= P(S(T')) \\ &= P((S_1(B_1), \ldots, S_8(B_8))) \\ &= P(T'') \ . \end{aligned}$$

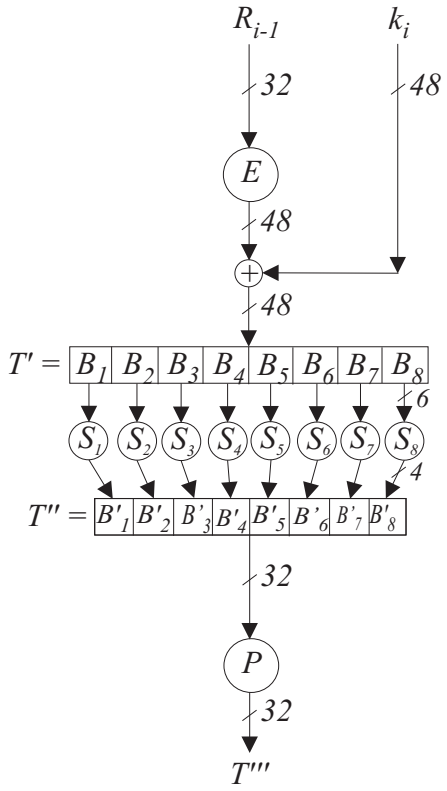


Fig. 4.3-1: DES inner round function G.

The inner round function G is illustrated in Fig. 4.3-1. A precise description of the initial bit permutation IP, the expansion function E, the substitutions S_j, the bit permutation P and how the single subkeys k_i are computed can be found in [Menezes96], [Schneier96] or [Fumy94]. A related animation can be found on [Kad97].

4.4 International Data Encryption Algorithm (IDEA)

IDEA (*International Data Encryption Algorithm*) is an encryption algorithm developed at ETH in Zurich, Switzerland. A preliminary draft of IDEA was published by Lai and Massey under the name PES (*Proposal Encryption Standard*) in 1990 [Lai91]. In 1991, after the publishing of the differential attack by Biham and Shamir[Biham93], Lai and Massey modified their block cipher in order to be resistent against this attack and named it IPES (*Improved Proposal Encryption Standard*). In 1992, the name IPES was changed in IDEA[Lai92].

IDEA is a 64-bit block cipher with a 128-bit key, and is generally considered to be very secure. It is considered among the best publicly known algorithms. IDEA is patented in the United States and in most of the European countries. The patent is held by the Swiss company Ascom-Tech.

4.4.1 Design concept of IDEA

The design concept of IDEA is based on mixing operations from three different algebraic groups. Supposing that m_i and m_j are two subblocks with 16 bit length, these three operations are:

- Bitwise XOR of two 16-bit subblocks m_i and m_j, denoted by $m_i \oplus m_j$ in $(\mathrm{GF}(2))^{16}$. For example:

$$\begin{array}{r} 1\,1\,0\,0\,1\,0\,0\,1\,1\,0\,1\,1\,0\,0\,0\,1 \\ \oplus\,0\,0\,1\,1\,1\,0\,0\,1\,1\,1\,1\,0\,1\,1\,0\,0 \\ \hline 1\,1\,1\,1\,0\,0\,0\,0\,0\,1\,0\,1\,1\,1\,0\,1 \end{array}$$

- Addition modulo 2^{16} in $\mathbb{Z}_2^{16}$ denoted by $m_i \boxplus m_j$. This is an addition of 16-bit numbers ignoring any overflow. For example:

$$\begin{array}{r} 1\,1\,0\,0\,0\,0\,0\,1\,1\,0\,0\,1\,1\,0\,1\,0\,1 \\ \boxplus\,1\,1\,1\,0\,0\,0\,0\,0\,1\,1\,0\,0\,1\,1\,0\,1\,0 \\ \hline 1\,0\,1\,0\,0\,1\,0\,0\,1\,1\,0\,0\,1\,1\,1\,1 \end{array}$$

- Multiplication modulo $2^{16}+1$ in $\mathbb{Z}^*_{2^{16}+1}$ denoted by $m_i \odot m_j$, where $0 \in \mathbb{Z}_2^{16}$ is associated with $2^{16} \in \mathbb{Z}^*_{2^{16}+1}$. With this supposition, 0 is equivalent to $-1 \mod 2^{16}+1$ and the multiplicative inverse modulo $2^{16}+1$ of 0 is 0. Actually, $\odot$ is multiplication of 16-bit numbers ignoring overflow. For example:

$$\begin{array}{r} 1\,0\,1\,0\,1\,0\,1\,1\,1\,1\,0\,0\,1\,1\,0\,1 \\ \odot\,1\,1\,1\,1\,0\,0\,0\,1\,1\,0\,1\,1\,1\,1\,0\,0 \\ \hline 1\,0\,0\,0\,0\,1\,0\,1\,0\,1\,0\,1\,0\,0\,1\,1 \end{array}$$

The mixing of three different group operations in IDEA enables confusion and diffusion of the input bits. The confusion obscures the relationship between the plaintext and the ciphertext. With the diffusion, the redundancy of the plaintext will be spreaded over the ciphertext. An attacker looking for these redundancies will have a hard job to find them.

4.4.2 IDEA Encryption

The plaintext block in IDEA is divided into four message blocks $m_0 m_1 m_2 m_3$ $(N = 4)$. Each message block m_t, with $0 \leq t < N-1$, has the bit size $n = 16$.

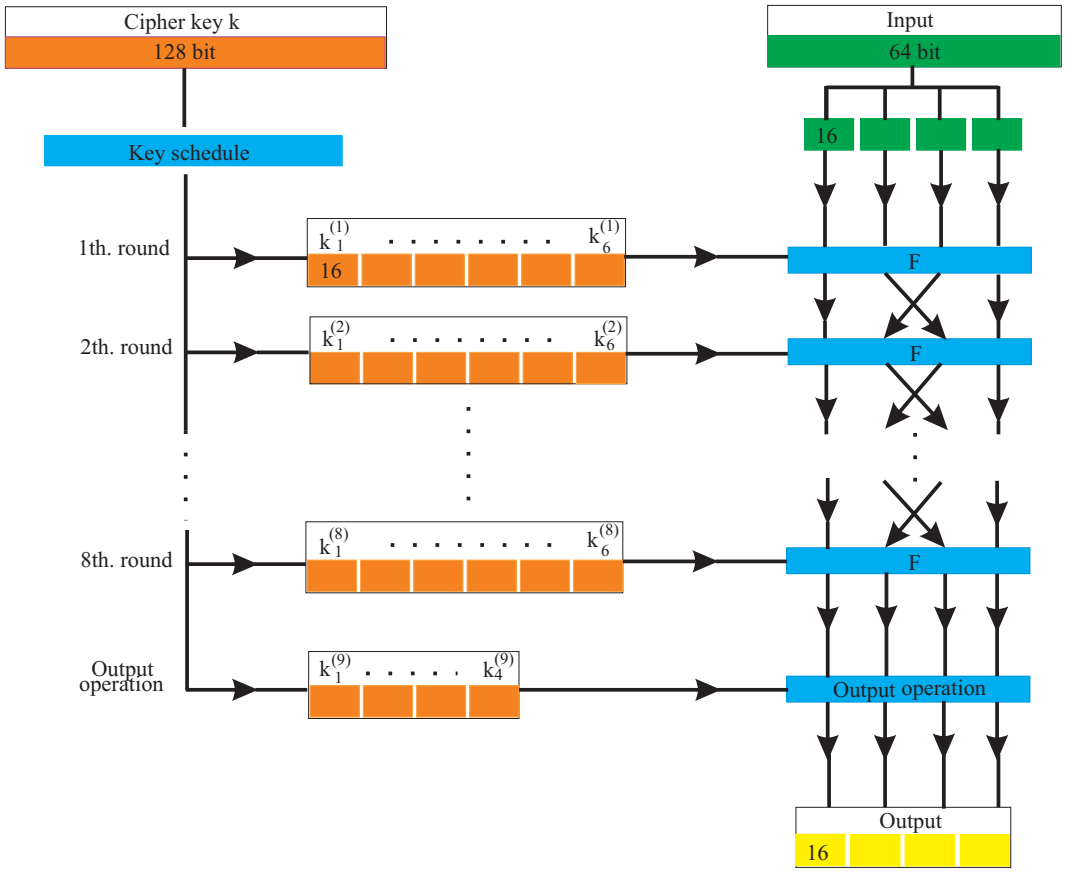


Fig. 4.4-1: The computational procedure of IDEA.

IDEA uses 52 subkeys, each of 16 bit length. Using IDEA, the encryption (resp. the decryption) of the plaintext (resp. the ciphertext) is performed in eight rounds (see Fig. 4.4-1) followed with an output transformation. Each of the 8 rounds requires 6 subkeys and in the output transformation only 4 subkeys are needed. So, altogether 52 subkeys are needed.

In the following we describe in detail the most important building blocks of IDEA: the key schedule algorithm, the round transformation F and the output transformation.

The IDEA Key Schedule Algorithm

The IDEA key schedule algorithm derives 52 16-bit subkeys from the 128-bit secret key k. First the 128-bit secret key k is divided into eight 16-bit subkeys. These are the six subkeys for the first round and the first two subkeys for the second round. Then the key k is rotated 25 bits to the left (that is, the operation $\mathrm{ShiftLeft}(k, 25)$ is performed) and again divided into eight subkeys. The first four are used in round two; the last four are used as the first four subkeys of round three, and so on until all 52 subkeys are generated.

Now we adopt the following notation: For each round $r\,(1 \leq r \leq 8)$, the four message blocks input to the round r are denoted $m_0^r\, m_1^r\, m_2^r\, m_3^r$ and the 6 subkeys used in this round are denoted $k_0^r k_1^r k_2^r k_3^r k_4^r k_5^r$. Hence, 48 subkeys are used in eight

rounds. The remaining 4 subkeys from the 52 subkeys, which are used in the output transformation, are denoted $k_0^o k_1^o\ k_2^o k_3^o$.

Round Transformation

As mentioned earlier, the IDEA algorithm consists of 8 round transformations and an output transformation. The round transformation F indexed by i takes four 16-bit subblocks and performs the group operations according to Fig. 4.4-2 with the use of 6 subkeys $k_j^{(i)}$, $1 \leq j \leq 6$ to output four 16-bit subblocks.

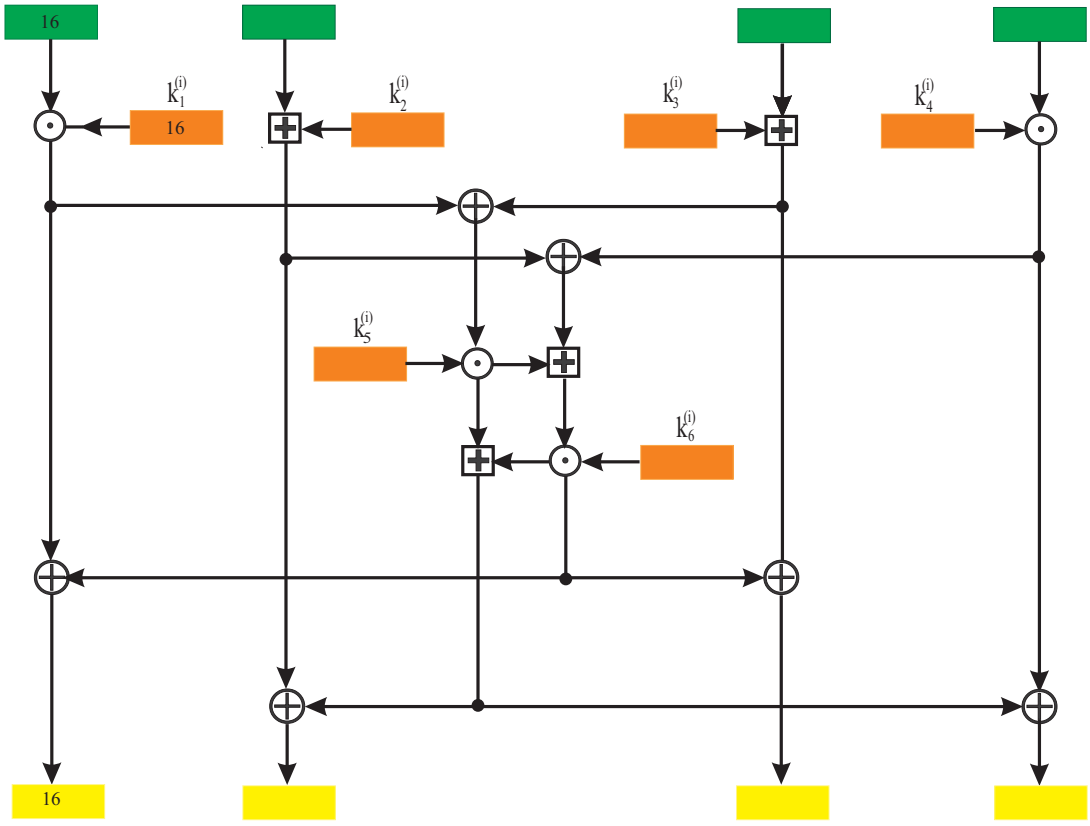


Fig. 4.4-2: The round transformation F.

In each round r, the sequence of events is as follows (see Fig. 4.4-2) [Schneier96]:

1. Multiply m_0^r and the first subkey k_0^r.
2. Add m_1^r and the second subkey k_1^r.
3. Add m_2^r and the third subkey k_2^r.
4. Multiply m_3^r and the fourth subkey k_3^r.
5. XOR the results of steps 1 and 3.
6. XOR the results of steps 2 and 4.
7. Multiply the results of step 5 with the fifth subkey k_4^r.

8. Add the results of steps 6 and 7.
9. Multiply the results of step 8 with the sixth subkey k_5^r.
10. Add the results of steps 7 and 9.
11. XOR the results of steps 1 and 9.
12. XOR the results of steps 3 and 9.
13. XOR the results of steps 2 and 10.
14. XOR the results of steps 4 and 10.

Between each round the second and third output message blocks are swapped. The output of the round is the four message blocks that are the results of steps 11, 13, 12, and 14. After swapping the inner blocks, the input to the next will then be the message blocks resulting from steps 11, 12, 13, and 14 successively. We further denote by $c_0^8\, c_1^8 c_2^8 c_3^8$ the output of the eight round (see Fig. 4.4-2).

IDEA Output Transformation

After performing the 8 round transformation of IDEA, an output transformation is performed. It is shown in Fig. 4.4-3. It takes the output of the eight round transformation as input and gives four 16-bit subblocks as output, which are attached and in this way form the ciphertext block.

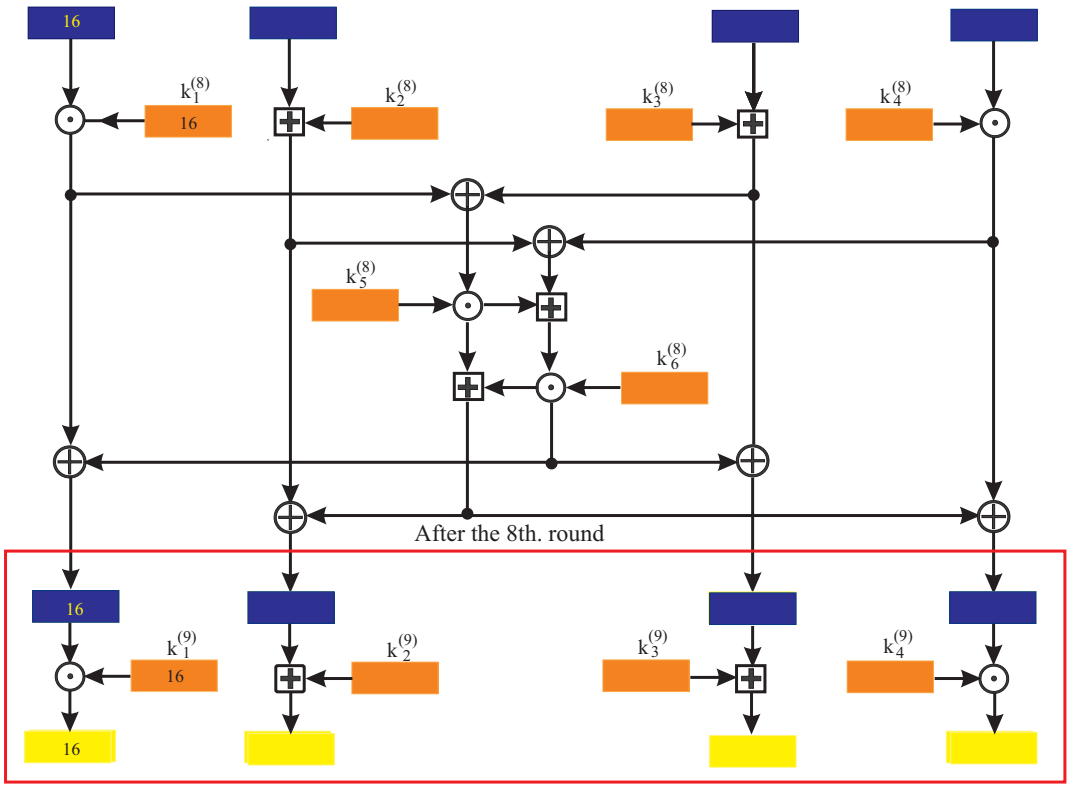


Fig. 4.4-3: The output transformation of IDEA.

The output transformation consists of the following operations:

1. Multiply c_0^8 and k_0^o.
2. Add c_1^8 and k_2^o.
3. Add c_2^8 and k_1^o.
4. Multiply c_3^8 and k_3^o.
5. Finally, the four message blocks c_0, c_1, c_2, and c_3 are attached to form the ciphertext.

4.4.3 IDEA Decryption

The IDEA decryption algorithm is the same as the encryption algorithm except for the used subkeys.

So the subkeys used for decryption are computed from those used for encryption according to Table 4.4-1 [Schneier96], where the notation $(k_i^j)^{-1}$ and $-k_i^j$ denote the multiplicative inverse modulo $2^{16}+1$ and the additive inverse modulo 2^{16} of k_i^j respectively, $0 \le i \le 5$ and $1 \le j \le 8$ or $j = o$:

Tab. 4.4-1: *Derivation of the decryption subkeys from the encryption subkeys.*

Round	Encryption subkeys	Decryption subkeys
1	$k_0^1 k_1^1 k_2^1 k_3^1 k_4^1 k_5^1$	$(k_0^o)^{-1} - k_1^o - k_2^o (k_3^o)^{-1} k_4^8 k_5^8$
2	$k_0^2 k_1^2 k_2^2 k_3^2 k_4^2 k_5^2$	$(k_0^8)^{-1} - k_1^8 - k_2^8 (k_3^8)^{-1} k_4^7 k_5^7$
3	$k_0^3 k_1^3 k_2^3 k_3^3 k_4^3 k_5^3$	$(k_0^7)^{-1} - k_1^7 - k_2^7 (k_3^7)^{-1} k_4^6 k_5^6$
4	$k_0^4 k_1^4 k_2^4 k_3^4 k_4^4 k_5^4$	$(k_0^6)^{-1} - k_1^6 - k_2^6 (k_3^6)^{-1} k_4^5 k_5^5$
5	$k_0^5 k_1^5 k_2^5 k_3^5 k_4^5 k_5^5$	$(k_0^5)^{-1} - k_1^5 - k_2^5 (k_3^5)^{-1} k_4^4 k_5^4$
6	$k_0^6 k_1^6 k_2^6 k_3^6 k_4^6 k_5^6$	$(k_0^4)^{-1} - k_1^4 - k_2^4 (k_3^4)^{-1} k_4^3 k_5^3$
7	$k_0^7 k_1^7 k_2^7 k_3^7 k_4^7 k_5^7$	$(k_0^3)^{-1} - k_1^3 - k_2^3 (k_3^3)^{-1} k_4^2 k_5^2$
8	$k_0^8 k_1^8 k_2^8 k_3^8 k_4^8 k_5^8$	$(k_0^2)^{-1} - k_1^2 - k_2^2 (k_3^2)^{-1} k_4^1 k_5^1$
Output transformation	$k_0^o k_1^o k_2^o k_3^o$	$(k_0^1)^{-1} - k_1^1 - k_2^1 (k_3^1)^{-1}$

The inverse round transformation used in the IDEA decryption algorithm is equivalent to the round transformation F, but it is used now with the appropriate decryption subkeys (Fig. 4.4-4).

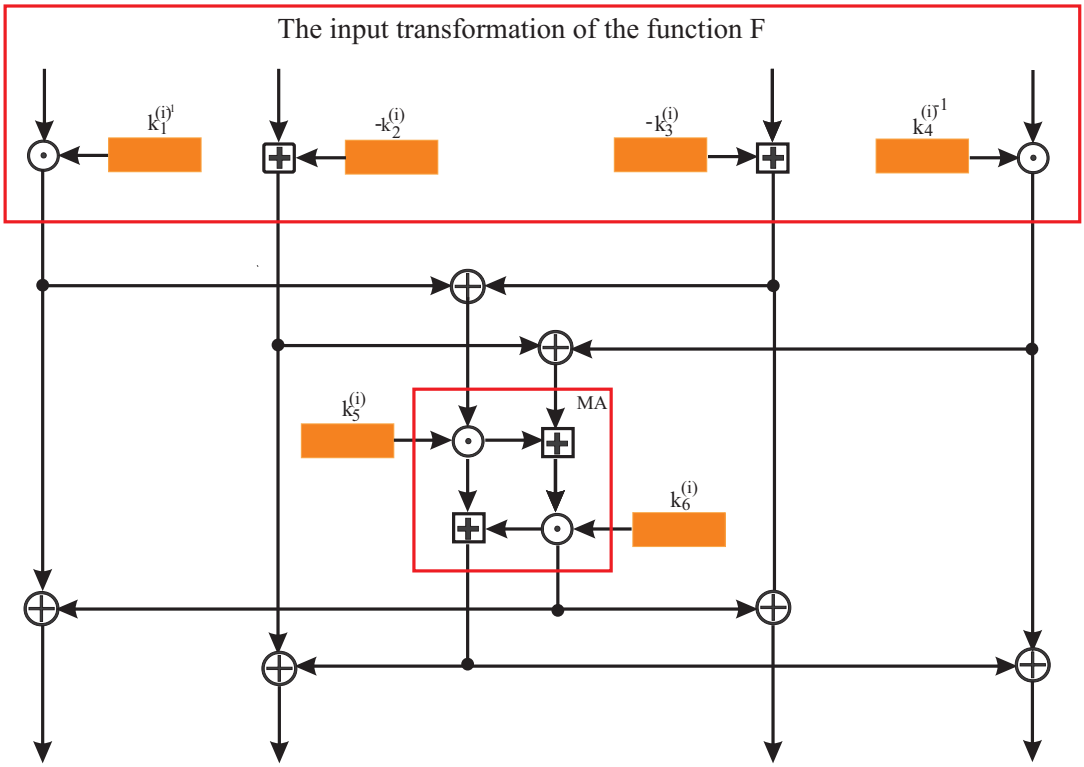


Fig. 4.4-4: The inverse transformation of F.

4.4.4 Security and Implementation Issues

From a security point of view, IDEA appears to be secure against known attacks except attacks related to the so called weak keys [Daemen94]. These keys are weak in the sense that their use can be detected only with a very small amount of effort. To avoid this attack, Daemon [Daemen94] proposes a slight modification to the key schedule algorithm. On the other hand, the key bit length of 128 strengthens the IDEA block cipher against an exhaustive search attack.

IDEA can work within any cipher mode of operation described in Section 4.2 (a related java applet can be found on the book homepage). From the efficiency point of view, IDEA is fast in software and in hardware implementation. The Swiss company Ascom-Tech has implemented IDEA on a chip and has reached an encryption rate of 177 Mbit/s.

4.5 Advanced Encryption Standard (AES)

The National Institute of Standards and Technology (NIST) has worked together with the industry and the cryptographic community to develop an Advanced Encryption Standard (AES). The AES will replace the former FIPS[7] Standard DES, which is vulnerable to many known attacks such as differential cryptanalysis [Biham93], linear cryptanalysis [Matsui94] and exhaustive search. The new

7 Federal Information Processing Standard.

developed standard must specify an encryption algorithm(s) capable of protecting sensitive government information well into the next decades. The algorithm(s) will be used by the U.S. Government and other private sectors.

4.5.1 Selection of Algorithms for AES

On January 2, 1997, NIST announced the initiation of the AES development effort and made a formal call for algorithms on September 12, 1997. The call described the requirements for candidate algorithm submission packages and the minimum requirements for acceptance that must be satisfied by the AES candidate, as well as the evaluation criteria to be used to appraise the candidate algorithms. The evaluation criteria belonged into three categories:

- Security: includes resistance of the algorithm to cryptanalysis, reliability of its mathematical basis, randomness of the algorithm output, etc.
- Cost: encompasses computational efficiency (speed) on various platforms, and memory requirements.
- Algorithm and implementation characteristics such as flexibility (ability of an algorithm to be implemented as a stream cipher or hash algorithm, for example), hardware and software suitability, etc.

Furthermore, the call declared, among other things, that the algorithm(s) must implement symmetric key cryptography as a block cipher and (at a minimum) support block sizes of 128-bits and key sizes of 128, 192, and 256 bits.

On August 20, 1998, fifteen AES candidate algorithms were established at the First AES Candidate Conference (AES1). These algorithms were subject to further study and research by the cryptographic community from around the world. Based on analysis and comments on the fifteen algorithms, NIST selected five algorithms from the fifteen at the Second AES Candidate Conference (AES2). The AES finalist candidate algorithms were MARS, RC6, Rijndael, Serpent, and Twofish.

These finalist algorithms received a second, more in-depth analysis on any aspect of the candidate algorithms, including the following topics: cryptanalysis, intellectual property, crosscutting analysis of all of the AES finalists, overall recommendations and implementation issues. On April, 2000, at the Third AES Candidate Conference (AES3), submitters of the AES finalists were invited to discuss comments on their algorithms. When the selection process finished, NIST studied all available information in order to make a selection for the AES. On October 2, 2000, NIST announced that it has selected the Rijndael algorithm to propose for the AES.

4.5.2 The Rijndael Algorithm: Some Notions

The winner of the AES selection process, the Rijndael algorithm, is a block cipher, designed by Joan Daemen and Vincent Rijmen at the Katholieke University Leuven in Belgium. This cipher has a variable block and key length. The Rijndael algorithm uses keys with a length of 128, 192, or 256 bits to encrypt blocks with a length of

128, 192 or 256 bits. All nine combinations of key length and block length are possible. Because of its importance, we will describe this algorithm in some detail.

state

Before we give a description of the AES standard, we introduce the notion of *state*[Fips01]. A state consists of four rows of bytes, each containing N_b bytes, where N_b is the block length divided by 32, that is $N_b = 4$.[8] In the state array denoted by the symbol s, each individual byte has two indices: its row number r, $0 \leq r < 4$ and its column number c, $0 \leq c < 4$. This allows an individual byte of the state to be referred to as either $s_{r,c}$ or $s[r, c]$.

In the AES algorithm, the basic processing unit is the byte. In what follows, a byte will be treated as a single entity and will be written in hexadecimal representation. For example, the byte 01100011 is written as $(63)_{16}$. For simplicity, we will drop off the index 16 and write the byte as two characters.

state array

At the start of the cipher operation and inverse cipher operation described later, the input – an array of bytes $in_0, in_1, \ldots\ in_{15}$ – is copied into the *state array* as illustrated in Fig. 4.5-1. The cipher or inverse cipher operations are then conducted on this state array, after which its final value is copied to the output – the array of bytes $out_0, out_1, \ldots\ out_{15}$.

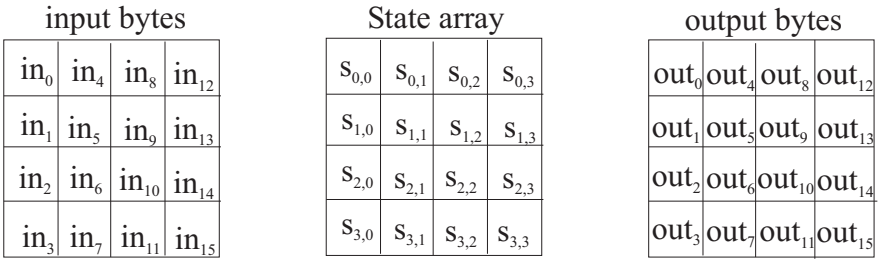


Fig. 4.5-1: State array input and output [Fips01].

Hence, at the beginning of the cipher or inverse cipher, the input array, denoted by in, is copied to the state array according to the scheme:

$$s[r, c] = in[r + 4c] \text{ for } 0 \leq r < 4 \text{ and } 0 \leq c < 4, \qquad \text{4.5-1}$$

and at the end of the cipher and inverse cipher, the state is copied to the output array, denoted by out, as follows:

$$out[r + 4c] = s[r, c] \text{ for } 0 \leq r < 4 \text{ and } 0 \leq c < 4. \qquad \text{4.5-2}$$

8 For 192-bits and 256-bits data blocks – as in original Rijndael algorithm – N_b will be 6 and 8 respectively.

Example 4.5-1:
We consider the input array in `00112233445566778899aabbccddeeff`, that is:

$$in_0 = \texttt{00},\ in_1 = \texttt{11}, \ldots, in_{14} = \texttt{ee},\ in_{15} = \texttt{ff}.$$

According to Eq. 4.5-1, the input array is copied to the state s as follows:

$s[0,0] = in[0] =$ `00`	$s[0,1] = in[4] =$ `44`	$s[0,2] = in[8] =$ `88}`	$s[0,3] =$ $in[12] =$ `cc`
$s[1,0] = in[1] =$ `11`	$s[1,1] = in[5] =$ `55`	$s[1,2] = in[9] =$ `99`	$s[1,3] =$ $in[13] =$ `dd`
$s[2,0] = in[2] =$ `22`	$s[2,1] = in[6] =$ `66`	$s[2,2] =$ $in[10] =$ `aa`	$s[2,3] =$ $in[14] =$ `ee`
$s[3,0] = in[3] =$ `33`	$s[3,1] = in[7] =$ `77`	$s[3,2] =$ $in[11] =$ `bb`	$s[3,3] =$ $in[15] =$ `ff`

4.5.3 AES Encryption

AES uses different structure than DES. It is not a Feistel cipher. In brief, it consists of subsequent similar rounds. The plaintext comes in as 16 bytes at the very top. The first operation is to XOR the plaintext with 16 bytes (128 bits) of round key. Each of the 16 bytes is then used as an index into an S-box table that maps 8-bit inputs into 8-bit outputs. The S-boxes are all identical. The bytes are then rearranged in a specific order that looks a bit complicated but has a simple structure. Finally, the bytes are mixed into groups of four using a linear mixing function. The term linear just means that each output bit of the mixing function is the XOR of several of the input bits. This completes a single round. As mentioned in the previous section, a full encryption consists of 10-14 rounds, depending on the size of the key. Like in DES and IDEA, there is a key schedule that generates the round keys.

Let us now have a detailed look at the algorithm (see the pseudo code given in Listing 4.5-1). The encryption consists of the following steps:

- An initial round key addition that is the XOR addition of the first round key and the input state written as in Eq. 4.5-1.
- $N_r - 1$ rounds: The resulting state array is transformed by implementing a round function 10, 12, or 14 times (depending on the key length). The round transformation is obtained by subsequently applying the transformations `SubBytes()`, `ShiftRows()`, `MixColumns()`, and `AddRoundKey()`.
- A final round which is different to the "normal" round by removing the `MixColumns()` transformation.

The final state is then copied to the output as described in Eq. 4.5-2. The round function is parameterized using the AES key schedule.

Listing 4.5-1: Ciphering with the use of AES.

```
Cipher(byte in[16], byte out[16], word w[4·(Nr + 1)])
begin
  byte state[4,4]
  state = in
  AddRoundKey(state, w[0, 3])
  for round = 1 step 1 to Nr-1
    SubBytes(state)
    ShiftRows(state)
    MixColumns(state)
    AddRoundKey(state, w[round.4,(round+1).3])
  end for
  SubBytes(state)
  ShiftRows(state)
  AddRoundKey(state, w[Nr ·4,(Nr + 1)·3])
  out = state
end
```

For the AES algorithm, the length of the cipher key k can be 128, 192, or 256 bits. The key length is represented by $N_k = 4$, 6, or 8, which reflects the number of 32-bit words (number of columns) in the cipher key. The number of rounds to be performed during the execution of the AES algorithm depends on the key size. The number of rounds is represented by N_r, where $N_r = 10$ when $N_k = 4, N_r = 12$ when $N_k = 6$, and $N_r = 14$ when $N_k = 8$. Table 4.5-1 summarizes the only key-block-round combinations conform to the standard.

Tab. 4.5-1: *Key-block-round combinations.*

	Key length (N_k words)	Block size (N_b words)	Number of rounds
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Why is the number of rounds N_r equal to 10 for a block length of 128 bits and a key length of 128 bits ($N_k = 4$)? Note first that no attacks, other than the exhaustive key search, have been found against the Rijndael algorithm with more than 6 rounds [Daemen99]. The answer of the former question is based on two security issues regarding the Rijndael algorithm.

- Two rounds of Rijndael provide **full diffusion** as introduced by Daemen and Rijmen, since every state bit depends on all state bits two rounds ago[9]. As mentioned above more than 6 rounds of the Rijndael algorithm are secure against all known attacks for a block length of 128 bits and a key length of 128 bits. To the 6 rounds Daemen and Rijmen added 4 rounds as a security margin or more precisely as adding a full diffusion at the beginning and at the end of the cipher.

9 Recall that a diffusion refers to rearranging or spreading out the bits in the message so that any redundancy in the plaintext is spread out over the ciphertext.

- Many cryptographic attacks against block cipher such as differential cryptanalysis or linear cryptanalysis exploit the propagation of input messages through n rounds in order to attack $n+1$ or $n+2$ rounds of the cipher. Thus, 4-round propagation structure can be used to attack 6 rounds of Rijndael. With 10 rounds, the number of rounds through which a propagation (of the input messages) has to be found will double.

For other key lengths, the number of rounds augments by one for every 32 bits of the cipher key. That is for 192-bits and 256-bits key lengths the number of rounds is 12 and 14 respectively (see Table 4.5-1). On the other hand, many attacks can be mounted against block ciphers by exploiting the knowledge of cipher key bits or the ability to use different cipher keys. Hence an increase of the key length leads to an increase of the range of possibilities available to the cryptanalyst. This facility can be compensated by increasing the number of rounds of the cipher.

The Key Expansion Algorithm

Using the cipher key k, the key expansion routine generates a key schedule with a total of $4 \cdot (N_r + 1)$ words. In this context, the word is a group of 32 bits. The resulting key schedule is a linear array of words denoted $[w_i]$ or $w[i]$, with i in the range of $0 \leq i < 4 \cdot (N_r + 1)$.

The key expansion routine is shown in the Listing 4.5-2.

Listing 4.5-2: The pseudo code for key expansion.

```
KeyExpansion(byte key[4· Nk], word w[4·(Nr + 1)], Nk)
begin
  word temp
  i = 0
  while (i < Nk)
    w[i] = word(key[4·i], key[4·i+1], key[4·i+2], key[4·i+3])
    i = i+1
  end while
  i = Nk
  while (i < 4·(Nr+1))
    temp = w[i-1]
    if (i mod Nk = 0)
      temp = SubWord(RotWord(temp)) XOR Rcon[i/Nk]
    else if (Nk > 6 and i mod Nk=4)
      temp = SubWord(temp)
    end if
    w[i] = w[i-Nk] XOR temp
    i = i + 1
  end while
end
```

Here, the `SubWord()` function is a function that takes a four-byte input. It uses the S-box (see Tab. 4.5-2) to substitute each of the four bytes. The function `RotWord()` performs a cyclic permutation over the input word $[a_0, a_1, a_2, a_3]$ and returns the output word $[a_1, a_2, a_3, a_0]$. The function `Rcon[i]` outputs the

four-byte word $[x^{i-1}, \texttt{00}, \texttt{00}, \texttt{00}]$, with x^{i-1} being powers of x in the field GF(2^8) with the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$.

The round keys are taken from the words generated by the key expansion routine as follows: the first round key consists of the first four words, the second of the following four words, and so on. From the pseudo code given in Listing 4.5-2, it can be seen that the first N_k words of the expanded key contain the cipher key k. Every following word, $w[i]$, is equal to the XOR of the previous word, $w[i-1]$, and the word N_k positions earlier, $w[i-N_k]$. For words in positions that are a multiple of N_k, a transformation is applied to $w[i-1]$ prior to the XOR, followed by an XOR with a round constant, `Rcon[i]`. This transformation consists of a cyclic shift of the bytes in a word `SubWord()`, followed by the application of a table lookup to all four bytes of the word `SubWord()`[Daemen99].

The key expansion routine for cipher key with 256 bits length ($N_k = 8$) is slightly different from those for 128-bit key ($N_k = 4$) or 192-bit key ($N_k = 6$). If $N_k = 8$ and $i - 4$ is a multiple of N_k, then `SubWord()` is applied to $w[i-1]$ prior to the XOR [Fips01].

Example 4.5-2:
The function `Rcon[10]` with $i = 10$ outputs the word $[x^9, \texttt{00}, \texttt{00}, \texttt{00}]$. x^9 in GF(2^8) is set to the value $x^9 \, mod x^8 + x^4 + x^3 + x + 1$, that gives the polynomial $x^5 + x^4 + x^2 + x$. The later can be written as `36`. Thus, `Rcon[10] = [36, 00, 00, 00]`.

Example 4.5-3: [Fips01]
Let the cipher key be set to `000102030405060708090a0b0c0d0e0f`. Regarding the pseudo code of the key expansion routine, we have $w[0] =$ `00010203`, $\texttt{w}[1] =$ `04050607`, $\texttt{w}[2] =$ `08090a0b`, $\texttt{w}[3] =$ `0c0d0e0f`, $\texttt{w}[4] =$ `d6aa74fd`. The first round key contains the cipher key.

The AES key expansion has been chosen for many reasons [Daemen99]. It ensures the diffusion of the cipher key into the round keys. It provides enough non-linearity by using the S-box substitution and prohibits therewith the determination of round key differences from cipher key differences only. One important feature of the key expansion is that a knowledge of any N_k consecutive words of the expanded key shall allow to regenerate the whole round keys. This is useful for applications which require key regeneration mechanisms. Other design criteria of the key expansion can be found in [Daemen99].

Round Transformation

The round transformation in AES consists of four different transformations: `SubBytes()`, `ShiftRows()`, `MixColumns()`, and `AddRoundKey()`. Each of them takes an input state of 128 bits and outputs a state of 128 bits. We now describe these transformations.

`SubBytes(state)` Transformation

The `SubBytes()` is a non-linear byte function and replaces each byte of the input state by another byte with the use of a substitution table called S-Box. It operates independently on each byte of the state and is formally composed of two byte-operations:

1. Take the multiplicative inverse in the finite field GF(2^8) with the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$.[10] The element `00` is mapped onto itself. The mapping $x \longrightarrow x^{-1}$ in GF(2^8) is chosen as an operation of the `SubBytes()` transformation, because it enables to protect the AES cipher against differential and linear cryptanalysis and to guarantee the invertibility of the cipher.

2. Apply the following affine transformation over GF(2):

$$b_i^{'} = b_i \oplus b_{(i+4) \mod 8} \oplus b_{(i+5) \mod 8} \oplus b_{(i+6) \mod 8} \oplus b_{(i+7) \mod 8} \oplus c_i$$

   for $0 \leq i < 8$, where b_i is the ith bit of the byte b, and c_i is the ith bit of a byte c with the value `63`. $\oplus$ denotes the exclusive-or operation. This invertible affine transformation is applied after the mapping $x \longrightarrow x^{-1}$ over GF(2^8) in order to avoid attacks such as the interpolation attack [Jakobsen97] that exploits mapping with very simple algebraic expression[11].

Based on these two operations, the S-box used in the `SubBytes(state)` transformation can be given as shown in Tab. 4.5-2.

10 See Section 2.4 in Chapter 2 to understand the arithmetic in GF(2^8).

11 Like the mapping $x \longrightarrow x^{-1}$ in GF(2^8).

Tab. 4.5-2: *S-box: substitution values for the byte xy (in hexadecimal format).*

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

For example, if $s_{1,1}$ =53, then the substitution value is determined by the intersection of the row with index '5' and the column with index '3' in the S-box table. This results in $s'_{1,1}$ having a value of ed.

ShiftRows() Transformation

In the ShiftRows(), the rows of the input state are cyclically shifted over different number of bytes. The first row, $r = 0$, is not shifted. The other rows are shifted according to Fig. 4.5-2.

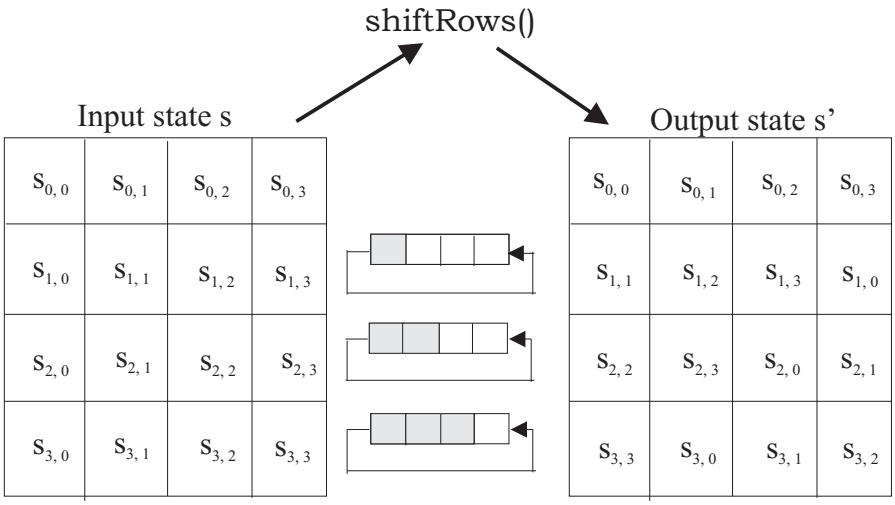


Fig. 4.5-2: The ShiftRows() transformation [Fips01].

Formally, the ShiftRows() transformation proceeds as follows:

$$s'_{r,c} = s_{r,(c+shift(r,4)) \mod 4} \text{ for } 0 \leq r < 4 \text{ and } 0 \leq c < 4, \qquad 4.5\text{-}3$$

where the shift value $shift(r, 4)$ depends on the row number r as follows: $shift(1, 4) = 1;\ shift(2, 4) = 2;\ shift(3, 4) = 3$.

Example 4.5-4:
After the use of the ShiftRows() transformation, the state s will be changed to the state s' both with values illustrated in the following table:

s	63cab7040953d051cd60e0e7ba70e18c
s'	6353e08c0960e104cd70b751bacad0e7

For example, $s'_{0,0} = s_{0,0} = 63,\ s'_{0,3} = s_{0,3} = ba$,

$$\begin{aligned} s'_{1,3} &= s_{1,(3+shift(1,4)) \mod 4} \\ &= s_{1,(3+1) \mod 4} \\ &= s_{1,0} \\ &= ca. \end{aligned}$$

The `ShiftRows()` transformation is a substitution operation of each byte of the rows ($r = 1$, 2, or 3) by another byte of the same row. It enables the resistance of the AES cipher against the square attack [Daemen97] and attacks using truncated differentials. The latter attack is a variant of the differential attack [Knudsen95].

`MixColumns()` Transformation

Whereas the `ShiftRows()` transformation affects the rows of the input state, the `MixColumns()` transformation operates on the state column-by-column. Each column of the state is considered as a polynomial over GF(8) and multiplied modulo $x^4 + 1$ with a fixed polynomial $c(x)$, given by

$$c(x) = \texttt{03}x^3 + \texttt{01}x^2 + \texttt{01}x + \texttt{02}. \tag{4.5-4}$$

Treating each column of the input state s as a four-term polynomial, the `MixColumns()` transformation outputs the state s' verifying the following equations:

$$\begin{aligned} s'_{0,c} &= (\texttt{02} \bullet s_{0,c}) \oplus (\texttt{03} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \\ s'_{1,c} &= s_{0,c} \oplus (\texttt{02} \bullet s_{1,c}) \oplus (\texttt{03} \bullet s_{2,c}) \oplus s_{3,c} \\ s'_{2,c} &= s_{0,c} \oplus s_{1,c} \oplus (\texttt{02} \bullet s_{2,c}) \oplus (\texttt{03} \bullet s_{3,c}) \\ s'_{3,c} &= (\texttt{03} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\texttt{02} \bullet s_{3,c}), \end{aligned} \tag{4.5-5}$$

where $\bullet$ denotes the multiplication in the finite field GF(2^8) with the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$ and $0 \leq c < (N_b = 4)$.

The same equations can be written in a matrix notation as follows:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{02} & \texttt{03} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{03} \\ \texttt{03} & \texttt{01} & \texttt{01} & \texttt{02} \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad 0 \leq c < (N_b = 4).$$

Example 4.5-5:
After the use of the `MixColumns()` transformation, the state s will be changed to the state s' both with values illustrated in Table 4.5-3.

Tab. 4.5-3: *An example of using the `MixColumns()` transformation.*

s	`6353e08c0960e104cd70b751bacad0e7`
s'	`5f72641557f5bc92f7be3b291db9f91a`

Example 4.5-6:
We now look at how the value $s'_{1,2}$ is obtained. From Eq.4.5-5, we have

$$\begin{aligned} s'_{1,2} &= s_{0,2} \oplus (\texttt{02} \bullet s_{1,2}) \oplus (\texttt{03} \bullet s_{2,2}) \oplus s_{3,2}. \\ &= \texttt{cd} \oplus (\texttt{02} \bullet \texttt{70}) \oplus (\texttt{03} \bullet \texttt{b7}) \oplus \texttt{51}. \end{aligned}$$

Recall that $\bullet$ is the multiplication in GF(2^8) with the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$. We calculate the value $\texttt{03} \bullet \texttt{b7}$ in GF(2^8). We write `03` and `b7` in binary representation, and we get the values 00000011 and 10110111 respectively. These two values can be written as polynomials $x+1$ for 00000011 and $x^7 + x^5 + x^4 + x^2 + x + 1$ for 10110111. Hence,

$$\begin{aligned} \texttt{03} \bullet \texttt{b7} &\equiv ((x+1) \cdot (x^7 + x^5 + x^4 + x^2 + x + 1)) \\ &\quad \bmod x^8 + x^4 + x^3 + x + 1 \\ &\equiv x^8 + x^7 + x^6 + x^4 + x^3 + 1 \mod x^8 + x^4 + x^3 + x + 1 \\ &\equiv x^7 + x^6 + x. \end{aligned}$$

Using the same procedure, we get

$$\texttt{02} \bullet \texttt{70} = x^7 + x^6 + x^5.$$

With the use of the binary representation, we have

$$\begin{aligned} s'_{1,2} &= 11001101 \oplus 11100000 \oplus 11000010 \oplus 01010001 \\ &= 10111110 \\ &= \texttt{be}. \end{aligned}$$

The `MixColumns()` transformation has the property that if the input state is applied with a single non-zero byte, the output state can have at most 4 non-zero bytes. This occurs because `MixColumns()` permutes the bytes of a column to all different columns. Hence the `MixColumns()` transformation provides a diffusion of the input bytes. In addition, `MixColumns()` transformation is invertible, because the polynomial $c(x)$ given in Eq. 4.5-4 is chosen in such a way that

$$\gcd(c(x), x^4 + 1) = 1,$$

enabling therewith the existence of the inverse[12] of $c(x)$ modulo $x^4 + 1$.

`AddRoundKey()` Transformation

The `AddRoundKey()` transformation is a simple bitwise XOR operation of a round key and the state s to output the new state s'. As previously mentioned the

12 The reader is referred to [Menezes96] to understand the multiplicative inverse of polynomials in Galois fields.

round key consists of 4 words and is generated from the key expansion algorithm. The state s' is obtained according to the following equation:

$$[s'_{0,c},\ s'_{1,c},\ s'_{2,c},\ s'_{3,c}] = [s_{0,c},\ s_{1,c},\ s_{2,c},\ s_{3,c}] \oplus [w_{(round \cdot 4)+c}],$$

where $0 \leq c \leq 4$, $0 \leq round \leq N_r$ and $[w_i]$ are the key schedule words.

4.5.4 AES Decryption

The structure of the AES deciphering algorithm is the same as the ciphering except for the use of the inverse transformations and changing their order (pseudo code in Listing 4.5-3). The key schedule remains the same as by ciphering. It is clear that without the knowledge of the cipher key k the `AddRoundKey()` cannot be applied and hence the decryption of messages would not be possible. Note that here the inverse lookup table of the S-box is needed.

Listing 4.5-3: Deciphering with AES.

```
InvCipher(byte in[16], byte out[16], word w[4·(Nr + 1)])
begin
  byte state[4,4]
  state = in
  AddRoundKey(state, w[4, (Nr + 1)·3])
  for round = Nr-1 step -1 downto 1
    InvShiftRows(state)
    InvSubBytes(state)
    AddRoundKey(state, w[round.Nb,(round+1).Nb-1])
    InvMixColumns(state)
  end for
  InvShiftRows(state)
  InvSubBytes(state)
  AddRoundKey(state, w[0, 3])
  out = state
end
```

We now describe the inverse transformations `InvShiftRows()`, `InvSubBytes()`, `InvMixColumns(state)`, and `AddRoundKey()`. The `AddRoundKey()` is its own inverse. We still use the symbols s for the input state to the corresponding transformation, and s' for the output state.

`InvShiftRows()` Transformation

The `InvShiftRows()` transformation is the inverse of the `ShiftRows()` transformation. The first row of the input state , $r = 0$, is not shifted. The other rows are shifted by $(4 - shift(r, 4))$ (see Fig. 4.5-3). Formally, the `InvShiftRows()` transformation proceeds as follows:

$$s'_{(r,c+shift(r,4)) \mod 4} = s_{r,c} \text{ for } 0 \leq r < 4 \text{ and } 0 \leq c < 4.$$

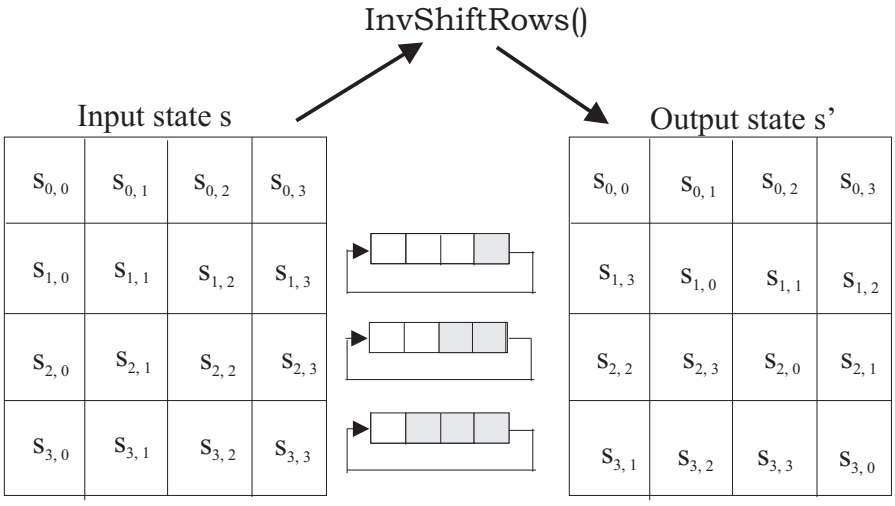


Fig. 4.5-3: The `InvShiftRows()` transformation [Fips01].

`InvSubBytes()` Transformation

`InvSubBytes()` is the inverse of the byte substitution transformation `SubBytes()`. Each byte of the input state is substituted according to the inverse S-box table presented in Tab. 4.5-4.

Tab. 4.5-4: *Inverse S-box: substitution values for the byte xy (in hexadecimal format).*

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

`InvMixColumns()` Transformation

The `InvMixColumns()` transformation is the inverse of the `MixColumns()` transformation. It operates on the state column-by-column. Each column of the state is considered as a polynomial over GF(8) and multiplied modulo x^4+1 with a fixed polynomial $c^{-1}(x)$, given by

$$c^{-1}(x) = \texttt{0b}x^3 + \texttt{0d}x^2 + \texttt{09}x + \texttt{0e}. \qquad \text{4.5-6}$$

Treating each column of the input state s as a four-term polynomial, the `MixColumns()` transformation outputs the state s' verifying the following equations:

$$\begin{aligned}
s'_{0,c} &= (\texttt{0e} \bullet s_{0,c}) \oplus (\texttt{0b} \bullet s_{1,c}) \oplus (\texttt{0d} \bullet s_{2,c}) \oplus (\texttt{09} \bullet s_{3,c}) \\
s'_{1,c} &= (\texttt{09} \bullet s_{0,c}) \oplus (\texttt{0e} \bullet s_{1,c}) \oplus (\texttt{0b} \bullet s_{2,c}) \oplus (\texttt{0d} \bullet s_{3,c}) \\
s'_{2,c} &= (\texttt{0d} \bullet s_{0,c}) \oplus (\texttt{09} \bullet s_{1,c}) \oplus (\texttt{0e} \bullet s_{2,c}) \oplus (\texttt{0b} \bullet s_{3,c}) \\
s'_{3,c} &= (\texttt{0b} \bullet s_{0,c}) \oplus (\texttt{0d} \bullet s_{1,c}) \oplus (\texttt{09} \bullet s_{2,c}) \oplus (\texttt{0e} \bullet s_{3,c})
\end{aligned}$$

with $0 \leq c < 4$.

The same equations can be written in a matrix notation as follows:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} \texttt{0e} & \texttt{0b} & \texttt{0d} & \texttt{09} \\ \texttt{09} & \texttt{0e} & \texttt{0b} & \texttt{0d} \\ \texttt{0d} & \texttt{09} & \texttt{0e} & \texttt{0b} \\ \texttt{0b} & \texttt{0d} & \texttt{09} & \texttt{0e} \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad 0 \leq c < (N_b = 4).$$

4.5.5 Security and Implementation Issues

In AES, we can recognize some of the same functional blocks as in DES. The XORs add key material to the data, the S-boxes provide nonlinearity, and the byte shuffle and mixing functions provide diffusion. AES is a very clean design with clearly separated tasks for each part of the cipher. It was designed to be resistant against all known attacks, especially the differential attacks [Biham93] and its variants (higher order differential [Knudsen95], truncated differential [Knudsen95]), the linear attack [Matsui94] and the square attack [Daemen97]. It is also proven that the AES has no security weakness to attacks related to the cipher key (related-key attacks [Biham93a], weak key attack [Daemen94]). Furthermore, AES has a mathematical basis (it is mainly based on operations over the Galois Field GF(8)), and thus its security can be analyzed using mathematical analysis.

From the efficiency point of view, the AES cipher is very fast – if efficiently implemented – on 8-bit processors (typical for smart cards) and on 32-bit-processors (typical for PCs) [Daemen99]. It is also fast on dedicated hardware. However, some limitations concern the implementation of the AES deciphering algorithm. For example, the deciphering algorithm is less suited for implementation on smart cards then the ciphering. The AES cipher can be used in many cryptographic applications. It can be used as a building block for MAC algorithms and hash functions, as a synchronous stream cipher, or as a pseudorandom number generator.

5 Public-Key Encryption

revolution in the development of cryptology

After conventional encryption, the other major form of encryption is public-key encryption, which has revolutionized communications security. The development of public-key cryptography is the greatest and perhaps the only true revolution in the entire history of cryptography. From its earliest beginnings right down to modern times, all cryptosystems have been based on the elementary tools of substitution and permutation. Public-key cryptography provides a radical departure from all that has been done before - public-key algorithms are based on mathematical functions rather than on substitutions and permutations. But more importantly, public-key cryptography is *asymmetric*, involving the use of two separate keys, in contrast to the conventional symmetric encryption, which uses only one key. The use of two keys has profound consequences not only in the area of confidentiality, but also in the techniques for key distribution and authentication.

This chapter introduces public-key encryption and concentrates on its use to provide confidentiality. Some important algorithms (RSA, ElGamal, ECC) are examined in detail. Many public-key schemes are based on number theory and finite field arithmetic, which have been introduced in Chapter 2.

5.1 Principles of Public-Key Cryptography

Diffie/Hellman's idea

Interestingly, the concept for this technique was developed and published before it was shown to be practical to adopt it. The concept of public-key cryptography evolved from an attempt to attack two of the most difficult problems associated with conventional encryption: the problems of key distribution and digital signatures. Diffie and Hellman achieved a breakthrough in 1976 by coming up with a method that addressed both problems and that was radically different from all previous approaches. Their paper "New Directions in Cryptography" ([Diffie76]) is a milestone in modern cryptology.

public and private key

In a public-key cryptosystem enciphering and deciphering are governed by distinct keys k_e and k_d, so that computing k_d from k_e is computationally infeasible (e.g. requiring 100^{100} instructions). The enciphering key k_e can thus be publicly disclosed without compromising the deciphering key k_d. Each user of the network can, therefore, place his enciphering key in a public directory. That is why this key is also called *public key*. A message m enciphered with the user's public key, can only be decrypted with his *private key* k_d, which is kept secret (we will not refer to this key as "secret key" in order to clearly differentiate it from the symmetric encryption key). This enables any user of the system to send a message to any other user enciphered in such a way that only the intended receiver is able to decipher it. *A private conversation can therefore be held between any two individuals regardless of whether they have ever communicated before, which is not the case when using*

a symmetric enciphering technique. Each one sends messages to the other enciphered in the receiver's public enciphering key and deciphers the messages he receives using his own secret deciphering key.

definition of public-key cryptosystem

Formally, a *public-key cryptosystem* is a pair of families $\{E_{k_e}\}_{k_e \in K}$ and $\{D_{k_d}\}_{k_d \in K}$ of algorithms representing invertible transformations $E_{k_e} : \{M\} \rightarrow \{M\}$ and $D_{k_d} : \{M\} \rightarrow \{M\}$ on a finite message space $\{M\}$, so that:

1. For every $k_e, k_d \in \{K\}$, the encryption E_{k_e} is the inverse operation of the decryption D_{k_d}. That means that for every message $m \in \{M\}$, $D_{k_d}\{E_{k_e}(m)\} = m$.
2. For every $k_e, k_d \in \{K\}$ and $m \in \{M\}$ the algorithms E_{k_e} and D_{k_e} are easy to compute.
3. For almost every $k_e \in \{K\}$, it is computationally infeasible to derive k_d from k_e.
4. For every $k_e, k_d \in \{K\}$, it is feasible to compute inverse pairs (k_e, k_d) from K.

Because of the third property, a user's enciphering key k_e can be made public without compromising the security of his secret deciphering key k_d. The cryptographic system is therefore split into two parts, a family of enciphering transformations and a family of deciphering transformations, in such a way that, given a member of one family, it is infeasible to find the corresponding member of the other.

The fourth property guarantees that there is a feasible way of computing corresponding pairs of inverse transformations when no constraint is placed on what either the enciphering or deciphering information is to be. In practice, the crypto equipment must contain a true random number generator for generating the key space K, together with an algorithm for generating the (k_e, k_d) pairs from its output.

generation and distribution of keys

Given a system of this kind, the problem of key distribution is vastly simplified. Each user I generates locally a pair of inverse transformations $k_{e,I}$ and $k_{d,I}$ on his terminal. The deciphering key $k_{d,I}$ must be kept secret, but need never be transmitted on any channel. The enciphering key $k_{e,I}$ can be made public by placing it in a public directory along with the user's name and address. Anyone can then encrypt the messages and send them to the user, but no one else can decipher messages intended for him. Public-key cryptosystems can thus be regarded as multiple access ciphers.

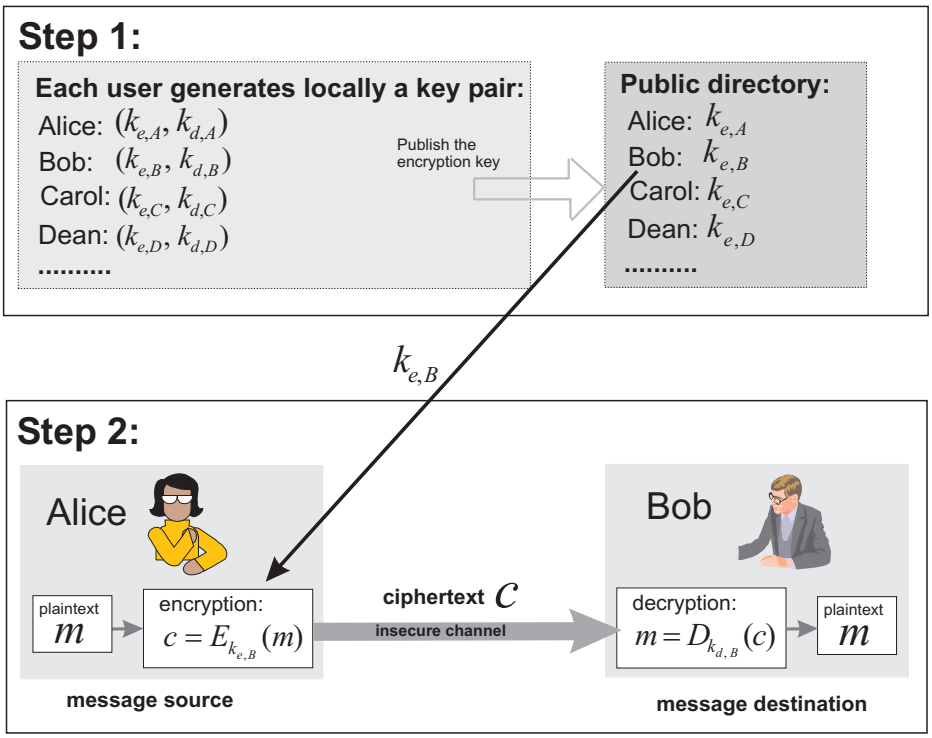


Fig. 5.1-1: Public-key encryption.

encryption and decryption

This process is illustrated in Fig. 5.1-1. When Alice wants to send an encrypted message to Bob (they have never exchanged a message before; actually, they do not know each other), she takes his public key $k_{e,B}$ from the public directory and gets the ciphertext c by enciphering the plaintext m with this key. The enciphered message c is sent on the insecure channel, where an attacker can get it, but no one else but Bob can decrypt c, since only he knows the corresponding deciphering key $k_{d,B}$.

protection of the public directory

It is crucial that the public file of enciphering keys be protected from unauthorized modification. This task is made easier by the public nature of the file. Read protection is unnecessary, but if an attacker X can replace Bob's public key $k_{e,B}$ with his public key $k_{e,X}$ secretly (no one notices this), then he can read all encrypted messages sent to Bob (actually, Alice now encrypts the message for Bob with $k_{e,X}$ instead of with $k_{e,B}$, and X has the deciphering key $k_{d,X}$).

use of PKCs

In the example above, we have an application of a public-key cryptosystem in which the sender Alice uses the receiver's public key. But depending on the application, Alice can use either her private key or receiver's public key, or both, to perform some type of cryptographic function. Thus, the public-key cryptosystems (PKCs) can be classified in three categories:

- Encryption/decryption: The sender encrypts a message with the recipient's public key.
- Digital signatures: The sender signs a message with its private key, i.e. the sender applies some crpyptographic algorithm on the message in order to assure that the message can not be altered by an attacker (see chapter 6).

- Key exchange: Two sides co-operate to exchange a session key. Several different approaches are possible, involving the private key(s) of one or both parties.

Some algorithms are suitable for all three applications (e.g. RSA, ElGamal), whereas others can be used for only one or two of these applications (e.g. Diffie-Hellman algorithm only for key exchange, or DSS only for signatures).

5.2 RSA Encryption Scheme

After Diffie and Hellman introduced the concept of public-key cryptography in their pioneering work in 1976 [Diffie76], the cryptographers were challenged to come up with a cryptographic algorithm that met the requirements for public-key systems. One of the first public-key algorithms was developed in 1977 by Ron Rivest, Adi Shamir and Len Adleman at MIT in Boston and first published in 1978 [Rivest78]. The Rivest-Shamir-Adleman (RSA) scheme is the first and still most important and widely accepted and implemented encryption/decryption algorithm that has been shown to be feasible for public-key encryption.

work of Rivest, Shamir and Adleman

The RSA scheme is a block cipher in which the plaintext and ciphertext are integers between 0 and $n-1$ for some n. It is based on an amazingly simple number-theoretical idea and yet it has been able to resist all cryptanalytic attacks. The idea is the clever use of the fact that, while it is easy to multiply two large primes, it is extremely difficult to factorize their product. Thus, the product can be published and used as encryption key. The primes themselves can not be recovered from the product. On the other hand, the primes are needed for decryption. So, the security of the RSA scheme is based on the problem of factorization of large numbers.

factorization problem

5.2.1 Description of the Algorithm

Let p and q be two distinct large random primes (typically, having about 100 digits in their decimal representation). The product of this large primes is denoted as n, i.e. $n = pq$. Because of primality of p and q, the Euler function of n can be computed as product of the Euler functions of p and q:

$$\varphi(n) = \varphi(p)\varphi(q) = (p-1)(q-1). \qquad \text{5.2-1}$$

The parameter n is called *modulus* of the system. When a user Alice wants to generate her key pair $(k_{e,A}, k_{d,A})$, she chooses a large random number $1 < e < \varphi(n)$ which is relativ prime to $\varphi(n)$, i.e. $gcd(e, \varphi(n)) = 1$, and computes the number d as multiplicative inverse of e modulo $\varphi(n)$, i.e. d satisfies the congruence

RSA key generation

$$ed \equiv 1 \bmod \varphi(n). \qquad \text{5.2-2}$$

The public key of Alice $k_{e,A}$ is the pair (e, n) and thus, e is referred to as *encryption exponent*. Her private key is $k_{d,A} = d$ and d is reffered to as *decryption exponent*.

RSA encryption If Bob wants to send an encrypted message to Alice, he uses her public key $k_{e,A} = (e, n)$. To encrypt, he raises the plaintext m to the power e and reduces modulo n:

$$c = m^e \bmod n. \tag{5.2-3}$$

RSA decryption Alice decrypts the text with her private key $k_{d,A} = d$: she raises the ciphertext c to the power d and reduces modulo n:

$$e = c^d \bmod n, \tag{5.2-4}$$

i.e. the encryption and decryption are performed as modular exponentiations modulo n. This is summarized in Fig. 5.2-1.

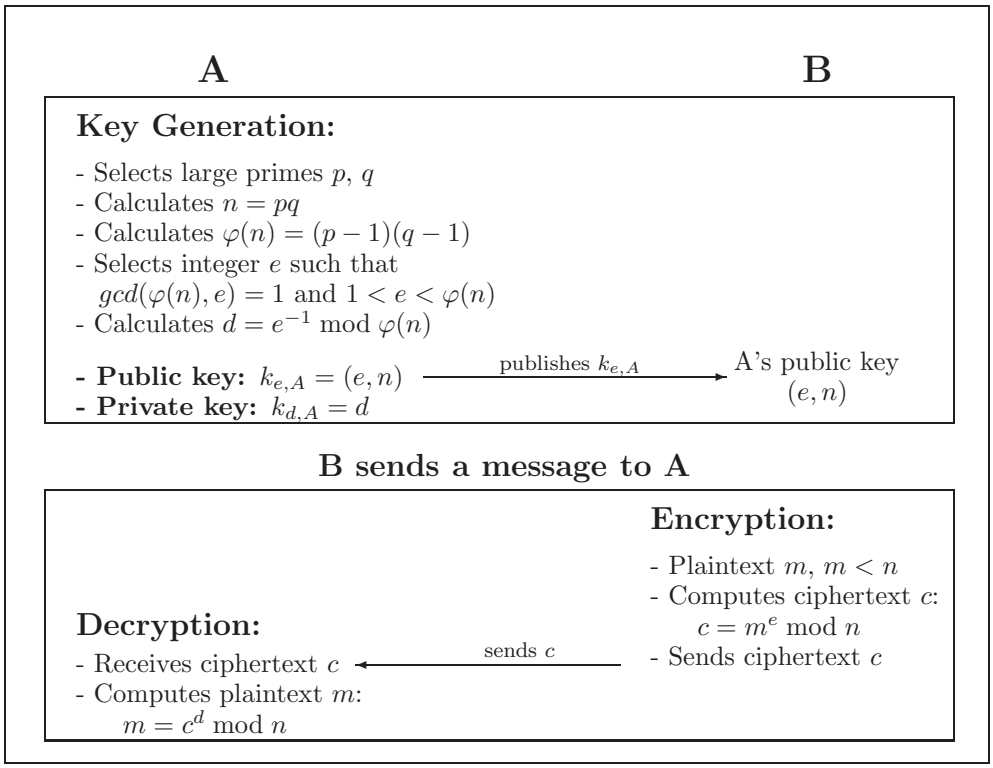


Fig. 5.2-1: Key generation, encryption and decryption with RSA.

proof that the decryption works We now show that decryption works as intended.

Since $ed \equiv 1 \bmod \varphi(n)$, there is an integer k such that $ed = 1 + k \cdot \varphi(n)$. In the proof we also make use of Euler's theorem (see Theorem 2.6-2): If $a \in \mathbb{Z}_n^*$, then $a^{\varphi(n)} \equiv 1 \bmod n$. For the decryption we have:

$$\begin{aligned} c^d \bmod n &\equiv (m^e)^d \bmod n \equiv m^{ed} \bmod n \\ &\equiv m^{1+k\cdot\varphi(n)} \bmod n \\ &\equiv (m^{\varphi(n)})^k \cdot m \bmod n \\ &\equiv (1)^k \cdot m \bmod n = m \bmod n \\ &\equiv m \qquad \text{(because } m < n\text{)}. \end{aligned} \tag{5.2-5}$$

We now illustrate how the RSA algorithm works using a simple example (see Fig. 5.2-2).

Example 5.2-1:
We take all parameters artificially small (and thus insecure) in order to show the operations comprehensibly. To generate a key pair, Alice first chooses two primes, say $p = 13$ and $q = 29$, which she keeps secret, and computes their product $n = pq = 377$, which she puts in the public directory. Further, she computes $\varphi(377) = (13-1)(29-1) = 336$ and selects an integer e so that $1 < e < \varphi(377) = 336$ and $\gcd(\varphi(n), e) = \gcd(336, e) = 1$. Let $e = 59$, and the pair $(e, n) = (59, 377)$ is her public key. Since the public and the private keys are related according to the equation $de \equiv 1 \bmod \varphi(n)$, she uses the Extended Euclidian algorithm to compute her private key d from the public key e : $d = e^{-1} \bmod \varphi(n) = 59^{-1} \bmod 336 = 131$. Notice that nobody but Alice can compute her private key d from the public key e, because nobody knows the primes p and q which are needed to compute $\varphi(n) = (p-1)(q-1)$ and apply it to compute d from the equation $de \equiv 1 \bmod \varphi(n)$ with the Extended Euclidian algorithm (see Section 2.3.5). Although n is known, it can not be factorized to determine p and q (for large integers n the factorization problem is very hard, see Section 2.6.2).

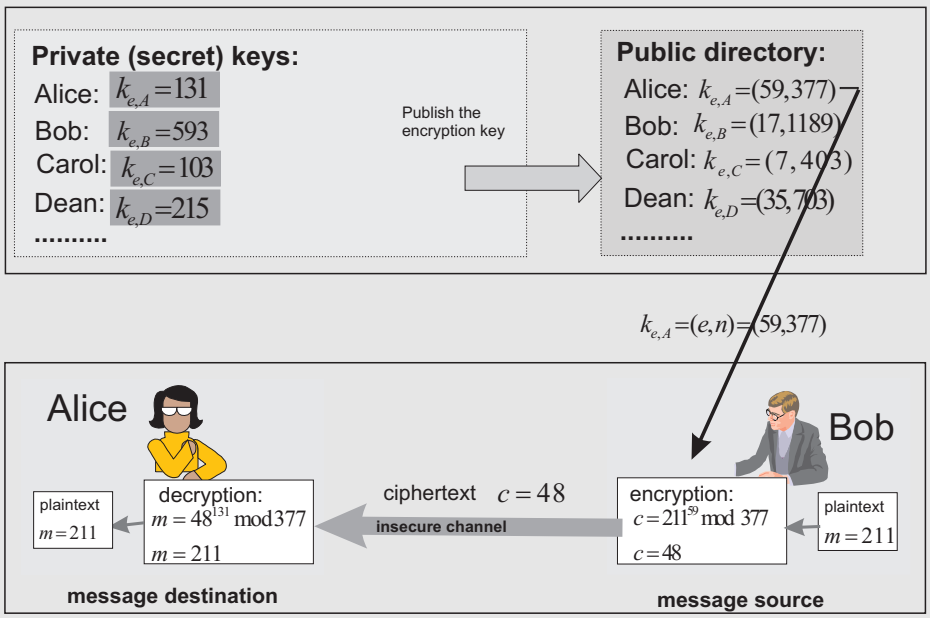


Fig. 5.2-2: Example with the RSA algorithm.

Now Bob wants to send the message M to Alice, which is encoded as $m = 211$. To encrypt m he uses Alice's public key $k_{e,A} = (e, n) = (59, 377)$ and computes $c = m^e \bmod n = 211^{59} \bmod 377 = 48$. When Alice receives the ciphertext $c = 48$, she uses her private key $k_{d,A}$ to recover the message m as $m = c^d \bmod n = 48^{131} \bmod 377 = 211$. To exercise, you can try this algorithm with much smaller numbers. For example, use the key pair $k_{e,A} = (13, 33)$ and $k_{d,A} = 17$ to encrypt and decrypt the message $m = 2$. Please use the Crypto-Calculator on the book web page to try the algorithm with other (and larger) numbers.

RSA as block cipher

We assume that the plaintext M is encoded as a decimal number m. Since m should be always smaller than n, in the case when $m \geq n$, the number m is divided into blocks of suitable size. A suitable size of the blocks is the unique integer i satisfying the inequalities $10^{i-1} < n < 10^i$. Then, the blocks are encrypted (and later decrypted) separately, allowing RSA to work as a block cipher in ECB or CBC mode. In the example above $n = 377$ implies that the block size equals 3. If we use, for example, $p = 3336670033, q = 9876543211, n = 32954765761773295963, \varphi(n) = 32954765748560082720, e = 1031, d = 31963885304131991$, the plaintext blocks will consist of 20 digits.

cryptosystem design

There are many aspects of the RSA cryptosystem to discuss, including the details of setting up the cryptosystem, the efficiency of enciphering and deciphering and the security issues. We now discuss some aspects of the cryptosystem design, that is, how the different items required are generated. In general, when we say that a random number is chosen, or that we select something randomly, then we are using a random number generator[13]. To select two large random primes p and q, one choses randomly an odd integer r of appropriate size (say 100 digits) and tests it for primality[14]. If the answer is negative, $r + 2$ is repeatedly tested. Once p and q have been chosen, candidates for d are tested by the Euclidian algorithm. When d satisfies $(d, \varphi(n)) = 1$, the chain of equations obtained from the Euclidian algorithm gives e immediately.

The operation needed for both encryption and decryption is *modular exponentiation*. Since the modulus n is very large, multiprecision arithmetic must be used to perform computations in $\mathbb{Z}_n$ and the time required will depend on the number of bits in the binary representation of n. The operation $a^b \bmod n$ can be done much faster than by repeatedly multiplying a by itself. Examples are the square-and-multiply algorithm, the windowing algorithm, the Lim/Lee algorithm and others, which perform the exponentiation in polynomial time. Although all operations of RSA can be carried out in polynomial time, these operations are still roughly 1000 times faster in DES than in RSA.

5.2.2 Security of RSA

factorization problem

The problem of computing the RSA decryption exponent d from the public key (e, n) and the problem of factoring n are computationally equivalent. So, one obvious attack on the cryptosystem is for cryptanalyst to attempt to factorize n. If this can be done, it is simple to compute $\varphi(n) = (p-1)(q-1)$ and then compute the decryption exponent d from e. Hence, it is necessary that $n = pq$ must be large enough, so that the factoring of n will be computationally infeasible. The primes p and q should have *approximately the same bit size*, but should not be close to one another. Current factoring algorithms are able to factor numbers having up

13 We will not discuss any details concerning random number generators here.

14 Primality tests are described in Section 2.6.1.

to 130 decimal digits[15]. Hence, it is recommended that one should choose p and q to have about 100 decimal digits and their product n will have about 200 digits. Many RSA implementations use a 512-bit modulus, which corresponds to about $512/\log_2 10 \approx 154$ decimal digits, and hence they do not offer good long-term security. For long-term security, 1024-bit or larger moduli should be used.

Here we stress that there is *no formal proof* that

- factorization is intractable or is intractable in the special case needed for RSA, and
- factorization is needed for cryptanalysis of RSA.

The second item means that it is not proven, that there is no cryptanalytic method avoiding factorization. In general, many other cryptanalytic attacks have been proposed against RSA cryptosystems, but none of them has turned out to be serious. We now briefly discuss some typical ones and also mention a few other aspects one should be aware of, in order to prevent certain rather obvious attacks.

low exponent attack

In order to improve the efficiency of encryption, it is desirable to select a small encryption exponent e (e.g. $e = 3$). But small encryption exponent should not be used if the same message is sent to many entities. To avoid such an attack, a pseudo-randomly generated bitstring of appropriate length (e.g. 64 bit) should be appended to the plaintext message. This process is sometimes referred to as *salting* the message. Also, if the message space is small or predictable, an attacker can decrypt a ciphertext c by simply encrypting all possible plaintext messages until c is obtained. This attack can also be prevented by salting the message. The decryption exponent d should be roughly the same size as n, because there is an efficient algorithm for computing d from the public information (e, n) in the case where d has up to approximately one-quarter as many bits as the modulus n.

multiplicative property

RSA has multiplicative property, which is sometimes referred to as *homomorphic property* of RSA. Let m_1 and m_2 be two plaintext messages, and c_1 and c_2 be their RSA encryptions respectively. Then the following equation holds:

$$(m_1 m_2)^e \equiv m_1^e m_2^e \equiv c_1 c_2 \bmod n. \qquad 5.2\text{-}6$$

In other words, the ciphertext corresponding to the plaintext $m = m_1 m_2 \bmod n$ is $c = c_1 c_2 \bmod n$. This property leads to the *adaptive chosen ciphertext* attack on RSA[16]. If one knows both ciphertexts c_1 and c_2, one knows the encryption of $m = m_1 m_2$. This attack can be prevented by imposing some structural constrains on plaintext messages in the way that the product of two plaintexts can not result in plaintext. For example, we can make the constraint that the first byte of each plaintext is the same as the last byte (we can achieve this artificially). If m_1 and m_2 satisfy this condition, then it is very inprobable that their product m satisfies it also.

15 For more information on factoring, see [Menezes96].

16 For details see [Menezes96].

So, the receiver can reject any encrypted plaintext which does not have the specific structure.

common modulus attack

To avoid the *common modulus attack*, each user in the system should choose his own RSA modulus n. When the same modulus n is used, each user could subsequently determine the decryption exponents of all other network users. Also, if a single message were encrypted and sent to two or more network users, then there is a technique by which an eavesdropper could recover the message with high probability using only publicly available information.

timing attacks

Apart from the attacks on RSA mentioned above, which belong to the so called mathematical attacks, another class of attacks are the *timing attacks*, which are based on the analysis of the running time of the decryption algorithm. These attacks are applicable not only to RSA, but also to other public-key cryptosystems. They are ciphertext-only attacks which are analogous to guessing the combination of a safe by observing how long it takes for someone to turn the dial from number to number. Although timing attacks are a serious threath, there are simple countermeasures that can be used to avoid them (forcing constant exponentiation time, adding a random delay, or blinding the ciphertext by a random number before performing exponentiation).

brute force attacks

Brute force attacks (trying all possible private keys) can also be performed, and the defense against them is the same for RSA as for other cryptosystems - namely, the key space used must be large. But some compromise between the security required and the speed should be made, because too large keys mean very slow systems (the time for the calculations involved depends on the key size).

5.3 The Discrete Logarithm Problem

Many public-key cryptosystems based on discrete logarithms have been proposed. When used as a basis for cryptosystems, the computation of discrete logarithms is assumed to be intractable. If we consider the equation $a^x = y$ for positive real numbers, the difficulty of determining the logarithm x from a and y to prescribed accuracy is approximately the same as determining y from a and x. With regard to *discrete logarithms*, the situation is entirely different. Modular exponentiation $a^x = y \bmod p$ can be carried out resonably fast, but the inverse operation, taking discrete logarithms, has much greater computational complexity.

5.3.1 The Problem of Discrete Logarithm in $\mathbb{Z}_p^*$

The general notion of discrete logarithms can be formulated as follows. Let g be an element of a finite group G and let y be another element of G. Then any integer x with $g^x = y$ is called a *discrete logarithm* of y to the base g. Clearly, *every element y of G has a discrete logarithm to the base g if and only if G is cyclic with the generator g*. For instance, in the multiplicative group of positive integers modulo 7 only the numbers 1, 2, and 4 have a discrete logarithm to the base 2, whereas all numbers have a discrete logarithm to the base 3. Since the results of the exponentiation of the base 3 are: $3^1 = 3, 3^2 = 2, 3^3 = 6, 3^4 = 4, 3^5 = 5$, and $3^6 = 1$, the discrete logarithms of all members of the cyclic group are: $\log_3 1 = 6, \log_3 2 = 2, \log_3 3 = 1, \log_3 4 = 4, \log_3 5 = 5$, and $\log_3 6 = 3$.

discrete logarithm

Therefore, the *discrete logarithm problem* in $\mathbb{Z}_p$ can be formally defined as follows: Given a triple (p, a, b), where p is prime, $a \in \mathbb{Z}_p$ is primitive element, and $b \in \mathbb{Z}_p^*$[17], find the unique integer $x, 0 \leq x \leq p-2$, so that $a^x \equiv b \bmod p$. This integer x is denoted as $x = \log_a b$.

discrete log problem

Of course, groups of small cardinality present no computational difficulties. There are some efficient algorithms of computing discrete logarithms in some special cases, but in general the known algorithms for computing discrete logs in group of order m are roughly of the same complexity in terms of m as the algorithms for factoring m.

5.3.2 Diffie-Hellman Key Exchange

purpose of the algorithm

Actually, the Diffie-Hellman algorithm was the first published public-key algorithm which appeared in the seminal paper by Diffie and Hellman that defined public-key cryptography [Diffie76]. The purpose of the algorithm is to enable two users to exchange a key securely over an insecure channel. The exchanged key can be used for subsequent encryption of messages with any symmetric algorithm (e.g. with DES). This technique makes use of the apparent difficulty of computing logarithms over a finite field.

Diffie-Hellman algorithm

For this scheme (see Fig. 5.3-1) there are two publicly known numbers: a prime number p and an integer g that is a primitive element (generator) of $\mathbb{Z}_p^*$ with $2 \leq g \leq p-2$. Suppose the users Alice and Bob want to exchange a key. Alice selects a random integer $a, a < p$ and computes the exponentiation

$$A = g^a \bmod p. \qquad 5.3\text{-}1$$

Similarly, Bob independently selects a random integer $b,\ b < p$ and computes

$$B = g^b \bmod p. \qquad 5.3\text{-}2$$

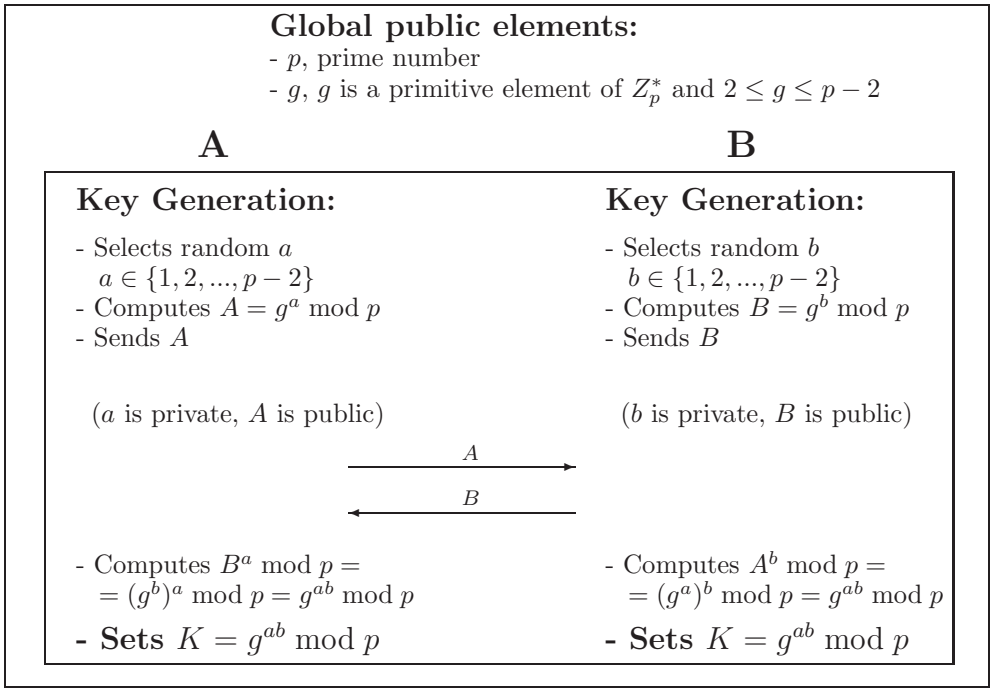


Fig. 5.3-1: The Diffie-Hellman key exchange algorithm.

17 We have definied $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \gcd(a, n) = 1\}$. Since p is prime, $\mathbb{Z}_p^* = \{a \in \mathbb{Z} \mid 1 \leq a \leq n-1\}$ see Section 2.3.7.

Each side keeps the random chosen value (a or b) private and makes the computed exponentiation value publicly available to the other side i.e. Alice receives the value B and Bob receives the value A. Alice computes the key as

$$K_a = B^a \bmod p. \tag{5.3-3}$$

Similarly, Bob computes the key as

$$K_b = A^b \bmod p. \tag{5.3-4}$$

It can be easily seen that these two calculations produce identical result $K_a = K_b = K$, which is the shared common key:

$$K_a = B^a \bmod p = (g^b)^a \bmod p = g^{ab} \bmod p$$
$$K_b = A^b \bmod p = (g^a)^b \bmod p = g^{ab} \bmod p.$$

From this scheme it is obvious that if logarithms $\bmod p$ can easily be computed, the system can be broken. An attacker knows $K_a = B^a \bmod p$ and $K_b = A^b \bmod p$. Since A and B are publicly exchanged, he tries to compute the exponents (discrete logs) a and b from these equations. If he could compute them, he could easily reveal the exchanged key as $K = g^{ab} \bmod p$ and eavesdrop the communication that follows.

cryptanalytic effort

If the prime p is slightly less than 2^l, then all quantities can be represented as l bit numbers. Then exponentiation takes at most $2l$ multiplications $\bmod p$ while taking logs requires $p^{1/2} \approx 2^{l/2}$ operations. The cryptanalytic effort therefore grows exponentially relative to the legitime effort. If $l = 200$, then at most 400 operations are required to compute A from a or K from A and b, yet taking logs $\bmod p$ requires 2^{100} or approximately 10^{30} operations.

5.4 ElGamal Encryption Scheme

In 1985 Taher ElGamal published a new public-key cryptosystem [Elgamal85], whose security is based on the difficulty of calculating discrete logarithms in a finite field. The ElGamal scheme can be used for both encryption and digital signatures. We now describe the encryption scheme.

key generation

To generate a key pair, first choose a prime p, so that the discrete logarithm problem in $\mathbb{Z}_p^*$ is intractable and let $g \in \mathbb{Z}_p^*$ be a random primitive element. Choose a random exponent x so that $x \in \mathbb{Z}_p^*$. Then calculate

$$y = g^x \bmod p. \tag{5.4-1}$$

The public key is the triple (p, g, y). Both g and p can be global public elements of the system, i.e. can be shared among a group of users. The private key is the exponent x. Because of the discrete logarithm problem it is computationally infeasible to reveal the private key x from the public key (y, g, p), i.e to resolve x from the Eq. 5.4-1.

encryption To encrypt a message m, $m \in 0, 1, ...p-1$, first choose a random number k, so that $k \in \{0, 1, ...p-2\}$ and k is relatively prime to $p-1$. Then compute

$$a = g^k \bmod p \qquad \text{5.4-2}$$
$$b = y^k m \bmod p.$$

The pair $c = (a, b)$ is the ciphertext. Actually, the plaintext m is "masked" by multiplying it with y^k, yielding b. The value g^k is also transmitted as a part of the ciphertext. Note that the ciphertext is twice the size of the plaintext.

decryption To decrypt the ciphertext $c = (a, b)$ compute

$$m = b/a^x \bmod p. \qquad \text{5.4-3}$$

The receiver, who knows the secret exponent x, can compute y^k from g^k. Then, he can "remove the mask" by dividing b by a^x to obtain m. We will now show that the

proof of the decryption decryption works correctly:

$$\begin{aligned} b/a^x \bmod p &\equiv (y^k m)/(g^k)^x \bmod p \\ &\equiv ((g^x)^k m)/g^{kx} \bmod p \\ &\equiv (g^{xk} m)/g^{kx} \bmod p \\ &\equiv m \bmod p \\ &= m. \end{aligned}$$

The first transformation in this equation can be done only with the knowledge of x, and since x is the private key of the receiver, only he is able to do this operation. The ElGamal encryption scheme is summarized in Fig. 5.4-1.

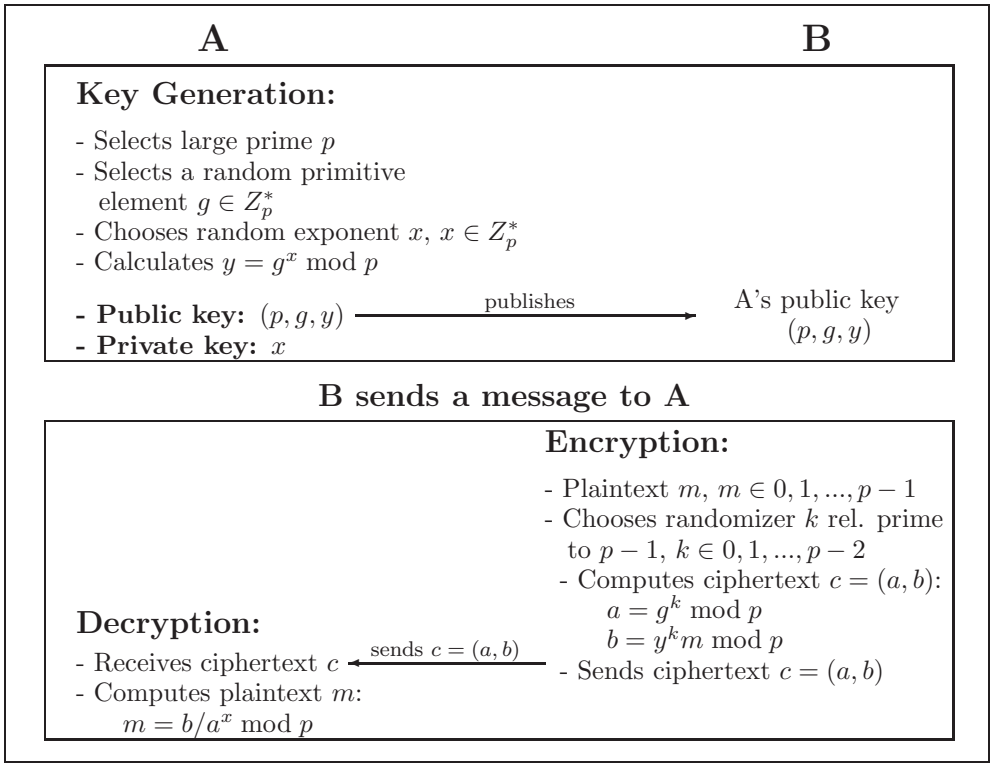


Fig. 5.4-1: Key generation, encryption and decryption with the ElGamal scheme.

Example 5.4-1:
In this example we will illustrate how the ElGamal scheme works. Suppose that the group $\mathbb{Z}^*_{2579}(p = 2579)$ and a random primitive element $g = 2$ are chosen and are shared among a group of users. The user Alice choses her private key $x = 765$, $x \in \mathbb{Z}^*_{2579}$. Then she computes her public key:

$$y = g^x \bmod p = 2^{765} \bmod 2579 = 949$$

and publishes it. Suppose, Bob wants to send the confidential message $m = 1299$ to Alice. Say $k = 853$ is the random integer he chooses. Then he encrypts the message $m = 1299$ with the public key y from Alice. He computes

$$a = g^k \bmod p = 2^{853} \bmod 2579 = 435$$
$$b = y^k m \bmod p = 949^{853} \cdot 1299 \bmod 2579 = 2396$$

and sends the ciphertext $c = (a, b) = (435, 2396)$ to Alice. Since she has the corresponding private key of y, she can decrypt the ciphertext c. She computes

$$m = b/a^x \bmod p = 2396 \cdot (435^{765})^{-1} \bmod 2579 = 1299$$

and gets the plaintext $m = 1299$.

If the plaintext m is larger than the system parameter p, then, similary as in RSA, m can be divided into blocks of appropriate size and each block will be encrypted and decrypted independently, working as block cipher in ECB or CBC mode.

non-deterministic property

This system differs from the other known systems due to the randomization with k in the enciphering operation. That means that the ElGamal cryptosystem is non-deterministic, since the ciphertext depends on both the plaintext m and the random value k chosen by the sender. The ciphertext for a given message m is not repeated, i.e. if we encipher a given message twice, we will not get the same ciphertext, as in the case of RSA encryption. The non-deterministic property of ElGamal scheme prevents attacks like a *probable text attack*, where if the intruder suspects that the plaintext is, for example m, then he tries to encipher m and finds out if it was really m. This attack, and similar ones, will not succeed since the original sender chooses a random k for enciphering, and different values of k will yield different values of the ciphertext c. The ElGamal system is also not-multiplicative, because due to the structure of the system, there is no obvious relation between the enciphering of the messages m_1, m_2, and $m_1 m_2$ (as is the case in RSA scheme), or any other simple function of m_1 and m_2.

efficiency of the system

Suppose that the system parameter p is of about the same size as that required for modulus n in the case of RSA. Then the size of the ciphertext in the ElGamal scheme is double the size of the corresponding RSA ciphertext (disadvantage). For the enciphering operation, two exponentiations are required, which is equivalent to

about $2 \log p$ multiplications in $\mathbb{Z}_p^*$. For the deciphering operation only one exponentiation (plus one division) is needed.

5.5 Elliptic Curve Cryptography (ECC)

Elliptic curves have been studied by mathematicians for more than a century. An extremely rich theory has been developed around them, and in turn they have been the basis of numerous new developments in mathematics. As far as cryptography is concerned, elliptic curves have been used for factoring and primality proving. The idea of using elliptic curves for public-key cryptosystems is due to Victor Miller [Miller85] and Neal Koblitz [Koblitz87] in the mid-eighties. As with all cryptosystems, and especially with public-key cryptosystems, it takes years of public evaluation before a reasonable level of confidence in a new system is established. The elliptic curve public-key cryptosystems (ECPKCs) seem to have reached that level now. In the last couple of years, the first commercial applications have appeared (email security, web security, smart cards, etc.). Before we look at how the ECPKCs work, we will give a short introduction to elliptic curves.

5.5.1 Elliptic Curves Over Real Numbers

definition of elliptic curves

Elliptic curves are not ellipses. They are called this because they are described by cubic equations, similar to those used for calculating the circumference of an ellipse. In general, an elliptic curve is the set of solutions of an equation of the form

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_5 \qquad 5.5\text{-}1$$

where the coefficients a_i are elements of some field ($\mathbb{R}, \mathbb{Z}$ or $\mathbb{Z}_p$) which satisfy some simple conditions in order to avoid singularities. Such an equation is said to be cubic, or of degree 3, because the highest exponent it contains is 3. The Eq. 5.5-1 is called *Weierstrass equation*. Also included in the definition of any elliptic curve is a single element denoted $\mathcal{O}$ and called *point of infinity* or the *zero point*.

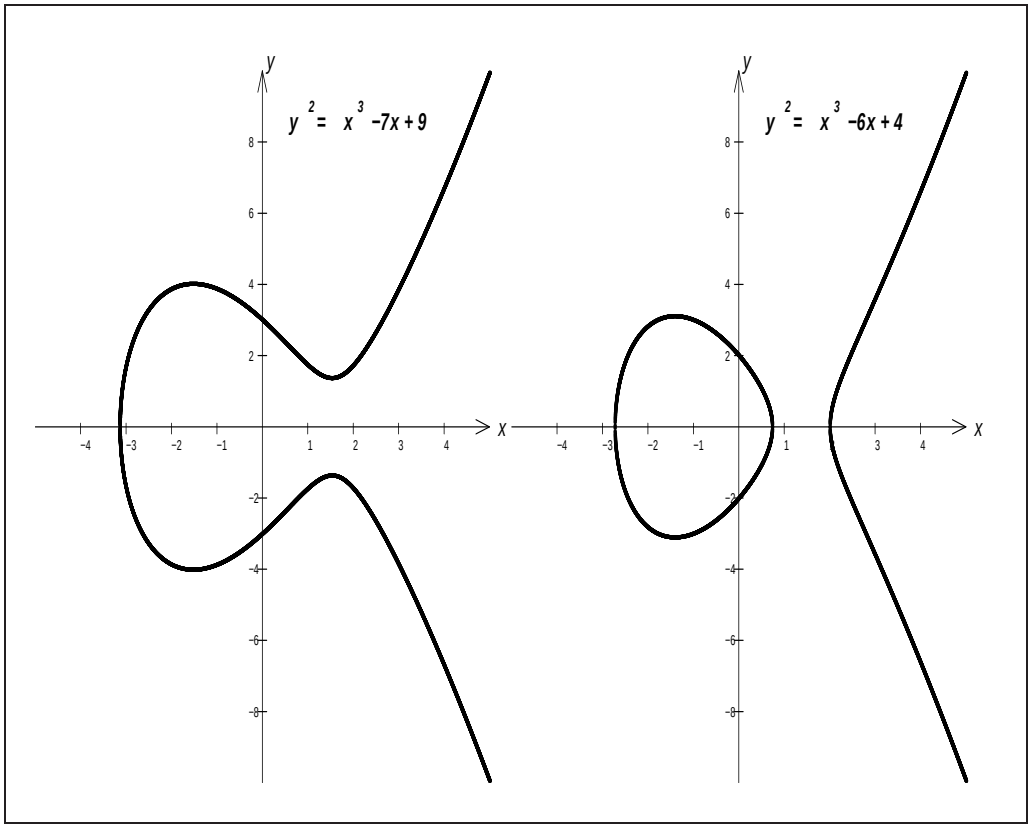


Fig. 5.5-1: Elliptic curves over real numbers.

In order to illustrate the properties of elliptic curves, we will now first examine the elliptic curves over real numbers, i.e. $a_i \in \mathbb{R}$, also called *continuous elliptic curves*. We will further refer to the following simplified form of the Weierstrass equation as *elliptic curve equation*:

elliptic curve equation

$$E : y^2 = x^3 + ax + b \qquad 5.5\text{-}2$$

where x, y, a, and b are real numbers. Each choice of the parameters a and b yields a different elliptic curve E, also denoted as $E(a, b)$. In Fig. 5.5-1 two curves are shown. This graph can be obtained by filling the values for x, and solving the quadratic equation in y. In particular cases the graph of the curve consists of two disjoint parts. To understand more about the elliptic curve forms, vary the parameter a and b on a continuous elliptic curve in the java applet on the book home page.

It now turns out that the set of solutions of an elliptic curve has some interesting properties. In particular, a *group operation* can be embedded in this set. Given an elliptic curve E and two points P and Q which lie on it, the operation regarded is some form of "addition" of these points resulting in a third point R which also lies on the curve. The operation *"addition of points"* is denoted as $R = P + Q$ and is geometrically defined in Fig. 5.5-2, but the coordinates of the sum point can be easily derived as a function of the coordinates of Q and R. In the definition of this operation, the following fact is used: *If three points on an elliptic curve lie on a straight line, their sum is $\mathcal{O}$ (point of infinity)*. Try to choose and add points on a continuous elliptic curve in the java applet on the book home page.

addition of points

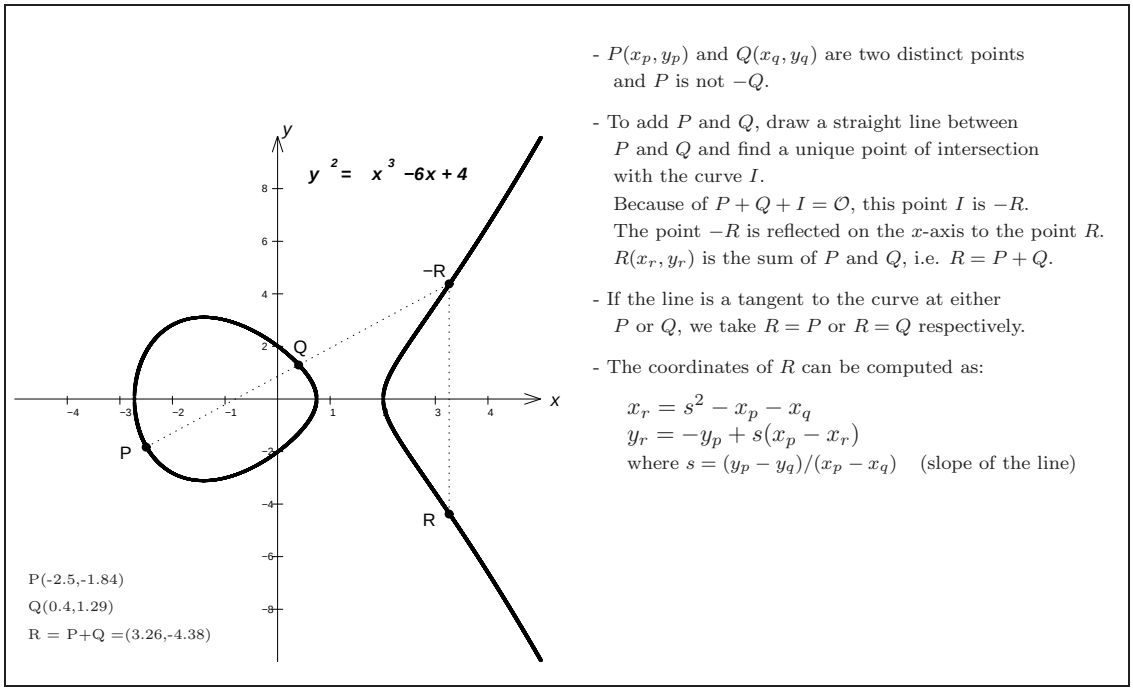


Fig. 5.5-2: Addition of distinct points on an elliptic curve.

group property of $(E, +)$

It can be easily shown that the structure $(E, +)$, where "+" represents the operation addition of points defined in Fig. 5.5-2 fulfills the requirements to be an abelian group:

1. For every two elements (points) $P, Q \in E$, the result $P+Q$ is also an element (point) of E. That means that the addition is an internal (binary) operation in the set of points on the curve E.

2. The operation addition of points is associative. That is, for all points $P, Q, R \in E$ we have $(P + Q) + R = P + (Q + R)$.

3. For any point P on the elliptic curve E, $P + \mathcal{O} = \mathcal{O} + \mathcal{P} = \mathcal{P}$ and $\mathcal{O} = -\mathcal{O}$. That means that the point of infinity $\mathcal{O}$ serves as an additive identity.

4. For each point $P \in E$, if there is an inverse point $P^{-1} \in E$, so that the result of the addition of these two elements is the group identity $\mathcal{O}$.

   The inverse point is also denoted as $-P$. For a given point $P(x_p, y_p) \in E$ the inverse is the point $-P$ with coordinates $(x_p, -y_p)$. A vertical line meets the curve at two points, $P_1 = (x_p, y_p)$ and $P_2 = (x_p, -y_p)$. It also meets the curve in the infinity point $\mathcal{O}$. Therefore, $P_1 + P_2 + \mathcal{O} = \mathcal{O}$ and $P_1 = -P_2$.

5. The operation addition of points is commutative. That is, for all points $P, Q \in E$ we have $P + Q = Q + P$.

avoiding singularities

Additional requirement for the elliptic curve $y^2 = x^3 + ax + b$ to form a group is that the expression $x^3 + ax + b$ should not have repeated factors, or equivalently, the determinant of the curve

$$D = 4a^3 + 27b^2 \qquad 5.5\text{-}3$$

should not be zero[18]. $D \neq 0$ ensures that the curve does not have singularities.

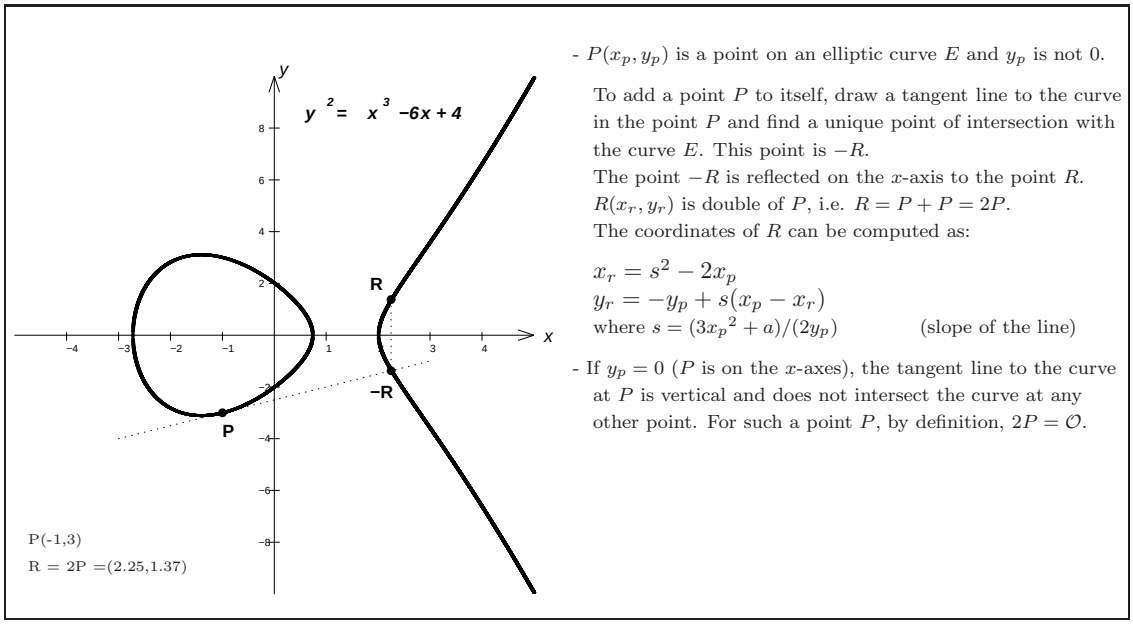


Fig. 5.5-3: Doubling of point P on an elliptic curve.

doubling of a point

scalar multiplication

In the group of elliptic curves, the operation *"doubling of a point"* can also be defined. Given a point P, computing $2P$ means computing $P + P$, i.e. addition of the point P to itself. This operation is also geometrically defined and is described in Fig. 5.5-3. Multiplication of a point $P \in E$ by a positive integer k is defined as a sum of the k copies of P. Thus, $2P = P + P$ (doubling), $3P = 2P + P$ (doubling and addition), $4P = 2 \cdot 2P$ (doubling), etc. Generally, the scalar multiplication can be iteratively computed as a chain of point doublings and point additions:

$$kP = P + (k-1)P. \qquad 5.5\text{-}4$$

18 The derivation of this equation is out of the scope of this book. You can find the details in [Menezes93].

5.5.2 Elliptic Curves Over Finite Fields

The group property is very important and actually this property enables us to do cryptography with elliptic curves. For this purpose, elliptic curves with a finite number of points are considered, i.e. elliptic curves over finite fields. Such curves are also called *discrete elliptic curves*. Actually, they are not "curves", similar to the continuous curves shown in Fig. 5.5-1, but rather a set of points which satisfy a given equation. Since these curves consist of a few discrete points, it is not clear how to "connect the dots" to make their graph look like a curve.

discrete elliptic curves

An elliptic curve over a field F_p can be constructed by choosing the variables a and b within the field F_p. The elliptic curve includes all points (x, y) which satisfy the elliptic curve equation modulo p:

definition of $E_p(a, b)$

$$y^2 \bmod p = x^3 + ax + b \bmod p.$$

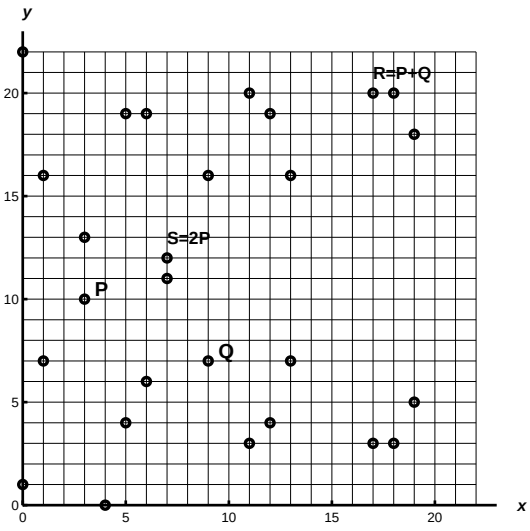
Note that the only difference between this equation and the Eq. 5.5-2 is that here the computations are performed $\bmod p$, i.e. in a field F_p. We can denote this curve as $E_p(a, b)$. There are finitely many points on such an elliptic curve, together with the infinity point $\mathcal{O}$. In general, the points which belong to the curve can be found in the following manner:

computing the points

- For each x with $0 \leq x < p$, calculate $x^3 + ax + b \bmod p$.
- For each result from the previous step, determine if it has a square root $\bmod p$.[19] If not, there is no point in $E_p(a, b)$ with this value of x. If so, there will be two values of y that satisfy the square root operation (exception: one value only when $y = 0$). These values (x, y) are the points on $E_p(a, b)$.

In the Fig. 5.5-4 the points of the curve $E_{23}(1, 1) : y^2 = x^3 + x + 1$ over F_{23} and their computation are shown. This curve has only 27 points. To exercise and learn, generate elliptic curves over different fields (vary p) and with different parameters (vary a and b). Note how the number of elements on the curve change. Use the java applet on the book home page.

19 It is out of the scope of this course to describe the algorithms which proove the existence of the square root $\bmod\ p$ and the computation of square roots. For this purpose, use the function $mod\_sqrt(a, p, 1)$ from the Crypto-Calculator.



- Elliptic curve $E_{23}(1,1) : y^2 = x^3 + x + 1$

$x = 0 \Rightarrow y = 1, y = 22$	$x = 12 \Rightarrow y = 4, y = 19$
$x = 1 \Rightarrow y = 7, y = 16$	$x = 13 \Rightarrow y = 7, y = 16$
$x = 2 \Rightarrow y$ does not exist	$x = 14 \Rightarrow y$ does not exist
$x = 3 \Rightarrow y = 10, y = 13$	$x = 15 \Rightarrow y$ does not exist
$x = 4 \Rightarrow y = 0$	$x = 16 \Rightarrow y$ does not exist
$x = 5 \Rightarrow y = 4, y = 19$	$x = 17 \Rightarrow y = 3, y = 20$
$x = 6 \Rightarrow y = 4, y = 19$	$x = 18 \Rightarrow y = 3, y = 20$
$x = 7 \Rightarrow y = 11, y = 12$	$x = 19 \Rightarrow y = 5, y = 18$
$x = 8 \Rightarrow y$ does not exist	$x = 20 \Rightarrow y$ does not exist
$x = 9 \Rightarrow y = 7, y = 16$	$x = 21 \Rightarrow y$ does not exist
$x = 10 \Rightarrow y$ does not exist	$x = 22 \Rightarrow y$ does not exist
$x = 11 \Rightarrow y = 3, y = 20$	

The set of all 27 points on the curve is: $(0,1)$, $(0,22)$, $(1,7)$, $(1,16)$, $(3,10)$, $(3,13)$, $(4,0)$, $(5,4)$, $(5,19)$, $(6,4)$, $(6,19)$, $(7,11)$, $(7,12)$, $(9,7)$, $(9,16)$, $(11,3)$, $(11,20)$, $(12,4)$, $(12,19)$, $(13,7)$, $(13,16)$, $(17,3)$, $(17,20)$, $(18,3)$, $(18,20)$, $(19,5)$, $(19,18)$.

Let $P = (3,10)$ and $Q = (9,7)$
Then $R = P + Q = (17,20)$
$S = 2P = (7,12)$

Fig. 5.5-4: Discrete elliptic curve $y^2 = x^3 + x + 1$ over F_{23}.

If p is prime and the expression $x^3 + ax^2 + b \bmod p$ contains no repeating factors (or, equivalentely, if the determinant $4a^3 + 27b^2 \bmod p$ is not 0), then the elliptic curve can be used to form a group. The group operation is, similar as in the case of continuous elliptic curves, the addition of points. The geometry used in elliptic curve groups over real numbers cannot be used for elliptic curve groups over F_p. However, the algebraic rules can be adopted for curves over F_p. In this case, the computing of the sum point coordinates is performed $\bmod\ p$. Unlike elliptic curves over real numbers, computations in the field F_p involve no round off errors — an essential property required for a cryptosystem. Now we summarize the addition rules for points on a discrete elliptic curve $E_p(a,b)$.

group property of $(E,+)$

Suppose $P = (x_p, y_p)$ and $Q = (x_q, y_q)$ are points on the curve $E_p(a,b)$. *"addition of points"* on the curve is defined as:

addition rules

- If $x_p = x_q$ and $y_p = -y_q$ (the points lie on vertical line), then $P + Q = \mathcal{O}$.
- Otherwise, $P + Q = R(x_r, y_r)$ where

$$x_r = s^2 - x_p - x_q \bmod p \qquad \text{5.5-5}$$
$$y_r = s(x_p - x_r) - y_p \bmod p$$

and

$$s = \left\{ \begin{array}{ll} (y_q - y_p)(x_q - x_p)^{-1} \bmod p, & if \quad P \neq Q \\ (3x_p^2 + a)(2y_p)^{-1} \bmod p, & if \quad P = Q \text{ (doubling of } P) \end{array} \right\}.$$

- $P + \mathcal{O} = \mathcal{O} + \mathcal{P} = \mathcal{P}$ for all $P \in E_p(a,b)$.

These equations hold except for the trivial case where P or Q are equal to the point of infinity. This definition of addition also includes the operation of doubling of points as a special case when $P = Q$. The only difference between the addition Eq. 5.5-5 and those for addition and doubling of points on a continuous elliptic

doubling of points

curve presented in the Fig. 5.5-2 and Fig. 5.5-3 is that here the operations are preformed $\mod p$, i.e. in the field F_p.

Example 5.5-1:
In this example, we compute the sum of the points P(3,10) and Q(9,7) on the curve $E_{23}(1,1)$:

$$\begin{aligned} s &= (7-10)(9-3)^{-1} = (-3)(6)^{-1} = 20 \cdot 4 = 80 \equiv 11 \bmod 23 \\ x_r &= s^2 - x_p - x_q = 11^2 - 3 - 9 = 109 \equiv 17 \bmod 23 \\ y_r &= s(x_p - x_r) - y_p = 11(3-(-6)) - 10 = 89 \equiv 20 \bmod 23. \end{aligned}$$

Thus, $R = P+Q = (17, 20)$. The double of the point $P(3, 10)$ can be computed as follows:

$$\begin{aligned} s &= (3 \cdot 3^2 + 1)(2 \cdot 10)^{-1} = 28 \cdot 20^{-1} = 5 \cdot 15 = 75 \equiv 6 \bmod 23 \\ x_s &= s^2 - x_p - x_p = 6^2 - 3 - 3 = 30 \equiv 7 \bmod 23 \\ y_s &= s(x_p - x_s) - y_p = 6(3-7) - 10 = -24 - 10 \equiv -34 \bmod 23 \\ &= 12 \bmod 23. \end{aligned}$$

The resulting point is $S = 2P = (7, 12)$.

elliptic curves over F_{2^m}

Another suitable field for cryptographic applications with elliptic curves is the field F_{2^m}. Elements of the field F_{2^m} are m-bit strings. The rules for arithmetic in this field can be defined by either *polynomial representation* or by *optimal normal basis representation*. An elliptic curve over F_{2^m} is constructed by choosing the elements a and b within F_{2^m}, and satisfying the following elliptic curve equation which is slightly adjusted for binary representation:

$$y^2 + xy = x^3 + ax + b. \qquad 5.5\text{-}6$$

There are finitely many points on this curve, also including the point of infinity $\mathcal{O}$. This curve also forms a group regarding the operation of addition of points, which is defined slightly differently than in Eq. 5.5-5. Since F_{2^m} operates on bit strings, computers can perform arithmetic in this field very efficiently. In a true cryptographic application, the parameter m must be large enough. Suitable choice today is $m = 160$. For simplicity, we will further discuss only ECPKCs defined over F_p.

5.5.3 Elliptic Curve Cryptosystems (ECCs)

The basis of every public-key cryptosystem is a hard mathematical problem that is computationally infeasible to solve. To construct a cryptosystem using elliptic curves, we need to find a ”hard problem” corresponding to factoring the product of two large primes or taking the discrete logarithm. This hard problem is the *elliptic curve*

discrete logarithm problem (ECDLP), which is based upon the intractability of scalar multiplication of points. Before some cryptosystems based on elliptic curves (ECCs) are described, the ECDLP is defined.

discrete log problem in $(E, +)$

Consider the equation $Q = kP$, where $Q, P \in E_p(a, b)$ and k is an integer, $k < p$. It is relatively easy to compute Q when k and P are given through combination of point doubling and point addition, but it is relatively hard to determine k when P and Q are given. So, if we use the additive notation to describe an elliptic curve group (as we have done in the previous sections), we can define the ECDLP in the following way:

- Given points P and Q from $E_p(a, b)$, find a number k so that $Q = kP$.

An elliptic curve group can also be described using the multiplicative notation, using the same operation as Eq. 5.5-5 for multiplying copies of the point P, yielding the point $Q = P \cdot P \cdot P..... \cdot P = P^k$. In this case, we can define the ECDLP in the following way:

- Given points P and Q from $E_p(a, b)$, find a number k so that $P^k = Q$.

Example 5.5-2:
In this example we try to find the discrete logarithm k of $Q = (4, 5)$ to the base $P = (16, 5)$ on the curve $E_{23} : y^2 = x^3 + 9x + 17$. One (naive) way to find k is to compute multiples of P until Q is found: $P = (16, 5), 2P = (20, 20), 3P = 2P + P = (14, 14), 4P = 2 \cdot 2P = (19, 20), 5P = 4P + P = (13, 10), 6P = 2 \cdot 3P = (7, 3), 7P = 6P + P = (8, 7), 8P = 2 \cdot 4P = (12, 17), 9P = 8P + P = (4, 5)$. Since $9P = (4, 5) = Q$, the discrete logarithm of Q to the base P is $k = 9$.

In real applications, k would be large enough (over 100 bits) so that it would be infeasible to determine k in this manner. There are several algorithms for computing discrete logarithms which can be applied to elliptic curve groups (see section 2.6.3), but these are exponential time algorithms. Thus, a key advantage of elliptic curve cryptosystems is that no subexponential algorithm is known that breaks the system[20]. Moreover, ECDLP is much harder than the DLP in $\mathbb{Z}_p^*$. We now describe analogous of some public- key cryptosystems based on the ECDLP.

Key Exchange with Elliptic Curves

The Diffie-Hellman key exchange algorithm can be applied on an elliptic curve group. In this case, the security of the algorithm is based on the elliptic curve discrete log problem.

20 The description of the algorithms and their computational complexity are out of the scope of this book. Overview of the algorithms and details can be found in [Menezes93].

parameters of the system

Suppose that Alice and Bob (A and B) want to agree upon a key which will later be used in conjunction with a classical cryptosystem. They first publicly choose a finite field F_p and an elliptic curve $E_p(a, b)$ defined over it. In order to enable enough security, the prime p should be $p \approx 2^{180}$. Then they choose a point $G(x_g, y_g)$ on the curve which serves as a "base". G plays the role of a generator in the group $E_p(a, b)$, but even if the point G is not a generator of the group, we would like the subgroup generated by G to be large, preferably of the same order of size as E itself. Consequently, the important criterion in selecting the point G is that the smallest value of n for which $nG = \mathcal{O}$ (G generates the group) is a very large prime number. $E_p(a, b)$ and G are parameters of the cryptosystem known to all participants.

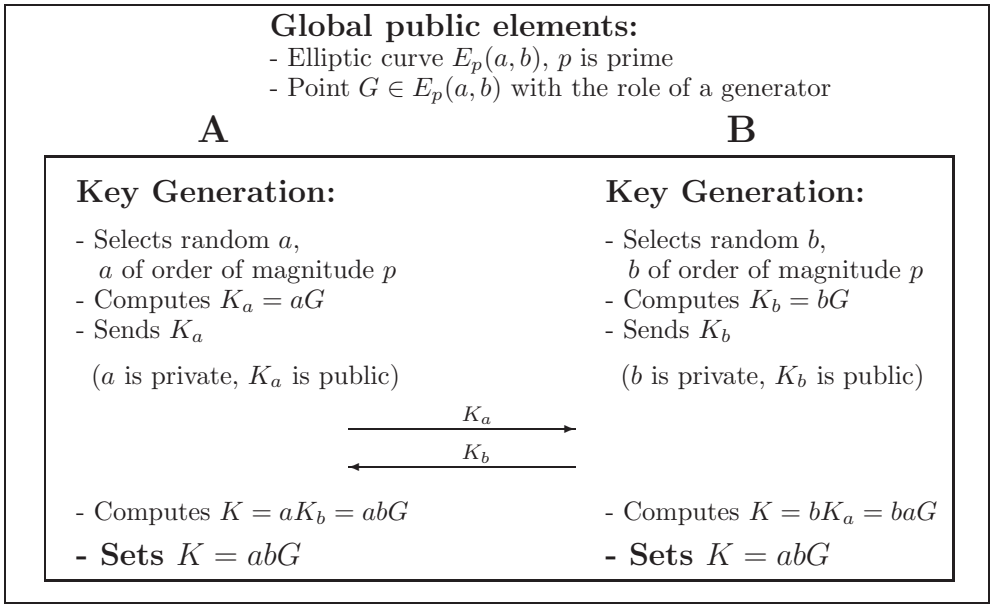


Fig. 5.5-5: Diffie-Hellman key exchange with ECC.

Diffie-Hellman ECC algorithm

To generate a key, Alice chooses a random integer a of order of magnitude p, which she keeps secret. She computes the point $K_a = aG \in E$, which she makes public. Bob does the same: he chooses a random b, which he keeps secret, computes the point $K_b = bG$, and makes it public. The secret key they use is then the point $K = abG$. Both users can compute this key: Alice knows bG (which is public knowledge) and her own secret a, so she computes $K = abG$; Bob receives aG and multiplies it with his own secret b. Without solving the ECDLP (finding a knowing aG and G or finding b knowing bG and G) there is no way to compute the key $K = abG$ knowing only aG and bG. The scheme is summarized in Fig. 5.5-5.

Note that the secret key K, which is a point on $E_p(a, b)$, is a pair of numbers (x_k, y_k). From this pair, a single number must be generated. One can simply use the x coordinate or some simple function of the coordinates.

Elliptic Curve Encryption/Decryption

To encrypt a given plaintext message m with some EC cryptosystem, it must represent a point P on the chosen curve $E_p(a, b)$, i.e. the message must be encoded in (x, y) coordinates of the point P. This point P will be encrypted. There are several approaches for this encoding, which we will not address here. Many approaches for encryption/decryption using elliptic curves have been analyzed in the literature. We now show only two of them: The ElGamal EC cryptosystem and the Menezes-Vanstone EC cryptosystem.

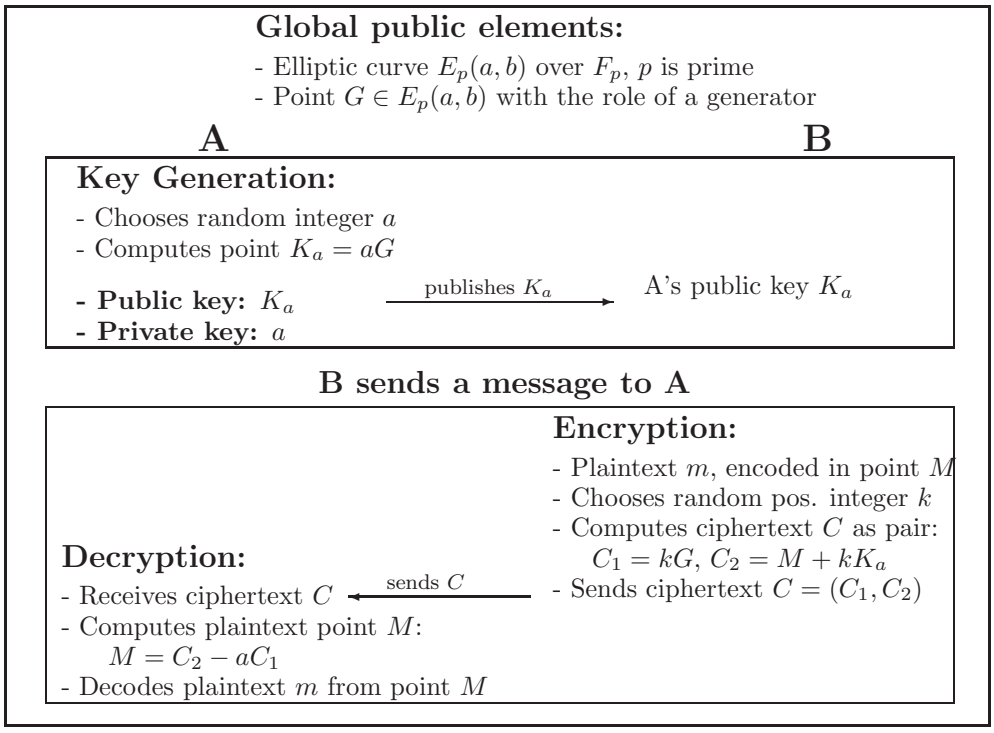


Fig. 5.5-6: ElGamal elliptic curve encryption scheme.

ElGamal EC cryptosystem

In the **ElGamal EC cryptosystem**, we start with the parameters of the cryptosystem, which are known to each user: A finite field F_p, an elliptic curve $E_p(a, b)$ defined over it, and a point G on the curve. Each user selects a private key as an integer a_i and computes the public key a_iG. If Bob wants to send a message to Alice, he needs her public key K_a, which is a point computed from Alice as $P_a = aG$ (note that a is Alice's secret). Suppose, he wants to encrypt the plaintext message m, which is encoded in the point $M \in E_p(a, b)$. Bob chooses a random positive integer k and produces the ciphertext C consisting of the pair of points $C_1 = kG$ and $C_2 = M + kK_a$ and sends the pair $C(C_1, C_2)$ to Alice. Note that Bob has used Alice's public key K_a to compute the ciphertext. He has masked the message M by adding kP_a to it. Nobody but Bob knows the value of k, so even though K_a is public, nobody can remove the mask kP_a.

To decrypt the message, Alice uses her private key a to multiply the first point of the ciphertext C_1 with it, and substracts the result from the second point:

$$C_2 - aC_1 = M + kK_a - a \cdot kG = M + kaG - akG = M. \qquad 5.5\text{-}7$$

The result M is a point on the curve $E_p(a, b)$ which can now be easily decoded to the corresponding plaintext m. To recover the message, an attacker would have to compute k from a given G and $K_a = kG$ which is assumed to be hard. There are some practical difficulties in implementing an ElGamal cryptosystem on an elliptic curve. The message has an expansion factor of (about) four. This happens since a ciphertext consists of two points on the elliptic curve, each with two coordinates. Another problem is that the plaintext m must be encoded (deterministically generated) in a point which lies on the curve E.

Menezes-Vanstone EC cryptosystem

A more efficient cryptosystem has been found by **Menezes** and **Vanstone**. In this variation, the elliptic curve is used for "masking", and plaintext and ciphertext are allowed to be arbitrary ordered points of nonzero field elements, i.e. *they are not required to be points on* E. This leads to a message expansion of factor two. We now briefly describe this cryptosystem, which is summarized in Fig. 5.5-7.

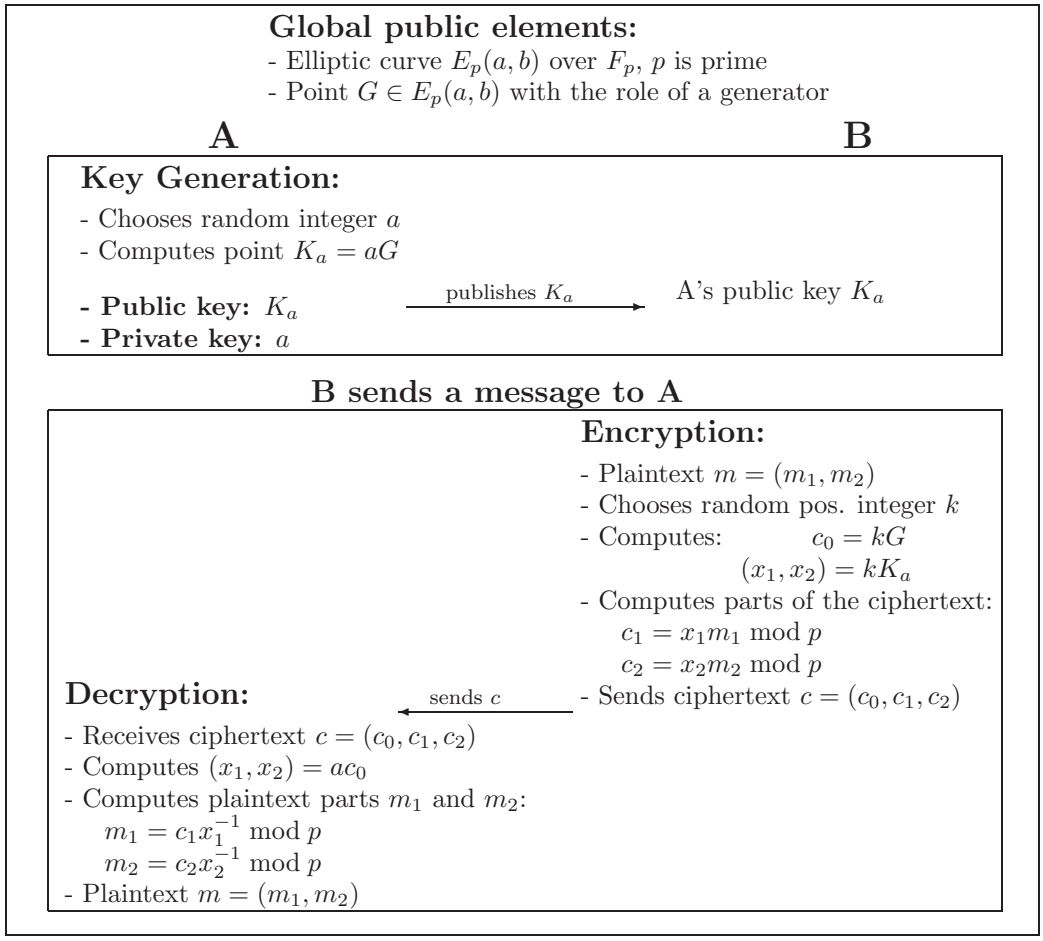


Fig. 5.5-7: Menezes-Vanstone Elliptic curve cryptosystem.

The global public elements of the system are an elliptic curve $E_p(a, b)$ and a point G on it. The curve must contain a cyclic subgroup H in which the discrete log problem is intractable. Each user chooses a large random integer a_i and keeps it secret, and publishes a_iG. So, Alice generates her key pair by choosing an integer a, which will be her private key, and computes the a-multiple of the base point $K_a = aG$, which will be her public key. Now Bob wants to send Alice the message $m = (m_1, m_2)$. In order to encrypt this plaintext, he first chooses a random positive integer k and computes $c_0 = kG$ and $(x_1, x_2) = kK_a$. Then he computes $c_1 = x_1 m_1 \bmod p$ and $c_2 = x_2 m_2 \bmod p$, which are parts of the ciphertext. The ciphertext consist of c_0, c_1 and c_2, i.e. $c = (c_0, c_1, c_2)$. To decrypt the ciphertext, Alice uses her private key a to compute $(x_1, x_2) = ac_0$ (because $ac_0 = akG = k \cdot aG = kK_a = (c_1, c_2)$) and then computes the plaintext parts $m_1 = c_1 x_1^{-1} \bmod p$ and $m_2 = c_2 x_2^{-1} \bmod p$. She gets the plaintext as $m = (m_1, m_2)$.

Example 5.5-3:
In this example we illustrate encryption and decryption according to the Menezes-Vanstone cryptosystem. Let the curve $E_{11}(1, 6) : y^2 = x^3 + x + 6$ over $\mathbb{Z}_{11}$ and the point $G(2, 7)$ be the public elements of the system. Alice chooses $a = 7$ for her private key and computes her public key: $K_a = aG = 7 \cdot (2, 7) = (7, 2)$. Suppose, Bob wants to send her the plaintext $m = (m_1, m_2) = (9, 1)$. Note that m is not a point on E. He first chooses a random value $k = 6$ and computes $c_0 = kG = 6 \cdot (2, 7) = (7, 9)$ and $(x_1, x_2) = kK_a = 6 \cdot (7, 2) = (8, 3)$. Then he computes c_1 and c_2 as: $c_1 = x_1 m_1 \bmod p = 8 \cdot 9 \bmod 11 = 6$ and $c_2 = x_2 m_2 \bmod p = 3 \cdot 1 \bmod 11 = 3$. The ciphertext Bob sends to Alice is $c = (c_0, c_1, c_2) = ((7, 9), 6, 3)$. When Alice receives the ciphertext c, she first computes $(x_1, x_2) = ac_0 = 7 \cdot (7, 9) = (8, 3)$, and then she computes the plaintext parts m_1 and m_2 as: $m_1 = c_1 x_1^{-1} \bmod p = 6 \cdot 8^{-1} \bmod 11 = 6 \cdot 7 \bmod 11 = 9$ and $m_2 = c_2 x_2^{-1} \bmod p = 3 \cdot 3^{-1} \bmod 11 = 3 \cdot 4 \bmod 11 = 1$. So, the plaintext is $(9, 1)$.

advantages of ECCs

All cryptosystems based on the discrete logarithm problem (DLP) can be converted to elliptic curve cryptosystems. Some slight technical modifications are necessary in order to adapt to the elliptic curve setting, but the underlying principles are the same as for other DLP-based systems. A very important observation is that the best known algorithms for the ECDLP have a complexity proportional to the square root of the group size, whereas DLP in $\mathbb{Z}_p$ and integer factorization can be solved in subexponential time. This implies that, for a certain level of security, the sizes of the parameter in ECC can be substantially smaller. For example, an elliptic curve group with a 175-bit size has a security that is equivalent to RSA with a 1024-bit modulus, or to systems based on DLP in $\mathbb{Z}_p$ with p a 1024-bit prime. The smaller block size has important implications on the resources that are needed to implement an EC cryptosystem. For example, far less chip area is needed for an elliptic curve processor than for an RSA processor.

To better understand the properties of elliptic curves and the operations addition of points and multiplication of a point with a scalar, as well as the group properties, please do some exercises with the java-applets on the book home page.

5.6 Other Public-Key Cryptosystems (PKCs)

We will now give a brief overview of several other public-key cryptosystems, which have some importance in cryptology.

Merkle-Hellman crypto system

The *Merkle-Hellman knapsack cryptosystem* was first described by Merkle and Hellman in 1978. Although this cryptosystem was broken in the early 1980's, it is still worth studying because of the underlying design technique. This system is based on the "hard" problem of subset sum, which is an NP-complete problem. That means, that there is no known polynomial time algorithm that solves it.

McEliece cryptosystem

The *McEliece cryptosystem* is based on algebraic coding theory and is still regarded as being secure. The NP-complete problem that is employed is decoding a general linear binary error-correcting code. One class of codes, the so called Goppa codes, are used as base of this cryptosystem.

Rabin's cryptosystem

The *Rabin's cryptosystem* gets its security from the difficulty of finding square roots modulo a composite number. This problem is equivalent to factoring. The Rabin public-key encryption scheme was the first example of a *provably secure encryption scheme* (there is a proof that breaking the scheme is equivalent to solving a computational problem which is widely believed to be difficult), which is a desirable property of any encryption scheme.

6 Digital Signatures

6.1 Introduction

The notion of a digital signature may prove to be one of the most fundamental and useful inventions of modern cryptography. A signature scheme provides a way for each user to sign messages so that the signatures can later be verified by anyone else. More specifically, each user can create a matched pair of private and public keys so that only he can create a signature for a message (using his private key), but everyone can verify the signature for the message (using the signer's public key). The verifier can convince himself that the message content has not been altered since the message was signed[21]. Also, the signer can not later repudiate having signed the message, since no one but the signer possesses the private key[22].

data integrity

non-repudiation

By analogy to the paper world, where one might sign a letter and seals it in an envelope, one can sign an electronic message using one's private key, and then seals the result by encrypting it with the recipient's public key. The recipient can perform the inverse operations of opening the letter and verifying the signature using his private key and the sender's public key, respectively. These applications of public-key technology, e.g. to electronic mail, are quite widespread today already.

Like a hand-written signature, the purpose of a digital signature is to guarantee that the individual sending the message really is the one who he or she claims to be[23]. Digital signatures are especially important for electronic commerce and are a key component for most authentication schemes. To be effective, digital signatures must be unforgeable. An example of what a digital signature looks like is given in Example 6.1-1.

authentication

Example 6.1-1: Digital Signature

```
-------BEGIN SIGNATURE------
LQB1awUBMVSiA5QYCuMfgNYjAQFAKgLjZkBfbeNEsbthba4Blrcnjad
mcKgNv+a5Kr4537y8RCd+RHm75yYh5xxA1ojELwNhhb7cltrp2V7Llb
xAelws4S87UX80cLBtBcN6AACf11qymC2h+Rb2j5SU+rmXWru+=QFGs
------END SIGNATURE------
```

21 *Data integrity*: The assurance that the data received was exactly the data sent.

22 *Non-repudiation*: Prevents a user from denying previous commitments or actions.

23 *Authentication* is any process through which one proves and verifies certain information. Sometimes one may want to verify the origin of a document, the identity of the sender, the time and date a document was sent and/or signed, the identity of a computer or user, and so on.

6.2 RSA Signatures

When public-key cryptography is used to encrypt a message, the sender encrypts the message with the public key of the intended recipient. When public-key cryptography is used to calculate a digital signature, the sender encrypts the message or the message digest[24] of the document with his or her own private key. Anyone with access to the public key of the signer may verify the signature.

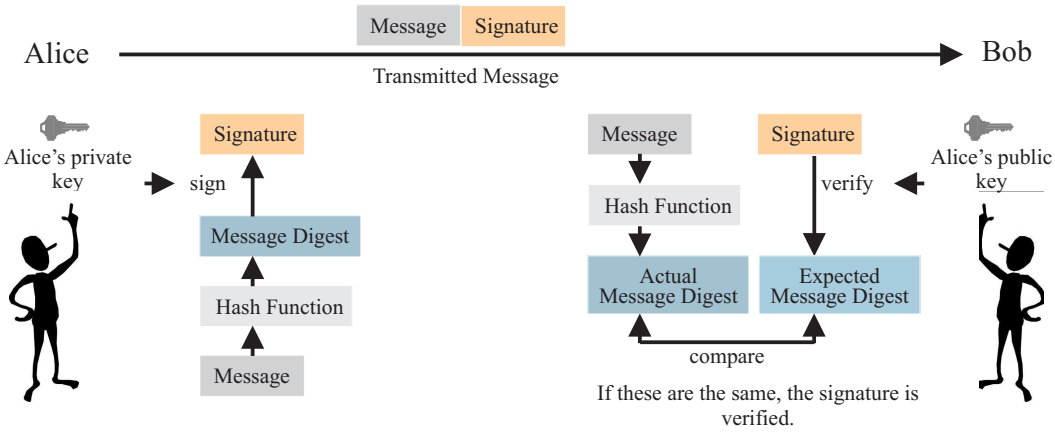


Fig. 6.2-1: RSA digital signature scheme using a hash function.

More specifically, suppose Alice wants to send a signed message to Bob (see Fig. 6.2-1):

- The first step is generally to apply a hash function to the message, creating a message digest. The hash function takes a message of an arbitrary length and shrinks it down to a fixed length.
- To create a digital signature, Alice signs the message digest instead of the message itself. This saves a considerable amount of computation time.
- Alice sends Bob the encrypted message digest and the message, which she may or may not encrypt using Bob's public key.
- In order for Bob to authenticate the signature he must apply the same hash function as Alice to the message she sent him, decrypt the encrypted message digest using $Alice$'s public key and compare the two.
- If the two are the same he has successfully authenticated the signature. If the two do not match there are a few possible explanations. Either someone is trying to impersonate Alice, the message itself has been altered since Alice signed it or an error occurred during transmission.

24 A hash function H is a transformation that takes an input m and returns a fixed-size string, which is called the hash value or message digest h. One can think of a message digest as a "digital fingerprint" of the larger document. Hash functions will be introduced in chapter 7.

6.2.1 Some Comments

As described above, everyone can read the message and verify the signature. This does not satisfy situations where A ($Alice$) wishes to retain the *secrecy* of the document. In this case she may wish to sign the document, then encrypt it using Bob's public key. B (Bob) will then need to decrypt the message using his private key and verify the signature on the recovered message using A's public key.

secrecy

Alternately, if it is necessary for *third parties* to validate the integrity of the message without being able to decrypt its content, a message digest may be computed on the encrypted message, rather than on its plaintext form.

third parties

There is a potential problem with this type of digital signature. A not only signed the message she intended to but also signed all other messages that happen to hash to the same message digest. When two messages hash to the same message digest, this is called a *collision*; the collision-free properties of hash functions are necessary security requirements for most digital signature schemes. A hash function is secure if it is very time consuming, if at all possible, to figure out the original message given its digest. However, there is an attack called the *birthday attack*[25] that relies on the fact that it is easier to find two messages that hash to the same value than to find a message that hashes to a particular value.

collision

birthday attack

In addition, someone could pretend to be A and sign documents with a key pair he claims is A's. To avoid scenarios such as this, there are digital documents called *certificates* that associate a person with a specific public key.

certificates

Digital timestamps may be used in connection with digital signatures to bind a document to a particular time of origin. It is not sufficient to just note the date in the message, since dates on computers can be easily manipulated. It is better that timestamping is done by someone everyone trusts, such as a certifying authority.

timestamps

6.2.2 Description of the algorithm

The RSA cryptosystem was developed by Ronald Rivest, Adi Shamir, and Leonard Adleman in 1977 ([RSA78]); RSA stands for the first letter in each of its inventors' last names. For more information on RSA, see Section 5.2 or [MOV96].

The key generation, signature generation, and signature verification procedures for RSA are given below.

RSA key generation

Each entity A does the following:

- Take two large primes, p and q, and compute their product $n = pq$; n is called the modulus.

25 Its name arises from the fact that for a group of 23 or more people the probability that two or more people share the same birthday is higher than 50%.

- Choose a number, e, less than n and relatively prime to $(p-1)(q-1)$, which means e and $(p-1)(q-1)$ have no common factors except 1.
- Find another number d such that $(ed-1)$ is divisible by $(p-1)(q-1)$.
- The values e and d are called the public and private exponents, respectively. The public key is the pair (n, e); the private key is (n, d).

The factors p and q may be erased or kept with the private key.

RSA signature generation

To sign a message m, A does the following:

$$s = m^d \, mod \, n$$

where d and n are A's private key. She sends m and s to B.

RSA signature verification

To verify the signature s, B exponentiates and checks that the message m is recovered:

$$m = s^e \, mod \, n$$

where e and n are A's public key.

6.3 ElGamal Signature Scheme

The ElGamal system is a public-key cryptosystem based on the discrete logarithm problem[26]. It consists of both encryption and signature variants[27].

6.3.1 Key generation

Each entity A does the following:

1. Generate a large random prime p and a generator α of the multiplicative group $\mathbb{Z}_p^*$.
2. Select a random integer $a, 1 \leq a \leq p-2$.
3. Compute $y = \alpha^a \mod p$.
4. A's public key is (p, α, y); A's private key is a.

26 For more information on the discrete logarithm problem, see Section 5.3.

27 For more information on the ElGamal cryptosystem and its security aspects, see Section 5.4

6.3.2 Signature generation

To sign a message m, A does the following:

1. Select a random secret integer $k, 1 \leq k \leq p-2$, with $gcd(k, p-1) = 1$.
2. Compute $r = \alpha^k \bmod p$.
3. Compute $k^{-1} \bmod (p-1)$.
4. Compute $s = k^{-1}(h(m) - ar) \bmod (p-1)$; h being a collision-free hash function.
5. The signature for the message m is the pair (r, s).

6.3.3 Signature verification

To verify A's signature (r, s) on m, B should:

1. Obtain an authentic copy of A's public key (p, α, y).
2. Verify that $1 \leq r \leq p-1$; if not, then reject the signature.
3. Compute $v_1 = y^r r^s \bmod p$.
4. Compute $h(m)$ and $v_2 = \alpha^{h(m)} \bmod p$.
5. Accept the signature if and only if $v_1 = v_2$.

6.4 DSA - Digital Signature Algorithm

The Digital Signature Algorithm (DSA) was proposed in August 1991 by the U.S. National Institute of Standards and Technology (NIST) and became a U.S. Federal Information Processing Standard (FIPS 186) in 1993. The FIPS 186 standard is also referred to as the *Digital Signature Standard* (DSS). The DSA was the first digital signature scheme accepted as legally binding by a government. The algorithm is a variant of the *ElGamal* signature scheme (see Section 6.3). The key generation, signature generation, and signature verification procedures for DSA are given below.

DSS

ElGamal

6.4.1 DSA key generation

Each entity A does the following:

1. Select a prime q such that $2^{159} < q < 2^{160}$.
2. Select a 1024-bit prime number p with the property that $q|p-1$. (The DSS mandates that p be a prime such that $2^{511+64t} < p < 2^{512+64t}$ where $0 \leq t \leq 8$. If $t = 8$ then p is a 1024-bit prime.
3. Select an element $h \in \mathbb{Z}_p^*$ and compute $g = h^{(p-1)/q} \bmod p$; repeat until $g \neq 1$ (g is a generator of the unique cyclic group of order q in $\mathbb{Z}_p^*$).
4. Select a random integer x in the interval $[1, q-1]$.
5. Compute $y = g^x \bmod p$.

6. A's public key is (p, q, g, y); A's private key is x.

6.4.2 DSA signature generation

To sign a message m, A does the following:

1. Select a random secret integer k in the interval $[1, q-1]$.
2. Compute $r = (g^k \bmod p) \bmod q$.
3. Compute $k^{-1} \bmod q$.
4. Compute $s = k^{-1}(h(m) + xr) \bmod q$ where h is the Secure Hash Algorithm (SHA-1)[28].
5. Is $s = 0$ then go to step 1 (if $s = 0$, then $s^{-1} \bmod q$ does not exist; s^{-1} is required in step 3 of signature verification).
6. The signature for the message m is the pair or integers (r, s).

6.4.3 DSA signature verification

To verify A's signature (r, s) on m, B should:

1. Obtain an authentic copy of A's public key (p, q, g, y).
2. Verify that r and s are integers in the interval $[1, q-1]$.
3. Compute $w = s^{-1} \bmod q$ and $h(m)$.
4. Compute $u_1 = h(m)w \bmod q$ and $u_2 = rw \bmod q$.
5. Compute $v = (g^{u_1} y^{u_2} \bmod p) \bmod q$.
6. Accept the signature if and only if $v = r$.

6.4.4 Security aspects

DSA signatures are 320 bits in size, since r and s are each integers less than q. The security of the DSA relies on two different but related discrete logarithm problems.

One is the discrete logarithm problem in $\mathbb{Z}_p^*$ where the number field sieve algorithm applies; this logarithm has a subexponential running time.

The second discrete logarithm problem works to the base q: given p, q, g, and y find x, such that $y \equiv g^x \pmod p$. For large p (e.g. 1024-bits), the best algorithm known for this problem is the Pollard rho-method, and takes about

$$\sqrt{\pi q/2}$$

steps.

28 See Section 7.4.3.

6.5 ECDSA - Elliptic Curve Digital Signature Algorithm

The elliptic curve analogue of the DSA is the ECDSA. Instead of working in a subgroup of order q in $\mathbb{Z}_p^*$, this algorithm works in an elliptic curve group $E(\mathbb{Z}_p)$. In February 2000 the US Secretary of Commerce approved the inclusion of the ECDSA in the US Government's Digital Signature Standard. The revised standard is FIPS 186-2 that specifies ECDSA by giving a reference to the ANSI X9.62 standard. The correspondence between some math notations used in DSA and ECDSA are shown in Table 6.5-1. Using Table 6.5-1 and Table 6.5-2, the analogies between DSA and ECDSA should be obvious ([JM99]).

Tab. 6.5-1: *Correspondence between DSA and ECDSA notation.*

DSA notation	ECDSA notation
q	n
g	P
x	x
y	Q

Tab. 6.5-2: *Correspondence between $\mathbb{Z}_p^*$ and $E(\mathbb{Z}_p)$ notation [JM99].*

Group	$\mathbb{Z}_p^*$	$E(\mathbb{Z}_p)$
Group elements	Integers $\{1, 2, \ldots, p-1\}$	Points (x, y) on E plus $\mathcal{O}$
Group operation	Multiplication modulo p	Addition of points
Notation	Elements: g, h Multiplication: $g \cdot h$ Inverse: g^{-1} Division: g/h Exponentiation: g^a	Elements: P, Q Addition: $p + Q$ Negative: $-P$ Subtraction: $P - Q$ Multiple: aP
Discrete logarithm problem	Given $g \in \mathbb{Z}_p^*$ and $h = g^a$ mod p, find a	Given $P \in E(\mathbb{Z}_p)$ and $Q = aP$, find a

The key generation, signature generation, and signature verification procedures for ECDSA are given below.

6.5.1 ECDSA key generation

Each entity A does the following:

1. Select an elliptic curve E defined over $\mathbb{Z}_p$. The number of points in $E(\mathbb{Z}_p)$ should be divisible by a large prime n.
2. Select a point $P \in E(\mathbb{Z}_p)$ of order n.
3. Select a statistically unique and unpredictable integer d in the interval $[1, q - 1]$.
4. Compute $Q = dP$.

5. A's public key is (E, P, n, Q); A's private key is d.

6.5.2 ECDSA signature generation

To sign a message m, A does the following:

1. Select a statistically unique and unpredictable integer k in the interval $[1, q-1]$.
2. Compute $kp = (x_1, y_1)$ and $r = x_1 \bmod n$ (here x_1 is regarded as an integer, e.g. by conversion from its binary representation). If $r = 0$, then go to step 1 (this is a security condition: if $r = 0$, then the signing equation $s = k^{-1}(h(m) + dr) \bmod n$, does not involve the private key d).
3. Compute $k^{-1} \bmod n$.
4. Compute $s = k^{-1}(h(m) + dr) \bmod n$, where h is the Secure Hash Algorithm (SHA-1)[29].
5. If $s = 0$, then go to step 1. (If $s = 0$, then $s^{-1} \bmod n$ does not exist; s^{-1} is required in step 3 of signature verification).
6. The signature for the message m is the pair of integers (r, s).

6.5.3 ECDSA signature verification

To verify A's signature (r, s) on m, B should:

1. Obtain an authentic copy of A's public key (E, P, n, Q).
2. Verify that r and s are integers in the interval $[1, q-1]$.
3. Compute $w = s^{-1} \bmod n$ and $h(m)$.
4. Compute $u_1 = h(m)w \bmod n$ and $u_2 = rw \bmod n$.
5. Compute $u_1P + u_2Q = (x_0, y_0)$ and $v = x_0 \bmod n$.
6. Accept the signature if and only if $v = r$.

6.5.4 Note

ANSI X9.62 prescribes that $n > 2^{160}$. The parameter n should have about 160 bits to obtain a security level similar to that of the DSA (with 160-bit q and 1024-bit p). If this is the case, then DSA and ECDSA signatures have the same size (320 bits).

consistencies with DSA

The important consistencies between ECDSA and DSA are as follows:

1. DSA and ECDSA are based on the ElGamal signature scheme and use the same signing equation: $s = k^{-1}(h(m) + dr) \bmod n$.

29 See Section 7.4.3.

2. The values that are relatively difficult to generate are the system parameters (p, q, and g for DSA; E, P, and n for ECDSA) which are public. The generation of a private key is relatively simple and generating the associated public key is straightforward.
3. DSA and ECDSA both use the SHA-1 as cryptographic hash function.

For more information on DSA and ECDSA see [JM99].

6.5.5 Security aspects

The security of the ECDSA is based on the following *elliptic curve discrete logarithm problem* (ECDLP): given an elliptic curve E defined over $\mathbb{Z}_p$, a point $P \in E(\mathbb{Z}_p)$ of order n, and a point $Q \in E(\mathbb{Z}_p)$, determine the integer $l, 0 \leq l \leq n-1$, such that $Q = lP$, provided that such an integer exists.

The ECDLP has been analyzed by mathematicians around the world, and no significant weaknesses have been reported. For more information on security aspects and key lengths, see Section 5.5.

6.6 Signatures with Additional Functionality

There are signature schemes that provide functionality beyond authentication and non-repudiation. To achieve additional features which the basic method does not provide, we combine a basic digital signature scheme with a specific protocol. For information on further signature schemes not mentioned here, see [MOV96].

6.6.1 Fail-stop signatures

A fail-stop signature scheme is a type of signature introduced by Birgit Pfitzmann and Michael Waidner to protect against the possibility that an enemy may be able to forge a user's signature. It is a variation of the one-time signature scheme, in which only a single message can be signed and protected by a given key at a time. The scheme is based on the discrete logarithm problem. In particular, if an enemy can forge a signature, then the actual signer can prove that forgery has taken place by demonstrating the solution of a supposedly hard problem. Thus the forger's ability to solve that problem is transferred to the actual signer.

The term "fail-stop" refers to the fact that a signer can detect and stop failures, i.e. forgeries. Note that if the enemy obtains an actual copy of the signer's private key, forgery cannot be detected. What the scheme detects are forgeries based on cryptanalysis. For more information on fail-stop signatures, see [Pfi96] and [MOV96].

6.6.2 Blind signatures

Blind signature schemes, first introduced by David Chaum ([CRS83]), allow a person to get a message signed by another entity without revealing any information about the message to the other entity.

Using RSA signatures blind signatures work as follows: Suppose entity A has a message m that she wishes to have signed by entity B, and she does not want B to learn anything about m. Let (n, e) be B's public key and (d) be his private key[30]. A generates a random value r such that $gcd(r, n) = 1$ and sends

$$m' = r^e \; m \; mod \; n$$

to B. The value m' is "blinded" by the random value r, and hence B can derive no useful information from it. B returns the signed value,

$$s' = (m')^d = (r^e m)^d \; mod \; n$$

to A. Since $s' = rmd$ mod n, A can obtain the true signature s of m by computing

$$s = s' r^{-1} \; mod \; n.$$

Now A's message has a signature she could not have obtained on her own. This signature scheme is secure provided that factoring and root extraction remain difficult. However, regardless of the status of these problems the signature scheme is unconditionally "blind" since r is random. The random r does not allow the signer to learn anything about the message even if the signer can solve the underlying hard problems.

Blind signatures have numerous uses, e.g. digital cash. Thus it is not surprising that there are now numerous variations of the blind signature scheme.

6.6.3 Undeniable signatures

Undeniable signature schemes, first introduced by Chaum and van Antwerpen, are non-self-authenticating signature schemes, where signatures can only be verified with the signer's consent. However, if a signature is only verifiable with the aid of a signer, a dishonest signer may refuse to authenticate a genuine document. Undeniable signatures solve this problem by adding a new component called the *disavowal protocol* in addition to the normal components of signature and verification.

The scheme is implemented using public-key cryptography based on the discrete logarithm problem.

30 For more information on RSA key generation, see Section 6.2.2.

Key generation

Each entity A does the following:

1. Take a random prime $p = 2q + 1$ where q is also a prime.
2. Generate a generator α of the multiplicative group $\mathbb{Z}_p^*$.
3. Select a random integer $\alpha \in \{1, 2, \ldots, q-1\}$.
4. Compute $y = \alpha^a \bmod p$.
5. A's public key is (p, α, y); A's private key is a.

Signature generation

To sign a message m, A does the following:

$$s = m^a \, mod \, p.$$

A sends m and s to B.

Signature verification

Verification is carried out by a challenge-response protocol. To verify A's signature s on m, B should:

1. Obtain an authentic copy of A's public key (p, α, y).
2. Select random secret integers $x_1, x_2 \in \{1, 2, \ldots, q-1\}$.
3. Compute $z = s^{x_1} y^{x_2} \bmod p$ and send z to A.
4. A computes $w = (z)^{a^{-1}} \bmod p$ (where $aa^{-1} \equiv 1 \pmod q$) and sends w to B.
5. B computes $w' = m^{x_1} \alpha^{x_2} \bmod p$ and accepts the signature if and only if $w = w'$.

One can prove that the signature verification works as required:

$$w \equiv (z)^{a^{-1}} \equiv (s^{x_1} y^{x_2})^{a^{-1}} \equiv (m^{ax_1} \alpha^{ax_2})^{a^{-1}} \equiv m^{x_1} \alpha^{x_2} \equiv w^{'} \, mod \, p.$$

The disavowal process is similar.

Signature disavowal

The disavowal process determines wheter A is attempting to disavow a valid signature s using the verification algorithm (presented above), or whether the signature is a forgery.

1. B obtains an authentic copy of A's public key (p, α, y).
2. B selects random secret integers $x_1, x_2 \in \{1, 2, \ldots, q-1\}$.
3. B computes $z = s^{x_1} y^{x_2} \bmod p$ and sends z to A.
4. A computes $w = (z)^{a^{-1}} \bmod p$ (where $aa^{-1} \equiv 1 \pmod q$) and sends w to B.
5. If $w = m^{x_1} \alpha^{x_2} \bmod p$, B accepts the signature s and the protocol halts.

6. B selects random secret integers $x_1^{'}, x_2^{'} \in \{1, 2, \ldots, q-1\}$, and computes $z^{'} = s^{x_1^{'}} y^{x_2^{'}} \bmod p$, and sends $z^{'}$ to A.
7. A computes $w^{'} = (z^{'})^{a^{-1}} \bmod p$ and sends $w^{'}$ to B.
8. If $w^{'} = m^{x_1^{'}} \alpha^{x_2^{'}} \bmod p, B$ accepts the signature s and the protocol halts.
9. B computes $c = (w\alpha^{-x_2})^{x_1^{'}} \bmod p$ and $c^{'} = (w^{'}\alpha^{-x_2^{'}})^{x_1} \bmod p$. If $c = c^{'}$, then B concludes that s is a forgery; otherwise, B concludes that the signature is valid and A is attempting to disavow the signature s.

7 Hash Functions and Authentication Codes

One of the most important area of network security is that of message authentication and the related topic of digital signatures. It is almost impossible to handle all cryptographic functions and protocols that have been proposed or implemented for this application in the last twenty years. As such, the purpose of this chapter is to provide a broad overview of the subject and introduce the most important design criteria. The basic approaches are surveyed, including the increasingly important area of secure hash functions which are used in many cryptographic protocols.

7.1 Authentication Functions

message authentication

Message authentication is a procedure to verify that received messages come from the alleged source and have not been altered. It may also verify sequencing and timeliness. Generally, message authentication is a notion for measures which deal with following items [Stallings99a]:

1. **Masquerade**: Insertion of messages into the network from a fraudulent source. This includes the creation of messages by an opponent that are purposed to come from an authorized entity. Also included are fraudulent acknowledgments of message receipt or nonreceipt.
2. **Content modification**: Changes to the contents of a message, including insertion, deletion, transposition, or any other modification.
3. **Sequence modification**: Any modification to a sequence of messages between parties, including insertion, deletion, and reordering.
4. **Timing modification**: Delay or replay of messages. In a connection-oriented application, an entire session or sequence of messages could be a reply of some previous valid session, or individual messages in the sequence could be delayed or replayed. In a connectionless application, an individual message (e.g. datagram) could be delayed or replayed.

A digital signature is an authentication technique that also includes measures to counter repudiation by either source or destination. Generally, a digital signature technique will also counter some or all the attacks listed in items 1-4, with the following item as additional:

- **Repudiation**: Denial of receipt of message by destination or denial of transmission of message by source.

authenticator

Any message authentication mechanism has two levels: At the lower level there must be some sort of function that produces an authenticator a value to be used to authenticate a message. This function is then used as primitive in a higher-level authentication protocol that enables a receiver to verify the authenticity of the message. In this section we are concerned with different types of functions that may be used to produce an authenticator:

- Hash functions,
- Message authentification codes (MAC).

The message encryption can also be seen as an authentication function because the ciphertext of the entire message serves as its authenticator. This is explained in the following.

7.1.1 Message Encryption as Authentication Function

authentication with conventional encryption

When A and B communicate using conventional (symmetric) encryption, we can say that the receiver B is assured that the message which came was generated by A (Fig. 7.1-1). The message must have come from A because A is the only other party that possesses the shared secret key K and therefore the only party with the information necessary to construct ciphertexts that can be decrypted with K. Futhermore, if m is recovered, B knows that none of the bits of m have been altered, because an opponent that does not know K would not know how to alter bits in the ciphertext to produce desired changes in the plaintext. So, we may say that conventional encryption provides authentication as well as confidentiality.

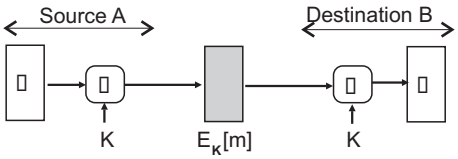


Fig. 7.1-1: Conventional encryption: confidentiality and authentication.

authentication with public-key encryption

The straightforward use of public-key encryption (Fig. 7.1-2 a) provides confidentiality but not authentication. The source A uses the public key $K_{e,B}$ of the destination B to encrypt the message m. Because only B has the corresponding private key $K_{d,B}$ only B can decrypt the message. This scheme provides no authentication because any opponent could also use B's public key to encrypt a message, claiming to be A.

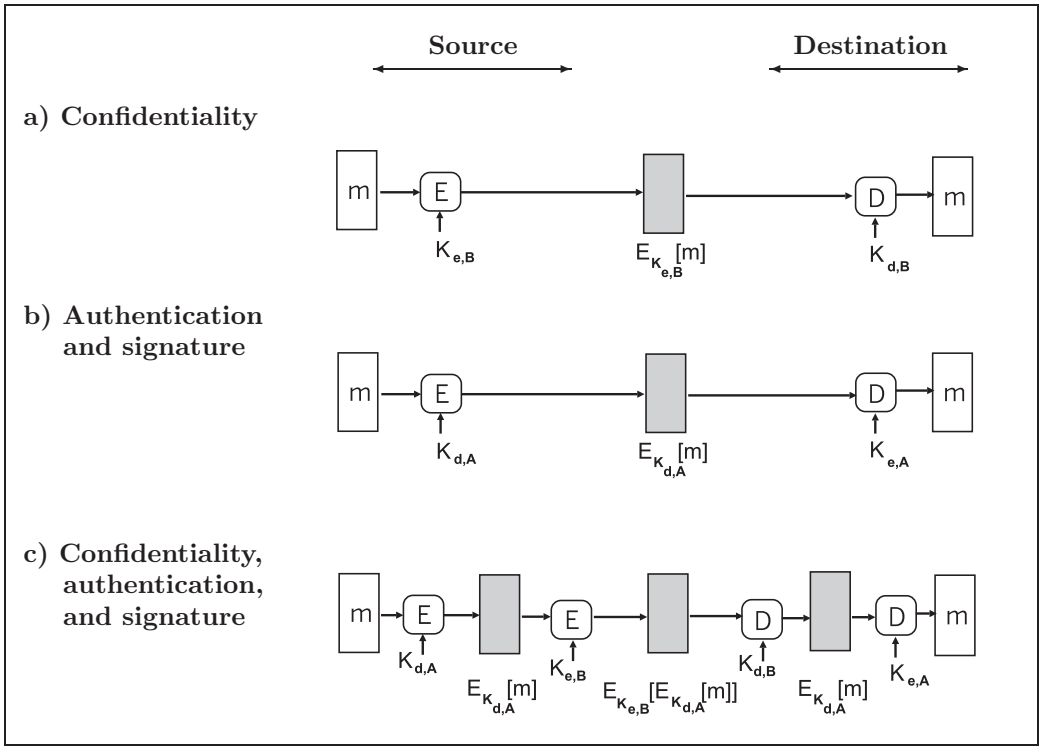


Fig. 7.1-2: Asymmetrical encryption: confidentiality and authentication.

Using the other key part, public-key encryption schemes can however provide authentication: A uses its private key $K_{d,A}$ to encrypt the message, and B uses A's public key $K_{e,A}$ to decrypt it (Fig. 7.1-2 b). This provides a measure for authentication: The message must have come from A because A is the only party that possesses $K_{d,A}$ and therefore the only party with the information necessary to construct ciphertexts that can be decrypted using $K_{e,A}$. This is the principle of constructing digital signatures, although for digital signatures some additional primitives are needed. Note that this scheme does not provide confidentiality. Anyone in possession of A's public key can decrypt the ciphertext.

To provide both confidentiality and authentication, A can encrypt m first by using her private key $K_{d,A}$, which provides message authentication, and then using B's public key $K_{e,B}$, which provides confidentiality (Fig. 7.1-1 c). Note that in this case, the public-key algorithm which is very complex and slow must be executed four times in each communication.

7.1.2 Message Authentication Code (MAC) as Authentication Function

cryptographic checksum

An alternative authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as *cryptographic checksum* or *message authentication code (MAC)*. This technique assumes that two communicating partners A and B share a common secret key K. When A has a message for B, it calculates the MAC as a function of the message m and the key K : $MAC = C(K, m)$, also denoted as $C_K(m)$. This MAC is appended to the message and both are transmitted to B. The process of concatenation is shown in the diagramms by the symbol

message authentication with MAC

$\|$. When B receives the message m and its MAC, it uses the same secret key K and performs the same calculation $C_K(m)$ on the received message m to generate a new MAC. Then, he compares the received MAC with the calculated one (Fig. 7.1-3 a). If the received MAC matches the calculated one, then the receiver B is assured that the message has not been altered, because only A and B know the secret key K. If an attacker O (Oscar) alters the message, but does not alter its MAC, then the MAC calculated from the receiver B differs from the received MAC which was calculated by A. Because the attacker O is assumed not to know the secret key, he can not alter the MAC for it to correspond to the alterations in the message. The receiver B is also assured that the message is from the alleged sender A, because no one else could compute a proper MAC to the message (no one else knows the secret key K).

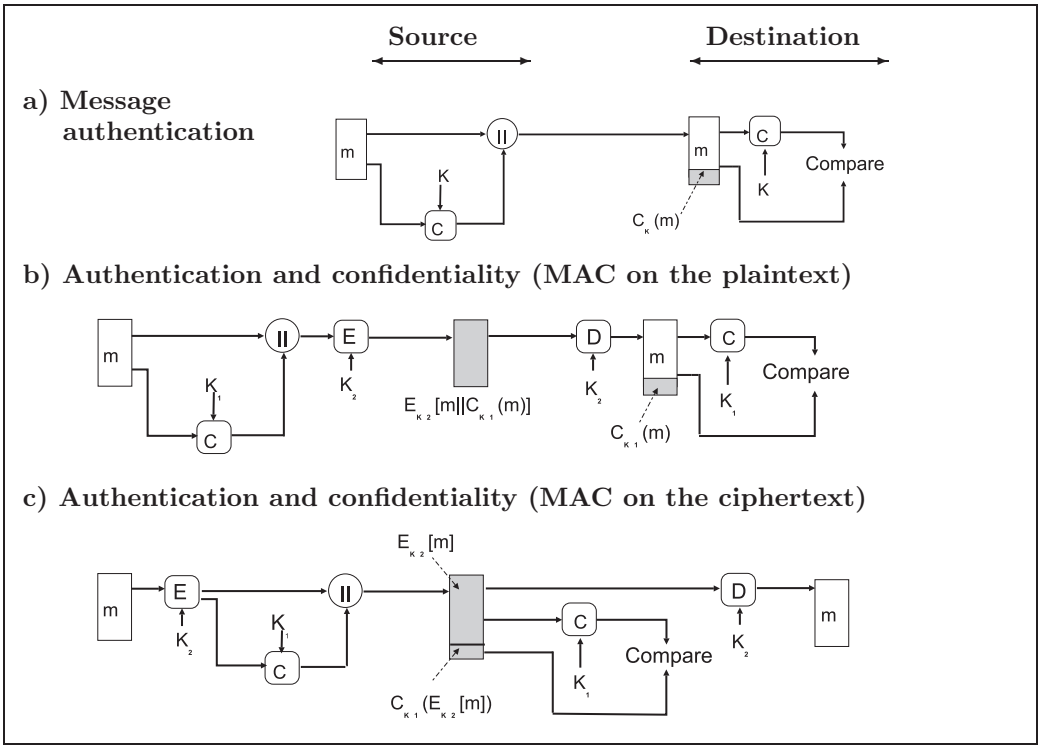


Fig. 7.1-3: Basic uses of MAC.

message authentication and confidentiality with MAC

Note that this process provides authentication but not confidentiality, because the message as a whole is transmitted unencrypted. Confidentiality can be provided by performing message encryption either after or before the MAC algorithm. In both cases, two separate keys K_1 (for MAC) and K_2 (for encryption) are needed, each of which is shared by the sender A and the receiver B. In the first case, the MAC is calculated with the message as input and is then concatenated to the message. The entire block is then encrypted (Fig. 7.1-3 b). In the second case, the message is encrypted first and then the MAC is calculated using the resulting ciphertext and is concatenated to the ciphertext to form the transmitted block (Fig. 7.1-3 c). Typically it is preferable to tie the authentication directly to the plaintext (case b). Note that the MAC does not provide a digital signature because both sender and receiver share the same key.

7.1.3 Hash Function as Authentication Function

hash code = hash value = message digest

A variation of the message authentication code is the one-way hash function. A hash function accepts a variable-size message m as input and produces a fixed-size hash code $h = H(m)$ as output. The hash code is a function of all the bits of the message and provides an error-detection capability: A change of any bit or bits in the message results in a change to the hash code. The hash code is often called *hash value* or *message digest (MD)*. Note that usually the hash function is publicly known.

message authentication with hash codes

In order to provide message authentication a hash code can be used in several ways (Fig. 7.1-4). In figure a) the message and the concatenated hash code is encrypted using conventional encryption, which enables confidentiality. Because only A and B have the secret key, the message must have come from A and has not been altered. The hash code provides redundancy required to achieve authentication. If some application does not require confidentiality, then for authentication only the hash code is encrypted using conventional encryption (fig. b). This can also be done using public-key encryption (fig. c). This provides authentication, but also digital signature because only the sender could have produced the encrypted hash code. If confidentiality as well as digital signature is desired, then the message plus the public-key encrypted hash code can be encrypted using symmetric key (fig. d). Another technique for providing authentication (fig. e) assumes that the two communicating parties A and B share a common secret value S. The hash value of $m||S$ is computed and appended to m. Because B knows S, it can recompute the hash value and verify it. Because the secret value itself is not sent, an opponent cannot modify an intercepted message. On this approach confidentiality can be added by encrypting the value $m||H(m||S)$ before sending it to the other side (fig. e).

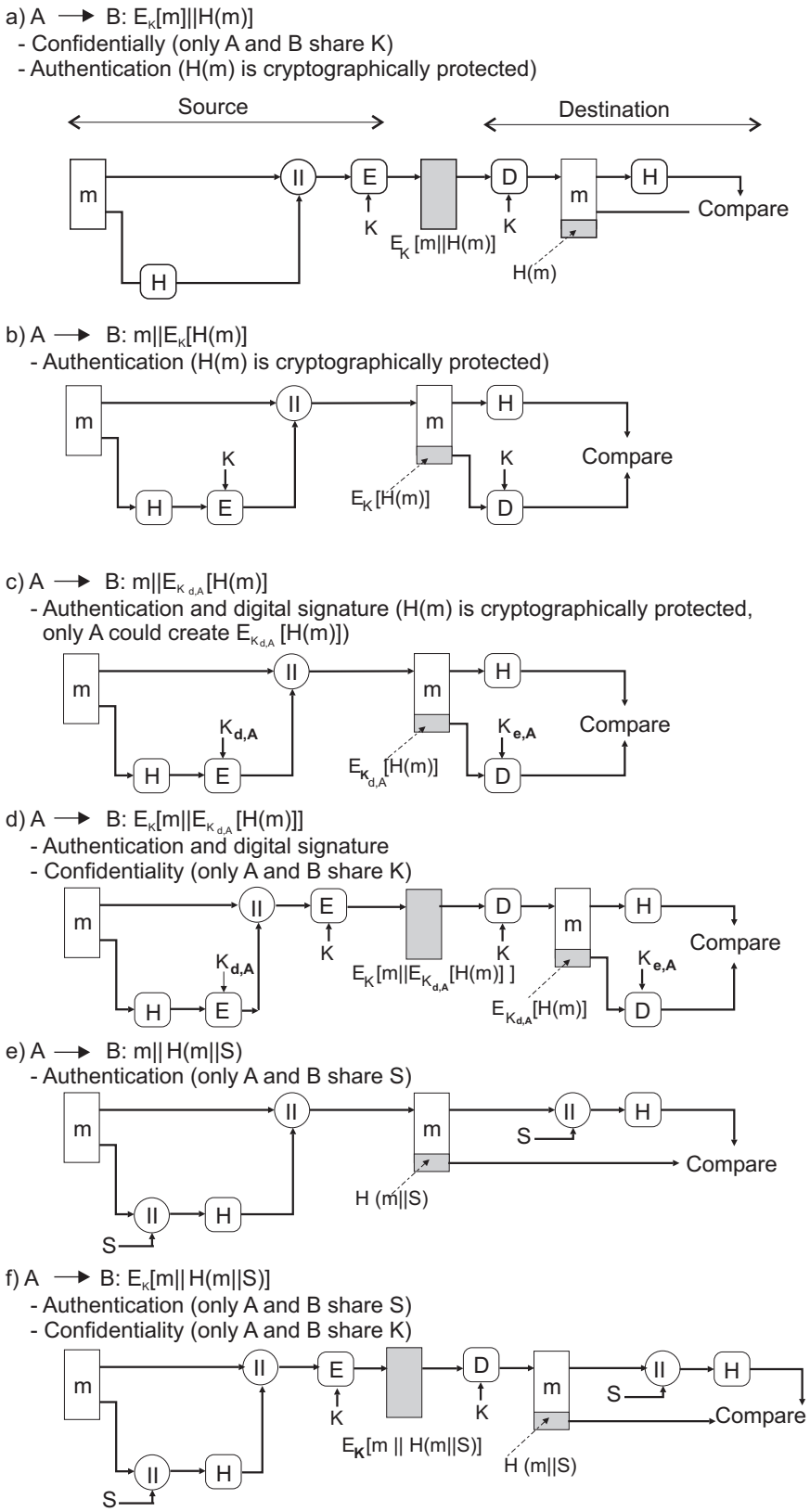


Fig. 7.1-4: Basic uses of hash functions.

7.2 Requirements for Hash Functions

The purpose of a hash function is to produce a ”fingerprint” of a file, message, or other block of data. Formally defined, a hash function is a function H that maps a

definition of hash function

message m of any bit length into a fixed-length *hash value* h (also called *message digest*), which serves as an authenticator:

$$h = H(m).$$

The hash value is appended to the message at the source and both are transmitted together. The receiver authenticates the message by recomputing the hash value and comparing it with the received hash value. Because the hash function itself is not considered to be secret, some additional protection of the hash value is required (Fig. 7.1-4).

collision-free hash function

It will be necessary for the hash function H to satisfy certain properties in order to prevent various frauds. Since hash values are used not only for message authentication, but are also a composing part of every digital signature scheme, we have to be careful that the use of a hash function H does not weaken the security of the signature scheme (because the message digest is signed, not the message). A *collision-free hash function* is one of the basic requirements for achieving security. Intuitively, *collision-free* means that two messages which have the same hash value can not be found. Because the notion "collision free" plays a central role in the security analysis of hash functions, we will define this notion more precisely.

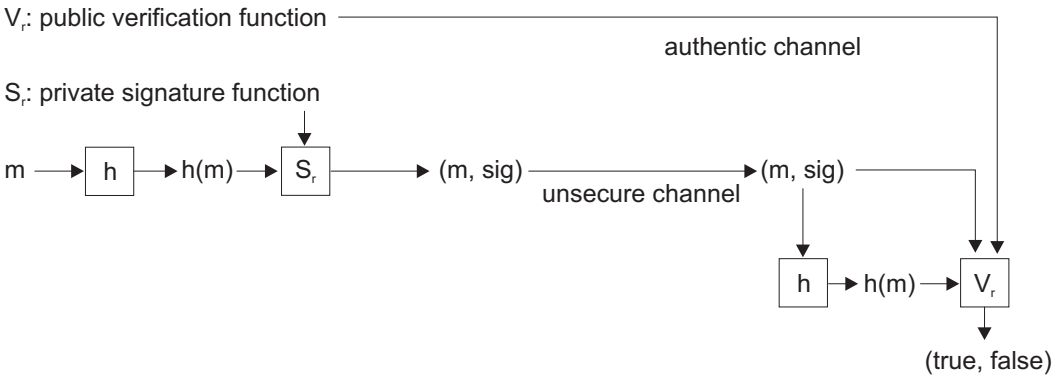


Fig. 7.2-1: Generation and verification of a digital signature with the aid of a hash function for compression.

Observe the general digital signature scheme in Fig. 7.2-1, where Alice sends a message m and its signature sig to Bob. The most obvious type of attack for an opponent Oscar is to start with a valid signed message (m, sig) where $sig = S_K(H(m))$. (The pair (m, sig) could be any message previously signed by Alice). Then he computes $h = H(m)$ and attempts to find message $m' \neq m$ such that $h(m') = h(m)$. If Oscar can do this, (m', sig) would be a valid signed message, i.e. a *forgery*. In order to prevent this type of attack, we require that the hash function H have the following collision-free property:

weak collision resistance

Definition 7.2-1: *Let m be a message. A hash function H is weakly collision-free for m if it is computationally infeasible to find a message $m' \neq m$ such that $H(m') = H(m)$. This property is called weak collision resistance.*

Another possible attack is the following: Oscar first finds any two messages $m \neq m'$ such that $H(m) = H(m')$. Oscar then gives m to Alice and persuades her to sign the message digest $H(m)$, obtaining sig. Then (m', sig) is a valid forgery. This motivates to require a different collision-free property:

strong collision resistance

Definition 7.2-2: *A hash function H is strongly collision-free if and only if it is computationally infeasible to find any two messages m and m' such that $m' \neq m$ and $H(m') = H(m)$. This property is called strong collision resistance.*

Observe that the hash function H is strong collision resistant if and only if it is computationally infeasible to find a message m such that H is not weakly collision-free for m.

It is often possible with certain signature schemes to forge signatures on random message digests h. Therefore, we observe another kind of attack. Suppose an attacker Oscar computes a signature on such a random h, and then he finds a message m such that $h = H(m)$. If he can do this, then (m, sig) is a valid forgery. To prevent this attack, the function H must be *one-way function*, i.e. the function H can not be inverted: $m = H^{-1}(h)$.

one-way property

Definition 7.2-3: *A hash function H is one-way if, given a message digest h, it is computationally infeasible to find a message m such that $H(m) = h$.*

One-way functions are extremely important cryptographic primitives. Probably the best known and simplest use of one-way functions is for passwords. Namely, in a multi-user computer system, instead of storing a table of login passwords, one can store, for each password p_i the value $h_i = H(p_i)$. At the login procedure passwords can be easily checked by computing $H(p_i)$ and comparing it with the stored value, but even the system administrator can not deduce any user's password by examing the stored table.

Existence of one-way functions is a necessary condition for the existance of most known cryptographic primitives, including secure encryption and digital signatures. But the current state of knowledge in complexity theory does not allow to prove the existance of one-way functions. So, we assume their existance.

properties required

We now summarize the properties a hash function H must have in order to be useful for message authentication:

1. H can be applied to a block of data of any size.
2. H produces a fixed-length output.
3. $h = H(m)$ is relatively easy to compute for any given m (both in hardware and software).

4. $m = H^{-1}(h)$ should be infeasible to compute for any given code h (one-way property).
5. Weak collision resistance.
6. Strong collision resistance.

The first three properties are relevant for the practical application of a hash function, and the second three for its security. The sixth property refers to how resistant the hash function is to a class of attacks known as the birthday attack, which we will shortly examine in the next section.

7.3 Size of the Hash Value (Message Digest)

In this section we determine a necessary security condition for hash functions that depends only on the size of the message digest produced by the hash function. This necessary condition results from a simple method of finding collisions which is informally known as the *birthday attack*. This terminology comes from the so-called *birthday paradox*, which says that in a group of 23 random persons, at least two will have the same birthday with a probability larger than 1/2. Before we state the security conditions, we describe the birthday paradox.

7.3.1 The Birthday Paradox

definition of the problem

The problem can be stated as follows: **What is the minimum value of k such that the probability that at least two persons in a group of k people have the same birthday is greater than 0.5?** So, we ask for the probability that *any* pair of persons in the group of k people have the same birthday. We will now try to give an answer to this question. As $P(n, k)$ we define the probability that we have at least one duplicate in k items, where each item is able to take on one of n equally likely (probable) values between 1 and n. Thus, we are looking for the smallest value of k such that $P(365, k) \geq 0,5$.

It is easier first to derive the probability that there are no duplicates, which we denote as $Q(365, k)$. If $k > 365$, then it is impossible for all values to be different. So, we assume $k \leq 365$. Now we consider the number of different ways N for having k values with no duplicates. We may choose any of the 365 values for the first item, any of the remaining 364 numbers for the second item, and so on. Hence, the number of different ways is

$$N = 365 \cdot 364 \cdot 363 \cdot \quad ..... \quad \cdot (365 - k + 1) = \frac{365!}{(365 - k)!} \quad .$$

The total number of possibilities is 365^k (each item can get any of 365 values, with or without duplicates). So, the probability of no duplicates is simply the fraction of the sets of values that have no duplicates out of all possible sets of values:

$$Q(365, k) = \frac{N}{365^k} = \frac{365!/(365 - k)!}{365^k} = \frac{365!}{(365 - k)!\ 365^k}$$

and

$$P(365,k) = 1 - Q(365,k) = 1 - \frac{365!}{(365-k)!\ 365^k}. \qquad 7.3\text{-}1$$

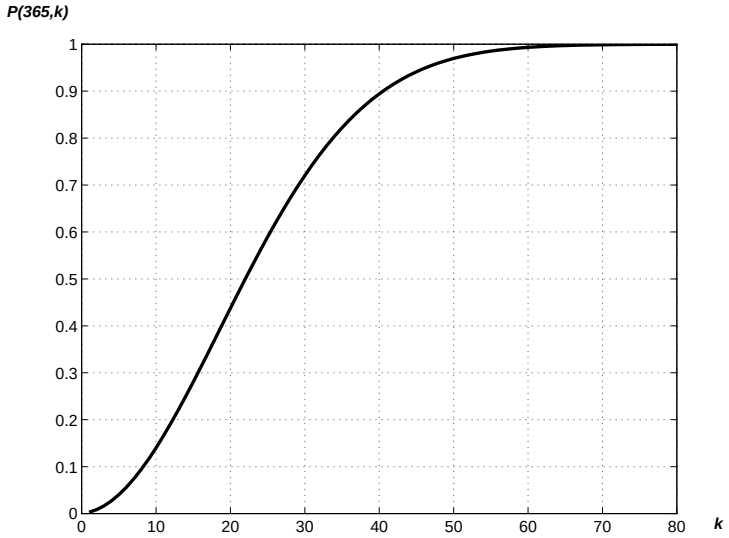


Fig. 7.3-1: The birthday paradox.

This function is plotted in Fig. 7.3-1. The probabilities seem surprisingly large to anyone who has not considered the problem before. We achieve a probability greater than 0.5 already by chosing $k = 23$, i.e. $P(365, 23) = 0.5073$. For $k > 60$, the probability for having a duplicate is almost 1.

general case of duplications

The birthday problem can be generalized to the following general problem of duplications: Given a random integer variable with uniform distribution between 1 and n and a selection of k instances ($k \leq n$) of the random variable, what is the probability $P(n,k)$ that there is at least one duplicate? The birthday problem is just a special case with $n = 365$. By the same reasoning as before, we can generalise the Eq. 7.3-1:

$$P(n,k) = 1 - \frac{n!}{(n-k)!\ n^k}. \qquad 7.3\text{-}2$$

In order to simplify Eq. 7.3-2, we use the well known inequality

$$(1-x) \leq e^{-x} \quad \text{for all} \ \ x \geq 0.$$

Then we can rewrite the Eq. 7.3-2 as:

$$\begin{aligned} P(n,k) &= 1 - \frac{n!}{(n-k)!\, n^k} \;=\; 1 - \frac{n\cdot(n-1)\cdot .....\cdot(n-k+1)}{n^k} \\ &= 1 - [\frac{n-1}{n}\cdot\frac{n-2}{n}\cdot\ .....\ \cdot\frac{n-k+1}{n}] \\ &= 1 - [(1-\frac{1}{n})\cdot(1-\frac{2}{n})\cdot\ .....\ \cdot(1-\frac{k-1}{n})] \\ &> 1 - [(e^{-1/n})\cdot(e^{-2/n})\cdot\ .....\ \cdot(e^{-(k-1)/n})] \\ &> 1 - e^{-[\frac{1}{n}+\frac{2}{n}+\ .....\ +\frac{k-1}{n}]} = 1 - e^{-\frac{1}{n}\cdot(1+2+....+(k-1))} \\ &> 1 - e^{-\frac{1}{n}\cdot\frac{k(k-1)}{2}}. \end{aligned}$$

Finally, we have the simplified relation for the probability $P(n,k)$:

$$P(n,k) > 1 - e^{-\frac{k(k-1)}{2n}}. \qquad 7.3\text{-}3$$

50 % probability for duplication

Using Eq. 7.3-3 we can find for which value of k we have $P(n,k) > 0,5$:

$$\begin{aligned} 1/2 &= 1 - e^{-\frac{k(k-1)}{2n}} \\ 1/2 &= e^{-\frac{k(k-1)}{2n}} = \frac{1}{e^{\frac{k(k-1)}{2n}}} \\ 2 &= e^{\frac{k(k-1)}{2n}} \quad \Rightarrow \quad \ln(2) = \frac{k(k-1)}{2n}. \end{aligned}$$

For large k we can replace $k\cdot(k-1)$ by k^2 and we get $\ln(2) = k^2/2n$, i.e.

$$k = \sqrt{2\cdot\ln(2)\cdot n} = 1,18\sqrt{n} \approx \sqrt{n}. \qquad 7.3\text{-}4$$

We can prove that Eq. 7.3-3 gives very good approximation: For $n = 365$ we get $k = 1,18\sqrt{365} = 22.54$, which is very close to the correct answer 23.

7.3.2 Lower Bound on the Sizes of Message Digest

The result of the Eq. 7.3-4 says that hashing just over $\sqrt{n}$ random values yields a collision with probability of 50%. The birthday attack imposes a lower bound on the size of message digests. Suppose we have a hash function H with 2^m possible outputs (i.e. an m-bit output). If H is applied to k random inputs, then the value of k so that the probability of at least one duplicate (i.e. $H(x) = H(y)$ for some inputs x, y) is 1/2 can be computed as:

$$k = \sqrt{2^m} \;=\; 2^{m/2}.$$

recommendable size for message digest

A 40-bit message would be very insecure, since a collision could be found with the probability of 50% with $\sqrt{2^{40}} \;=\; 2^{20}$ (about a million) random hashes. It is usually suggested that the minimum acceptable size of a message digest is 128 bits. In this case the birthday attack will require over 2^{64} hashes. The choice of a 160-bit message digest for use in the digital signature standard (DSS) was motivated by these considerations.

7.4 Construction and Classification of Hash Functions

compression function

Hash functions are usually implemented as a sequence of similar compression steps (iterations) through which a message m, is processed block-wise via a *compression function* to a hash value $h(m)$, as illustrated in Fig. 7.4-1. An input message m of arbitrary finite length is divided into fixed-length n-bit blocks m_i. This preprocess typically involves appending extra bits (padding) as necessary to attain an overall bit length which is a multiple of the block length n, and often includes for security reasons a block indicating the bit length of the unpadded input. Each block m_i then serves as an input to an internal fixed-size compression function, which computes a new intermedia result of bit length k (k is fixed), as a function of the previous k-bit intermediate result and the next input block m_i. If h_i denotes the partial result after the i^{th} iteration, the general process for the input $m = m_1 m_2 .....m_b$ can be represented as:

general model

$$h_0 = IV; \quad h_i = f(h_{i-1}, m_i) \text{ for } 1 \leq i \leq b\,; \quad h(m) = g(h_b)\,. \qquad 7.4\text{-}1$$

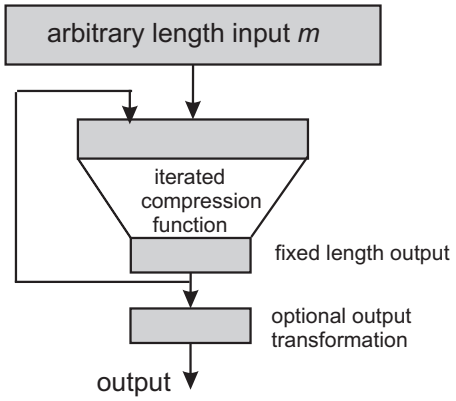


Fig. 7.4-1: General model for an iterated hash function.

classification

Regarding the design of the compression function, the preprocessing and the output transformation, we distinguish between four categories of iterated hash functions:

1. Hash functions based on symmetric block ciphers,
2. Hash functions based on modular arithmetic,
3. Dedicated hash functions, and
4. Provable secure hash functions.

Hash functions based on symmetric block ciphers make it possible to use cryptographic techniques which are already implemented and they use the know how for designing new block ciphers that already exist. The efficiency of a hash function based on modular arithmetic (e.g. asymmetric cipher) would usually not be acceptable. The third group contains dedicated designs for hash functions. These techniques were developed for a more efficient software implementation. Dedicated hash functions have become more and more important in recent years. Finally the category of hash functions must be mentioned whose security is provable under certain assumptions. However, the constructions known so far do not have any practical meaning due to their inefficiency. In the following we concentrate on hash functions which are based on symmetric ciphers and dedicated hash functions.

7.4.1 Hash Functions Based on Block Ciphers

motivation

Motivation for constructing hash functions from block ciphers is that if an efficient implementation of a block cipher is already available (either in hardware or in software), then using it as hash function may provide additional functionality almost without any additional cost. The idea is that if the block algorithm is secure, then the one-way hash function will also be secure. The block cipher is iteratively used as an internal compression function.

security

If (n, r) denotes a block cipher defining an invertible function from n-bit plaintext to n-bit ciphertext using an r-bit key, then the hash functions constructed from this block cipher are divided into those producing single-length (n-bit) and double-length ($2n$-bit) hash values. The motivation for double-length hash functions is that many n-bit block ciphers exist of size approximately $n = 64$, and single-length hash-codes of this size are not collision resistant. In the simplest case, the size of the key used in such hash functions is approximately the same as the block length of the cipher (i.e. n bits). In other cases, hash functions use larger (e.g. double-length) keys. One useful measure for hash functions based on block ciphers is the *hash rate*, or the number of n-bit message blocks processed per encryption. The higher the hash rate, the faster the algorithm.

hash rate

construction

Suppose that using the encryption function $E_K(m)$ we want to produce the hash value h of the message m which can be subdivided into b subblocks m_i which are processed individually and h_0 is some random initial value IV. The general scheme is as follows (see Fig. 7.4-2):

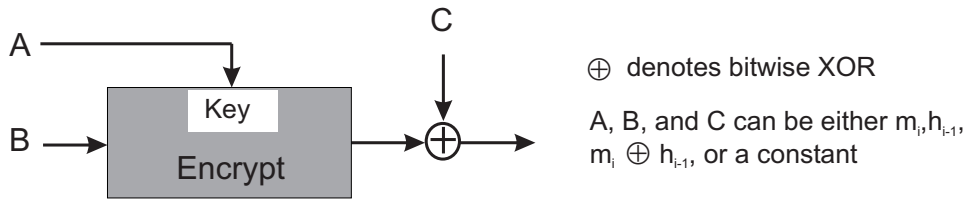


Fig. 7.4-2: General hash function based on symmetric block cipher.

$$h_0 = IV, \quad \text{where } IV \text{ is a random initial value}$$
$$h_i = E_A(B) \oplus C \quad \text{(for all blocks } i\text{)}$$
$$h = h_b$$

where A, B and C can be either $m_i, h_{i-1}, (m_i \oplus h_{i-1})$, or a constant. The three different variables can take on one of four possible values, so that there are 64 total schemes of this type. 15 are trivially weak because the result does not depend on one of the inputs. 37 are insecure for more subtle reasons. Bellow the remaining 12 secure schemes are listed:

1. $h_i = E_{h_{i-1}}(m_i) \oplus m_i$
2. $h_i = E_{h_{i-1}}(m_i \oplus h_{i-1}) \oplus m_i \oplus h_{i-1}$
3. $h_i = E_{h_{i-1}}(m_i) \oplus h_{i-1} \oplus m_i$
4. $h_i = E_{h_{i-1}}(m_i) \oplus m_i$
5. $h_i = E_{m_i}(h_{i-1}) \oplus h_{i-1}$
6. $h_i = E_{m_i}(m_i \oplus h_{i-1}) \oplus m_i \oplus h_{i-1}$
7. $h_i = E_{m_i}(h_{i-1}) \oplus m_i \oplus h_{i-1}$
8. $h_i = E_{m_i}(m_i \oplus h_{i-1}) \oplus h_{i-1}$
9. $h_i = E_{m_i \oplus h_i}(m_i) \oplus m_i$
10. $h_i = E_{m_i \oplus h_i}(h_{i-1}) \oplus h_{i-1}$
11. $h_i = E_{m_i \oplus h_i}(m_i) \oplus h_{i-1}$
12. $h_i = E_{m_i \oplus h_i}(h_{i-1}) \oplus m_i.$

The first scheme was described in 1985 by S.M. Matyas, C.H. Mayer, and J. Oseas from IBM and is also known as Matyas-Mayer-Oseas hash function. The second scheme is known as N-Hash (described in 1990 by Miyaguchy, Ohta, and Iwata) and was proposed as an ISO standard. The fifth scheme was proposed by Carl Meyer, but in the literature it is commonly called Davies-Meyer hash function. The first, second, third, fourth, ninth, and eleventh schemes have a hash rate of 1 i.e. the key length equals the block length. The others have a rate of k/n, where k is the key length.

Davis-Meyer Hash Function

Davis-Meyer hash function

Given a message m which can be divided into b blocks m_i, its *Davis-Mayer hash* value h can be computed with iterations as shown in Fig. 7.4-3 [Winternitz84]. The hash value has the same length as the block value.

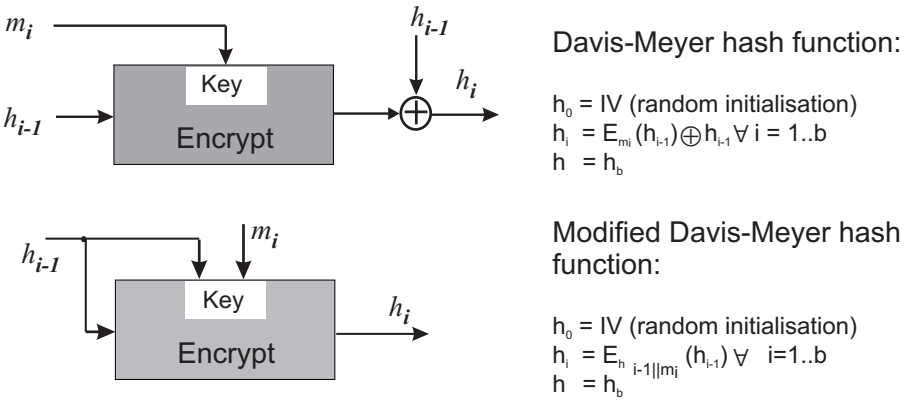


Fig. 7.4-3: Davis-Meyer hash functions.

Modified Davis-Meyer hash function

Lai and Massey modified the Davies-Meyer technique to work with the IDEA (International Data Encryption Standard) cipher [Lai92]. IDEA has 64-bit block size and 128-bit key size. This function hashes the message in blocks of 64 bits and produces 64-bit hash values (see Fig. 7.4-3). No other attack on this scheme is known than brute force. There are also other modifications of the Meyer hash function, for example parallel Davies-Meyer (hash rate 1 which produces hash values twice the block length), Tandem Davies-Meyer, and Abreast Davies-Meyer (both have hash rate 1/2 and 128-bit hash values).

Matyas-Meyer-Oseas Hash Function

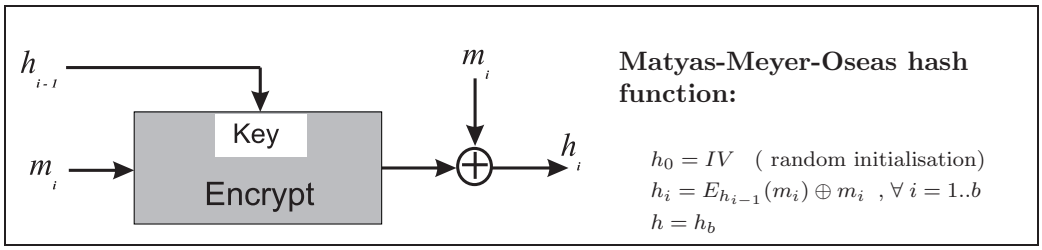


Fig. 7.4-4: Matyas-Meyer-Oseas hash function.

Matyas-Meyer-Oseas hash function

This hash function was presented in 1985 by Matyas, Meyer and Oseas [Matyas85]. It is a strong one-way function and has the hash rate 1. The key length equals the block length. The principle is shown in Fig. 7.4-4. Input of the algorithm is a bit-string (message) m, which is divided into n-bit blocks and padded, if necessary, to complete the last block. The padded message consists of bn-bit blocks: $m_1 m_2 .... m_b$. A constant n-bit initial value IV must be pre-specified.

Meyer-Shilling Hash Function

Meyer-Shilling hash function

This hash function was first developed by C.H. Meyer and M. Shilling at IBM [Meyer88] and is sometimes also known as MDC-2. It has a hash rate of $1/2$ and produces a hash value twice the length of the block size. It is shown in Fig. 7.4-5. It has two hash values h_i and g_i which are initialized with different random initial values h_0 and g_0. The algorithm is patented and is secure with respect to current computing power. The specifications use DES as block function, although every encryption algorithm can be used.

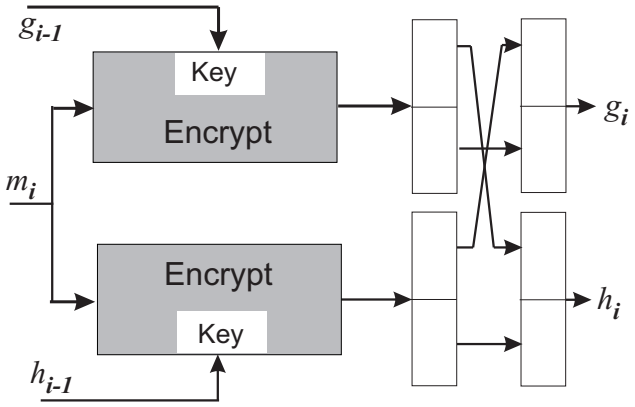


Fig. 7.4-5: Meyer-Shilling hash function

Many other hash functions based on block ciphers have been proposed: Quisquater-Girault, MDC-4, AR, Miyaguchi-Preneel etc. The interested reader can find more information in the original litareture or in [Menezes96a] and [Schneier96a].

7.4.2 Hash Functions Based on Modular Arithmetic

The basic idea of hash functions based on modular arithmetic is to construct an iterated hash function using $\mathrm{mod}\ n$ arithmetic as basis for the compression function. Motivating factors are re-use of existing software or hardware for modular arithmetic and scalability to match required security levels. A significant disadvantage however is that the computation of these functions is very slow.

Exponentiation modulo $n = pq$

exponentiation modulo $n = pq$

If $n = pq$ is the product of two large primes and e is chosen so that $\gcd(e, \varphi(n)) = 1$, then the modular exponentiation function E defined by $E(m) = m^e \bmod n$ is a trapdoor one-way function. If we don't have the trapdoor (additional information needed to invert the function), we can not determine the unknown m when $E(m)$ is given. The only practical method to find m is to use the exponent d with $ed \equiv 1 \bmod \varphi(n)$ and calculate $c^d \bmod n$. Thus, inverting E without the trapdoor appears to be as difficult as factoring n. This is the basis of the MASH (Modular Arithmetic Secure Hash) hash function which has been proposed for inclusion in a draft ISO/IEC standard. The bit length of the modulus n affects the security, and also determines the block size of the messages to be processed and the size of the hash value (e.g. 512-bit modulus yields a 512-bit hash value).

Exponentiation in a finite field

exponentiation in a finite field

For any prime power q the multiplicative group of the finite field $GF(q)$ of order q is cyclic. If q is a large prime power and a is a primitive element of $GF(q)$ (i.e. generator of the cyclic multiplicative group), then the function defined as $E(m) = a^m \bmod q$ is a one-way function. In practice it appears that the only fields that are used for implementations are $GF(p)$ where p is large prime or $GF(2^n)$ for large n. Finding m from $E(m)$ would mean to find the discrete logarithm of $E(m)$ to the base a in $GF(q)$. The computation of $E(m) = a^x \bmod p$ requires at most $2 \log_2 x$ operations while the best general algorithms known for extracting logarithms modulo p require a precomputation of the order of $\exp((\ln p)^{1/2}(\ln \ln p)^{1/2})$ operations. Arithmetic in $GF(2^n)$ for large n can be performed faster than arithmetic over large primes, so there is some reason to prefer the one-way function $E(m) = a^m$ in $GF(2^n)$. In this case n determines the block size of the messages to be processed and the size of the hash value.

Squaring modulo n

squaring modulo n

If $n = pq$ where p and q are two large primes, then the function $E(m) = m^2 \bmod n$ is a trapdoor one-way function, where the trapdoor information is the knowledge of the prime factors. Inverting $E(m)$ (finding m from $E(m)$) means finding a square root of $E(m)$ modulo n. The operation of computing square roots modulo n can be performed efficiently when n is a prime, but is difficult when n is a composite integer whose prime factors are unknown (computationally equivalent to factoring). The integer n determines the block size of the messages to be processed and the size of the hash value.

7.4.3 Dedicated Hash Functions

dedicated hash functions

Dedicated or customized hash functions are those which are specially designed for the explicit purpose of hashing, with optimized performance in mind, and without being constrained to reusing existing system components such as block ciphers or modular arithmetic. Those which have received the greatest attention in practice are based on the MD4 hash function. MD4 is the fourth function in the serie of message digest algorithms and was designed specifically for software implementation on 32-bit machines. Security concerns about MD4 motivated the development of MD5 shortly later, as a more conservative variante of MD4. Other important subsequent variants are SHA-1, RIPEMD-128 and RIPEMD-160.

Message Digest 4 (MD4)

MD4

MD4 is a one-way hash function designed by Ron Rivest [Rivest90]. The algorithm has a message of arbitrary length as input and produces a 128-bit hash value. The original MD-4 design goals were that breaking it should require roughly brute-force effort. MD4 is suitable for high-speed software implementation because it is based on a simple set of bit manipulations on 32-bit operands. After the algorithm was introduced, it was cryptanalytic attacked by Merkle, Biham, and others. Collisions have been found in 2^{20} compression function computations. After this attack, Rivest strengthened the algorithm and the result is MD5.

Message Digest 5 (MD5)

MD5

MD5 was designed as an improved version of MD4 [Rivest92]. Although more complex than MD4, it is similar in design and also produces a 128-bit hash. Since MD5 is one of the most used hash algorithms, we will now describe it in some more detail to illustrate how it works.

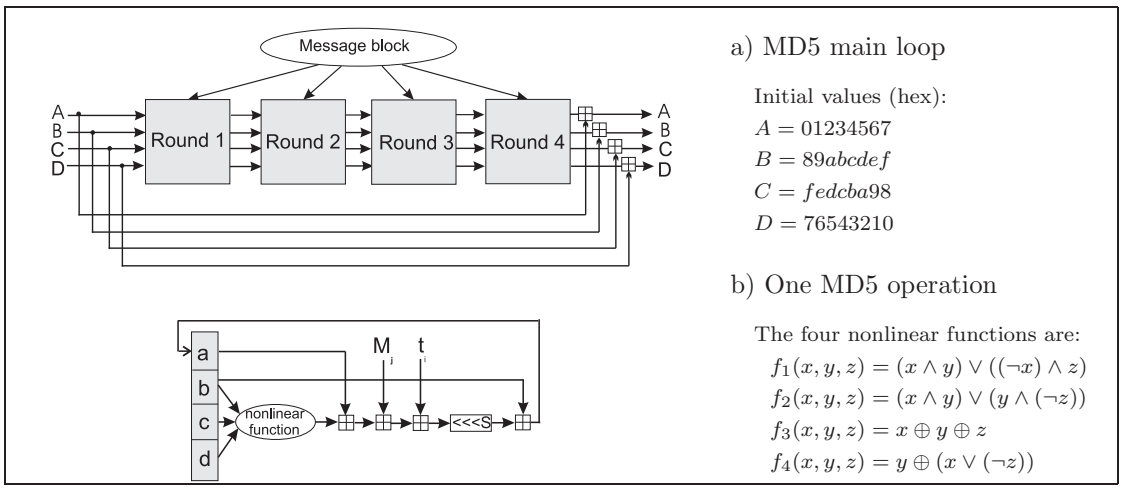


Fig. 7.4-6: MD5 hash function.

After some initial preprocessing, MD5 processes the input text in 512-bit blocks (if necessary the message is padded until its length is a multiple of 512), divided into 16 32-bit subblocks. The output of the algorithm is a set of four 32-bit blocks, which concatenate to form a single 128-bit hash value. The 32-bit variables $A, B, C,$ and D, also called chaining variables, are initialized with fixed values (see Fig. 7.4-6). Then, the main loop of the algorithm begins and the rounds are repeated for every 512-bit message block. The four variables are copied into different variables: a gets A, b gets B, c gets C and d gets D. The main loop has four rounds which are very similar. Each round uses different operation 16 times. Each operation performs nonlinear function on three of a, b, c, and d. Then it adds the result to the fourth variable, a subblock of the text M_j and a constant t_i. Then it rotates that result a variable number of bits to the right and adds the result to one of a, b, c, or d. Finally, the result replaces one of a, b, c, or d.

rounds in MD5

There are four nonlinear functions $f_i(x, y, z)$, one used in each operation (a different one for each round). These functions are designed so that if the corresponding bits of x, y and z are independent and unbiased, then each bit of the result will be independent and unbiased. Let M_j represents the jth subblock of the message ($j = 0, .., 15$), and $<<< s$ represents a left circular shift of s bits[31], then the operations in each round of MD5 (shown in Fig. 7.4-6 under b) can be represented as:

operations in each round

$$
\begin{aligned}
F_1(a, b, c, d, M_j, s, t_i) \text{ denotes } \quad & a = b + ((a + f_1(b, c, d) + M_j + t_i) <<< s) \\
F_2(a, b, c, d, M_j, s, t_i) \text{ denotes } \quad & a = b + ((a + f_2(b, c, d) + M_j + t_i) <<< s) \\
F_3(a, b, c, d, M_j, s, t_i) \text{ denotes } \quad & a = b + ((a + f_3(b, c, d) + M_j + t_i) <<< s) \\
F_4(a, b, c, d, M_j, s, t_i) \text{ denotes } \quad & a = b + ((a + f_4(b, c, d) + M_j + t_i) <<< s).
\end{aligned}
$$

The number of shift bits s in each round, the constants t_i and the order of the subblock M_i have been determined by the designer, and the four rounds (64 steps) are shown in Table 7.4-1.

31 The constants t_i are chosen in the following way: In step i, t_i is the integer part of $2^{32} \cdot abs(\sin(i))$, where i is in radians.

Tab. 7.4-1: *Round operations in MD5.*

Round 1	Round 2	Round 3	Round 4
$F_1(a,b,c,d,M_0,7,d76aa478)$	$F_2(a,b,c,d,M_1,5,f61e2562)$	$F_3(a,b,c,d,M_5,4,fffa3942)$	$F_4(a,b,c,d,M_0,6,f4292244)$
$F_1(d,a,b,c,M_1,12,e8c7b756)$	$F_2(d,a,b,c,M_6,9,c040b340)$	$F_3(d,a,b,c,M_8,11,8771f681)$	$F_4(d,a,b,c,M_7,10,432afff97)$
$F_1(c,d,a,b,M_2,17,242070db)$	$F_2(c,d,a,b,M_{11},14,265e5a51)$	$F_3(c,d,a,b,M_{11},16,6d9d6122)$	$F_4(c,d,a,b,M_{14},15,ab9423a7)$
$F_1(b,c,d,a,M_3,22,c1bdceee)$	$F_2(b,c,d,a,M_0,20,e9b6c7aa)$	$F_3(b,c,d,a,M_{14},23,fde5380c)$	$F_4(b,c,d,a,M_5,21,fc93a039)$
$F_1(a,b,c,d,M_4,7,f57c0faf)$	$F_2(a,b,c,d,M_5,5,d62f105d)$	$F_3(a,b,c,d,M_1,4,a4beea44)$	$F_4(a,b,c,d,M_{12},6,655b59c3)$
$F_1(d,a,b,c,M_5,12,4787c62a)$	$F_2(d,a,b,c,M_{10},9,02441453)$	$F_3(d,a,b,c,M_4,11,4bdecfa9)$	$F_4(d,a,b,c,M_3,10,8f0ccc92)$
$F_1(c,d,a,b,M_6,17,a8304613)$	$F_2(c,d,a,b,M_{15},14,d8a1e681)$	$F_3(c,d,a,b,M_7,16,f6bb4b60)$	$F_4(c,d,a,b,M_{10},15,ffeff47d)$
$F_1(b,c,d,a,M_7,22,fd469501)$	$F_2(b,c,d,a,M_4,20,e7d3fbc8)$	$F_3(b,c,d,a,M_{10},23,bebfbc70)$	$F_4(b,c,d,a,M_1,21,85845dd1)$
$F_1(a,b,c,d,M_8,7,698098d8)$	$F_2(a,b,c,d,M_9,5,21e1cde6)$	$F_3(a,b,c,d,M_{13},4,289b7ec6)$	$F_4(a,b,c,d,M_8,6,6fa87e4f)$
$F_1(d,a,b,c,M_9,12,8b44f7af)$	$F_2(d,a,b,c,M_{14},9,c33707d6)$	$F_3(d,a,b,c,M_0,11,eaa127fa)$	$F_4(d,a,b,c,M_{15},10,fe2ce6e0)$
$F_1(c,d,a,b,M_{10},17,ffff5bb1)$	$F_2(c,d,a,b,M_3,14,f4d50d87)$	$F_3(c,d,a,b,M_3,16,d4ef3085)$	$F_4(c,d,a,b,M_6,15,a3014314)$
$F_1(b,c,d,a,M_{11},22,895cd7be)$	$F_2(b,c,d,a,M_8,20,455a14ed)$	$F_3(b,c,d,a,M_6,23,04881d05)$	$F_4(b,c,d,a,M_{13},21,4e0811a1)$
$F_1(a,b,c,d,M_{12},7,6b901122)$	$F_2(a,b,c,d,M_{13},5,a9e3e905)$	$F_3(a,b,c,d,M_9,4,d9d4d039)$	$F_4(a,b,c,d,M_4,6,f7537e82)$
$F_1(d,a,b,c,M_{13},12,f9987193)$	$F_2(d,a,b,c,M_2,9,fcefa3f8)$	$F_3(d,a,b,c,M_{12},11,e6db99e5)$	$F_4(d,a,b,c,M_{11},10,bd3af235)$
$F_1(c,d,a,b,M_{14},17,a679438e)$	$F_2(c,d,a,b,M_7,14,676f02d9)$	$F_3(c,d,a,b,M_{15},16,1fa27cf8)$	$F_4(c,d,a,b,M_2,15,2ad7d2bb)$
$F_1(b,c,d,a,M_{15},22,49b40821)$	$F_2(b,c,d,a,M_{12},20,8d2a4c8a)$	$F_3(b,c,d,a,M_2,23,c4ac5665)$	$F_4(b,c,d,a,M_9,21,eb86d391)$

As we see, each step has a unique additive constant. Each step adds in the result of the previous step, and this promotes a faster avalanche effect. The left circular shift amounts s in each round have been approximately optimized to yield a faster avalanche effect. The four shifts used in each round are different from the ones used in other rounds. While no collisions for MD5 have yet been found, collisions have been found for the MD5 compression function, but this does not lead to attacks against MD5 in practical applications. Regardless of this weakness, MD5 is widespread used in practice.

avalanche effect

Secure Hash Algorithm (SHA)

The secure hash algorithm (SHA), based on MD4, was proposed by the US National Institute for Standards and Technology (NIST) for certain US federal government applications [NIST92]. SHA is also used in the Digital Signature Standard (DSS). The standard is the Secure Hash Standard (SHS) and SHA is the algorithm used in this standard.

SHA

When a message of any length $< 2^{64}$ bits is input, the SHA produces 160 bit hash value (longer than MD5!). SHA is designed to be computationally infeasible to recover a message corresponding to a given message digest, or to find two different messages which produce the same message digest.

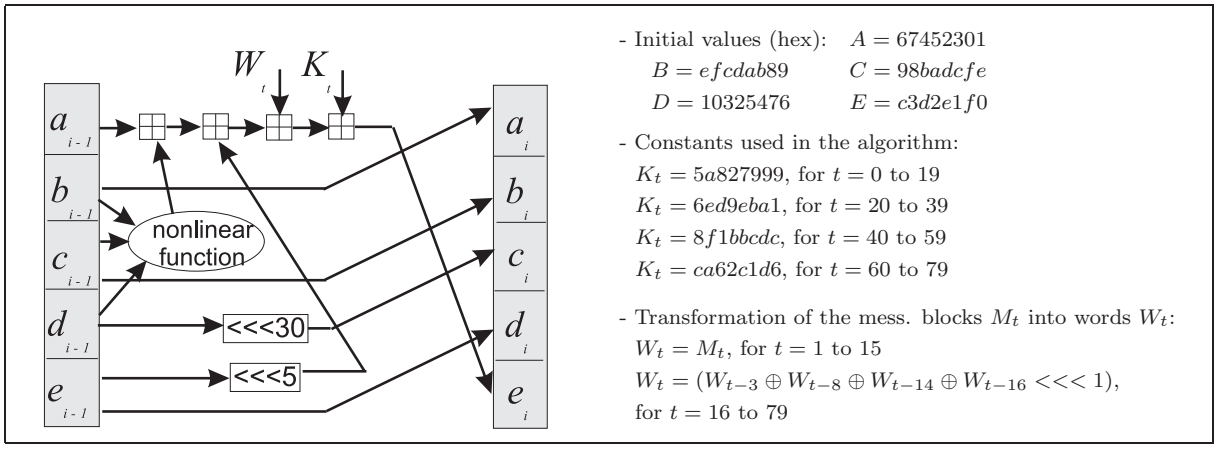


Fig. 7.4-7: One SHA operation.

We now will briefly describe how SHA algorithm works to produce the 160-bit hash value. First the message is padded to make it a multiple of 512 bits. Five 32-bit variables A, B, C, D, and E (needed to produce a 160-bit hash value) are initialized as shown in Fig. 7.4-7. Then the main loop of the algorithm begins. It processes message blocks of 512 bits at a time and processes all blocks of the message. First the five variables are copied into different variables: a gets A, b gets B, and so on. The main loop has four rounds of 20 operations each (MD5 has four rounds of 16 operations). Each operation performs a nonlinear function on three of a, b, c, d, and

main loop of SHA

e, and then does shifting and adding similar to MD5. The nonlinear functions of SHA are:

$$f_t(x,y,z) = (x \wedge y) \vee ((\neg x) \wedge z), \quad \text{for } t = 0 \text{ to } 19$$
$$f_t(x,y,z) = x \oplus y \oplus z, \quad \text{for } t = 20 \text{ to } 39$$
$$f_t(x,y,z) = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z), \quad \text{for } t = 40 \text{ to } 59$$
$$f_t(x,y,z) = x \oplus y \oplus z, \quad \text{for } t = 60 \text{ to } 79.$$

The four constants K_t used in the algorithm are shown in Fig. 7.4-7. The message block is transformed from 16 32-bit words (M_0 to M_{15}) to 80 32-bit words W_0 to W_{79} (see Fig. 7.4-7). If t is the operation number (from 1 to 80) and $<<< s$ represents a left circular shift of s bits, then the main loop is:

```
FOR t = 0 to 79
    TEMP = (a <<< 5) + f_t(b,c,d) + e + W_t + K_t
    e = d
    d = c
    c = b <<< 30
    b = a
    a = TEMP
END
```

After this, a, b, c, d, and e are added to A, B, C, D, and E respectively and the algorithm continues with the next block of data. The final output is the concatenation of A, B, C, D, and E.

comparison of SHA and MD5

SHA has additional expand transformation (M_t to W_t), an extra round, and better avalanche effect. A significant effect of the expansion of 16-word message blocks to 80 word block in the compression function is that any two distinct 16-word blocks yield 80-word values which differ in a larger number of bit positions, significantly expanding the number of bit differences among message words input to the compression function. The redundancy added by this preprocessing evidently adds strength. The designers of SHA did not publish the design criteria for the algorithm. There are no known cryptographic attacks against SHA. Because it produces a 160-bit hash, it is more resistant to brute-force and birthday attacks than the 128-bit hash functions.

Other Dedicated Hash Functions

RIPEMD-128

RIPE-MD: The research and Development in Advanced Communication Technologies in Europe (RACE) programm was launched by the European Community to support pre-normative work in communication standards and technologies. As part of this effort, RACE established the RACE Integrity Primitive Evaluation (RIPE) group, consisting of six leading European cryptography research groups, to put together techniques to meet the anticipated security requirements for Integrated Broadband Communication (IBC). Within this project, the RIPEMD (RIPE Message Digest) hash function was developed, also denoted as RIPEMD-128. The

algorithm is a variation of MD4, designed to resist known cryptanalytic attacks, and produces a 128-bit hash value [RACE92].

RIPEMD-160

Taking into account the knowledge gained in the analysis of MD4, MD5, and RIPEMD-120, the hash algorithm *RIPEMD-160* was developed. The RIPEMD-160 compression function differs from MD4 in the number of words of chaining the variable, the number of rounds, the round functions themselves, the order in which the input words are accessed, and the amounts by which results are rotated. The overall RIPEMD-160 compression function maps 21-word inputs to 5-word outputs. Each input block is processed in parallel by distinct versions of the compression function. The 160-bit outputs of the separate lines are combined to give a single 160-bit output.

Tab. 7.4-2: *Summary of the most used hash functions.*

Hash function	Bitlength	Rounds x Steps per round	Relative Speed
MD4	128	3 x 16	1.00
MD5	128	4 x 16	0.68
RIPEMD-128	128	4 x 16 twice (in parallel)	0.39
SHA-1	160	4 x 20	0.28
RIPEMD-160	160	5 x 16 twice (in parallel)	0.24

HAVAL

HAVAL is a variable-length hash function presented in 1993 by Zheng et al. [Zheng93]. It is based on MD5 and processes messages in blocks of 1024 bits (MD5's blocks are 512 bit), and has eight 32-bit chaining variables (MD5 has four). HAVAL replaces MD5's simple nonlinear functions with highly nonlinear 7-variable functions. Each round uses a single function, but in every step a different permutation is applied to the inputs. It has a new message order and every step uses a different additive constant. It has a variable number of rounds, from three to five, each having 16 steps, and it can produce a hash length of 128, 160, 192, 224, or 256 bits. The variable number of rounds and the variable-length output result in 15 versions of this algorithm.

Snerfu

Snerfu is a one-way hash function designed by Ralph Merkle [Merkle90]. The heart of the algorithm is a function H, which hashes 512-bit message blocks into 128-bit values. The next block is appended to the hash of the previous block and is hashed again. After the last block the first 128 bits are appended to a binary representation of the length of the message and is hashed one final time. The function H is based on a reversible block cipher function that operates on 512-bit blocks and randomizes data in several passes. It is recommended to use Snerfu with at least eight passes. However, in this case the algorithm is significantly slower than either MD5 or SHA.

7.4.4 Provable Secure Hash Functions

provable security

Most hash functions used in the practice are heuristically secure. But a desired feature of cryptographic primitives is the provable security. A cryptographic method is said to be *provably secure* if it can be shown that breaking the method is essentially as difficult as solving a well-known problem, such as integer factorisation or the computation of discrete logarithms. Hash functions can also be designed using some of this hard solvable number theoretic problems as kernel of the compression function. A challenge in this approach is that all possible attacks lead to the ability to solve the referenced problem, which is considered infeasible given current knowledge and an opponent with bounded resources.

Chaum-van Heijst-Pfitzmann (CHP) Hash Function

CHP hash function

One example for a provably secure hash function is the Chaum-van Heijst-Pfitzmann (*CHP*) hash function [Chaum92], based on the discrete logarithm problem. It is defined as follows: Suppose p is a large prime and $q = (p-1)/2$ is also prime. Let α and β be two primitive elements of $\mathbb{Z}_p$. The value $\log_\alpha(\beta)$ is not public, and we assume that it is computationally infeasible to compute this value. The hash function

$$h: \quad \{0, ..., q-1\} \times \{0, ..., q-1\} \quad \rightarrow \quad \mathbb{Z}_p \setminus \{0\}$$

is defined as follows:

$$h(x_1, x_2) = \alpha^{x_1} \beta^{x_2} \bmod p.$$

This hash function will be secure, if a particular discrete logarithm can not be computed. It can be proved[32] that if one collision $h(x_1, x_2) = h(x_3, x_4)$ for $(x_1, x_2) \neq (x_3, x_4)$ of the CHP hash function h can be found, then the discrete logarithm $log_\alpha \beta$ can be computed efficiently. This hash function is not fast enough to be of practical use, but it is conceptually simple and provides a nice example of a hash function that can be proved to be secure under a reasonable computational assumption.

7.5 Message Authentication Codes (MAC)

A message authentication code (MAC), also known as *cryptographic checksum*, is a function C of a variable-length message m and a secret key K shared only by sender and receiver, that produces a fixed-length value MAC that serves as authenticator:

$$MAC = C(K, m) \quad \text{or} \quad MAC = C_K(m).$$

Actually, MAC is a key-dependent one-way hash function whose specific purpose is message authentication. MACs have the same properties as the one-way hash

32 The proof is out of the scope of this book. The proof is explained in [Stinson95b].

functions discussed previously, but they also include a key. This implies the main difference between message authentication with MACs and hash functions: Everyone can prove whether a message m is authentic if the authentication is performed with a hash function, but only someone with an identical key can verify the MAC.

properties

A MAC should have the following properties:

- Ease of computation: For a known function C, given a key K and an input $m, C_K(m)$ is easy to compute.
- Compression: The function C maps the input m of arbitrary length to an output $C_K(m)$ of fixed bit length n.
- Computation resistance: Given text-MAC pairs (m_i, MAC_i) it is computationally infeasible to compute any text-MAC pair (m, MAC) for any new input $m \neq m_i$. The feature of computation resistance implies the property of *key non-recovery* (it must be computationally infeasible to recover K from one or more pairs (m_i, MAC_i) for that K.

In general, the MAC function is a many-to-one function. The domain of the function consists of messages of some arbitrary length, whereas the range consists of all possible MACs and all possible keys. If an n-bit MAC is used, then there are 2^n possible MACs whereas there are N possible messages with $N >> 2^n$. Furthermore, with a k-bit key, there are 2^k possible keys.

MACs are very useful to provide authenticity without secrecy. They can be used to authenticate files between users, or if the files of some user have been altered (perhaps by a virus). A user can compute the MAC of his files and store it in a table, and use it later for comparison. This can not be achieved by using a hash function for message authentication, because, for example, the virus can compute the new hash value of the altered file and replace the table entry (since a key is not used). An easy way to turn a one-way hash function into a MAC is to encrypt the hash value with a symmetric algorithm. Vice versa, any MAC can be turned into a one-way hash function by making the key public.

Requirements for MACs

security of MACs

In assessing the security of a MAC function, we need to consider the types of attacks that may be mounted against it. An objective of an adversary for a MAC algorithm is the following: **Without prior knowledge of the key K, compute a new text-MAC pair $(m, C_K(m))$ for some text $m \neq m_i$, when one or more pairs $(m_i, C_K(m_i))$ are given.** So, the opponent should not be able to construct a new message to match a given MAC, it is however not necessary to reconstruct the key K. Similar to the situation for signature schemes, the following attack scenarios exist for MACs:

- Known-text attack: One or more text-MAC pairs $(m_i, C_K(m_i))$ are available.
- Chosen-text attack: One or more text-MAC pairs $(m_i, C_K(m_i))$ are available for m_i chosen by the adversary.
- Adaptive chosen-text attack: The m_i may be chosen by the adversary as above, and successive choices based on the results of prior queries are allowed.

The function $C_K(m)$ should be uniformly distributed in the sense that for randomly chosen messages m and m', the probability that $C_K(m) = C_K(m')$ is 2^{-n}, where n is the number of bits in the MAC. This requirement deals with the need to thwart a brute-force attack based on chosen plaintext. That is, if we assume that the opponent does not know the key K but does have access to the MAC function and can present messages for MAC generation, then the opponent could try various messages until finding one that matches a given MAC. If the MAC function is uniformely distributed, then a brute-force method would require on an average 2^{n-1} attempts before finding a message that fits a given MAC.

Key recovery of the MAC itself is, of course, the most damaging attack. A brute-force attempt to discover the authentication key requires no less effort than that required to discover the decryption key of the same length.

7.6 Some MAC algorithms

Compared to the large number of hash functions, relatively few MACs have been proposed. Many of these are for historical reasons based on block ciphers. Many iterated MACs can be described as iterated hash functions (see Fig. 7.4-1 and Eq. 7.4-1). In this case, the MAC key is generally part of the output transformation g. It may also be an input to the compression function in the first iteration, and be involved in the compression function f in every stage.

7.6.1 MAC based on block ciphers

CBC-MAC The most commonly used MAC algorithm based on block cipher makes use of the CBC-mode. Input of the algorithm (Fig. 7.6-1) is the message m and the secret MAC-key K. The message m is first padded if necessary, and then divided in b blocks $m_1, ..., m_b$. If E_K denotes encryption using the algorithm E with the key K, then the block H_b is computed as follows:

$$
\begin{aligned}
H_1 &= E_K(m_1) \\
H_i &= E_K(H_{i-1} \oplus m_i) \quad \text{for } \; 2 \le i \le b \; , \\
H &= H_b .
\end{aligned}
$$

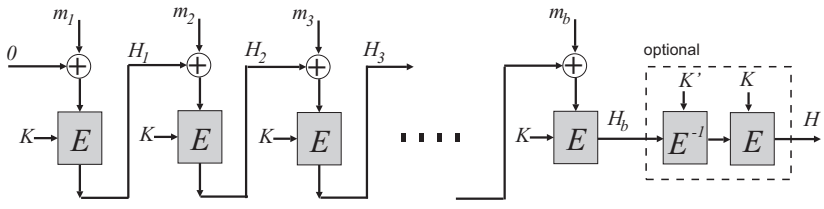


Fig. 7.6-1: CBC-based MAC algorithm.

Optionally, to increase the strength of a MAC, we can use a second key $K' \neq K$ and compute $H'_b = E^{-1}_{K'}(H_b)$ and finally $H = E_K(H'_b)$. This reduces the threat of an exhaustive key search, without impacting the efficiency. RIPE-MAC is a variant of CBC-MAC. Both versions, *RIPE-MAC1* (64-bit MAC) and *RIPE-MAC3* (64-bit MAC), differ in their internal encryption function E being either single DES or two-key triple-DES, requiring a 56- or 112-bit key K.

RIPE-MAC1
RIPE-MAC3

7.6.2 Constructing MACs from Hash Functions

A common suggestion is to construct a MAC from a hash algorithm by simply including the secret key as part of the hash input. But the construction of a MAC from a hash function requires careful analysis, because some security risks arise. For example, it is insecure to use the *secret prefix method* for constructing MACs from hash functions. In this method, the MAC of the message m is $MAC = H(K||m)$, i.e. the key K is appended at the beginning of the message, and then the hash value is computed. It is also insecure to construct a MAC using the key K as the initial value IV of some iterated hash functions, or using the key as a suffix, i.e. $MAC = H(m||K)$ (*secret suffix method*). More secure is to append the key at both the start and the end of the hash computations: $MAC = H(K||p||m||K)$. Here p is a string used to pad K to the length of one block. For example, if H is MD5 and K is 128 bits, p is a 384-bit string. This method is known as the *envelope method with padding*. One can also use the following secure construction also known as *hash-based MAC*: $MAC = H(K||p_1||H(K||p_2||m))$, where p_1 and p_2 are distinct strings of sufficient length to pad K out to the full block length of the compression function.

MACs from hash functions

7.6.3 Dedicated (Customized) MACs

Dedicated MACs are algorithms designed for the specific purpose of message authentication. We will mention here only three of them.

MAA (Message Authenticator Algorithm) is a customized MAC algorithm involving 32-bit operations for all computations (designed for 32-bit machines). It uses a message m of arbitrary length and a 64-bit MAC-key as input and produces 32-bit MAC on m. The main loop consists of two parallel interdependent streams of computations. This algorithm is an ISO standard.

MAA

In **MD5-MAC** the compression function of the MD5 hash function is rearranged to depend on K, implying the secret key be involved in all iterations, thus providing additional protection.

MD5-MAC

The **IBC-hash** MAC algorithm is interesting because it is provably secure and the chance of a successful attack can be quantified. The heart of the algorithm is the iteration $h_i = ((m_i \bmod p) + v) \bmod 2^n$. The secret key is the pair p and v, where p is an n-bit prime and v is a random number less than 2^n. The values m_i are derived by a carefully specified padding procedure. The probabilities of breaking both the

IBC-hash

one-wayness and the collision-resistance can be quantified and the security level can be chosen by changing the parameters.

8 Entity Authentication

8.1 Introduction

Computer systems are now used in almost all aspects of business and commerce and there are various applications where it is necessary for a computer to verify the identity of a person before allowing the person access (e.g. ATMs[33]). Conventional methods of identification based on possession of ID cards or exclusive knowledge – like a PIN[34] or a password – are not altogether reliable. ID cards can be lost, forged, or misplaced; passwords can be forgotten or compromised. Biometric authentication on the contrary is based on the use of physiological or behavioural characteristics. One of the earliest and best-known biometric technologies is fingerprint recognition. New biometric applications include face, hand, finger, iris, retina, signature, and voice recognition.

8.2 Entity Authentication

Entity authentication is a process whereby one party, the *verifier*, is assured of the identity of a second party, the *claimant*, involved in a protocol and that the claimant has actually participated in the protocol. Typically, the verifier is presented with the claimed identity of the claimant (e.g. that the claimant is Alice) and the goal is to verify that the identity of the claimant is indeed Alice.

There are three main categories of criteria that may be used to authenticate a user:

- something the user *knows* e.g. a password or another secret fact;
- something the user *has* e.g. a smart card or token;
- something the user *is* e.g. a fingerprint or retinal pattern.

8.2.1 Authentication based on what the user knows

This method requires the user to provide information or responses to questions and therefore involves asking the user a question, then checking the reply against a reply stored in the system and if there is a match, the user is authenticated. For example, a user logs onto a system and then provides a claimed identity, such as a user name. The system then requests the password which, when entered correctly, serves to verify the user's identity.

Password mechanisms suffer the same problems as any other authenticator using knowledge as proof of identity. If an impostor finds, guesses, or steals a user's password, he has the key to an account, and can use it until the password is changed.

33 Automated teller machines.

34 Personal Identification Number.

8.2.2 Authentication based on what the user has

This method of user authentication, common to most mechanisms outside of those used for computer systems, requires the possession of an object. Such a token, e.g. keys, magnetic cards, must be shown to the authentication mechanism for the user to be granted access. When used as the only means of authentication, the *what-the-user-has* method can be compromised by someone losing a valid token.

To strengthen *what-the-user-has* methods of authentication, *what-the-user-knows* methods are typically added, such as requiring a code to use the token. ATMs are a common example.

8.2.3 Authentication based on what the user is

The third method of user authentication requires the authentication device to measure some physical characteristic of the user being identified. The characteristics could be either a user's physiological traits, e.g. fingerprints, hand geometry, etc. or his behavioural characteristics, e.g. voice and signature. This method of identification is called *biometrics*. The primary advantage is that a biometrics cannot be misplaced or forgotten; it represents *something that the user is*.

8.3 Password-based authentication

Password-based authentication still is the dominant mechanism for identity authentication of computer users, even though passwords are susceptible to attack: Users tend to choose passwords that are easy to remember, which in the case of textual passwords normally implies that they are easy to obtain by searching through a carefully designed dictionary of candidate passwords. In one case study of over 14,000 Unix passwords, almost 25 % of the passwords were found by searching for words from a carefully designed dictionary consisting of only $3 \cdot 10^6$ words [Monrose99a].

8.3.1 Password selection

In traditional authentication schemes a password is a string of characters, usually of length six or more. For selecting good passwords, four techniques are generally known:

- User education: Users are instructed on how to select good passwords. In general, this strategy does not work well because users do not follow these guidelines.
- Password generators: The system chooses a random password for each user. From a theoretical point of view, this is the perfect solution. In practice, however, users will tend to forget passwords, or, worse, they will write them down or store them somewhere.
- Reactive password checking: Special programmes are run periodically by the system administrators to find weak passwords and the corresponding users are informed and asked to change their password.

- Proactive password checking: When a user selects a password, the system checks immediately whether it is acceptable. If the password is obvious (e.g. equal to the login name), or belongs to a given set of dictionaries, it is not accepted and the user is asked to choose a new one.

Proactive password checking is considered to be the best approach.

8.3.2 Attacks

The security of a password system depends on how difficult it is for an attacker to determine a valid password. The basic attacks that authentication protocols need to guard against are *replay attacks* (an attacker records messages and resends them at a later time), *password search*, *password guessing and dictionary attacks*.

Password search

The password length provides an absolute bound to the number of possible passwords in a system. Knowing the composition of allowable passwords, one can compute the number of possible passwords allowed by a particular system by q^l, where q is the size of the character space and l the length of the password.

Small increases in length can dramatically increase the number of possible passwords, which is usually too large to enable an exhaustive search[35].

It would seem that passwords should be as long as possible. However, users must be able to remember their passwords.

Password guessing and dictionary attacks

The space of passwords that are likely to be selected in practice is much smaller than the number of possible passwords, since users tend not to use a wide variety of symbols in their passwords. Making the attacker's job even easier is the tendency to select words or names as passwords. Not only does this practice make little use of character space, it also drastically limits the number of guesses an attacker has to make. Words found in a dictionary, for instance, are poor passwords and knowledge about a user can provide valuable clues to an attacker.

To prevent dictionary attacks, it is now common practice for system administrators to invoke reactive password checkers on an existing collection of user chosen passwords to identify weak password choices, or to use proactive checkers to filter out certain classes of poorly chosen passwords when the user inputs his password for the first time (see Section 8.3.1). Password checkers successfully increase the uncertainty of user's passwords thus making them move beyond the reach of dictionary and exhaustive search attacks, but the resulting passwords are not necessarily easy for users to remember.

35 For detailed information about the number of possible passwords and the time required to test each, see [Menezes96a], page 392.

8.3.3 Salting

Password guessing attacks are still likely to succeed when conducted on a large collection of passwords, even if proactive and reactive checkers are being used. To make dictionary attacks less effective, an n-bit random number (called the *salt*) is appended to a user's password and the concatenated string is then encrypted using a one-way hash function. Both the encrypted string and the salt are stored in a password table. When a user tries to authenticate and enters his password, the salt is retrieved from the password table, prepended to the password, and the concatenated string encrypted. The result is compared to that stored in the password table and authentication succeeds if they match. The difficulty of exhaustive search on any particular user's password is unchanged by salting since the salt is given in cleartext in the password file. However, salting increases the complexity of a dictionary attack against a large set of passwords simultaneously, by requiring any precomputed dictionary of hashed passwords to contain 2^n variations of each trial password. Salting also ensures that users with the same textual passwords will have different encrypted entries in the password table.

8.3.4 One-time passwords

In 1981 Lamport proposed a one-time password authentication scheme based on a one-way function.

A wants to identify itself to B by use of a one-time password:

1. One-time setup:
   a. User A begins with a secret w. Let H be a one-way function.
   b. A constant t is fixed (e.g. $t = 100$ or 1000), defining the number of identifications to be allowed (thereafter, the system is restarted with a new w to avoid replay attacks).
   c. A transfers $w_0 = H^t(w)$, in a manner guaranteeing its authenticity, to the system B. B initializes its counter for A to $i_A = 1$.

2. Protocol messages: The i^{th} identification, $1 \leq i \leq t$, proceeds as follows:

   $$A \rightarrow B : A, i, w_i (= H^{t-i}(w)).$$

   Here $A \rightarrow B$: X denotes A sending the message X to B.

3. Protocol actions: To identify itself for session i, A does the following:
   a. A computes $w_i = H^{t-i}(w)$, and transmits (1) to B.
   b. B checks that $i = i_A$, and that the received password w_i satisfies: $H(w_i) = w_{i-1}$. If both checks succeed, B accepts the password, sets $i_A \leftarrow i_A + 1$, and saves w_i for the next session verification.

Lamport's scheme can withstand an intruder obtaining useful information for breaking the system by tampering with or eavesdropping on the communication links. Lamport's scheme is practical and can be implemented with microcomputers.

However, in order to enforce security, users should change their passwords periodically.

8.4 Challenge-response

The main difficulty in designing secure password mechanisms arises from the fact that the space of passwords from which most users tend to choose is small and much easier to attack by guessing than, for example, random cryptographic keys. Guessing attacks are most effective when a large number of guesses can be made automatically and each guess verified to see whether the guess was correct. A standard way to address this issue is to use salting as described in Section 8.3.3.

More effective solutions to the problem associated with time-invariant passwords are those that involve the use of challenge-response mechanisms. Challenge-response is a common authentication technique whereby an entity is prompted (the challenge) to provide some private information (the response). The time-variant challenge[36] is typically a number chosen randomly and secretly by one entity at the beginning of the protocol.

8.4.1 Challenge-response based on symmetric encryption

In case of challenge-response protocols based on symmetric-key encryption both users (claimant and verifier) are required to share a symmetric key. In systems with a small number of users, each pair of users may share a key a priori. In larger systems there may be need for an on-line trusted third party (TTP) providing session keys to two entities. Therefore, each entity also needs to share a key with the TTP.

In the following, challenge-response based on symmetric-key encryption according to ISO/IEC 9798-2 is specified. Provided that both users A and B share a secret key, the claimant verifies its identity by demonstrating knowledge of the shared key by decrypting a challenge (and possibly additional data) using the key.

Unilateral authentication based on a timestamp

$$A \rightarrow B : E_K(t_A, B^*),$$

where t_A denotes a timestamp generated by A , E_K a symmetric encryption algorithm with a key K shared by A and B, and optional message fields are denoted by an asterisk (*). Upon reception and decryption, B verifies that the timestamp t_A is acceptable, and optionally verifies the received identifier as its own. The identifier B here prevents an adversary from re-using the message immediately on A, in the case that a single bidirectional key K is used.

36 For detailed information about time-variant parameters and random numbers, see [Menezes96a], pp. 397.

Unilateral authentication based on random numbers

Timestamps may be replaced by a random number, at the cost of an additional message:

$$A \leftarrow B: \ r_B,$$
$$A \rightarrow B: \ E_K(r_B, B^*),$$

where r_B denotes a random number generated by B. B decrypts the received message and verifies that the random number matches that sent in message one. Optionally, B checks that the identifier in message two is its own; this prevents a reflection attack in the case of a bidirectional key K. To prevent chosen-text attacks on the encryption scheme E_K, A may (as below) embed an additional random number in message two. Another possibility is to restrict the form of the challenges; the critical requirement is that they are non-repetitive.

Mutual authentication based on random numbers

$$A \leftarrow B: \ r_B,$$
$$A \rightarrow B: \ E_K(r_A, r_B, B^*),$$
$$A \leftarrow B: \ E_K(r_B, r_A),$$

where r_A denotes a random number generated by A. Upon reception of message two, B carries out the checks as above and, in addition, recovers the decrypted r_A for inclusion in message three. Upon decryption of message three, A verifies that both random numbers match those used earlier. The second random number r_A in message 2 serves both as a challenge and to prevent chosen-text attacks.

8.4.2 Challenge-response based on public-key encryption

Public-key encryption may also be used for challenge-response protocols. In this case, a claimant demonstrates knowledge of its private key in two ways:

1. the claimant decrypts a challenge encrypted under its public key, or
2. the claimant digitally signs a challenge.

The public-key pair used for identification should not be used for other purposes, since that could compromise security.

Challenge-response based on public-key decryption

$$A \leftarrow B: \ h(r), B, P_A(r, B),$$
$$A \rightarrow B: \ r,$$

where P_A denotes the public-key encryption algorithm A, and h a one-way hash function. B chooses a random number r, computes the *witness*[37] $x = h(r)$, and computes the challenge $e = P_A(r, B)$. B sends message one to A. A then decrypts e to recover $r'$and B', computes $x' = h(r')$, and terminates if $x' \neq x$ or if B' is unequal to its own identifier B. Otherwise, A sends $r = r'$ to B. Entity authentication of A is successful, if B verifies that the received r agrees with that sent earlier. The use of the witness eliminates chosen-text attacks.

Challenge-response based on digital signatures

In the following, challenge-response based on signatures according to ISO/IEC 9798-3 is specified. If the verifier has the authentic public key of the claimant a priori, certificates may be omitted. Otherwise, it is assumed that the verifier has appropriate information to verify the validity of the public key contained in a received certificate.

Unilateral authentication based on a timestamp

$$A \rightarrow B : cert_A, t_A, B, S_A(t_A, B),$$

where r_A denotes a random number and t_A a timestamp generated by A. S_A denotes A's signature mechanism and $cert_A$ the public-key certificate containing A's signature public key. Upon reception, B verifies that the timestamp is acceptable, the received identifier B is its own, and checks that the signature over these two fields is correct by using A's public key extracted from $cert_A$ after verifying the latter.

Unilateral authentication based on random numbers

Timestamps may be replaced by a random number, at the cost of an additional message:

$$A \leftarrow B: \ r_B,$$
$$A \rightarrow B: \ cert_A, r_A, B, S_A(r_A, r_B, B),$$

B verifies that the cleartext identifier is its own, and verifies that A's signature is valid over the cleartext random number r_A, the same number r_B as sent in message one, and its identifier. The use of the signed r_A prevents chosen-text attacks.

37 x is evidence for the knowledge of r without disclosing it.

Mutual authentication based on random numbers

$$A \leftarrow B: \ r_B,$$
$$A \rightarrow B: \ cert_A, r_A, B, S_A(r_A, r_B, B)),$$
$$A \leftarrow B: \ cert_B, A, S_B(r_B, r_A, A),$$

Processing of messages one and two is as above and message three is processed analogously to message two.

8.5 Zero-knowledge

Challenge response protocol itself uses something similar to zero-knowledge concepts but uses other encryption techniques, like public key techniques, or symmetric-key techniques. As we will see in this chapter, zero-knowledge protocols on the other hand do not rely on digital signatures or public key-encryption, and avoid the use of block ciphers, sequence numbers, and timestamps. They make use of random numbers not only as challenges, but also as commitments to prevent cheating.

A natural way of establishing a person's identity is to ask him to supply a proof of knowledge of a fact that this person is supposed to know. In zero-knowledge (ZK) protocols, the claimant (called a *prover* in the context of zero-knowledge protocols) A demonstrates knowledge of a secret to the verifier B, without revealing any information whatsoever that is of use to B in conveying this knowledge to others.

General structure of zero-knowledge protocols

For a large class of zero-knowledge protocols the procedure is as follows:

$$A \rightarrow B: \ witness,$$
$$A \leftarrow B: \ challenge,$$
$$A \rightarrow B: \ response.$$

The prover claiming to be A selects a random element from a pre-defined set as its secret commitment, and therewith computes an associated (public) witness. This defines a set of questions all of which A claims to be able to answer and only the legitimate party A, with knowledge of A's secret, is truly capable of answering all the questions. The answer to any one question provides no information about A's long-term secret. B's subsequent challenge selects one of these questions. A provides its response, which B verifies for correctness.

The protocols of Feige-Fiat-Shamir, Guillou-Quisquater, and Schnorr provide solutions to the identification problem. Each one has advantages and disadvantages with respect to various performance criteria and for specific applications (see [Menezes96a], p. 416).

8.5.1 Feige-Fiat-Shamir identification protocol

The Feige-Fiat-Shamir identification protocol enables an entity to identify itself by providing knowledge of a secret using a zero-knowledge proof. The protocol reveals no information whatsoever about A's secret identification value(s).

1. Selection of system parameters. After selecting two secret primes p and q each congruent to $3 \, mod \, 4$, the trusted third party T publishes the common modulus $n = pq$ for all users. n has to be computationally infeasible to factor.

2. Selection of per-entity secrets. Each entity A does the following
   a. select k random integers $s_1, \ldots, s_k$ in the range $1 \leq s_i \leq n-1$, and k random bits $b_1, \ldots, b_k$.
   b. compute $v_i = (-1)^{b_i} \cdot (s_i^2)^{-1} \, mod \, n$ for $1 \leq i \leq k$.
   c. A identifies itself by conventional means to T, which then registers A's public key $(v_1, \ldots, v_k; n)$, while only A knows its private key $(s_1, \ldots, s_k)$ and n. Therewith the one-time set-up phase is completed.

3. Protocol messages. The three messages involved in each of the t rounds are
   a. $A \rightarrow B : (x = \pm r^2 \, mod \, n)$
   b. $A \leftarrow B : (e_1, \ldots, e_k), e_i \in \{0, 1\}$
   c. $A \rightarrow B : (y = r \cdot \prod_{e_j=1} s_j \, mod \, n)$.

4. Protocol actions. A identifies itself to B by t executions of the following steps and B accepts the identity only if all t executions are successful.
   a. A chooses a random integer r, $1 \leq r \leq n-1$, and a random bit b. A then computes $x = (-1)^b \cdot r^2 \, mod \, n$ and sends the witness x to B.
   b. B sends the random k-bit vector $(e_1, \ldots, e_k)$, i.e. the challenge, to A.
   c. A computes the response $y = r \cdot \prod_{j=1}^{k} s_j^{e_j} \, mod \, n$ and sends it to B.
   d. B computes $z = y^2 \cdot \prod_{j=1}^{k} v_j^{e_j} \, mod \, n$, and checks that $z = \pm x$ and $z \neq 0$.

8.5.2 Guillou-Quisquater identification protocol

The Guillou-Quisquater (GQ) identification scheme – an extension of the Fiat-Shamir protocol – involves three messages between the claimant A whose identity is to be proven, and the verifier B.

1. Selection of system parameters.
   a. A trusted third party T selects random RSA-like primes p and q yielding a modulus $n = pq$.
   b. T defines a public exponent $v \geq 3$ with $gcd(v, \phi) = 1$ where $\phi = (p-1)(q-1)$, and computes its private exponent $s = v^{-1} \, mod \, \phi$.
   c. System parameters (v, n) are made available for all users.

2. Selection of per-user parameters.

   a. Each entity A is given a unique identity I_A from which $J_A = f(I_A)$, satisfying $1 < J_A < n$, is derived using a known redundancy function f.
   b. T gives the secret $s_a = (J_A)^{-s} \, mod \, n$ to A.

3. Protocol messages. The three messages involved in each of the t (often $t = 1$) rounds are
   a. $A \rightarrow B$: $I_A, x = r^v \, mod \, n$
   b. $A \leftarrow B$: e, $1 \leq e \leq v$
   c. $A \rightarrow B$: $y = r \cdot s_A^e \, mod \, n$.

4. Protocol actions. A identifies itself to B by t executions of the following steps and B accepts the identity only if all t executions are successful.
   a. A selects a secret random integer r, i.e. the commitment, $1 \leq r \leq n-1$, and computes the witness $x = r^v \, mod \, n$.
   b. A sends the pair of integers (I_A, x) to B.
   c. B selects the challenge e (a random integer), $1 \leq e \leq v$ and sends it to A.
   d. A computes the response $y = r \cdot {s_A}^e \, mod \, n$ and sends it to B.
   e. B receives y, constructs J_A from I_A using f, computes $z = {J_A}^e \cdot y^v \, mod \, n$ and accepts A's proof of identity if both $z = x$ and $z \neq 0$.

8.5.3 Schnorr identification protocol

An alternative to the Fiat-Shamir and GQ protocols is the Schnorr identification protocol, whose security is based on the intractability of the discrete logarithm problem. The basic idea is that A proves knowledge of a secret a (without revealing it) in a time-variant manner (depending on a challenge e), identifying A through the association of a with the public key v via A's authenticated certificate.

1. Selection of system parameters.
   a. A suitable prime p is selected such that $p - 1$ is divisible by another prime q.
   b. An element β is chosen, $1 \leq \beta \leq p - 1$, having multiplicative order q.
   c. Each party obtains an authentic copy of the system parameters (p, q, β) and the public key of the trusted third party T, allowing verification of T's signature $S_T(m)$ on messages m.
   d. A parameter t, $2^t < q$, is chosen.

2. Selection of per-user parameters.
   a. Each claimant A is given a unique identity I_A.
   b. A chooses a private key a, $0 \leq a \leq q-1$, and computes $v = \beta^{-a} \, mod \, p$.

c. A identifies itself by conventional means to T, transfers v to T with integrity, and obtains a certificate $cert_A = (I_A, v, S_T(I_A, v))$ from T binding I_A with v.

3. Protocol messages. The three messages involved are
   a. $A \rightarrow B : cert_A, x = \beta^r \, mod \, p$
   b. $A \leftarrow B : e, 1 \leq e \leq 2^t < q$
   c. $A \rightarrow B : y = ae + r \, mod \, q$.

4. Protocol actions. A identifies itself to B as follows
   a. A chooses a random number r, i.e. the commitment, $1 \leq r \leq q - 1$, computes the witness $x = \beta^r \, mod \, p$, and sends protocol message one to B.
   b. B authenticates A's public key v by verifiying T's signature on $cert_A$, and sends a never previously used random challenge e, $1 \leq e \leq 2^t$, to A.
   c. A verifies $1 \leq e \leq 2^t$, and sends the response $y = ae + r \, mod \, q$ to B.
   d. B computes $z = \beta^y v^e \, mod \, p$, and accepts A's identity if $z = x$.

8.6 Biometrics

8.6.1 Introduction

Biometrics is the science of automatically identifying individuals based on their distinctive physiological or behavioural characteristics and unlike PINs or passwords, biometrics cannot be lost, stolen, or overheard. A physiological characteristic is a relatively stable physical feature such as a fingerprint, iris pattern, or retina pattern. A behavioural characteristic, e.g. signature, keystroke pattern, voice, on the other hand, has some physiological basis, but also reflects a person's emotional state (see Fig. 8.6-1).

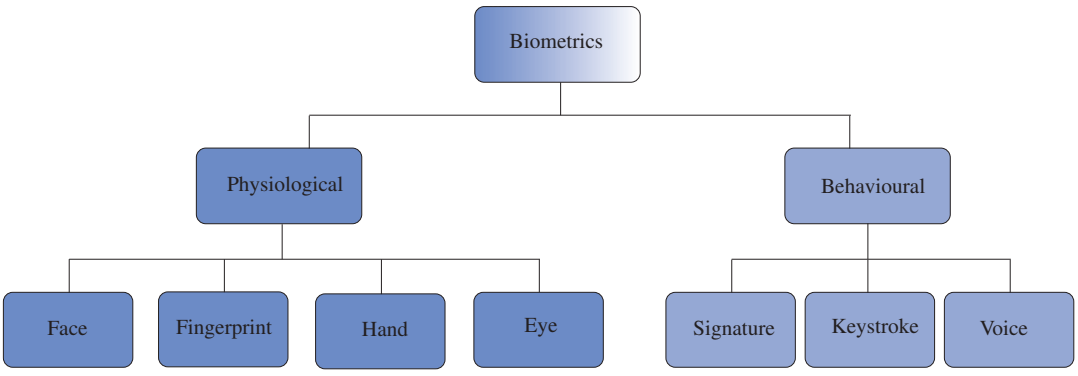


Fig. 8.6-1: Biometrics ecompasses both physiological and behavioural characteristics.

Access management based on biometric techniques can be applied to:

- access to banks and strategic structures,
- user recognition for banking transactions via ATMs,
- user recognition for information applications residing in databases (consultation via remote access, the Internet or a WAN), or
- control of any access to information-technology applications.

Any human physiological or behavioural characteristic could be a biometrics preconditioned if it has the following properties ([Jain99a]):

- universality - every person should have the characteristic,
- uniqueness - no two persons should be the same in terms of the characteristic,
- permanence - the characteristic should be invariant with time, and
- collectability - the characteristic can be measured quantitatively.

8.6.2 Authentication and Identification

Biometric systems can be used for identifying an individual from all those enrolled in the system, by authenticating a claimed identity.

- Identification:
  Identification is a one-to-many comparing process: a biometric sample of an unknown person is presented to the system. The system compares the sample (template) with a database of reference templates of known individuals. On the basis of the comparison, the system then picks the one that matches the characteristics of the unknown individual.
- Authentication:
  Authentication is a one-to-one comparing process: a person presents a biometric sample and a claim (e.g. name, number) that a particular identity belongs to the system. The sample is then compared against a single reference template of a single enrollee whose identity is being claimed. The algorithm either accepts (match) or rejects (non-match) the claim. The reference template does not have to reside in a large database. It can be carried within a smart card or other security device.

Biometric systems that rely on identification are mainly used in law enforcement, forensics, and intelligence. These applications include identifying faces from mug shots, fingerprints, and surveillance images. Authentication is used during point-of-sale transactions, to control access to computers or to secure buildings.

8.6.3 Architecture and functionality

A biometric system is an automated system capable of

- capturing a biometric sample from a user,
- extracting biometric data from that sample,
- comparing the biometric data with that contained in one or more reference templates,

- deciding how well they match, and
- indicating whether or not an identification or authentication of identity has been achieved.

Logically, a biometric system can be divided into two stages: the enrollment module and the identification module (Fig. 8.6-2).

In both processes, the first stage is for the user to present his biometric feature (e.g. fingerprint) to the system, and a biometric sample (e.g. fingerprint image) is captured. The second stage converts this biometric sample to biometric data (e.g. minutiae[38] coordinates) for matching. The final stage of the enrollment process is to form the reference template for the individual; in the case of authentication or identification the final stage is to compare the biometric data with the reference template.

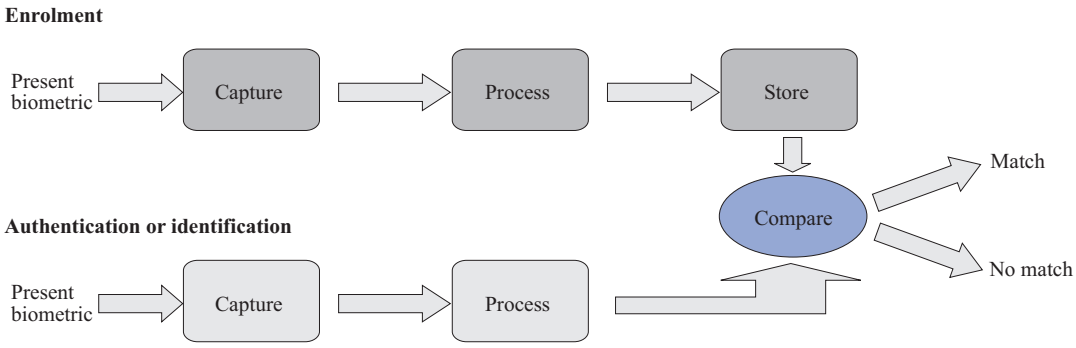


Fig. 8.6-2: Basic biometric process.

- Enrollment produces a biometric data record for storage and future matching operations through a series of capture/process steps. **enrollment**
- Authentication compares (on a one-to-one basis) newly captured resp. processed biometric samples against previously enrolled biometric samples from a known individual. This step answers the question "Is this person who he claims to be?". **authentication**
- Identification compares (on a one-to-many basis) newly captured resp. processed biometric samples against a database of previously enrolled samples. This answers the question "Who is this person?". **identification**
- Capture obtains the raw biometric data (like a bitmapped finger image) from a biometric capture device (like a finger scanner). **biometric data**
- Process extracts a unique biometric identifier (like fingerprint details) from the raw biometric data. **biometric identifier**

38 Small details found in finger images such as ridge endings or bifurcations.

8.6.4 Error statistics

The most commonly discussed measure of a biometric's performance is its identifying power. Biometric systems can never provide an absolutely certain identification because analysis of physiological and behavioural characteristics has a natural range of variation. The presentation of a correct/incorrect password in a password-based authentication system always correctly results in acceptance/denial of an identity authentication claim. On the other hand, even if a legitimate biometric identifier is presented to a biometric-based authentication system, the correct authentication may not be guaranteed due to sensor noise and limitations of feature extractor and matcher (*false rejection*). Similarly, there is a possibility that an impostor will be incorrectly accepted by a biometric-based authentication system (*false acceptance*).

The corresponding error rates are called *false rejection rate* (FRR) and *false acceptance rate* (FAR). They determine the quality of a biometric system (see Fig. 8.6-3).

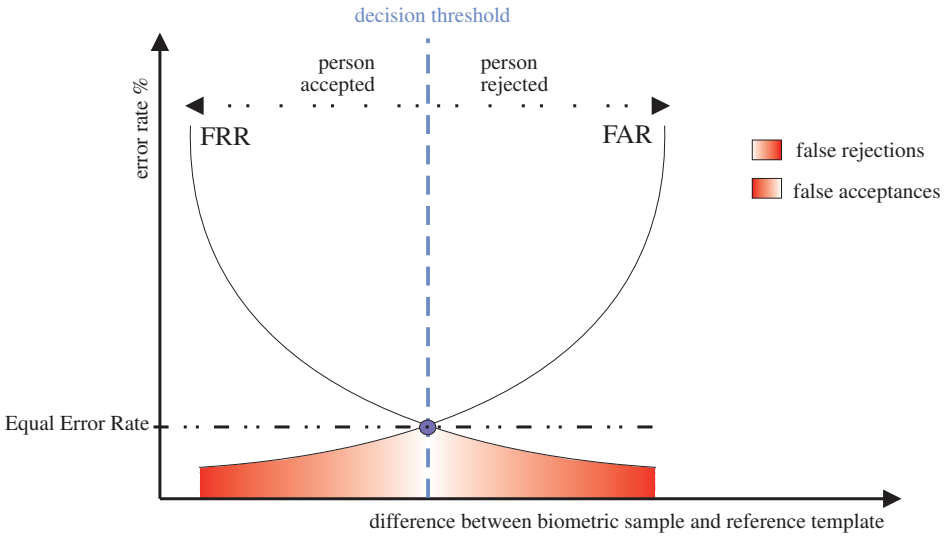


Fig. 8.6-3: False rejections and false acceptances.

False Acceptance Rate (FAR)

FAR FAR is the probability that a biometric system will incorrectly identify an individual or will fail to reject an impostor. It is stated as follows:

$$FAR = NFA/NIIA$$

or

$$FAR = NFA/NIVA$$

where FAR is the false acceptance rate, NFA is the number of false acceptances, NIIA is the number of impostor identification attempts, and NIVA is the number of impostor verification attempts.

False Rejection Rate (FRR)

FRR is the probability that a biometric system will fail to identify an enrollee, or verify the legitimate claimed identity of an enrollee. It is stated as follows: **FRR**

$$FRR = NFR/NEIA$$

or

$$FRR = NFR/NEVA$$

where FRR is the false rejection rate, NFR is the number of false rejections, NEIA is the number of enrollee identification attempts, and NEVA is the number of enrollee verification attempts.

In a perfect biometric system, both error rates would be zero. Unfortunately, biometric systems are not perfect, such systems operate between the two extremes. For most applications, the system parameters are set to achieve a desired false acceptance rate, which results in a corresponding false rejection rate. The parameter setting depends on the application. For a bank's ATM, where the overriding concern may be to avoid irritating legitimate customers, the false-reject rate will be set low at the expense of the false-alarm rate. On the other hand, for systems that provide access to a secure area, the false-alarm rate will be the overriding concern.

Because system parameters can be adjusted to achieve different false acceptance rates, it often becomes difficult to compare systems that provide performance measurements based on different false acceptance rates.

8.6.5 Attacks

There are several possible sources of attacks on a biometric system, which are described below (see Fig. 8.6-4).

1. Fake biometric at the sensor: In this mode of attack, a possible reproduction of the biometric being used will be presented to the system. For example, a fake finger or a copy of a signature.
2. Resubmission of old digitally stored biometrics signal: In this mode of attack, an old recorded signal is replayed into the system bypassing the sensor. For example, presentation of an old copy of fingerprint image or recorded audio signal of a speaker.
3. Override feature extract: The feature extractor could be attacked with a Trojan horse to change it to produce feature sets of choice.
4. Tampering with the feature representation: After the features have been extracted from the input signal, they are replaced with a synthesized feature set of choice assuming the representation is known. Often the two stages of feature extraction and matcher are inseparable and this mode of attack is extremely difficult. However, if minutiae are transmitted to a remote matcher (e.g. over the Internet) than this threat is very real.

5. Override matcher: The matcher is attacked to produce the desired result.
6. Tampering with stored templates: The database of enrolled templates is available locally or remotely. This database can also be distributed over several servers. The stored template attacker tries to modify one or more templates in the database which could result in at least denial of service for the corrupted template.
7. Channel attack between stored templates and the matcher: The templates from the stored database are sent to the matcher through a channel which could be attacked to change the contents of the templates before they reach the matcher.
8. Decision override: If the final result can be overridden with the choice of result from the attacker, the final outcome is very dangerous. Even if the actual pattern recognition system has an excellent performance characteristic, it can be rendered useless by a simple exercise of overriding the result.

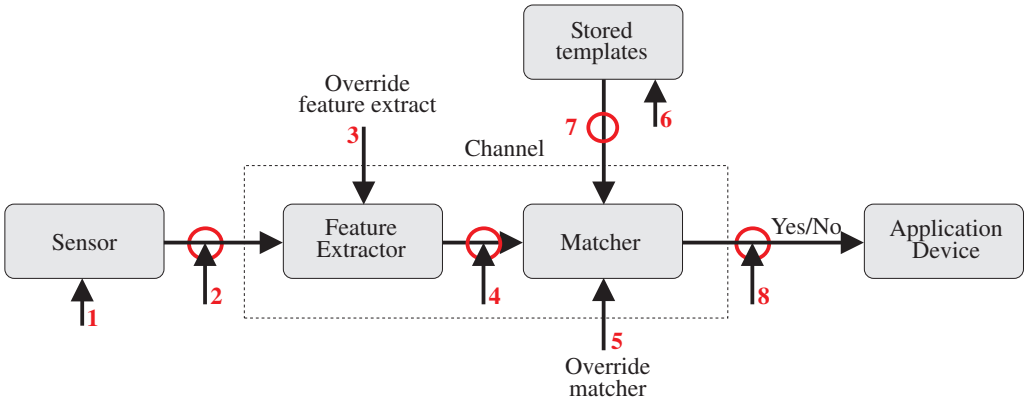


Fig. 8.6-4: False rejections and false acceptances.

There are several techniques to prevent attacks at various points. For instance, sensing finger conductivity or pulse can stop simple attacks at point 1. Encrypted communication channels can eliminate at least remote attacks at point 4. The simplest way to stop attacks at points 5, 6 and 7 is to have the matcher and the database reside in a secure location. Of course even this cannot prevent attacks in which there is collusion. Cryptography again can help at point 8.

9 Key Management Techniques

9.1 Introduction

cryptographic keys

Security services based on cryptographic mechanisms often assume cryptographic keys to be distributed to the parties which are involved in communication before securing the communication. The secure management of these keys is one of the most critical elements when integrating cryptographic functions into a system. Even the most ingenious security concept will be ineffective if the key management is weak. Key management includes the

- generation,
- certification and authentication,
- establishment and distribution,
- escrow/recovery,
- storage, update and destruction,

of keying material. Key management techniques depend on the underlying cryptographic techniques, the intended use of the keys and the implied security policy. The appropriate protection of keys is subject to a number of factors, such as the type of application for which the keys are used, the threats they face, or the different states the keys may assume. Primarily, depending upon their type, keys have to be protected against disclosure, modification, destruction and replay.

9.2 Key Generation

data encryption keys
key encryption keys
master keys

There are several possibilities to generate keys used in cryptographic systems. The main criteria for a chosen method is the key lifetime and its application. Cryptographic keys are often differentiated in data encryption keys (DEK), key encryption keys (KEK) and master keys. Keys which are usually used to secure communication should have a limited lifetime, whereas data encryption keys may be used for a longer period. Master keys, which may have a long lifetime, are exclusively used to encrypt KEKs and should be stored in secure hardware components. Every key generation must be carried out in such a way that unauthorized persons have no access to the generation process. Therefore the keys must be generated in a trustworthy site or from authorized users of a cryptographic system in a trusted environment. Beside the key generation environment, the generation process is very important for the security of a system. In order to evaluate the security of a cryptographic algorithm usually made assumption is that every key occurs with the same probability. In this respect, an ideal case to generate keys is a true random processes like tossing a coin or throwing a dice. Such manual methods are expensive and cannot be used to generate lots of keys or keys with a short lifetime. But in some special cases e.g. the generation of master keys, the use of these secure methods is recommended. A possibility to generate keys automatically is the use of methods based on radioactive

sources, quantum effects in semiconductors or resistance noise. Due to the mentioned problems with true random processes, cryptographic keys are often generated by pseudorandom processes. Most common software for generating pseudorandom numbers available on many computer systems is usually too predictable to be used for this purpose. One possibility of generating cryptographic keys is to choose a suitable pseudorandom generator (which guarantees unpredictable values) and to initialize it with a manually produced master value. Such a key generator expands one true random key into several pseudorandom keys.

9.3 Certification and Authentication

Certificates are issued for authentication purposes. A credential containing identifying data together with other information (e.g. public keys) is rendered unforgeable by some certifying information (e.g. a digital signature provided by a *Key Certification Center*). Certification may be an online service where some certification authority provides interactive support and is actively involved in the key distribution processes; or it may be an offline service so that certificates are issued to each entity at some initial stage. The three main types of authentication are entity authentication or identification, message content authentication, and message origin authentication. The term verification refers to the process of checking the appropriate claims, i.e. the correct identity of an entity, the unmodified message content, or the correct source of a message. The validity of a certificate can be verified using some public information (e.g. a public key of a *Key Certification Center*), and can be carried out without the assistance of the certification authority, so that the trusted party is only needed for issuing the certificates.

authentication

9.4 Key Establishment

Key establishment is the process of making a key available to one or more entities. Key establishment techniques include secret key agreements and key transport for secret, and public keys. For many environments, protocols that allow to establish keying material in one pass are of particular interest. If entity authentication is a requirement, typically a setup phase is needed prior to key establishment. The establishment of keys can be rather complex. Key establishment protocols are influenced by the nature of the communication links, the trust relationships involved and the used cryptographic techniques. The entities may either communicate directly or indirectly, they may belong to the same or to different security domains, and they may or may not use the services of a trusted authority. The following conceptual models illustrate how these different aspects affect key establishment.

9.4.1 Point-to-Point Key Establishment

The basic mechanism of every key establishment technique is the point-to-point key establishment (see Fig. 9.4-1). If based on symmetric cryptographic techniques, point-to-point key establishment requires that the two parties involved already share a key that can be used to protect the keying material to be established. If based on asymmetric techniques, point-to-point key establishment typically requires that each of the parties has a public key with its associated private key, and that the authenticated public key is known to the other party:

- For data integrity or data origin authentication, the recipient requires the sender's corresponding public key certificate. **data integrity**
- For confidentiality, the sender requires a public key certificate of the intended recipient. **confidentiality**

Fig. 9.4-1: Point-to-point key establishment.

9.4.2 Key Establishment Within One Domain

One of the simplest forms of key establishment employs a single *Trusted Third Party* (TTP) (e.g. a *Key Distribution Center* (KDC), a *Certification Authority* (CA)) for the entire system (see Fig. 9.4-2). This authority may offer key management services such as the generation, certification, distribution and translation of keying material. When entities use asymmetric techniques for the secure exchange of information, each entity may need to contact its authority to get an appropriate public key certificate. In situations where the communicating partners trust each other and can mutually authenticate their public key certificates, no authority is needed. When symmetric cryptography is used between two such entities, the sender and the receiver are required to share keys with the authority. Key establishment then is initiated in one of two ways: **Trusted Third Party**

- By one entity generating the key and sending it to a *Key Translation Center* (KTC). The KTC receives an enciphered key from one entity, deciphers it and reciphers it using the key shared between itself and the other entity. Then it may either forward the re-enciphered key directly, or send it back to the first entity, who forwards it to the second entity.
- By one entity asking a KDC to generate a key for subsequent distribution. The KDC then either distributes the key directly to both entities, or sends it back to the initiator, who forwards it to the other entity.

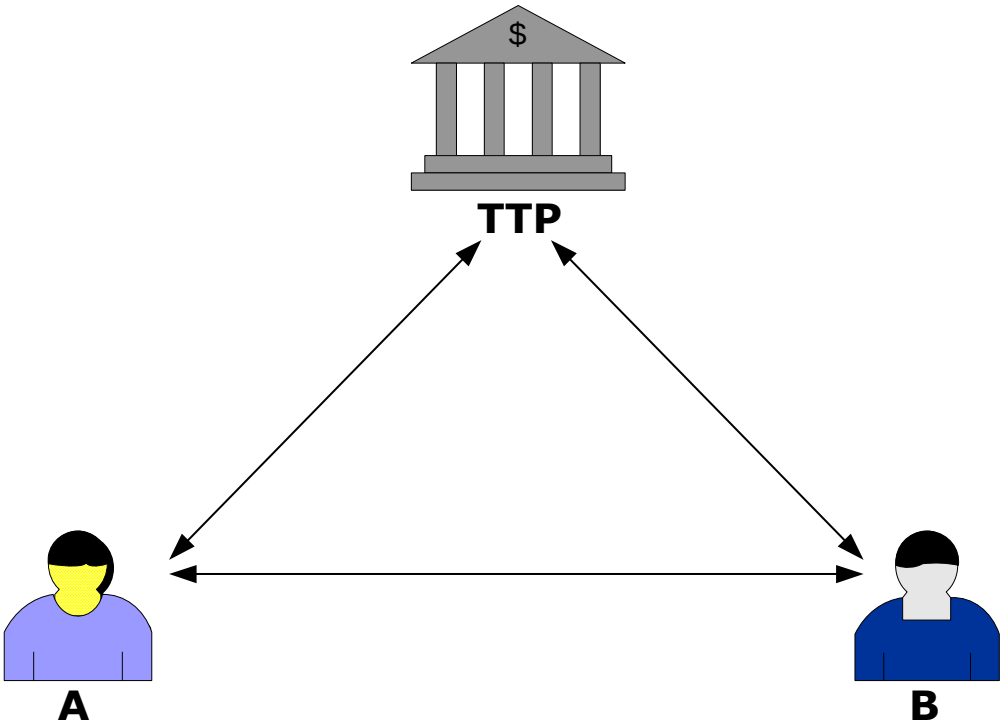


Fig. 9.4-2: Key establishment using a *Trusted Third Party*.

9.4.3 Key Establishment Between Domains

The third model involves entities belonging to different security domains (see Fig. 9.5-1). By definition, each domain has its own authority. If A and B either trust each other or each entity trusts the authority of the other's domain, then keys may be distributed according to the model described above. When the entities use asymmetric techniques and do not have access to a common directory service that offers public key certificates, each entity shall contact its respective authority to get its partner's public key certificate. The authorities of A and B may exchange the public key certificates of entities A and B and forward them to A and B. A second approach for public key certificates is cross certification exchanging, where the CAs certify the public keys of the other CAs. When the entities use symmetric techniques, at least one of them has to contact its authority to receive a secret key for communication. The authorities then establish a common secret key to be used by the entities. This key may be distributed by one authority to both entities using the other authority as a KTC, or via one of the entities which is responsible for forwarding the key to the other entity.

9.5 Key Distribution

Key distribution refers to procedures by which keys are securely provided to parties legitimately asking for them. The fundamental problem of key exchange or distribution is to establish keying material to be used in symmetric and asymmetric mechanisms whose origin, integrity, and confidentiality can be guaranteed. As a result of diverse decisions appropriate to different circumstances, a large variety of

key distribution protocols exist. Therefore it is necessary to explicate key distribution techniques in order to understand which goals the different techniques achieve, and on which assumption they depend.

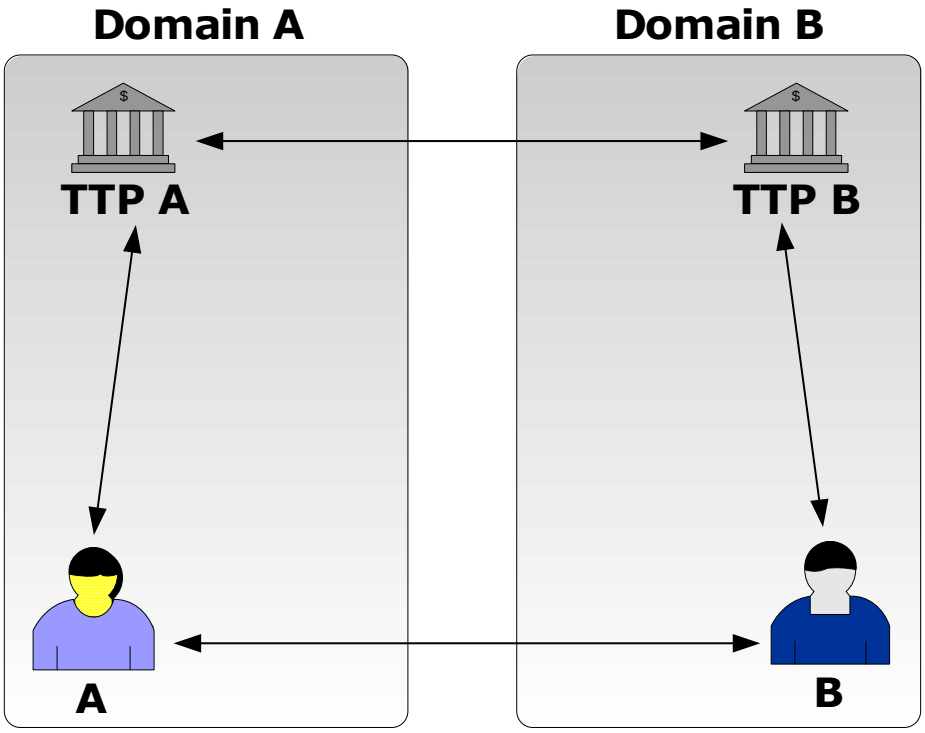


Fig. 9.5-1: Key establishment between two domains.

9.5.1 Techniques for Distributing Public Keys

Protocols involving public key cryptography are typically described assuming *a priori* possession of (authentic) public keys of appropriate parties. This allows full generality among various options for acquiring such keys. Alternatives for distributing explicit public keys with guaranteed or verifiable authenticity, including public exponentials for Diffie-Hellman key exchange (or more generally, public parameters), include the following.

point-to-Point delivery

- Point-to-Point delivery over a trusted channel: Authentic public keys of other users are obtained directly from the associated user by personal exchange, or over a direct channel, originating at that user, and which guarantees integrity and authenticity. This method is suitable if used infrequently, or in small closed systems. A related method is to exchange public keys and associated information over an untrusted electronic channel, and provide authentication of this information by communicating a hash from it via an independent authentic channel. One of the disadvantages of this method is the cost of the trusted channel.

public key register

- Direct access to a trusted public file (e.g. public key register): A public database, with trusted integrity, may be set up to contain the name and authentic public key of each system user. This may be implemented as a public key register operated by a trusted party. Users acquire keys directly from this register. While remote access to the register over unsecured channels is acceptable against passive adversaries, a secure channel is required for remote access in the presence of active adversaries. One method of authenticating a public file is by tree authentication of public trees. Authentication trees provide a method for making public data available with verifiable authenticity, by using a tree structure in conjunction with a suitable hash function, and authenticating the root value.

online trusted server

- Use of an online trusted server: An online trusted server provides access to the equivalent of a public file storing authentic public keys, returning requested individual public keys in signed transmissions; confidentiality is here not required. The requesting party possesses a copy of the servers' signature verification public key, allowing verification of the authenticity of such transmissions. Disadvantages of this approach are that the trusted server must be online, or that the trusted server may become a bottleneck.

offline server

- Use of an offline server and certificates: In a one-time process, each party A contacts an offline trusted party referred to as a *Certification Authority* (CA), to register its public key and obtain the CA's signature verification public key. The CA certifies entity A's public key by binding it to a string identifying A, thereby creating a certificate. Parties obtain authentic public keys by exchanging certificates or extracting them from a public directory. Public key certificates are a vehicle by which public keys may be stored, distributed or forwarded over unsecured media without danger of undetectable manipulation. The objective is to make one entity's public key available to others such that its authenticity and validity are verifiable.

guaranteeing systems

- Use of systems implicitly guaranteeing authenticity of public parameters: In such systems, including identity-based systems and those using implicity certified keys, by algorithmic design, modification of public parameters results in detectable, non-compromising failure of cryptographic techniques.

9.5.2 Secret Key Transport Mechanisms

Secret key transport based on symmetric techniques makes use of (long-term) symmetric keys shared *a priori* between sender and receiver. The ISO/IEC[39] ([ISO96]) specifies twelve such mechanisms, some based on point-to-point key transport and others based on mechanisms involving a third party. Secret key transport using public key techniques assumes that the sender possesses *a priori* an authentic public encryption key of the intended recipient. Also in this case ISO/IEC specifies mechanisms whose major properties are the number of passes, key control, key authen-

39 International Organization for Standardization/International Electrotechnical Commission

tication, key confirmation, and entity authentication. In the following sections, a number of typical examples for secret key transport mechanisms is given.

Secret Key Transport using Symmetric Techniques

In the basic key transport mechanism in Fig. 9.5-2, the keying matrial k is supplied by entity A

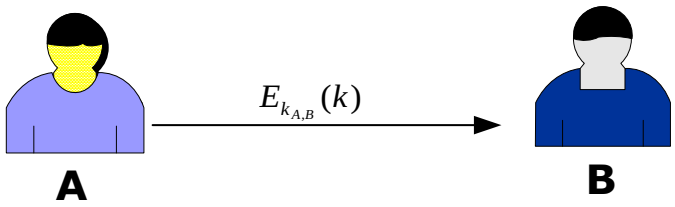


Fig. 9.5-2: Basic mechanism using symmetric techniques.

1. A sends B the keying material k enciphered with the shared key $k_{A,B}$.
2. B deciphers the message and thus obtains k.

This basic mechanism only provides implicit key authentication to A. However, the mechanism can easily be extented to also provide unilateral authentication of entity A, implicit key authentication and a freshness guarantee to B, and protection of replay back to A. In the mechanism shown in Fig. 9.5-3, two additional fields are transferred in the enciphered part:

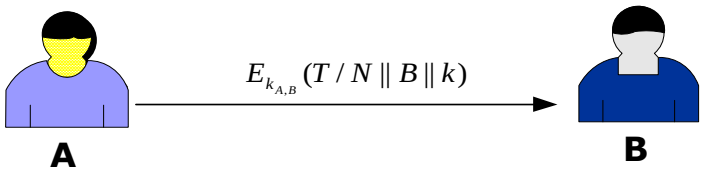


Fig. 9.5-3: Secret key transport with unilateral authentication.

1. A sends B a timestamp T or a sequence number N, the distinguishing identifier B (in order to prevent a so-called substitution attack, i.e. the reuse of this message from A by an adversary masquerading as B), and the keying material k. The data fields are enciphered with $k_{A,B}$.
2. B deciphers the message, checks the correctness of its distinguishing identifier, checks the timestamp or sequence number, and obtains the key k.

Third Party Secret Key Transport Using Symmetric Techniques

Third Party secret key transport mechanisms based on symmetric techniques assume that both entities A and B *a priori* share a secret key (denoted by $k_{A,T}$ and $k_{B,T}$, respectively) with the TTP. Furthermore, the third party is requested to be online with at least one of the entities. In the following key transport mechanism, the keying material k is supplied by a KDC:

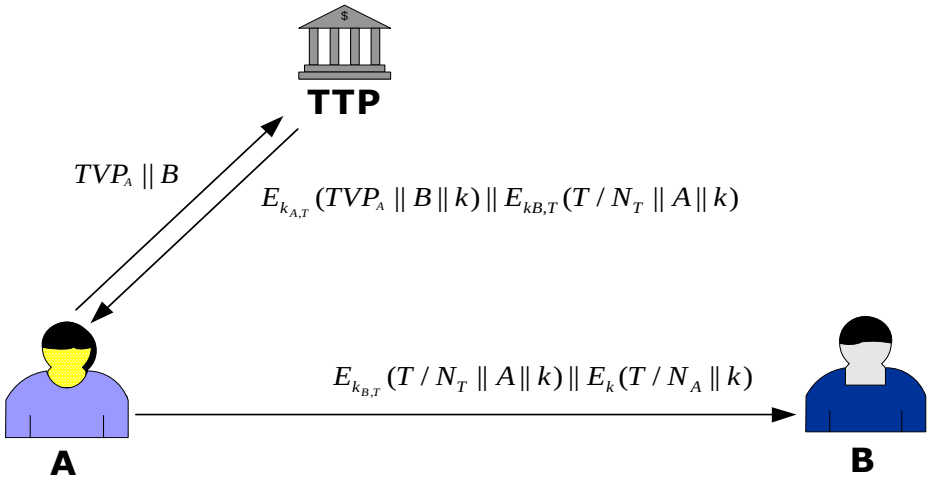


Fig. 9.5-4: Secret key transport with TTP.

1. A requests keying material from the KDC by sending a message to the KDC that contains a time variant parameter $\mathrm{TVP_A}$ (a random number, timestamp, or sequence number) and the distinguishing identifier of the recipient B.
2. The KDC returns a protected message to A that contains k enciphered for A and enciphered for B:

$$E_{k_{A,T}}(\mathrm{TVP_A} \parallel \mathrm{B} \parallel k) E_{k_{B,T}}(T/N_T \parallel \mathrm{A} \parallel k).$$

3. On receipt of this message, A deciphers the first part, checks that the time variant parameter sent to the TTP was used in constructing the message, checks the distinguishing identifier, and obtains k. If all checks are positive, A forwards the second part of the message to B followed by a data field $E_k(T/N_A \parallel k)$ which enables B to authenticate A and to check the integrity of the retrieved key k.
4. B deciphers the message, checks the correctness of the timestamp or sequence number, and obtains the keying material k. The distinguishing identifier indicates to B that k was requested by A. Then B deciphers the second part of the message and checks the correctness of the time variant parameter and of its distinguishing identifier.

This mechanism provides entity auhentication of A and key confirmation to B. By adding a fourth message, it can be extended to provide mutual entity authentication and mutual key confirmation.

Point-to-Point Secret Key Transport Using Public Key Techniques

ElGamal and RSA key transport are examples for a one-pass key transport mechanism based on asymmetric techniques. For the mechanism shown in Fig. 9.5-5 it is required that A has access to an authenticated copy of B's public encryption key $k_{e,B}$.

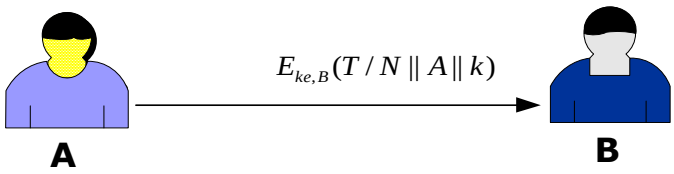


Fig. 9.5-5: ElGamal key transport.

1. A sends B a timestamp T or a sequence number N, the distinguishing identifier A, and the keying material k. The data fields are enciphered with B's public key $k_{e,B}$.
2. B deciphers the received message, checks the correctness of the time variant parameter and associates the recovered keying material with A.

The mechanism provides replay protection and implicit key authentication for A since only B is able to recover the key k. In contrast to the analogue mechanism based on symmetric techniques, B has no assurances regarding the source of the keying material. To also provide authentication of the initiator A, the key token is signed in addition to encipherment:

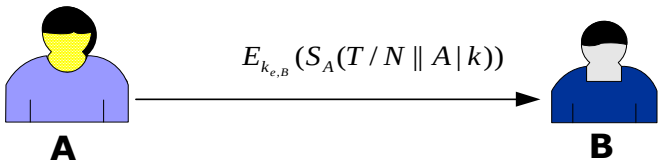


Fig. 9.5-6: Key transport with originator signature.

1. A forms a data block consisting of a timestamp or a sequence number, the recipient's distinguished identifier, and the keying material k. Then A signs the data and enciphers the signature using B's public encryption key. The result is transferred to B.
2. B deciphers the received message and verifies integrity and origin of the key token. Then B validates that he is the intended recipient of the token and that the token has been sent timely. If all verifications are successful, B accepts the keying material k.

This mechanism provides replay protection, unilateral entity authentication (based on A's digital signature), and mutual implicit key authentication (since only B is able to decipher and only A is able to sign). Depending on the needs of the enviroment, the signature may also be applied after encipherment.

9.5.3 Key-Exchange Algorithms

Diffie-Hellman

The first and best known key agreement protocol is the *Diffie-Hellman* (DH) key exchange. The protocol gets it's security from the difficulty of calculating discrete logarithms in a finite field, as compared with the ease of calculating exponentiation in the same field. DH can be used for key distribution - entities A and B can use this algorithm to generate a secret key - but it cannot be used to encrypt and decrypt messages. First, A and B agree on large primes, n and g, such that g is primitive $\mod n$. These two integers don't have to be secret; A and B can agree on them over a public channel. Then the following steps are performed:

1. A chooses a large random integer x and sends $BX = g^x \mod n$,
2. B chooses also a large random integer y and sends $AY = g^y \mod n$,
3. A computes $k = Y^x \mod n$,
4. B computes $k' = X^y \mod n$.

Both k and k' are equal to $g^{xy} \mod n$. No one listening on the channel can compute this value since they only know n, g, X and Y. Unless they can compute the discrete logarithm and recover x or y, they do not solve the problem. So k is the secret key that both entities A and B computed independently. The DH key exchange protocol can easily be extented to work with three or more people. The mechanism described above is the traditional DH key agreement scheme which provides neither key authentication nor key confirmation and is vulnerable to the man-in-the-middle attack. The classic man-in-the-middle attack on Diffie-Hellman key exchange is as follows:

man-in-the-middle attack

1. A sends his public key to B. An adversary intercepts this key and sends his own public key to B.
2. B sends his public key to A. The adversary intercepts also this key and sends his own public key to A.
3. When A sends a message to B, encrypted with the replaced public key of the adversary, the adversary intercepts the message, decrypts it with his private key, and re-encrypts the message with public key of B, and sends it to B.
4. When B sends a message to A, encrypted with the replaced public key of the adversary, the adversary intercepts the message, decrypts it with his private key, and re-encrypts the message with public key of A, and sends it to A.

This attack works, because A and B have no way to verify that they are communicating with each other. Assuming the adversary does not cause any noticeable network delays, A and B believe they communicate securely, while the adversary reads all traffic. This attack can be prevented by using protocols which provide authentication (e.g. key exchange with digital signatures).

ElGamal

ElGamal key agreement is a one-pass variant of the DH protocol where one entity uses a static public key agreement key and the other entity needs to have access to an authentic copy of this key. For ElGamal key agreement, following steps are performed:

1. A chooses a large random integer x, computes $g^x \mod n$ and sends the result to B. Furthermore, A computes, using x and B's public key agreement key g^b, the shared key as $k = (g^b)^x \mod n$.
2. On receipt of the message, B computes the same shared key using his private key agreement key b: $k = (g^x)^b \mod n$.

If static public key agreement keys are distributed using certificates, ElGamal key agreement provides unilateral key authentication.

Station-to-Station Protocol

The DH key exchange is vulnerable to a man-in-the-middle attack. One way to prevent this problem is to have entity A and B sign their messages to each other. The *Station-to-Station protocol* (STS) assumes that A has a certificate with entity B's public key and that B has a certificate with entity A's public key. These certificates have been signed by a trusted authority. The following steps are performed in order to generate a secret key k:

1. A generates a random number x, computes $g^x \mod n$ and sends the result to B.
2. B in turn generates a random number y, computes $g^y \mod n$ and the shared key $k = (g^x)^y$. B then signs the concatenation of g^x and g^y and the distinguished identifier of A and encrypts the signature using the computed key k. The result is appended to g^y and transferred to A.
3. A computes the shared key $k = (g^y)^x$, deciphers the encrypted data and verifies B's signature. If verification is successful, A sends B an analogously constructed enciphered signature.
4. B decrypts the received message and verifies A's signature.

The STS protocol provides secret key establishment with mutual entity authentication and mutual key confirmation. Another variant of the STS protocol is being considered for standardization with the Internet security protocol (IPsec).

Encrypted Key Exchange

The *Encrypted Key Exchange* (EKE) protocol provides security and authentication on computer networks, using both symmetric and public key cryptography in a way that a shared secret key is used to encrypt a randomly generated key. Within this protocol A and B share a common password P and using this protocol they can authenticate each other and generate a common session key k. In the following enumeration the basic EKE protocol is described:

1. A generates a random public-key/private-key key pair and encrypts the public key k', using a symmetric algorithm and P as the key: $E_P(k')$. A sends $B(A, E_P(k'))$.
2. B decrypts the message to obtain k'. Then B generates a random session key k and encrypts it with the public key received from A and P as the key and sends A: $(E_P(E_{k'}(k)))$.
3. A decrypts the message to obtain k and generates a random string R_A. Then A encrypts it with k and sends B: $E_k(R_A)$.
4. B decrypts the message to obtain R_A and generates another random string R_B and encrypts both strings with k. Then B sends the result to A: $E_k(R_A, R_B)$.
5. A decrypts the message to obtain R_A and R_B. Assuming the string R_A A received from B is the same as the one A sent to B in step (3), A encrypts R_B with k and sends it to B: $E_k(R_B)$.
6. B decrypts the message to obtain R_B and assuming the string R_B B received from A is the same B sent to A in step (4), the protocol is complete. Both parties are able to communicate using k as the session key.

EKE can be implemented with a variety of public-key algorithms like RSA or ElGamal.

9.6 Key Escrow/Key Recovery

A key escrow or key recovery scheme is an encrytion system with a backup decryption capability available under special prescribed conditions. Such a scheme allows law enforcement authorities, users, or other authorized persons to decrypt ciphertext with the help of key escrow/recovery information supplied by one or more TTPs. In a general key escrow/recovery scenario one assumes that the communicating entities A and B are in different domains and that there is the requirement to be able to recover decipherment keys from either domain independently. A key escrow/recovery scheme typically includes the following mechanisms:

- The sending entity prepares for transmission the enciphered message and some additional information which enables specific third parties to recover the decryption key (should this later become necessary). For this, the sending entity may need to involve a TTP.
- The recipient checks the correct format of the recovery parameters received with an encrypted message and rejects messages where this checks fails.

In a simple example for such a scheme, the sender A takes the session key k used to encipher the message m, encrypts this key with the public encryption key $k_{T,Ap}$ of the involved TTP and in addition with the public key $k_{T,Bp}$ of the receiver B's TTP. The enciphered message is then transmitted together with two encipherments of the session key:

$$E_{k_{T,Ap}}(k) \parallel E_{k_{T,Bp}}(k) \parallel E_k(m).$$

With such a protocol, both TTPs independently are able to release the session key k.

9.7 Storing, Updating and Destroying Keys

Storing Keys

Storage of keying material refers to a key storage facility which provides secure storage of keys for future use, e.g. confidentiality and integrity for secret keying material, or integrity for public keys. Secret keys must be protected by physical security (e.g. by storing it within a cryptographic device) or enciphered by keys that have physical security. For all keying material, unauthorized modification must be detectable by suitable authentication mechanisms.

Updating Keys

Assuming an encrypted data link where users want to change keys daily, the effort to distribute a new key every day is laborious. An easier solution is to generate a new key from the old key, and this process is called key updating. This can be done using a one-way function. Two entities sharing the same key and both operating on it using the same one-way function, will get the same result (note, that even an adversary gets the same result if he can access the old key). This result can be used to create a new key, but it is obvious that the new key is only as secure as the old one.

Key Destruction

Key destruction refers to procedures by which parties are assured of the secure destruction of keys that are no longer needed. Destroying keys means eliminating all records of this key, such that no information remaining after the deletion provides any usable information about the destroyed key (note: when deleting a file on most computer systems, the file isn't really deleted; the only thing deleted is an entry in the disk's index file, used for telling the machine that the file is there). A key may be destroyed by overwriting it with a new key or by zeroizing it. Keying material stored on magnetic media should either be zeroized or the media itself should be destroyed.

10 Public Key Infrastructure

10.1 Introduction

A major advantage of asymmetric key cryptography over symmetric key cryptography (see [Kaderali00b], chapter 4) is that the key distribution problem is easier to solve. Symmetric key distribution systems are expensive and hard to manage. In high-security applications with imminent man-in-the-middle attacks, symmetric systems require expensive and cumbersome secure communication lines, face-to-face meetings or courier services. In asymmetric cryptosystems the public key can be distributed without the fear of compromising the secret private key. Nevertheless, key management in public key cryptography is still a difficult and complex issue.

Many currently emerging applications in the field of information technology rely on the principles of asymmetric key cryptography. The basic security related features that public-key systems can supply are confidentiality, data integrity, authentication, and non-repudiation. Typical real-world examples are:

Secure Email The need for a secure messaging environment for the Internet is of paramount importance. Although in the past the public awareness for the problems concerning insecure email was very low, the spread of details about global surveillance systems like ECHOLON immediately produced great concern about this issue. ECHOLON is a code word for a global automated communication interception system, operated by the intelligence agencies of the United States, the United Kingdom, Canada, Australia, and New Zealand. The consortium is led by the American National Security Agency (NSA). Some estimates state that ECHOLON intercepts up to 3 billion communications everyday. These include phone calls, emails, internet downloads, satellite transmission, etc.

Secure electronic payment At the moment, many payments in Internet-based e-commerce transactions are based on credit cards. According to the credit card company Eurocard the number of credit card frauds has risen to an alerting degree. Eurocard stated that in the year 2000 the fraud rate increased by 32%. The security of electronic credit card payments can be increased by applying asymmetric key cryptography. For example, a mechanism for authentication of involved parties (customers, merchants, banks) can be provided. Furthermore, the credit card and payment information should be encrypted during the transaction. Today's most popular credit card payment system was introduced by Visa International and MasterCard in 1996 under the name Secure Electronic Transaction (SET). SET employs asymmetric cryptography for key exchange and digital signature (two different key pairs).

Access control At the moment the prevailing method of access control in corporate and open networks is to employ weak authentication with passwords. Passwords that can be remembered (and thus used) by human users, even if they

have a reasonable length, normally have such a low entropy[40] that dictionary attacks are readily successful. Even though sophisticated methods for useful password selection exist, these methods are often too cumbersome for casual users or users simply do not bother to use them. Hence, it can be advisable to replace low entropy passwords with large entropy asymmetric keys.

Authorization Allowing a user to access a computer system is a special form of authorization. Other forms of authorizations are, e.g. the authorization to provide medical advice over the Internet, the authorization to view the content of a video on demand stream, the authorization to spend money in the name of a company, etc. Such authorizations can be realized with so-called authorization certificates, which bind a special form of authorization to a public key. The holder of the corresponding private key is then able to prove that she is allowed to carry out the certified action.

Electronic Signature The recent evolution of the Internet into an open and global communication platform has greatly stimulated electronic commerce and Internet-based business-to-business transactions. An increasing number of transactions are carried out online which leads to a demand for an electronic equivalent of traditional contracts. Especially politicians from the leading industry nations were under pressure from businesses and providers of e-commerce solutions to quickly adopt legislation of electronic signatures. Emerging electronic signature acts include the use of digital signatures as a legal replacement of hand-written signatures. In february 2001, the German Bundestag approved the adoption of the European electronic signature directive. Before this law can be put into practice, the "Bürgerliches Gesetzbuch" and the "Zivilprozessordnung (ZPO)" have to be adjusted. The directive uses the term *electronic signature* instead of *digital signature* and defines electronic signature as follows:

> *"Electronic signature" means data in electronic form which are attached to or logically associated with other electronic data and which serve as a method of authentication.*

Furthermore, the directive also defines an advanced electronic signature:

> *"Advanced electronic signature" means an electronic signature which meets the following requirements: (a) it is uniquely linked to the signatory; (b) it is capable of identifying the signatory; (c) it is created using means that the signatory can maintain under his sole control; and (d) it is linked to the data to which it relates in such a manner that any subsequent change of the data is detectable.*

40 Only random passwords have maximum entropy.

In practice, this will most often be achieved by digital signatures with explicit support of non-repudiation.

It can be seen from the examples above, that asymmetric key cryptography is applied in diverse disguises. For all these applications to work effectively, sophisticated key management and distribution systems have to be constructed. The key management system for applications of asymmetric key cryptography is called *public-key infrastructure* (PKI). A typical PKI consists of hardware, software, the people working to administer and maintain the infrastructure, as well as policies regarding security, privacy and liability.

10.2 Basics of PKI

Most public-key applications rely on the fact that a public key is uniquely bound to a real world person. For these application it is absolutely essential, that the public keys which are distributed are authentic. This means that a party, which receives a public key of an entity must be assured that this entity controls the corresponding private key. In [Kaderali00b], section 7.5.1, techniques for the distribution of public keys were introduced. In practice, the association of a public key to an entity is most often achieved through a public-key certificate issued by a *trusted third party (TTP)*.

10.2.1 Identity Certificates and Trusted Third Parties

Identity certificates, originally introduced by Kohnfelder [Kohnfelder76] in 1976, bind a public key to the name of the owner of the corresponding private key and are signed by a trusted third party. A certificate is a data structure that consists of a name and a public key which form the data part, and a digital signature over the data part carried out by a TTP (see Fig. 10.2-1). Hence, identity certificates use a name to identify a key holder. A key holder is defined as the person or other entitiy that controls the private key corresponding to the public key. Identity certificates can be distributed over unsecure channels like open networks.

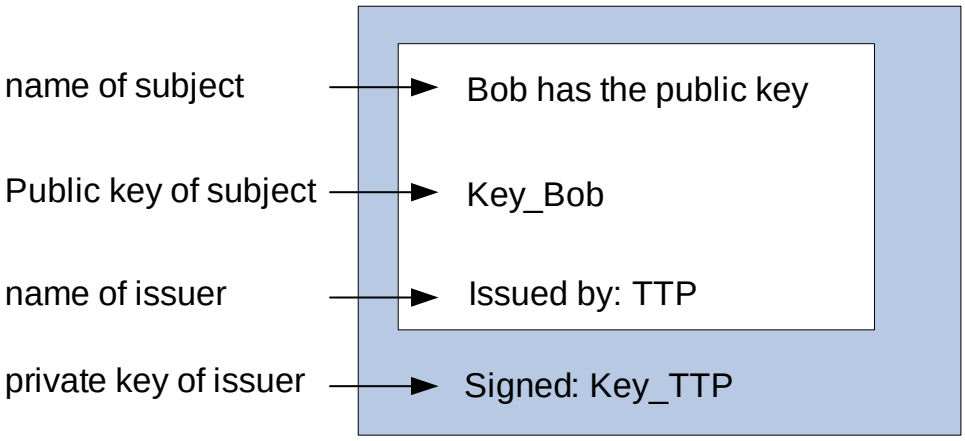


Fig. 10.2-1: Structure of an identity certificate.

Fig. 10.2-2 shows two parties, Alice and Bob, where Alice wants to send a confidential message to Bob. Alice is not able to authenticate Bob's public-key herself and asks a trusted third party for a certificate which conveys Bob's public key. Alice can accept the certificate and thus indirectly authenticate Bob if the following two conditions are met:

1. Alice authenticates the TTP, i.e. she must know the TTP's public key.
2. Alice must trust the TTP, that it correctly authenticates Bob before it creates a certificate.

Only if both conditions are met, the authentication is successful. If Alice does not authenticate the TTP (check the public key), she is susceptable to man-in-the-middle-attacks. If she does not trust the TTP, she should better not accept the certificate because the TTP could have been very inaccurate concerning the authentication of Bob.

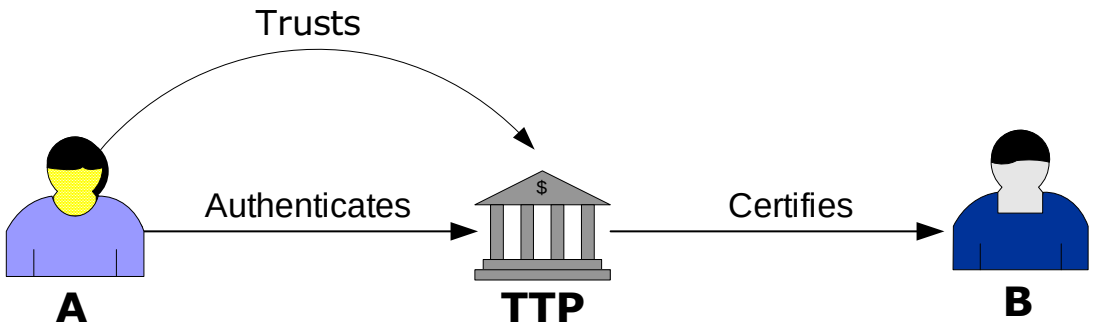


Fig. 10.2-2: Certificate from trusted third party (TTP).

While the methods available for authentication are well understood, the notion of trust has not yet been clearly defined. In contrast to other fundamental concepts of computer security like privacy and integrity, trust often remains an ambigious term. There are many notions of trust in the field of computer security, e.g.

- *System trust*: trust as assurance in the correct and secure functioning of software, computer systems, and legal systems.
- *Entity trust*: trust as belief in the benevolent, honest, competent, and predictable behaviour of autonomous agents (human or software).
- *Dispositional trust*: trust across a broad spectrum of situations and entities. Example: a person assumes that irrespective of whether people are good or bad, one will obtain better outcomes by trusting them - hence, one should generally trust them.

Trust is always expressed in a relation to an entity *and* to an action. If an entity is *trustworthy* it means that this entity is *able* and *willing* to act in other entities' best interests.

If we regard trust as a personal relation between two entities Alice an Bob, the following properties seem to be reasonable:

1. Trust is not symmetric. If Alice trusts Bob it does not mean that Bob also trusts Alice.
2. Trust is not transitive. If Alice trusts Bob, and Bob trusts Carol, Alice does not necessarily trust Carol.

Trust that is not transitive is also referred to as *direct trust*. Nevertheless, in practice trust is often transitive and is then referred to as *indirect trust*. For instance, professional organizations and recommendation services impart trust by virtue of their reputation, and insurance companies impart trust by increasing the financial predictability of outcomes. In these two examples trust flows from reputable institutions to other principals. Generally, certain kinds of trust are transitive, while others are not.

A PKI that manages keys in a community/organisation with an already existing trust relationship between the trusted third parties and the users is referred to as *closed PKI*. A typical example is a corporate PKI. In an *open PKI* there is no "natural" trust relationship between trusted third parties and users. Open PKIs are thus much harder to realize. A typical example is a PKI for electronic signature applications.

Currently, there are two important certificate formats: the X.509 format which was standardized by ITU-T [IT97b], and the Open PGP format which was standardized by IETF RFC 2440 [RFC2440]. Both formats are described in further detail in Section 10.3). In PKIs based on X.509 certificates a trusted third party is called certificate authority (CA) which is an institution responsible for certifying users of the PKI. CAs in open PKIs which provide certificates as a commercial service are called trust centers. In PGP every user can also act as a certificate authority and is then referred to as introducer. The structure in which all users certify each other and rely on certificates issued by other users is known as web of trust.

The goal of identity certificates is to bind a public key to an entity. Each entity must be represented by a unique identifier to distinguish it from other entities. X.509 and PGP employ different concepts to achieve globally unique identifiers.

Distinguished names

X.509 certificates rely on the X.500 [IT97a] approach of distinguished names. X.500 and X.509 are standards which were published by the ITU in 1988. X.500 describes a distributed international public directory which is based on the client-server principle. The directory is organized in a hierarchical tree structure (directory information tree (DIT)) with a global authority as root. The original purpose of X.500 was to built a global database of named entities: people, computers, printers, etc. These entities are the leaves of the information tree. The nodes of the DIT represent countries and organisations which are responsible for the following parts (nodes and leaves) of the information tree. Originally, X.509 certificates were used to specify which entity was allowed to modify which part of the X.500 tree. This was achieved by binding the name of the entity to a public key (originally a password). The names in X.500 have a special format and are called distinguished names. The distinguished name is a complete path from the root to the referred

entity. Fig. 10.2-3 shows an exemplary information tree with distinguished names. Table 10.2-1 shows the allowed attribute types for distinguished names.

It must be noted that the X.500 plan of a single-rooted open online-telephone book is unlikely to ever come to fruition. Firstly, collections of employee and hardware information are considered valuable or even secret assets. Most organizations and corporations will certainly not publish sensitive data in an open directory. Secondly, the idea of a distinguished name, which must be globally unique, is also not likely to occur. The required global naming discipline is simply not achievable, both for economical and political reasons. There is one exception to this rule: the Domain Name System of the Internet which is a global namespace holding unique names.

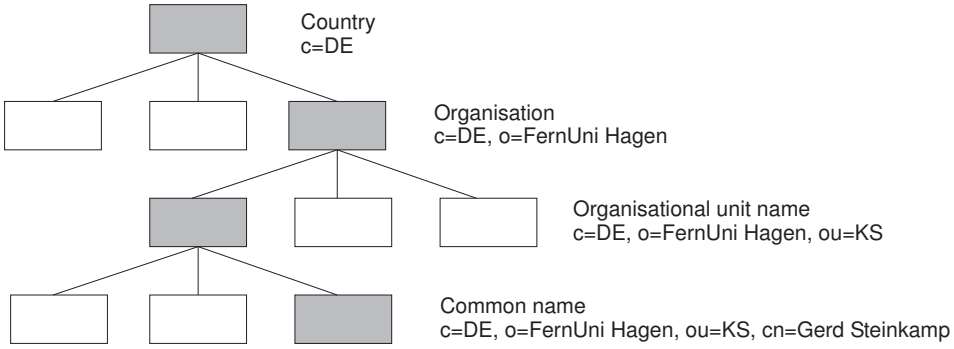


Fig. 10.2-3: X.500 directory structure with distinguished name.

Tab. 10.2-1: *Attributes for distinguished names.*

Attribute	Meaning
c	country name
o	organization name
ou	organizational unit name
s	surname
cn	common name
l	locality name
sp	state or province name
st	street address
t	title
sn	serial number
bc	business category
d	description

Names in the Domain Name System

The DNS namespace is under the control of the Internet Corporation for Assigned Names and Numbers (ICANN). The ICANN is a non-profit organization formed by a broad coalition of the Internet's business, technical, academic, and user communities. The tasks of ICANN include the coordination of the technical management of the Internet's domain name system and the allocation of the IP address space. Hence, in contrast to an X.500 distinguished name, a DNS name is truly unique, which is guaranteed by a single authority, the ICANN. A single root could be accomplished, simply because it was setup before the Internet gained its international economical and political importance. An examplary DNS name is shown in Fig. 10.2-4.

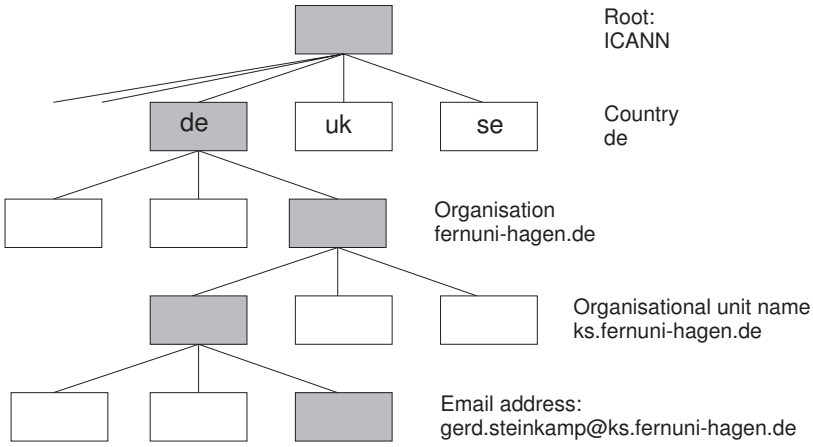


Fig. 10.2-4: A name in the DNS.

Creation and storage of key pairs and certificates

For each of the different public-key applications like encryption, authentication, electronic signatures an individual key pair has to be created. The reason for this is that the different applications require seperate considerations with respect to

- creation, storage and backup,
- expiry and key lifetime.

A decryption key has different backup requirements than an electronic signature key. The decryption key must be backed up and stored over the whole lifetime of the encrypted documents. A signature private key, on the other hand, *must never* be backed up! Non-repudiation can only be achieved if only the owner has access to the private signing key. Otherwise, impersonation can occur. Hence, the signature key must be securely stored in a way, that only the rightful owner of the key can control it. For instance, this can be achieved by storing the private key on a tamper-resistant smart card which is secured by the owner's biometrics. A signature private key requires a relatively short lifetime because it must remain uncompromised. This is in complete contrast to the decryption key, which expires with the encrypted document. Hence, in some cases it never expires. If the decryption key expires prior to the document, the document is lost. It can be seen, that the distinct functions of

encryption and electronic signature cannot be truly and effectively provided using just one key pair.

An electronic signature key must never be used for challenge-response authentication because it would compromise security by allowing chosen-plaintext attacks. For example, Bob challenges Alice with a random number r_b encrypted under Alice's public key, and Alice is required to respond with the decrypted random number. If Bob challenges Alice with $h(x)$, where x is a message unknown to Alice, Alice's response to this authentication request may unwittingly provide to Bob her electronic signature on the hash value of the message x.

There are two options for the creation of the key pair:

- The key pair is created by the certificate authority. The CA also creates the certificate which binds the public key to the holder's identity. If this is a key pair for electronic signatures, the user has to trust the CA that it does not copy the private key during creation. This is not a big issue in closed PKIs but has to be considered in open PKIs (like that conforming to the European directive for electronic signatures).
- The key pair is created by the user. The user uses his own equipment to create the key pair. Afterwards, he either creates a certificate himself (e.g. PGP) or contacts a certificate authority to testify the binding of his identity to the public key.

As already mentioned, in high-security applications the private key has to be securely stored. The environment, in which keys are stored and cryptographic operations are carried out is called personal security environment (PSE). The private key must never leave the PSE. Encrypted documents are decrypted in the PSE, because the private decryption key is needed for this operation. Documents are signed inside the PSE for the same reason. Furthermore, the PSE may also hold the corresponding public-key certificate. Another important function is the generation of key pairs which requires high-quality random numbers that the PSE must be able to generate. This may be a reason to delegate key generation to a professional CA which can afford the required cryptographically sound equipment. Alternatively, user workstations can generate random numbers by evaluating random events produced by the user, like mouse movements or keyboard typing. A simple PSE ist just some memory space on the user's personal computer, secured by a password. A high-security PSE is a hardware device like a smart card that is physically protected against tampering and secured with biometrics against unauthorized access.

Although devices like smart cards seem to be well suited as a PSE, the following problem has to be considered in electronic signature applications: The PSE must be able to display the document to be signed. In the smart card example, a sensible place to put the display would be the card reader (here the PSE consists of the smart card and the card reader). Nevertheless, for most signature applications the display integrated in the card reader will simply be too small. It may be sufficient to display the value of an electronic cheque, but it will not be able to show a multipage electronic contract. Hence, such a large document must be displayed outside the PSE.

This introduces a major security risk, because now the user has to trust equipment that is outside his PSE to correctly display the document to be signed. In the case of the Internet-connected PC, all it needs to let the user sign an arbitrary document is a trojan horse. The trojan horse would display the document the user thinks he is signing, but would send a different document (in practice the hash value of the document) to the PSE. A more general term often used instead of PSE is trusted computing base (TCB). Electronic signature applications need a TCB (which includes the display) to provide the required evidence for non-repudiation that a court is willing to accept.

Time stamping and Notarization

Time stamping is a service in which a trusted third party - a time stamp authority (TSA) - signs a message, in order to provide evidence that it existed prior to a given time. Time stamping is mandatory for non-repudiation applications like the electronic signature. For instance, a user cannot claim that a transaction was later forged after compromise of a private key occured, because the existence of the signed time stamp indicates that the transaction in question could not have been created after the indicated time.

Time stamping is a special form of notarization. A notary service is a more general service which is not only able to testify the existence of a document at a given time, but also vouches for the truth of more general statements at specified points in time.

Certificate Distribution, Expiration and Revocation

There are two basic mechanisms for the distribution of certificates:

1. In the pull model the acceptor of authentication information (e.g. a computer that authenticates a user) pulls the certificate from a certificate directory, which often is a database holding all relevant certificates. Depending on the size of the PKI this can be a single directory server or a distributed database consisting of multiple directory servers.
2. In the push model the certificates are sent out to all users upon certificate creation. Alternatively, the user that acts as a signer (e.g. in an authentication situation) may provide an appropriate certificate as evidence when needed.

In many applications certificates are not issued for an infinite time period but are bound to a validity window. An expiration date shall help to reduce the risk of compromise of the private key (precaution against cryptanalysis). A certificate becomes invalid or insecure in the following situations:

- The expiration date of the certificate has been reached.
- A change in the owner's relationship to the trusted third party has occured or the owner's access rights have changed. For instance, this happens when the owner leaves or changes his role within an organization. If the distinguished name of a user in an X.509 based PKI changes (or the email address in PGP), a new certificate has to be created.

- The user looses his private key. In this case he must be provided with a new key pair and a new certificate. Alternatively, if the application supports key recovery, the user may be provided with the recovered key. In order to enable key recovery the private keys of all users must be stored in a confidential location. The procedure of storing copies of private keys is also referred to as key escrow. As mentioned before, key escrow only makes sense for decryption keys.
- The private key has been compromised. Typical reasons for key compromise are:
  - an attacker got access to the password securing the private key.
  - an attacker got access to the private key itself, e.g. by cryptanalysis.

Certificate revocation is the mechanism under which an issuer can revoke the association (key-name) before the end of its documented lifetime. The *revocation state* indicates the validity or cancellation of its association. The "freshness" of the revocation information is essential for the security of the application. It is up to the operator of the PKI to define the meaning of freshness. The following issues have to be considered [Daniel00]:

- All verifiers (users of the PKI) must be able to correctly determine the revocation state of a certificate within well-known time bounds.
- The costs associated with the management, retrieval, and verification of certificates should increase at a rate slower than increases in the size of the serviced community.
- Any revocation service must be able to support guarantees consistent with existing security policies and requirements.

A frequently employed means for revocation is to list the revoked certificate in a certificate revocation list (CRL). The CRL identifies the revoked certificates by their serial number and is concatenated with a time stamp which indicates its freshness. The list is signed by the responsible revocation authority, which is normally the certificate authority that issued the certificate, to guarantee integrity. The CRL can be published over the same distribution channels as the certificates. This can either occur in push or pull mode. In push mode, the notification of certificate revocation (here the CRL) is automatically sent to all users of the PKI. In pull mode, the verifier of a certificate has to check if the certificate is still valid, by requesting the CRL from the responsible revocation authority. The advantage of CRLs is simplicity, the disadvantage the high communication costs. An option to reduce the size of the CRLs ist to publish only new entries. A CRL that only contains new entries is called delta-CRL. The users of the PKI have to cache all revocation information and update it according to the issued delta-CRLs.

Often, a compromise between security and scalability has to be found. The more frequently revocation information has to be published (e.g. in form of a CRL), the higher the communication costs become. In some high-security applications the verifiers are obliged to contact an online revalidation service before relying on a certificate. The revalidation service’s answer to a request is a statement that the certificate is either valid or invalid. It must be noted, that in such a system the power

of certificates is considerably reduced. A certificate does not suffice by itself to provide authentication information. Every certificate in the system is stale by default and must be refreshed by online revalidation.

Although popular in commercial PKIs due to their simplicity, CRLs have been criticized in the research community. Rivest [Rivest98] made three propositions which should help to improve on conventional revocation mechanisms.

Proposition 10.2-1: *Recency requirements must be set by the acceptor, not by the certificate issuer (CA).*

The acceptor of the certificate is the one who is taking the risk. Hence, he should decide what a satisfactory recency requirement is.

Corollar 10.2-1: *Periodically-issued CRLs are wrong, because they are inconsistent with Proposition 1.*

CRLs are issued by the CA which also dictates the update interval. The acceptor of the certificate must accept the provided recency evidence.

Proposition 10.2-2: *The signer can (and should) supply all evidence the acceptor needs, including recency information.*

Instead of having the *acceptor* query the CA for CRLs, the *signer* is asked to obtain any necessary evidence, and present it with his signature. One advantage of this approach is, that it allows the acceptor, which often is a server, to be implemented in a stateless manner. For example, the server may reply: "Sorry, please make sure that all of your evidence is at most one week old", and then forget about the request. The server does not have to store state information about that request, it simply rejects it and expects the user to come back later with appropriate evidence. A stateless server design is less vulnerable to denial-of-service attacks.

Proposition 10.2-3: *The simplest form of "recency evidence" is just a (more-) recently issued certificate.*

This proposition leads to a considerable organisational overhead if certificates have to be periodically reissued. In practice, the more efficient solution would be to implement a revalidation service. Revalidation has the advantage, that it can be automated, e.g in form of a software agent. A certification service, on the other hand, in most cases cannot be automated.

Certificate Policy and Certificate Practice Statement

A certificate policy is a set of rules that define the applicability of a certificate to a community/organisation with certain security requirements. The certificate policy of an organisation leads to a certificate practice statement (CPS). The CPS explicitely states a CA's policies concerning the issuance, maintenance and revocation of certificates. Furthermore, it may contain information about legal aspects and liabilities towards entities relying on the certificates. The CPS can be seen as a verbose version of the certificate policy. As much as possible, a certification practice statement should indicate any of the widely recognized standards to which the CA's

practices conform. The level of details makes a CPS proprietary and thus it normally only applies to a single organization. A certificate policy, on the other hand, applies more broadly than a CPS. If a particular certificate policy is widely recognized and imitated, it has great potential as the basis of automated certificate acceptance in many systems. Interconnection of PKIs is thus carried out on basis of certificate policies, and not on certificate practice statements. A CA with a single CPS may support multiple certificate policies used for different applications and purposes. Also, multiple different CAs, with non-identical certification practice statements, may support the same certificate policy.

10.2.2 Certification Structures

A more general certification path than that presented in Fig. 10.2-2 can consist of several TTPs and certificates as shown in Fig. 10.2-5. The chain at the top of the graphic is based on direct trust. Alice can only authenticate Bob if she authenticates the first TTP and directly trusts *each* TTP in the chain. The trust vectors originating from Alice and ending in a TTP are also called trust anchors. In the example at the bottom of Fig. 10.2-5 Alice only sets up one trust anchor to the nearest TTP and the trust propagates from each TTP to its successor. Which of the two trust models can be applied depends on the application including the related security policies and the participants.

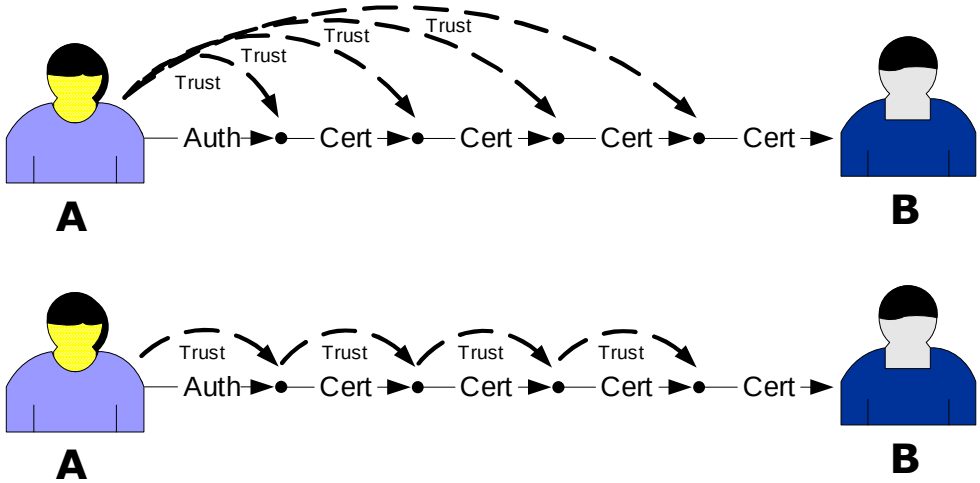


Fig. 10.2-5: A certificate chain relying on direct trust (top) and indirect trust (bottom).

The certificate path shown in Fig. 10.2-5 consists of all certificates $TTP\{E\}$ of the path. $TTP\{E\}$ is certificate issued by a TTP for entity E, which can either be a user or another TTP. The certificate path to Bob may look as follows:

$$TTP_1\{TTP_2\}, TTP_2\{TTP_3\}, \cdots, TTP_{n-1}\{TTP_n\}, TTP_n\{Bob\}.$$

The trust model at the bottom of Fig. 10.2-5 is typical for CA-based PKIs. In these PKIs trust is often a form of system trust which is transitive. The web of trust, on the other hand, is based on entity trust which is produced by social relationships. This form of trust is not transitive and the trust model at the top of Fig. 10.2-5 applies.

In the following, important certification structures are described.

Single-CA

In the model shown in Fig. 10.2-6 one single-CA is responsible for certifying all end entities (EE). The arrows represent certificates. The geographical distribution of the participants must be locally constrained if the certificate policy requires physical attendance for certification. It is obvious that the CA will become a bottleneck because the task of authenticating users cannot be automated beyond a certain degree. Obviously, this model is only feasible for small applications with a limited number of participants.

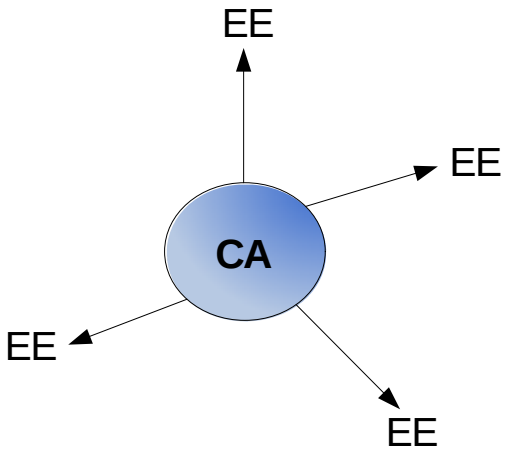


Fig. 10.2-6: Single-CA.

Single-CA plus Registration Authorities

This model still consists of a single-CA but allows for a greater number of participants who can also be geographically distributed (see Fig. 10.2-7). There are multiple registration authorities (RA) which are trusted by the CA to verify the mapping between a name and a key. Hence, the CA delegates the task of user registration and authentication to the RAs and performs the cryptographical tasks itself (e.g. key generation, signing). This model is more convenient for the users because they do not have to travel to the CA's location for registration. Each RA has a key pair associated with it. The CA knows the RAs' public keys which it uses to authenticate messages sent by the RAs. If the CA receives a valid signed request from an RA, it creates a certificate.

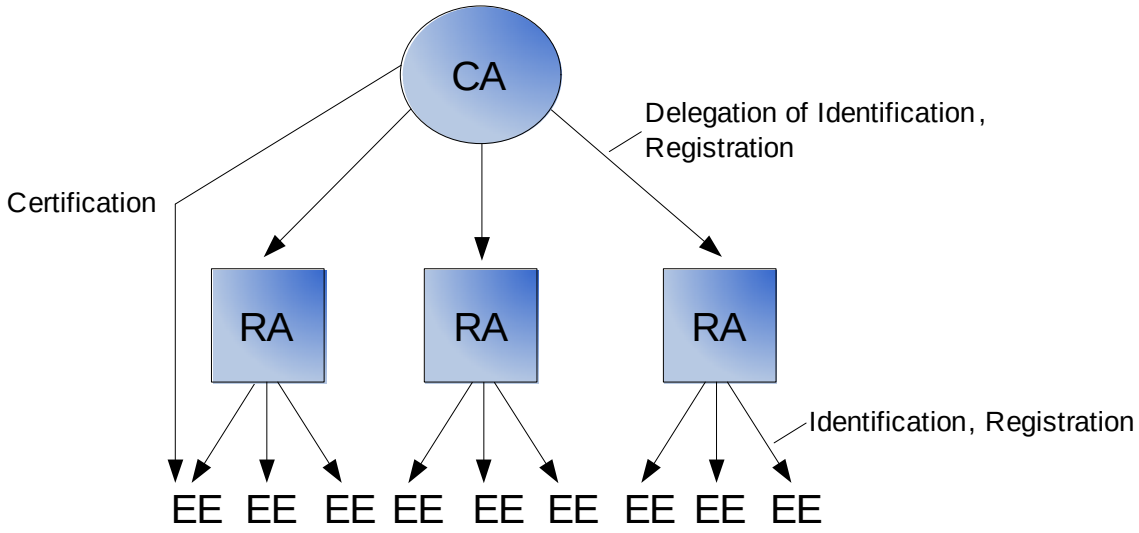


Fig. 10.2-7: Single-CA plus registration authorities (RA).

Oligarchy of CAs

Instead of being configured with the public key of a single-CA, everything is configured with public keys from multiple CAs. For example, there can be dozens of organisations from which one can obtain a certificate. This model is currently implemented in web browsers (open PKI). The advantage of this over the single-CA model is that competition among CAs should prevent abusive pricing for obtaining certificates. The disadvantage of this model is that it is less secure than the single-CA model. In the single-CA model security depends only on the single-CA's key. When there are multiple CAs the security depends on all keys. Compromise of any of the dozens of keys is as serious as compromise of the single key in the single-CA model.

Subordinated hierarchy

In this model there are multiple CAs arranged in a tree-like hierarchy (see Fig. 10.2-8). The CA at the top of the hierarchy, which is referred to as *root CA*, certifies all other subordinated CAs. Each CA certifies one or several CAs at the following level of the hierarchy. Hence, the root CA delegates the duty of certification to other CAs. The subordinated CAs are normally strictly limited in terms of the name subtree they can certify. Hence, this hierarchical model resembles the structure of large corporations with a root authority at the top level and subordinated departments at the lower levels. In the model displayed in Fig. 10.2-8 the user (EE) does not need to trust the CA closest to her, but she must trust the root CA, hence the root CA is the user's trust anchor. This trust transitively propagates over all subordinated CAs. If the private key of the root CA is compromised, all certificates in the hierarchy become invalid. In practice, the number of CAs in the chain is restricted (often to 3) because trust quickly degrades as it is propagated across a long chain. The advantage of this model is that the workload of certification is distributed across the hierarchy. Each CA only certifies entities which are somehow related to it (e.g. are working in a related department of a large corporation).

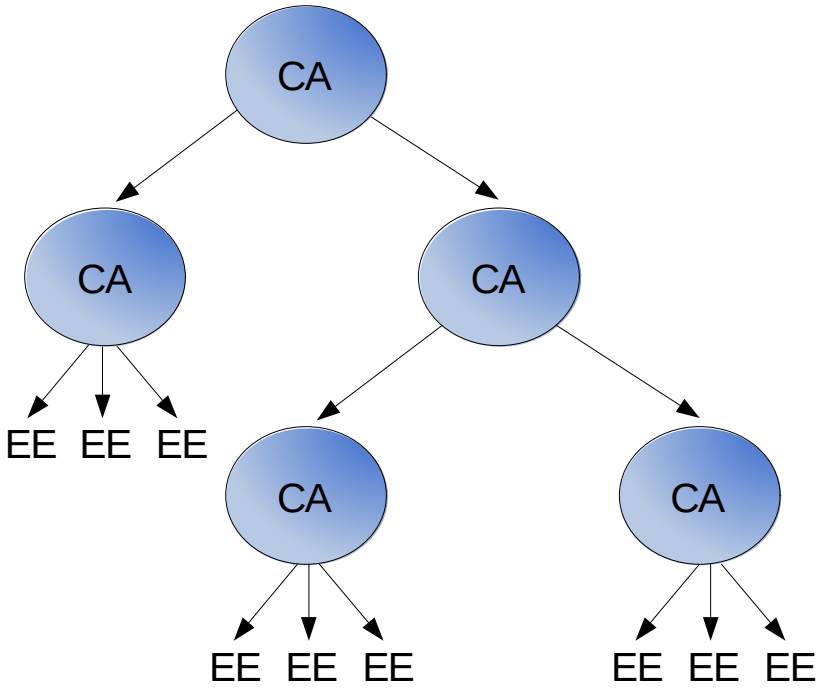


Fig. 10.2-8: Hierarchy of CAs.

Cross-certified Mesh

A cross-certified mesh envolves when PKIs belonging to different security domains want to cooperate. A security domain is defined as a system under the control of a single authority which the entities therein trust. Each security domain has its own security and privacy policy defined either implicitly or explicitly by its authority. To enable communication between two domains the root CAs cross-certify each other (see Fig. 10.2-9). Nevertheless, also subordinated CAs can cross-certify each other. This is often done to reduce the length of a high-usage certification path (short-cut). It must be noted, that cross-certificates are not necessarily bidirectional.

The problems connected with cross-certification are:

- Cross-certification can only be carried out between domains, which agreed on a common certificate policy. In practice, this is a big problem, because certificate policies of different organisations are not compatible by default. Fig. 10.2-10 shows two incompatible certificate policies.

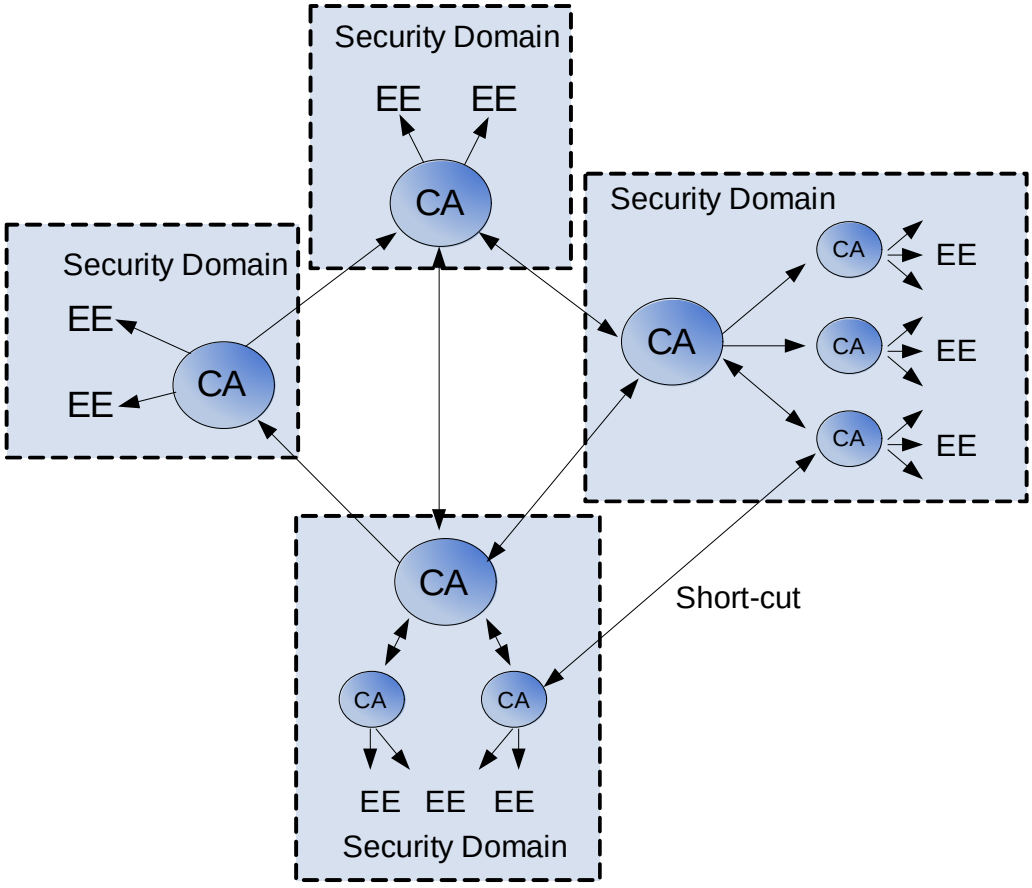


Fig. 10.2-9: Cross-certified mesh.

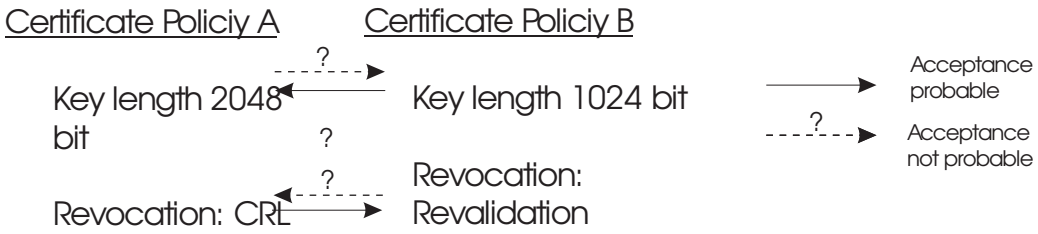


Fig. 10.2-10: Comparison of two incompatible certificate policies.

- In this model there is in general more than one certification path between two entities. In practice, there are no mechanisms which can handle more than one certification path.
- The cross-certification is often transparent to the users (EE) of the PKI. The chain along which trust is transitively propagated can become quite long. It is not obvious, how secure these chains really are.

Bridge CA

Cross-certification does not scale well to larger numbers of CAs. The organisational overhead of linking security domains by cross-certification grows as n^2, where n is the number of domains to be fully connected. The bridge CA model can help to reduce the costs and problems of linking independent security domains. Fig. 10.2-11 shows a bridge CA that cross-certifies with each security domain. It must be noted that the bridge CA is not a root CA but a federal neutral institution. It acts as a peer rather than a superior to the root CAs in the different security domains. A bridge CA is thus able to connect diverse existing PKIs like corporate PKIs, banking PKIs, and government PKIs. Nevertheless, a uniform certificate policy is still needed. In practice a bridge CA provides the certificate policy to which all connecting PKIs must conform. This model appears to be the prime candidate to transform today's PKI island solutions into a national PKI network. If bridge CAs in different continents and countries cross-certify, this model may even extend to a global open PKI.

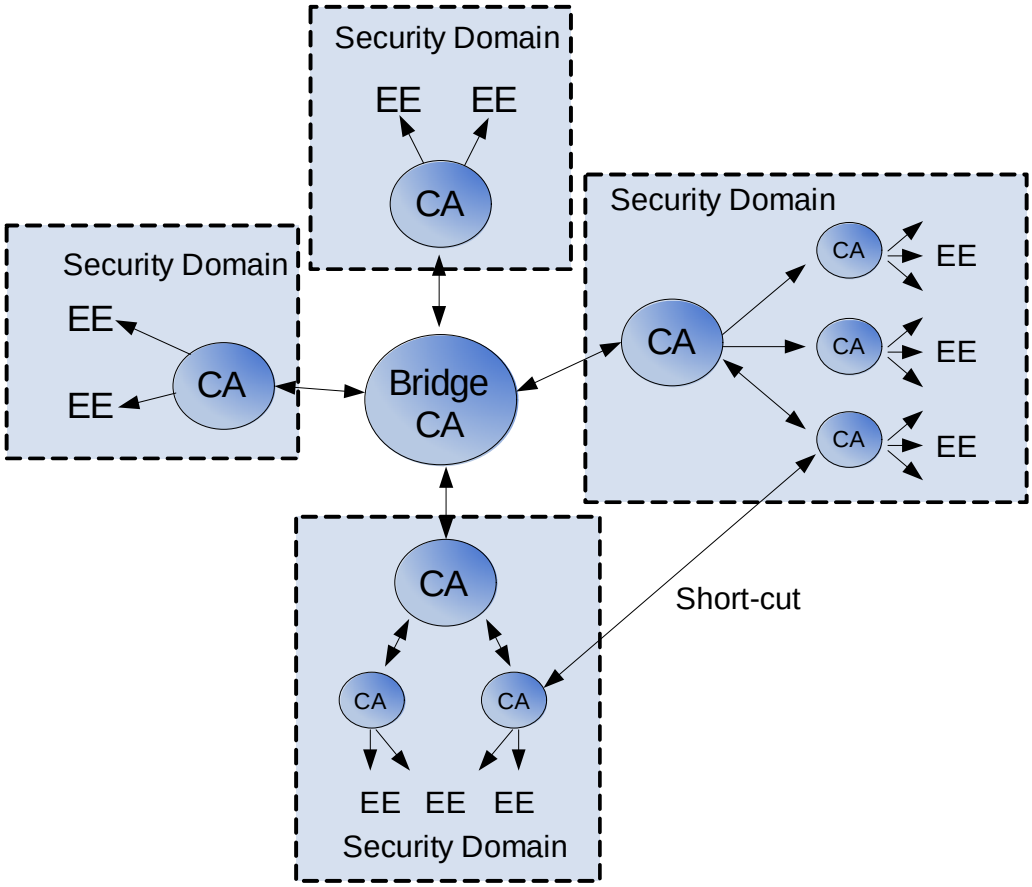


Fig. 10.2-11: Bridge CA.

Egalitarian structure

In the egalitarian model, as shown in Fig. 10.2-12, all entities are both, certificate authority and end entity. Each entity decides which other entities it wants to trust with regard to the certification of other entities. This model was made popular by Phil Zimmerman and his email encryption programme *Pretty Good Privacy (PGP)* which coined the expression Web of Trust. It must be noted that the egalitarian model is totally distributed. There is no central institution that must be relied upon. Hence, every participant can freely set his trust anchors on every other participant, who will than act as an introducer for other entities to him. This results in a certification graph with multiple parallel paths of arbitrary length. The model can thus be compared with the cross-certified mesh model, but the structure of the graphs and the semantics of the nodes and edges is slightly different. The structure of the graph often resembles social relationships between the entities, e.g. friendship, work relation, family relation, etc.

Note that Fig. 10.2-12 only shows certificates issued by the users of the Web of Trust. The actual trust anchors are not displayed in this graphic. While the certification graph is unique for all participants (ideal case), the trust graph which contains the trust relations is specific (subjective) for every user. For an example of a combined certification and trust graph see Fig. 10.3-1 in which Alice's subjective trust graph is superposed on the certification graph.

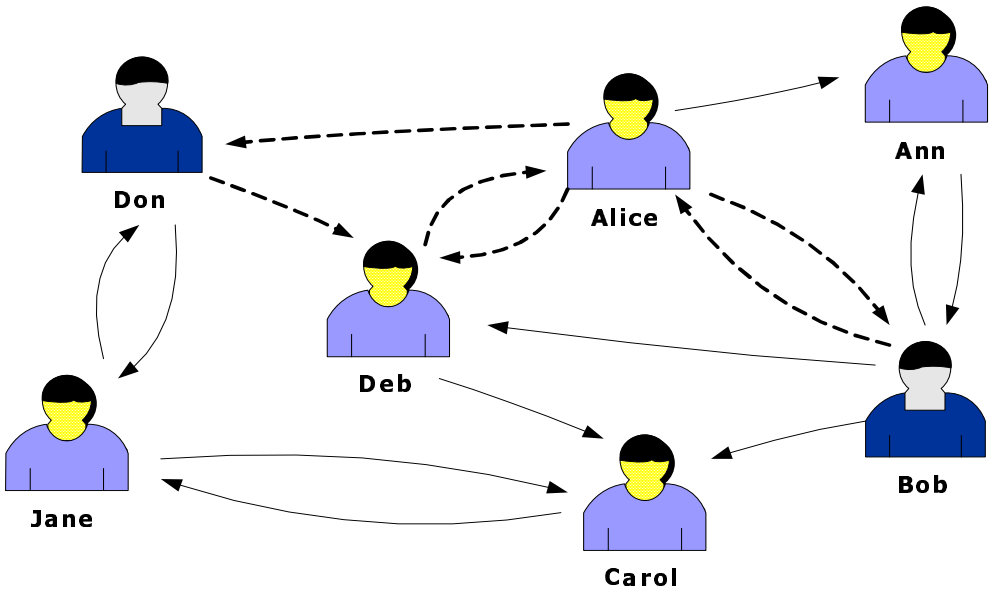


Fig. 10.2-12: An example for an egalitarian certification structure: The Web of Trust.

10.2.3 Attribute Certificates

Identity certificates like X.509 certificates only solve one part of the problems in public-key applications: They bind a person to a public key and thus help to authenticate an entity. They give no information about which action the certified entity is allowed to perform. The mapping of entities to rights is completely left to the application itself.

Example 10.2-1:
Let us consider an application that employs strong authentication to grant access to a computer system (see Fig. 10.2-13). All users of the system are provided with a smart card containing a X.509 identity certificate. The X.509 certificate was issued by a CA belonging to the organization who owns the computer network (closed PKI). When a user wants to get access to the computer system she inserts the smart card into a card reader and enters her password to proove that she is the rightful owner of the private key which is also stored on the smart card. Each computer in the network is configured with the public key of the CA. The authentication can now be performed as a challenge-response based on digital signatures (see [Kaderali00b], Section 8.4.2)[41]. After the authentication process the computer system knows the name of the user, but it does not know if the user is allowed to access the system. Hence, the system must look-up the name of the user in an access control list (ACL). The ACL is held in a directory server (light-weight database). The directory server and the ACL are administered by the system administrator, who is the responsible person for access control. If the user name is listed in the ACL the system grants access to the user.

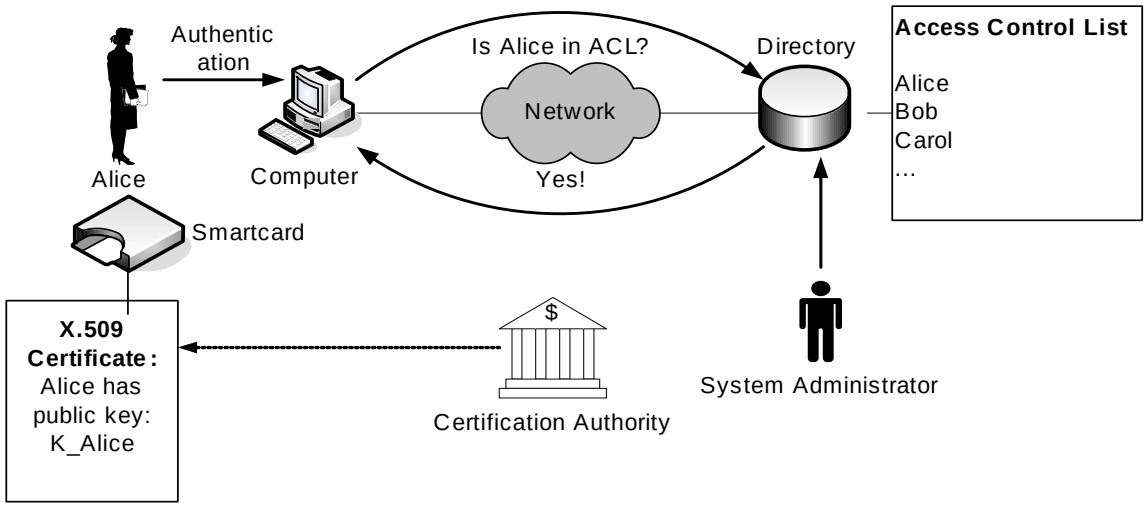


Fig. 10.2-13: Example for strong authentication with certificates.

In the example above, Alice presents a piece of evidence - an identity certificate - to the computer system. This is referred to as push model. This model conforms with the rule, that the signer should supply all evidence the acceptor needs, including recency information (see Section 10.2.1). Nevertheless, sometimes it is more convenient to request the certificate from a corresponding network service (e.g. a directory server (pull model)). Alice does not present any evidence that she is allowed to access the network. Hence, the access right is pulled by the accessed computer from the directory.

This example shows, that the PKI alone (in this case a single-CA) does not suffice to make decisions about access rights. There has to be a second mechanism, a directory server holding the ACL, to successfully grant access requests. The maintenance of such a directory server leads to organizational overhead, and also represents a critical point of failure. There is always the chance that the directory server fails, which would prohibit any user from working with the system. To eliminate this risk, redundancy mechanisms have to be installed which further add to the complexity of the system. Furthermore it must be noted, that every access request leads to messages sent over the network from the accessed computer to the directory server and back. For an attacker, the directory server is a valuable target. If the attacker is able to get access to the directory he is able to add and remove arbitrary users.

The integrity of the access information can be achieved with a digital signature. One approach is, to include the access authorization inside the X.509 identity certificate (in the extensions fields). Nevertheless, this is often not a viable method, because the certificate authority is seldom the authority responsible for granting the appropriate rights. This can be seen in our example where we have a CA responsible for issueing identity certificates and a system administrator who grants access rights. These are obviously different entities. Furthermore, the dynamics of the two types

41 The private key must never leave the smart card. The signature is computed on the smart card itself.

of certificates may be different. Identity certificates often have a validity period of several years. The access right on the other hand may only be valid for a much shorter time period, e.g. a day or a week. If the identity certificate has to be reissued or revoked in synchronization with the authorization information, this may have a severe impact on the performance of the access control system.

Recognizing that identity certificates are not the ideal place to put authorization information, an alternative approach is to place the authorization information in an extra attribute certificate. This attribute certificate is a structure consisting of a data part which holds the attribute information and the subject (name of an entity) to which the attributes are bound, and a signature part containing a signature over the data part carried out by an attribute authority (AA).

In our example, the access control system can be simplified by introducing attribute certificates carrying the authorization information besides the identity certificates. These attribute certificates can either be stored in the directory server or, like the identity certificate, on the user's smart card. Fig. 10.2-14 shows how attribute certificates can help to reduce the complexity of the access control system. It can be seen that the directory server was removed. Instead of maintaining a list with authorized users, each user gets an attribute certificate signed by the system administrator (AA), which binds the right to access the computer system to the person holding the certificate. The integrity of the certificate is maintained by the system administrator's signature. The system administrator himself is certified by the CA. Each computer on the network is configured with this certificate and the CA's public key.

When Alice wants to access the network, she presents two items of evidence: the identity certificate which states that Alice is bound to the key stored on the smart card, and the attribute certificate which states that Alice is granted access to the network. After the authentication process the accessed computer does not query a directory server, but processes the attribute certificate. If the signature checks out ok, Alice is granted access to the network.

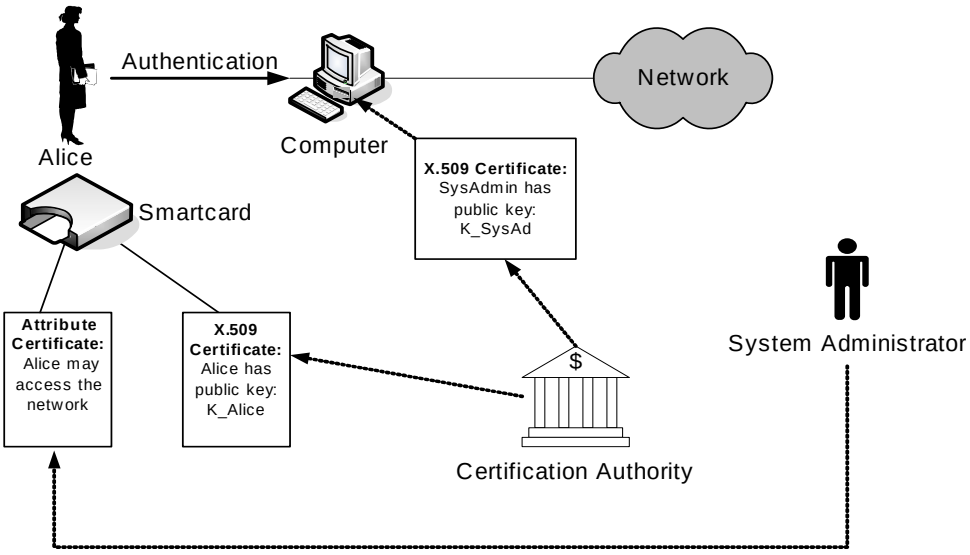


Fig. 10.2-14: Access control with attribute certificates.

The distribution of authorization information is just one of many possible uses for attribute certificates. Basically, an attribute certificate can be used to bind any kind of information to an entity. A proposal for the standardization of attribute certificates was made by the ANSI X9 committee. ITU-T included this proposal in X.509v3 [IT97b].

10.2.4 Authorization and Delegation Certificates

Besides the two types of certificates already presented there is a third important kind of certificate - the authorization certificate. The main application for such certificates is authorization in distributed environments, like the Internet, where no central trusted authorities exist. An important new mechanism provided by authorization certificates is the possibility to delegate certified rights to other entities.

An authorization certificate is a digitally signed piece of information that assigns a subject, usually represented in the form of a public key, one or more permissions, which allows the subject to perform specified actions on one or more specified objects in a target system. Authorization certificates are also referred to as *signed capabilities*. Fig. 10.2-15 shows a typical authorization certificate.

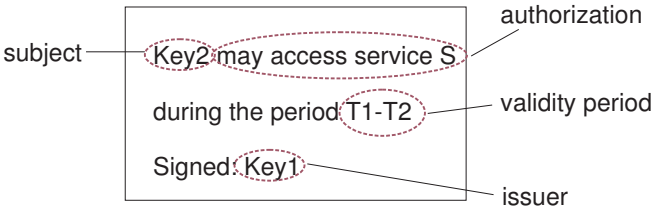


Fig. 10.2-15: An authorization certificate.

A delegation certificate is an authorization certificate, that allows the subject to delegate the assigned permissions further. A delegation certificate has the following meaning:

S_{Key1}(during the validity period $T_1 - T_2$, if I have any of the rights R I give the rights also to *Key2*).

$S_{\text{Key1}}(...)$ denotes a signed message that includes both the signature and the original message. The key that signed the certificate (*Key1*) is the issuer and the key to whom the rights are given (*Key2*) is the subject of the certificate, and the rights R given by the certificate are the authorization. With the delegation certificate, the issuer delegates the rights to the subject. The certificate becomes invalid outside the validity period $T_1 - T_2$. The validity period can be exploited to regulate the amount of trust the issuer places into the subject. By specifying extremely short validity periods the issuer can force the subject to frequently make online connections to the issuer. In practice, authorization and delegation certificates are often treated as synonyms.

An important difference to attribute certificates is, that delegation certificates are key-oriented. Subjects in delegation certificates are not identified by a name, but by their public key. The chance that two persons have the same public key (e.g. 512 bit RSA keys) is so low that it can be neglected. Hence, a public key is truly a globally unique identifier.

The use of the public key as identifier leads to a considerable simplification: the same authorization that requires the combination of an identity certificate and an attribute certificate can be expressed in a single delegation certificate (see Fig. 10.2-16). There is no need for a trusted CA and every key may delegate its rights to any other key. Hence, delegation certificates allow egalitarian certification structure comparable to the web of trust. In order to bound the delegation chain, an issuer can specify whether he allows the granted rights to be delegated.

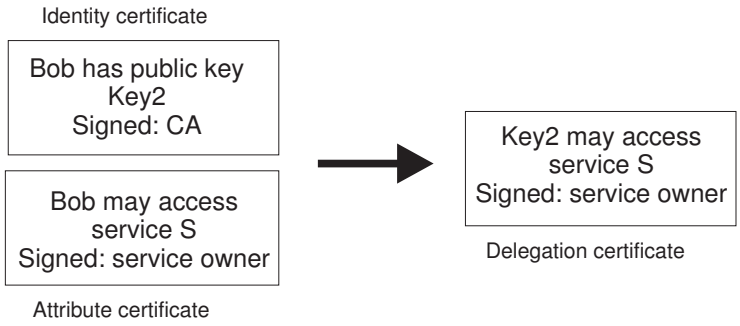


Fig. 10.2-16: Delegation certificates avoid trusted CAs.

If we use delegation certificates in our access control example, the system can be further simplified. Fig. 10.2-17 shows the system and it can be observed, that the CA was removed from the system. Instead, the system administrator directly binds the access authorization to Alice's public key. Alice only has to present one piece of evidence to the accessed computer: the authorization certificate. Each computer securely stores the reduced ACL, which only consists of the administrator's public key. When an authorization certificate is presented by a user, the accessed computer employs the administrator's public key to verify the access authorization.

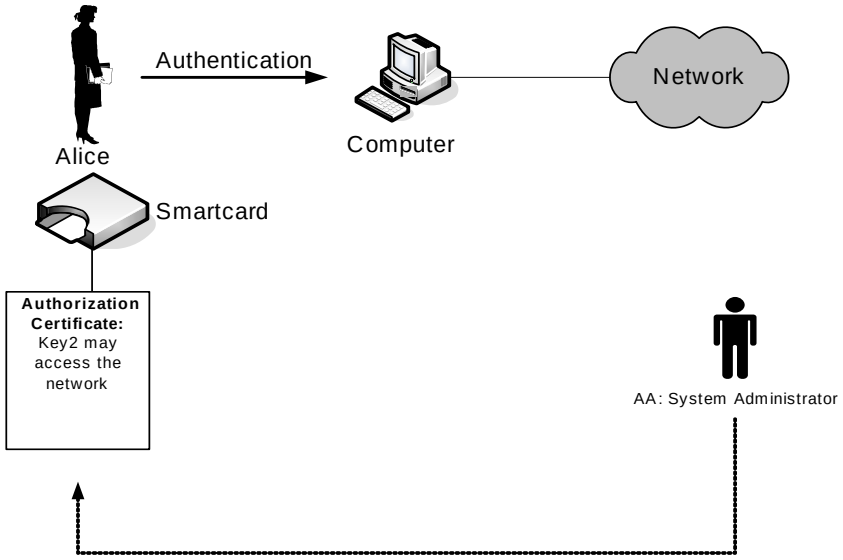


Fig. 10.2-17: The access control system employing authorization certificates.

Normally, the administrator would not allow Alice to further delegate this access right. On the other hand, there can be situations where such a delegation could be useful. If Alice wants to delegate the access right she does not have to reveal a password - thus compromising system security - but can create a delegation certificate herself (see Fig. 10.2-18). According to the trust in the other entity, she can limit the access rights and the validity period ($T3 - T4 < T1 - T2$, $R3 \subseteq R2$).

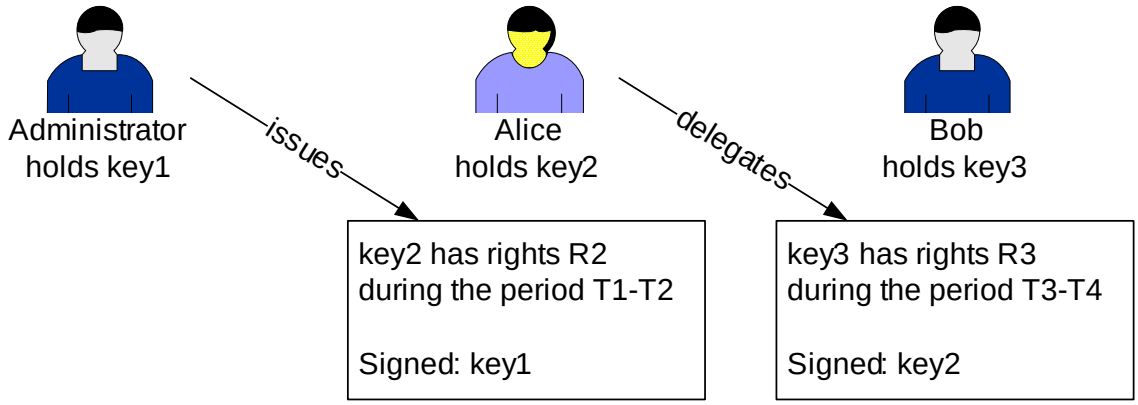


Fig. 10.2-18: Delegation of access rights $T3 - T4 < T1 - T2$, $R3 \subseteq R2$.

10.3 Important PKI Standards

10.3.1 X.509

The X.509 version 1 certificate format (X.509v1) was first published in 1988 by the ITU-T. It was first extended in 1993 to become X.509v2. As a result of attempting to deploy certificates within the Internet, X.509v2 was revised to hold additional extension fields. The resulting X.509v3 format was officially released in 1996 [IT97b]. X.509 certificates are formatted according to the ASN.1 abstract syntax notation. The details of ASN.1 are beyond the scope of this book.

X.509 Identity Certificates

An X.509 certificate has the following structure:

```
Certificate ::=      SEQUENCE {
tbsCertificate       TBSCertificate,
signatureAlgorithm   AlgorithmIdentifier,
signature            BIT STRING }
```

The field `tbsCertificate` holds the data part, the field `signatureAlgorithm` identifies the chosen signature algorithm and the field `signature` carries the signature of the certificate.

The signature algorithm is identified by an `Object Identifier`, OID.

```
Algorithm Identifier ::= SEQUENCE {
algorithm                OBJECT IDENTIFIER,
parameters               ANY DEFINED BY algorithm   OPTIONAL }
```

The field `algorithm` specifies a combination of hash function and signature algorithm. An application has to list supported signature algorithms, including the allowed parameters. The field `parameter` holds the parameters of the signature algorithm.

The field `signature` carries the signature, which is created by employing the identified algorithm in combination with the CA's private (signature) key on the data section (`tbsCertificate`).

The field `tbsCertificate` consists of a sequence of subfields, which make up the data part of the certificate (ASN.1 syntax).

```
TBSCertificate         ::= SEQUENCE {
   version                  [0] EXPLICIT Version DEFAULT v1,
   serial number            CertificateSerialNumber,
   signature                AlgorithmIdentifier,
   issuer                   Name,
   validity                 Validity,
   subject                  Name
   subjectPublicKeyInfo     SubjectPublicKeyInfo,
   issuerUniqueID           [1] IMPLICIT UniqueIdentifier   OPTIONAL
   subjectUniqueID          [2] IMPLICIT UniqueIdentifier   OPTIONAL
   extensions               [3] EXPLICIT Extensions         OPTIONAL }
```

{version} Designates the version: v1, v2, or v3.

{serial number} A unique number is used to identify the certificate, e.g. for use in certificate revocation lists.

{signature} Used for identification of the employed signature algorithm. It must not be confused with the field holding the actual signature. The structure and content is identical to the field `signatureAlgorithm`.

{issuer} This field holds the name of the issuer (CA). The name must be a distinguished name conforming to X.500-Syntax (see Table 10.2-1).

{validity} This field specifies the validity window and holds the subfields `notBefore` and `notAfter`.

{subject} Here the distinguished name of the subject is given.

{subjectPublicKeyInfo} This field specifies the subject's public key (subfield `subjectPublicKey`) and the according algorithm (subfield `algorithm`) that must be used with this key.

{issuerUniqueID} This field contains an optional bit string used to make the CA name unambigious in the case that the same name was reassigned to different enities through time.

{subjectUniqueID} Same as above for subject.

{extensions} X.509v3 certificates may contain extension fields that can hold arbitrary data. An application can theoretically define an unlimited number of extension fields. For instance, the extension fields may hold authorization information. The drawbacks of placing authorization information in identity certificates have been illustrated in Section 10.2.3. An extension field has three parts: extension type (`extnId`), extension value (`extnValue`), and criticality indicator (`critical`). The extension type is a gobally unique identifier that references the syntax and semantics of the extension value. The extension value holds the actual value of an extension field. Finally, the criticality indicator is a flag that instructs a certificate-using application whether it

is safe to ignore the extension field if it does not recognize the extension type. X.509v3 defines several standard extension fields, typical examples are fields that contain information about key usage (electronic signature, encryption, etc.), certification path constraints, CRLs, and certificate policies. Besides the standard extensions, applications may define private extensions for their own use (e.g. email address, server URL, etc.).

X.509 Attribute Certificates

The ITU-T X.509 standard also defines an attribute certificate format. It has the following structure:

```
AttributeCertificateInfo  ::= SEQUENCE {
   version                Version DEFAULT v1,
   subject                CHOICE {
      baseCertificateID   [0] IssuerSpecial,
      subjectName         [1] GeneralNames },
   issuer                 GeneralNames,
   signature              AlgorithmIdentifier,
   serial number          CertificateSerialNumber,
   attrCertValidityPeriod AttCertValidityPeriod,
   attributes             SEQUENCE OF Attribute,
   issuerUniqueID         IMPLICIT UniqueIdentifier   OPTIONAL
   extensions             EXPLICIT Extensions         OPTIONAL }
```

{version} The version number differentiates between different versions of the attribute certificate. Currently, the version number is v1.

{subject} This field conveys the identity of the certificate's subject, referenced either by the `serialNumber` associated with the subject's X.509 identity certificate or by the `subjectName`.

{issuer} This is the name of the Attribute Authority (AA) who created the attribute certificate.

{serialNumber} This number uniquely identifies the attribute certificate.

{signature} This field identifies the cryptographic algorithm used to digitally sign the attribute certificate.

{attCertValidityPeriod} This field conveys the time period during which the attribute certificate is considered valid.

{attributes} This field contains the actual attributes that are bound to the subject.

{issuerUniqueID} This field uniquely identifies the isuuer of the attribute certificate in those instances where the issuer name is not sufficient.

{extensions} This field allows the addition of new fields to the attribute certificate.

The X.509v3 standard leaves a lot of room for applications to adjust the certificates to their requirements. Many of the standard and extension fields can take a wide range of options. Nevertheless, this flexibility makes it extremely difficult to produce independent implementations that are able to operate with one another. To overcome this problem, templates for certificates and procedures are created, which are referred to as profiles. A profile defines which extension fields must be supported and what values the certificate fields might carry for a certain type of application.

10.3.2 PKIX

The IETF Internet X.509 Public Key Infrastructure (PKIX) Working Group formed in 1995 to develop standards for an Internet PKI based on X.509 certificates. These standards consist of X.509v3 and CRL profiles, as well as several protocols for PKI services. The specifications are documented in a family of RFC documents. These RFCs can be grouped in five areas:

Profiles This first area involves profiles of the X.509v3 certificate and the X.509v2 CRL standards for the Internet.

Operational protocols These protocols can be used by participants to retrieve certificates. Protocols defined so far use LDAP (Lightweight Directory Access Protocol), HTTP, FTP, and X.500 for certificate distribution.

Management protocols This area covers management protocols, in which different entities in the system exchange information needed for proper management of the PKI. For example, a management protocol can convey information about a revocation request from a client to a CA. The registration information from a registration authority (RA) to a CA is another application for a management prototcol.

Policy outline This section deals with certificate policies and certificate practice statements, covering the areas of PKI security not directly addressed in the rest of PKIX. RFC 2527 [Chokhani99] defines a certificate policy and certificate practices framework.

Time stamping and data certification PKIX provides time stamping via a Time Stamping Protocol (TSP). Furthermore, the working group proposes a data certification service to provide notary functionalities.

10.3.3 SPKI

The Simple Public-Key Infrastructure (SPKI) working group was built by the IETF to standardize a PKI based on authorization and delegation certificates as described in Section 10.2.4. The SPKI team addresses the problem that the name of a key holder which is conveyed by traditional identity certificates is rarely of security interest. A user of a certificate has to know whether a given key holder has been granted some specific authorization.

There are two types of certificates in SPKI:

1. The basic certificate is an authorization certificate which communicates a permission from one principal to another principal.
2. The second type is a name certificate which maps a name in the issuer field to either a key or a name in the subject field.

In this book, we will only describe the authorization certificate. An SPKI authorization certificate is a 5-tuple and contains the following fields (similar to the certificate displayed in Fig. 10.2-15):

```
<Issuer, Subject, Delegation, Authorization, Validity>
```

Issuer The cryptographic key issuing this certificate and granting the permission or rights it communicates.

Subject The cryptographic key acquiring the permission or rights.

Delegation A flag noting whether the Subject is also acquiring the right to delegate all or part of the permission it acquires through this certificate.

Authorization {<tag>} A field specifying the permission being communicated.

Validity A specification of the dates or online conditions under which the certificate is assumed to be valid. The dates have the form `YYYY-MM-DD HH:MM:SS`.

The starting point of a SPKI certificate chain is always an ACL entry. An ACL entry is like a certificate, except that it has no issuer and is not signed. It is protected in the application by other means.

Example 10.3-1:
In the access control example shown in Fig. 10.2-17 the ACL only holds the public key of the administrator and is securely stored in every computer that can be accessed. The administrator delegates the right to access the computer network to Alice, but does not allow her to further delegate this right.

The data format chosen for SPKI is called S-expression. This is a parenthesized expression with the limitations that empty lists are not allowed and the first element

in any S-expression must be a string, called the "type" of the expression. Alice's SPKI certificate may look as follows:

```
(cert
   (issuer
      (public-key
         (rsa-pkcs1-md5
            (e #03#)
            (n
             |ANHCG85jXFGmicr3MGPj53FYYSY1aWAue6PKnpFErHhKMJa4HrK4WSKTO
             YTTlapRznnELD2D7lWd3Q8PD0lyi1NJpNzMkxQVHrrAnIQoczeOZuiz/yY
             VDzJ1DdiImixyb/Jyme3D0UiUXhd6VGAz0x0cgrKefKnmjy410Kro3uW1|)))
       )
   (subject
      (public-key
         (rsa-pkcs1-md5
            (e #11#)
            (n
             |ALNdAXftavTBG2zHV7BEV59gntNlxtJYqfWIi2kTcFIg
             IPSjKlHleyi9s5dDcQbVNMzjRjF+z8TrICEn9Msy0vXB0
             0WYRtw/7aH2WAZx+x8erOWR+yn1CTRLS/68IWB6Wc1x8h
             iPycMbiICAbSYjHC/ghq2mwCZO7VQXJENzYr45|)))
 )
   (tag (access acme.org/alice/* (read write) )
   (not-before "1998-03-01_12:42:17")
   (not-after "2012-01-01_00:00:00")
)
{KDQ6Y2VydCg2Omlzc3Vlcig0Omhhc2gzOm1kNTE2OlLfZteG8j83WwjWVnQ
4zUIpKSg3OnN1YmplY3QoMTE6b2JqZWN0LWhhc2goNDpoYXNoMzptZDUxNjq
zMpKVIr5I3MiwcffCMCxNMTI6cnVuZW1hY3MuZXhlKSkpKDM6dGFnMTA6dml
ydXMtZnJlZSkp}
```

Issuer and subject are identified by their public keys. The public keys consist of an identifier (including hash function) and the public-key parameters e and n. PKCS1 [Kohnfelder78] is a document published by RSA which describes how RSA is employed for encryption and digital signatures. The public exponent e is encoded in hexadecimal notation, the modulus n is encoded in base64 notation[42]. The authorization to access the computer network with read/write permissions in her home directory is given within the `tag` field. The certificate is digitally signed by the administrator (base64 encoded). If the administrator would allow Alice to delegate her access right the certificate would change to

```
...
     (propagate)
     (tag (access acme.org/alice/* (read write) )
...
\
```

42 65 character subset of ASCII, each character represents 6 bits.

10.3.4 OpenPGP

Pretty Good Privacy (PGP) is an email encryption programme written by Phil Zimmerman in 1991 at MIT. Zimmerman wanted to create a solution that could be easily used over the Internet. He released the programme as freeware and also published the source code. PGP soon became a great success and currently is the only working PKI solution for the global Internet. PGP does not rely on "official" certification authorities, but allows any user to certify other users. This model became publicly known as *Web of Trust* (see Fig. 10.2-12). Due to the immense success the PGP message and certificate format were published as Internet standards in RFC 2440 [Callas98] under the name OpenPGP. It must be noted that OpenPGP not only supports the Web of Trust certification model but also the hierarchical certification structures based on CAs. Hence, OpenPGP is not only used by private users for the Internet but also by large corporations.

An OpenPGP certificate is a kind of meta certificate which may contain several keys, user IDs and signatures. OpenPGP certificates are normally referred to as "PGP Key", not "PGP certificate". The general format of an OpenPGP(v4) key looks as follows:

```
Primary-Key
      [Revocation Self Signature]
      [Direct Key Self Signature...]
       User ID [Signature ...]
      [User ID [Signature ...] ...]
      [[Subkey [Binding-Signature-Revocation]
           Primary-Key-Binding-Signature] ...]
```

Entries in square brackets are optional and ellipses indicate repetition. An OpenPGP key holds one primary key and may contain several subkeys. If the key contains subkeys, the primary key *must* be a signing key. The subkeys may be keys of any type (encryption keys, signing keys). The primary key may contain several user IDs, which may carry several signatures. The Primary-key-name binding must include a self-signature. This is the part `User ID [Signature]` without brackets. Hence, a PGP key at least consists of a primary key, a user ID and a self-signature carried out by the primary key. Futhermore, there are some optional fields:

{Revocation Self Signature} A PGP key carrying this signature is revoked. The signature is calculated directly on the key being revoked. A revoked key is not to be used. Only revocation signatures by the key being revoked, or by an authorized revocation key, should be considered valid revocation signatures.

{Direct Key Self Signature} This signature is calculated directly on a key. Such a signature may contain information about the key that non-self certifiers want to make about the key itself (e.g. authorized revocation key), rather than the binding between a key and a name.

{Signature} There are four types of signatures which declare the amount of certainty in the authentication of the public key to be signed:

1. Generic certification of a User ID and Public Key packet. The issuer of this certification does not make any particular assertion as to how well the certifier has checked that the owner of the key is in fact the person described by the User ID.
2. Persona certification of a User ID and Public Key. The issuer of this certification has not done any verification of the claim that the owner of this key is the User ID specified.
3. Casual certification of a User ID and Public Key. The issuer of this certification has done some casual verification of the claim of identity.
4. Positive certification of a User ID and Public Key. The issuer of this certification has done substantial verification of the claim of identity.

{Binding-Signature-Revocation} The signature is calculated directly on the subkey being revoked. A revoked subkey is not to be used. Only revocation signatures by the primary signature key that is bound to this subkey, or by an authorized revocation key, should be considered valid revocation signatures.

{Primary-Key-Binding-Signature} This signature is a statement by the primary signing key and indicates that it owns the subkey. This signature is calculated directly on the subkey itself, not on any User ID.

Every key holder signs the binding of primary key and user ID himself, which is referred to as self-certificate.

Example: Bob's PGP key:

```
Primary Key: DSA (Signature)
   User ID: bob<bob@acme.org>
      signed by: primary key (self-certificate)
      signed by: Alice
      signed by: VerySane CA
   User ID: bob<bob@acme.com>
      signed by: primary key (self-certificate)
      signed by: Carol
   Subkey: RSA (Encryption & Signature)
      signed by: primary key
```

In this example the primary key is only used to certify other users. The RSA subkey is used for email encryption, data integrity and data origin authentication.

The PGP keys can be distributed to other users either by personal exchange or by uploading the key to a directory server. Other users can download the keys from users they want to communicate with. Key known to the user are stored in keyrings. Public keys are stored in the public keyring, private keys are stored in the secret

keyring. The secret keyring is protected by a password (also passphrase or mantra) and should be stored in a safe place (e.g. on a floppy disk). PGP software normally includes a front-end to display the keys available in the user's keyring.

If a user wants another user to act as an introducer, he must assign a trust value to this user's key (he thus sets a trust anchor). PGP supports the following trust levels:

- untrustworthy
- don't know
- marginal
- full.

If the user is fully trusted, he can act as an introducer. If the user is marginally trusted, another signature from a (marginally) trusted user is needed before this user may act as an introducer. PGP supports the direct trust model as displayed at the top of Fig. 10.2-5.

Example:
Fig. 10.3-1 shows an example for PGP's Web of Trust. Alice authenticates and certifies Carol and Don who both certify Jane. Although Alice fully trusts Jane she has not authenticated herself and she has to rely on the authentication carried out by Carol and Don. Since Alice only marginally trusts Carol and Don, there have to be two signatures on Jane's key before Alice is willing to accept the key as valid. After Jane is authenticated Alice can use her as introducer for Bob. Since Alice fully trusts Jane a single signature on Bob's key from Jane suffices to authenticate him.

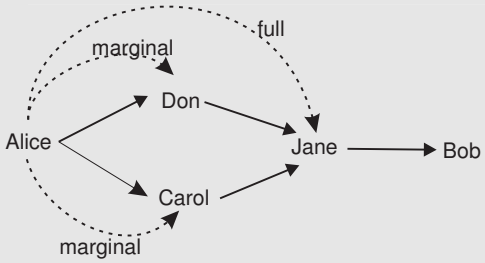


Fig. 10.3-1: Example for PGP's Web of Trust: Alice indirectly authenticates Bob.

Assignments

Assignments for Chapter "Introduction"

Assignment 1: **15 P.**

a) In which categories can encryption systems be divided? **2 P.**

b) Recall Example 1.4-1. Define an encryption function E_{k_3} and a decryption function D_{k_3} which are not the identity functions. **3 P.**

c) Recall Example 1.4-1. Encrypt the message sequence $m = m_1, m_3, m_2$ under the key k_2. **3 P.**

d) Explain the following two attacks: **4 P.**

- Chosen-ciphertext attack
- Chosen-plaintext attack.

e) Recall Example 1.4-2 with $k = 7$. Encrypt the message sequence $m = 7, 5, 1$. **3 P.**

Assignment 2: Suppose there are two persons A and B using an asymmetric-key encryption system. Both have generated a matched pair of private and public keys: $(k_{d,A}, k_{e,A})$ for A and $(k_{d,B}, k_{e,B})$ for B. After the key generation process A and B exchange their public keys: $k_{e,A}$ from A to B and $k_{e,B}$ from B to A. Suppose entity A wants to send a message m, encrypted as ciphertext c, to B. **4 P.**

a) How can A generate the ciphertext c from the message m? **1 P.**

b) How can B decrypt the ciphertext c to receive the message m? **1 P.**

c) B isn't able to decrypt the ciphertext c. What could have happened? **2 P.**

Assignment 3: Install the Crypto-Calculator on your PC or use the online version of the Crypto-Calculator. You can find the links on the book home page. Download the manual for the Crypto-Calculator. **6 P.**

a) Compute the function $2345^{6789} \mod 377$ using the function `mod.exp`. **? P.**

b) What cryptographic encryption and decryption functions are implemented in the Crypto-Calculator. What types of inputs and outputs are available? **2 P.**

c) Encrypt the message 193243434 with the key 1599999 for all three encryption functions. Perform the reverse operation as well and write the result in a table. **2 P.**

12 P. **Assignment 4**: As mentioned in the course, Cryptography is the science of the methods of encryption and decryption. One simple example for an encryption algorithm is the **Caesar encryption algorithm** invented by Caesar. There are many slightly different variants of the algorithm. We will use the following variant for this assignment:

As **alphabet** we use only upper case letters (A-Z). The characters of the alphabet in a given plaintext are encrypted by a cyclic shift of each character depending on a given **key**. The key is a choosen character from the alphabet and the position of this character in the alphabet determines by how many positions each character must be shifted, e.g. for the key "B" a cyclic shift of two characters takes place ($A \rightarrow C$, $B \rightarrow D$, ..., $Z \rightarrow B$). Characters (especially blanks) which are not part of the alphabet are left untouched by the encryption algorithm.

Example:
Plaintext: CRYPTOGRAPHY and CRYPTOANALYSIS

Encryption key: B

Encrypted text: ETARVQITCRJA and ETARVQCPCNAUKU

5 P. a) Encrypt the plaintext "CRYPTOGRAPHY AND CRYPTOANALYSIS" with the given algorithm and the key "E".

4 P. b) A special case of the Caesar encryption algorithm is the so called **ROT13 encryption algorithm**. ROT13 uses the key "M" for encryption which shifts each character by 13 places. What is special about the ROT13 encryption?

3 P. c) Why should "Z" not be used as encryption key?

12 P. **Assignment 5**: The following text is encrypted with the variant of the Cesar encryption algorithm given in Assignment 1.

LQ WKHLU DSSOLFDWLRQ FUBSWRORJLFDO PHWKRGV DUH QRW ORQJHU DOLJQHG WR WKH DFTXLVLWLRQ RI SULYDFB RQOB. WKH UDQJH RI DSSOLFDWLRQV WRGDB FRQWDLQV DOVR OLPLWHG DFFHVV IRU HADPSOH WR SDB-WY, HOHFWURQLF VLJQDWXUH VFKHPHV WR HQVXUH DXWKHQWLFLWB RU DOVR WR EXLOG VBVWHPV IRU HOHFWURQLF SDBPHQWV.

Determine the used encryption key and the plaintext.

Solution hints: Instead of trial-and-error you can use a cryptoanalytic approach. Calculate a frequency distribution for the characters of the encrypted text and compare it with the frequency distribution of the English language.

In various English texts of 1000 characters, the alphabet occurs with about the following relative frequencies:

A	B	C	D	E	F	G	H	I	J	K	L	M
73	9	30	44	130	28	16	35	74	2	3	35	25

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
78	74	27	3	77	63	93	27	13	16	5	19	1

Assignments for Chapter "Mathematical Background"

Assignment 6: Prove that $(\mathbb{Z}_p^*, \times_p)$ is a group. ($\times_p$ is the multiplication modulo p, p is prime). **5 P.**

Assignment 7: Compute the following expressions: **6 P.**

a) $17^{-1} \mod 101$ **1.5 P.**

b) $28^{-1} \mod 75$ **1.5 P.**

c) $21/2 \mod 28$ **1.5 P.**

d) $701/357 \mod 1234.$ **1.5 P.**

Assignment 8: In the following the set $\mathbb{Z}_{14}^*$ will be dealt with in greater detail. **5 P.**

a) Find the elements of the set $\mathbb{Z}_{14}^*$. **1 P.**

b) Compute the order of the elements of $\mathbb{Z}_{14}^*$. **3 P.**

c) Find the generators of $\mathbb{Z}_{14}^*$. **1 P.**

Assignment 9: Determine x from the following system of simultaneous congruences: **8 P.**

$$x \equiv 12 \mod 25$$
$$x \equiv 9 \mod 26$$
$$x \equiv 23 \mod 27.$$

Assignment 10: Let $p(x) = x^5 + x^2 + 1 \in \mathrm{GF}(2)$ be an irreducible polynomial. **8 P.**

a) Construct $\mathrm{GF}(2^5)$ over the polynomial $p(x)$. The elements of $\mathrm{GF}(2^5)$ should be represented as polynomials in $\mathrm{GF}(2)$. **2 P.**

2 P. b) Compute $(x^4 + x^2 + x + 1) \cdot (x^2 + 1)$ in GF(2^5).

1 P. c) Find the inverse of $(x^2 + 1)$ in GF(2^5).

3 P. d) Prove that $p(x)$is a primitive polynomial.

4 P. **Assignment 11**: Determine whether the number 1729 is prime or not using the following tests:

2 P. a) Fermat's test.

2 P. b) Miller-Rabin test.

3 P. **Assignment 12**: Use Pollard's rho algorithm to factorize 221.

4 P. **Assignment 13**: Using the "baby step" "giant step" method, determine x from the equation $7^x = 5$ in $\mathbb{Z}_{17}^*$.

Assignments for Chapter "Stream Ciphers"

20 P. **Assignment 14**: We consider an LFSR with the feedback polynomial

$$c(x) = x^5 + x^2 + 1 \in \mathrm{GF}(2)[x]$$

and an initial state

$$\underline{s}_0 = (s_0, s_1, s_2, s_3, s_4)^T = (1, 0, 0, 0, 0)^T \in \mathrm{GF}(2)^5.$$

Further, we consider a nonlinear filter generator (NLFG) with the feedback polynomial c, an initial state

$$\underline{s}_0 = (s_0, s_1, s_2, s_3, s_4)^T = (1, 0, 0, 0, 0)^T \in \mathrm{GF}(2)^5$$

of the LFSR, the filter function $f : \mathrm{GF}(2)^3 \to \mathrm{GF}(2)$, given in algebraic normal form (ANF) as

$$f(x_1, x_2, x_3) = x_1x_2 + x_3,$$

and the tapping sequence

$$\Gamma = (\gamma_1, \gamma_2, \gamma_3) = (0, 2, 3).$$

The operations of f in the ANF are addition and multiplication modulo 2.

a) Which functions and procedures are available in the Crypto-Interpreter to handle LFSRs? Describe the functionality, the input and output types! **4 P.**

b) Build a table with the current state **5 P.**

$$\underline{s}_t = (s_t, s_{t+1}, \ldots, s_{t+4})^T \in \mathrm{GF}(2)^5$$

of the LFSR at time t, $0 \leq t \leq 10$.

c) What is the period of the output sequence $s = s_0, s_1, \ldots$ of the LFSR? **2 P.**

d) Is the feedback polynomial c primitive? Give a reason for your answer. **2 P.**

e) Compute the output of the filter function f for all possible input values. **2 P.**

f) Determine the output sequence $z = z_0, z_1, \ldots$, of the nonlinear filter generator for $0 \leq t \leq 10$. **5 P.**

Assignment 15: We consider the Boolean function $f : \mathrm{GF}(2)^3 \rightarrow \mathrm{GF}(2)^2$, **20 P.**

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3,$$

from assignment 1. Now the inputs $s_t \in \mathrm{GF}(2)$, $t \geq 0$, for f are from a true random source, i.e. the s_t are balanced and statistically independent for any $t \geq 0$. The output z_t is given by

$$z_t = f(s_t, s_{t+2}, s_{t+3})$$

for $t \geq 0$.

a) Give an exact mathematical description of " the s_t are balanced and statistically independent for any $t \geq 0$" ! **2 P.**

b) Which inputs are needed to produce the output elements (z_t, z_{t+1})? **2 P.**

c) Compute a truth table for the output (z_t, z_{t+1}) for all possible input values. Arrange the input s_t according to their occurence in time. **8 P.**

d) Which inputs lead to the output $(z_t, z_{t+1}) = (0, 0)$? **4 P.**

e) What is the value of the conditional probability **4 P.**

$$P(s_{t+3} = 0 | z_t = 0 \wedge z_{t+1} = 1)?$$

Assignments for Chapter "Block Ciphers"

6 P. **Assignment 16**: Do some exercises with the Java-Applet on the modes of operation on the book home page and write down some results (plain-ciphertext pairs, IVs, key) for each algorithm and mode.

4 P. a) Choose some texts, encrypt them with DES and IDEA in ECB, CBC, OFB and CFB mode.

2 P. b) Insert some transmission error and observe how this error affects the decryption in each mode.

11 P. **Assignment 17**: We consider in this exercise the DES algorithm.

3 P. a) What is the bit length of the key k, the plaintext m and the ciphertext c for the DES algorithm?

2 P. b) Which functions can be found in the Crypto-Interpreter for the DES Algorithm? Describe the functionality, the input and output types!

3 P. c) Encrypt the message sequence $m_0 = 456$, $m_1 = 312$ and $m_2 = 456$ with DES in the ECB mode under the key $k = 4534$. Write down the encryption and decryption process and the ciphertext sequence c_0, c_1 and c_2.

3 P. d) Encrypt the message sequence $m_0 = 456$, $m_1 = 312$ and $m_2 = 456$ with DES in CBC mode under the key $k = 4534$ and initial value $\mathrm{IV} = 3241$. Write down the encryption and decryption process and the ciphertext sequence c_0, c_1 and c_2.

5 P. **Assignment 18**: In this exercise we consider the IDEA key schedule algorithm. Suppose that the cipher key is $k = \texttt{0001000200030004000500060007000 8}$ in hexadecimal. Compute the six subkeys $k_1^{(j)}$, $k_2^{(j)}$, $k_3^{(j)}$, $k_4^{(j)}$, $k_5^{(j)}$, $k_6^{(j)}$, $k_7^{(j)}$, and $k_8^{(j)}$ for each round j, $1 \leq j \leq 8$, and the four keys for the output operation.

7 P. **Assignment 19**: Sketch a picture (or block-diagram) which describes roughly the encryption algorithm in AES! What are the most important steps?
(As example, see how the IDEA algorithm is sketched in Fig. 4.4-1 and Fig. 4.4-2).

Assignment 20: **5 P.**

a) Prove the following statement: When an attacker can factorize the modulus n of the RSA-scheme, then he can compute the secret key $k_{d,A}$ of the user A. **2 P.**

b) Show how the receiver of an RSA-encrypted message can speed up the decryption operation using the Chinese Reminder theorem! **3 P.**

Assignment 21: Can the pair $(e = 1123, n = 117739)$ be an RSA encryption key? If so, then compute the decryption key! **3 P.**
For the computations use the functions of the Crypto-Calculator (`isprime, gcd, factor, rcp` and other).

Assignments for Chapter "Public-Key Encryption"

Assignment 22: Can we use the group $\mathbb{Z}_{23}^*$ and the element $g = 9$ in it as public elements of ElGamal encryption scheme? Explain why! **2 P.**
If you require some computation use the functions from the Crypto-Calculator (e.g. `mod_exp`).

Assignment 23: **5 P.**

a) Compute all points on the elliptic curve $E_{11}(7, 6)$. **2.5 P.**

b) On the curve $E_{11}(7, 6)$ compute the point $Q = 3P$ if the point P is $P(5, 1)$! Explain all computational steps and do not use the Java Applet! Because in the book we have not dealt with the algorithms for finding a square root of element in Z_n, we square all elements of Z_{11} in order to take the square root. You can use the following table for finding the required square roots: **2.5 P.**

y	0	1	2	3	4	5	6	7	8	9	10
y^2 in Z_{11}	0	1	4	9	5	3	3	5	9	4	1

Assignment 24: For exchanging confidential messages Alice and Bob use the ElGamal encryption scheme with elliptic curves. As public elements of the system they use the curve $E_{17}(7, 8)$ and the point $G(2, 8)$ which is a generator of the group $(E_{17}(7, 8), +)$. **5 P.**
Alice knows that Bob's public key is $K_b = (12, 16)$. She wants to send him the message "Christmas" which is encoded to the point $M(13, 1)$ (using some algorithm). Suppose that Alice uses the random parameter $k = 3$ for masking the message.

For the computational operations in the elliptic curve group please use the Java-Applet on the course homepage!

1.5 P. a) Which ciphertext C does Alice send?

2 P. b) Can you guess Bob's private key?

1.5 P. c) Show how Bob decrypts the message?

Assignments for Chapter "Digital Signatures"

6 P. **Assignment 25**: What are the properties of an electronically signed document? The explanation of the properties is also required.

4 P. **Assignment 26**: Compute the RSA signature (without hash function) of $m = 11111$ using $n = 28829$ and the smallest possible exponent.

10 P. **Assignment 27**: RSA signatures: To send the signed message $m = (11001011)_2$ to Bob, Alice does the following: To create a matched pair of private and public keys, she takes two primes, $p = 13$ and $q = 19$, to compute her public key $e = 25$. Solve the following tasks:

2 P. a) Compute Alice's private key d.

2 P. b) Create the digital signature for the message m without applying a hash function to the message.

3 P. c) Alice has sent the signed message m to Bob. How can Bob verify Alice's signature?

3 P. d) Message m and signature s have not been encrypted by Alice before transmission, enabling an attacker to alter two bits of the message. Bob receives $m' = (11010011)_2$. How can he decide whether the message has been altered or not since it was signed by Alice?

Assignments for Chapter "Hash Functions and Authentication Codes"

9 P. **Assignment 28**: Explain why it is necessary for a hash function to be like the three properties shown below. Which kind of attacks can we avoid with these properties?

a) Weakly collision free **3 P.**

b) Strongly collision free **3 P.**

c) One-way function. **3 P.**

Assignment 29: Explain why it is recommended that the minimal size of a hash value should be 128 bits! **2 P.**

Assignment 30: **4 P.**

a) Explain how hash functions are generally constructed! **2 P.**

b) How can hash functions be classified with respect to the design of the compression function? Explain briefly what is the main characteristic of each class! **2 P.**

Assignment 31: **4 P.**

a) Explain how MACs based on block ciphers work! **2 P.**

b) Explain how can we construct a MAC from a hash function! **2 P.**

Assignment 32: Use the online Crypto-Calculator to compute the following hash values and comment the results: **3 P.**

a) Hash of ”Die Pruefung werde ich bestehen” with MD5 **0.5 P.**

b) Hash of ”die pruefung werde ich bestehen” with MD5 **0.5 P.**

c) Hash of ”Die Pruefung werde ich bestehen” with SHA **0.5 P.**

d) Hash of ”die pruefung werde ich bestehen” with SHA **0.5 P.**

e) Hash of ”Die Pruefung werde ich bestehen” with squaring modulo 10552043297 **0.5 P.**

f) Hash of ”die pruefung werde ich bestehen” with squaring modulo 10552043297 **0.5 P.**

Assignments for Chapter "Entity Authentication"

6 P. **Assignment 33**: Illustrate the functionality of the Feige-Fiat-Shamir identification protocol by means of an example considering the *selection of system parameters*, the *selection of per-entity secrets*, and the *protocol actions*.

6 P. **Assignment 34**: Illustrate the functionality of the GQ identification protocol by means of an example considering the *selection of system parameters*, the *selection of per-entity secrets*, and the *protocol actions*.

Assignments for Chapter "Key Management Techniques"

6 P. **Assignment 35**: Give a short description of the following procedures: storing, updating, and destroying of keys.

10 P. **Assignment 36**: The Diffie-Hellman key exchange protocol can be extended to work with three or more persons. Write down the steps which must be done for the DH key exchange with three persons.

Assignments for Chapter "Public Key Infrastructure"

6 P. **Assignment 37**: Examine the certificate you have received from the FernUniversität in Hagen. In the case you don´t have a certificate, take one from https://ca.fernuni-hagen.de/ and summarize your X.509 certificate attributes.

4 P. **Assignment 38**: Use your favorite web browser and look at the web site https://meine.deutsche-bank.de. Why does your bowser trust the certificate issued by this webserver?

Assignment 39: You have opened (browsed) the web page https://ca.fernuni-hagen.de/ and your browser displays a warning about an unknown certificate. The following details are shown: **5 P.**

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=DE, ST=Nordrhein-Westfalen, L=Hagen,
        O=FernUniversitaet in Hagen,
                OU=Zentrum fuer Medien und IT ,
                CN=Certification Authority (CA) 2005
                /emailAddress=caadmin@fernuni-hagen.de
        Validity
            Not Before: Oct  4 12:28:01 2005 GMT
            Not After : Oct  4 12:28:01 2007 GMT
        Subject: C=DE, ST=Nordrhein-Westfalen,
                 O=FernUniversitaet in Hagen,
                 OU=Zentrum fuer Medien und IT,
                 CN=ca.fernuni-hagen.de
                 /emailAddress=caadmin@fernuni-hagen.de
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:a4:3a:68:64:f2:94:1f:85:97:13:1b:6d:0e:c7:
                    95:f4:9d:46:6c:cc:9b:5c:98:2a:97:07:34:5b:9b:
                    ae:c3:b9:f2:51:e8:46:35:05:5d:e8:ff:7c:de:0f:
                    51:64:0d:e5:db:21:52:ab:bb:d2:66:18:d9:17:0f:
                    21:66:ba:4c:bb:65:47:b1:8f:5d:6e:1f:bc:94:eb:
                    e9:5a:e7:df:b1:68:fa:7b:11:2f:5a:da:12:da:00:
                    40:9d:3c:51:91:af:34:d6:73:89:77:83:0d:6b:58:
                    1b:34:04:8a:4c:9f:62:2b:66:8b:d3:1c:a4:a4:92:
                    29:1d:7b:6e:39:74:3d:e4:d5
                Exponent: 65537 (0x10001)
    Signature Algorithm: md5WithRSAEncryption
        3a:c6:9e:73:d6:db:0a:27:44:9f:57:c0:3b:8b:25:fd:2c:28:
        db:ce:20:b1:fe:e2:8a:64:ad:62:86:c0:65:e4:a4:cf:9e:42:
        91:50:be:46:14:ae:9a:64:1c:7c:1d:2a:bb:06:f1:96:35:94:
        e6:f1:84:c7:6e:3e:f8:9c:93:53:a6:a6:44:2b:00:06:9e:6f:
        10:cc:5a:e8:4d:f4:c3:7b:d0:66:92:b6:72:76:c1:f2:d3:b5:
        f7:11:c4:ef:c6:a3:05:bc:10:39:2c:3a:5a:8e:e5:2c:87:48:
        d6:a2:3e:22:92:fd:61:40:fb:f9:44:64:35:c9:63:41:74:d2:
        a1:9e
```

Your browser computes the sha1-Fingerprint: `68 d5 8c 0e c9 8d 74 e7 31 99 c4 78 74 0c 65 af b9 44 55 03`

Can you trust this certificate?

14 P. Assignment 40:

2 P. a) Why should an OpenPGP key carry a self-certificate?

12 P. b) Alice and Bob are friends. Alice wants to send Bob an encrypted and signed e-mail using PGP. She does not know Bob's public key. Give a short statement about the following methods of authentication explaining how secure they are.

1. Alice asks Bob to send her his public key in an e-mail.
2. Alice receives Bob's public key in an e-mail. She phones Bob and asks him to dictate the hash value (fingerprint) of his public key.
3. A PGP public key server is a giant public key ring containing the public keys of many PGP users. Alice downloads Bob's public key from a public key server. The key is signed by a CA which Alice does not know.
4. Bob has an authentic copy of Alice's public key. Bob sends his public key in an encrypted e-mail to Alice.
5. In any e-mail conversation with Alice so far (e.g. the last 10 e-mails), Bob has included a fingerprint of his public key. Now, he sends an e-mail to Alice including his public key.
6. Bob mails a diskette with his public key to Alice (traditional mail).

16 P. Assignment 41:

14 P. a) Create an SPKI certificate which conveys the following authorization to Alice:

"Alice may use the support of the company BigBucks during 01.05.2001 - 31.05.2001. She may delegate this authorization to her colleagues."

Use the Crypto-Calculator to create the digital signature. The certificate is issued and signed by BigBucksSupport. Alice's public key is (128 bit RSA):

```
n: 133742360663487486655573698906059112007
e: 30103164899134424358636143543737677
```

BigBucksSupport holds the following key pair (128 bit RSA):

```
n: 19993932509812921571414962404567286697 3
e: 42682163418151075735531010069604871163
d: 16228149633033681858112004945737503275 9
```

Solution hints: Procedure:

1. Use MD5 as hash algorithm.
2. You may freely make up a tag you think is suitable.

3. Use decimal notation for all keys and the signature.
4. The certificate has the following structure:

```
(cert
   (issuer
      (public-key
         (rsa-md5
            (e #...#)
            (n #...#)
             ...
       )
     .....
)
{signature in decimal notation}
```

5. For the actual signing operation it is usefull to format the certificate into a single line, because the Crypto-Calculator only can handle single line strings:

   ```
   cert(issuer(public-key(rsa-md5(e #...#) ... ) ... ) ... )
   {signature}
   ```

   These two lines must be present in your solution.

6. If you write a PROCESS (which is advisable), you should also include the code in your solution. Otherwise, you should include a listing with the commands for the Crypto-Calculator.

b) How does the certificate change, when DSA keys are employed instead of RSA keys? **2 P.**

Solutions for Assignments

Solution for Assignment 1:

a) Encryption systems can be subdivided into the following two categories: symmetric-key (secret-key) and asymmetric-key (public-key) encryption. The methods can be further divided into block ciphers (operating on blocks of fixed length) and stream ciphers (operating on symbols).

b) Let, for example, the following bijection be assigned to key k_3

$$E_{k_3}(m_1) = c_3, E_{k_3}(m_2) = c_2 \text{ and } E_{k_3}(m_3) = c_1.$$

The decryption function D for key k_3 is:

$$D_{k_3}(c_1) = m_3, D_{k_3}(c_2) = m_2 \text{ and } D_{k_3}(c_3) = m_1.$$

c) The ciphertext sequence c for the message sequence $m = m_1, m_3, m_2$ is:
$c = c_1, c_2, c_3$.

d) The explanation of the two attacking types:

- Chosen-ciphertext attack: The attacker tries to deduce the decryption key by pairs of plaintexts and ciphertexts, which (ciphertexts) are chosen by him.
- Known-plaintext attack: The attacker has a quantity of chosen plaintext and corresponding ciphertext and tries to determine the secret key k or decrypt further ciphertexts.

e) The ciphertext of the message sequence $m = 7, 5, 1$ with $k = 7$ is $c = 4, 2, 8$.

Solution for Assignment 2:

a) The encryption of the message m is done by $c = E_{k_{e,B}}(m)$.

b) The decryption is done by $m' = D_{k_{d,B}}(c)$.

c) A possible error is: During the exchange of the public keys, an attacker C has intercepted the transmission from B to A and has inserted a self generated public key $k_{e,C}$. A has used this public key to encrypt the message m. This attack can be detected if the keys are certified.

Solution for Assignment 3:

a) 239

b) The following three cryptographic encryption functions are implemented in the Crypto-Calculator:

- encrypt_des
- encrypt_idea
- encrypt_rc5.

For decryption purposes the following three decryption functions can be used:

- decrypt_des
- decrypt_idea
- decrypt_rc5.

All these functions have Integer as input as well as output.

c) The result of the encryption and decryption is illustrated in the table below:

Message	Key	Function	Result
193243434	1599999	encrypt_des	17269227889117512911
193243434	1599999	encrypt_idea	15143491676147334603
193243434	1599999	encrypt_rc5	14237518539867853697
17269227889117512911	1599999	decrypt_des	193243434
15143491676147334603	1599999	decrypt_idea	193243434
14237518539867853697	1599999	decrypt_rc5	193243434

Solution for Assignment 4:

a) HWDUYTLWFUMD FSI HWDUYTFSFQDXNX

b) A ROT13-encrypted text can be decrypted by a second run of the ROT13 algorithm.

c) Due to the cyclic shift operation the encrypted text is identical to the plaintext.

Solution for Assignment 5:

In the encrypted text of 242 characters, alphabet occurs with the following frequencies:

A	B	C	D	E	F	G	H	I	J	K	L	M
1	8	0	20	1	15	5	24	4	5	6	21	0

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	14	6	17	24	9	1	13	16	25	5	2	0

From the solution hints we know that the character "E" occurs most frequently in the English language. The characters which occurs most frequently in the encrypted text are "W" (25), "H" (24) and "R" (24). First we assume that "W" in the encrypted text corresponds to "E" in the plaintext which results in the decryption key "R" and the plaintext:

TY ESPTC LAAWTNLETZY NCJAEZWZRTNLW XPESZOD LCP YZE
WZYRPC LWTRYPO EZ ESP LNBFTDTETZY ZQ ACTGLNJ ZYWJ. ESP
CLYRP ZQ LAAWTNLETZYD EZOLJ NZYELTYD LWDZ WTXTEPO
LNNPDD QZC PILXAWP EZ ALJ-EG, PWPNECZYTN DTRYLEFCP
DNSPXPD EZ PYDFCP LFESPYETNTEJ ZC LWDZ EZ MFTWO
DJDEPXD QZC PWPNECZYTN ALJXPYED.

Obviously this text makes no sense, so we try "H", which results in the decryption key "C" and the plaintext:

IN THEIR APPLICATION CRYPTOLOGICAL METHODS ARE NOT
LONGER ALIGNED TO THE ACQUISITION OF PRIVACY ONLY. THE
RANGE OF APPLICATIONS TODAY CONTAINS ALSO LIMITED
ACCESS FOR EXAMPLE TO PAY-TV, ELECTRONIC SIGNATURE
SCHEMES TO ENSURE AUTHENTICITY OR ALSO TO BUILD
SYSTEMS FOR ELECTRONIC PAYMENTS.

After only two attempts the Caesar encrypted text was decrypted.

Solution for Assignment 6:

Let $\mathbb{Z}_p^* = \{1,\ 2,\ 3, \ldots,\ p-1\}$.

- The multiplication modulo $\times_p$ is a binary operation.

  For all $a, b \in \mathbb{Z}_p^*$

  $$a \times_p b = (a \cdot b) \mod p.$$

  $a \cdot b > 0$, then $a \cdot b$ can be written as (see Theorem 2.3-1):

  $$a \cdot b = qp + r, \text{ with } 0 \leq r < p. \qquad (1)$$

  $(a \cdot b)$ cannot be a multiple of p. That is $r \neq 0$. This can be proven as follows:

  Suppose that $r = 0$, then equation (1) can be written as:

  $$\begin{aligned}(a \cdot b) \mod p &= qp \mod p \\ &\equiv 0 \mod p.\end{aligned}$$

That is $a \cdot b = 0$. This implies that $a = 0$ or $b = 0$. Since $a, b \in \mathbb{Z}_p^*$, a resp. b is not equal 0, thus $r \neq 0$. This is in contradiction to our supposition, namely that $r = 0$.

Thus, for $0 < r < p$, we can write:

$$\begin{aligned}(a \cdot b) \mod p &\equiv (qp + r) \mod p \\ &= qp + r + kp \quad \text{(with } k \in \mathbb{Z}\text{)} \\ &= r + (q + k)p \\ &\equiv r \mod p \\ &= r \in \mathbb{Z}_p^*.\end{aligned}$$

Thus, $a \times_p b \in \mathbb{Z}_p^*$.

- $\times_p$ is associative.

  For all a, b, $c \in \mathbb{Z}_p^*$

$$\begin{aligned}(a \times_p b) \times_p c &= (((a \cdot b) \mod p) \cdot c) \mod p \\ &= (((a \cdot b) \cdot c) \mod p) \\ &= (a \cdot (b \cdot c) \mod p) \\ &= (a \cdot ((b \cdot c) \mod p)) \mod p \\ &= a \times_p ((b \cdot c) \mod p)) \\ &= a \times_p (b \times_p c).\end{aligned}$$

- 1 is an identity element of $(\mathbb{Z}_p^*, \times_p)$, since for all $a \in \mathbb{Z}_p^*$

$$\begin{aligned}a \times_p 1 &= (a \cdot 1) \mod p \\ &= a \mod p \\ &= a, \quad \text{and}\end{aligned}$$

$$\begin{aligned}1 \times_p a &= (1 \cdot a) \mod p \\ &= a \mod p \\ &= a.\end{aligned}$$

  That is, for all $a \in \mathbb{Z}_p^*$, $1 \times_p a = a \times_p 1 = 1$.

- Each element a of $(\mathbb{Z}_p^*, \times_p)$ has an inverse a^{-1}, since for every $a \in \mathbb{Z}_p^*$ $\gcd(a, p) = 1$.

  It follows that $(\mathbb{Z}_p^*, \times_p)$ is a group.

Solution for Assignment 7:

a) We use the Extended Euclidean algorithm to find x and y so that x and y are solution to the equation

$$ax + by = \gcd(a, b).$$

Throughout this solution we use the following notation:

Let $r_0 = a$, $r_1 = b$, $r_2 = r_0 \mod r_1$ and

$$q_1 = \lfloor \frac{r_0}{r_1} \rfloor.$$

If $r_2 \neq 0$, then:

$r_3 = r_1 \mod r_2$ and

$$q_2 = \lfloor \frac{r_1}{r_2} \rfloor.$$

We continue with this notation until $r_i = 0$:

$r_{i+1} = r_{i-1} \mod r_i$ and $q_i = \lfloor \frac{r_{i-1}}{r_i} \rfloor$, for $1 \leq i \leq n$.

We start with $x_0 = 1$, $y_0 = 0$, $x_1 = 0$, $y_1 = 1$ and compute x_{i+1} and y_{i+1} in every further iteration as:

$$x_{i+1} = q_i x_i + x_{i-1},$$
$$y_{i+1} = q_i y_i + y_{i-1}.$$

For $1 \leq i \leq n$, let n be defined such that $r_{n+1} = 0$. Then, the integers x and y (the solutions of the equation) can be computed as:

$$x = (-1)^n x_n$$
$$y = (-1)^{n+1} y_n.$$

$17^{-1} \mod 101$:

We want to find an element y so that $17y \equiv 1 \mod 101$.

$17y \equiv 1 \mod 101$ means that there exists an integer k so that

$$17y = 1 + 101k.$$

We write

$$7y - 101k = 1.$$

We put $x = -k$ and write

$$101x + 17y = 1.$$

The previous equation has a solution since $\gcd(101, 17) = 1$. Using the notation, introduced above, we obtain the following table

i	0	1	2	3	4
r_i	101	17	16	1	0
q_i		5	1	16	
x_i	1	0	1	1	
y_i	0	1	5	6	

From the table we get $n = 3$, since $r_4 = 0$. Then we have

$$x = (-1)^3 x_3 \text{ and}$$
$$y = (-1)^{3+1} y_3.$$

Regarding the table, we have $x_3 = 1$ and $y_3 = 6$. So $x = -1$ and $y = 6$. Thus, the inverse of 17 modulo 101 is $y = 6$.

We verify that

$$\begin{aligned} \gcd(101, 17) &= 101 \cdot (-1) + 17 \cdot 6 \\ &= 1 \end{aligned}$$

and

$$17 \cdot 6 \equiv 1 \mod 101.$$

b) $28^{-1} \mod 75$:

We proceed in the same manner as in the previous paragraph. Since $\gcd(28, 75) = 1$, the inverse of 28 modulo 75 is an element y so that

$$75x + 28y = 1.$$

i	0	1	2	3	4	5
r_i	75	28	19	9	1	0
q_i		2	1	2	9	
x_i	1	0	1	1	3	
y_i	0	1	2	3	8	

After using the Extended Euclidean, we can derive from the table above that $n = 4$, since $r_5 = 0$. Then we get

$$x = (-1)^4 x_4 \text{ and}$$
$$y = (-1)^{4+1} y_4.$$

Regarding the table, we have $x_4 = 3$ and $y_4 = 8$. So $x = 3$ and $y = -8$.

Note that $-8 \equiv 67 \mod 75$.

Thus, the inverse of 28 modulo 75 is $y = 67$.

We verify that

$$\begin{aligned} \gcd(28, 75) &= 75 \cdot 3 + 28 \cdot (-8) \\ &= 1 \end{aligned}$$

and

$$28 \cdot 67 \equiv 1 \mod 75.$$

c) $21/2 \mod 28$:

$21/2 \mod 28 = (21 \cdot 2^{-1}) \mod 28$, where 2^{-1} is the inverse of 357 modulo 28.

2^{-1} modulo 28 does not exist, since $\gcd(2, 28) = 2 \neq 1$. Hence $21/2 \mod 28$ does not exist.

d) $701/357 \mod 1234$:

$701/357 \mod 1234 = (701 \cdot 357^{-1}) \mod 1234$, where 357^{-1} is the inverse of 357 modulo 1234.

First we determine 357^{-1} modulo 1234. Since $\gcd(1234, 357) = 1$, the inverse of 357 modulo 1234 is an element y such that

$$1234x + 357y = 1.$$

After using the Extended Euclidean, we get the following table:

i	0	1	2	3	4	5	6	7
r_i	1234	357	163	31	8	7	1	0
q_i		3	2	5	3	1	7	
x_i	1	0	1	2	11	35	46	
y_i	0	1	3	7	38	121	159	

From the table we obtain $n = 6$, since $r_7 = 0$. Then we get:

$$x = (-1)^6 x_6 \text{ and}$$
$$y = (-1)^{6+1} y_6.$$

Regarding the table, we have $x_6 = 46$ and $y_6 = 159$. So $x = 46$ and $y = -159$.

Note that $-159 \equiv 1075 \mod 1234$.

Thus, the inverse of 357 modulo 1234 is $y = 1075$. We verify that

$$\begin{aligned}\gcd(357, 1234) &= 375 \cdot (-159) + 1234 \cdot 46 \\ &= 1\end{aligned}$$

and

$$357 \cdot 1075 \equiv 1 \mod 1234.$$

Now we get

$$\begin{aligned}701/357 \mod 1234 = (701 \cdot 357^{-1}) \mod 1234 &= 701 \cdot 1075 \mod 1234 \\ &= 753575 \mod 1234 \\ &= 835.\end{aligned}$$

Solution for Assignment 8:

a)

$$\begin{aligned}\mathbb{Z}_{14}^* &= \{a \in \mathbb{Z}_{14} \mid \gcd(a, 14) = 1\} \\ &= \{1,\ 3,\ 5,\ 9,\ 11,\ 13\}.\end{aligned}$$

b) The order of an element g of $\mathbb{Z}_{14}^*$ is the least positive integer δ such that $g^\delta = 1$. The order of the elements of $\mathbb{Z}_{14}^*$ are summarized in the table below:

i	0	1	2	3	4	5	6
$1^i \mod 14$	1						
$3^i \mod 14$	1	3	9	13	11	5	1
$5^i \mod 14$	1	5	11	13	9	3	1
$9^i \mod 14$	1	9	11	1			
$11^i \mod 14$	1	11	9	1			
$13^i \mod 14$	1	13	1				

From the table we obtain the following result:

The order of 1 is 1, the order of 3 is 6, the order of 5 is 6, the order of 9 is 3, the order of 11 is 3, and the order of 13 is 2.

c) The elements 3 and 5 generate all elements of $\mathbb{Z}_{14}^*$. That is there are generators of $\mathbb{Z}_{14}^*$. Note that the number of generators of $\mathbb{Z}_{14}^*$ is

$$\begin{aligned}\varphi(\varphi(\mathbb{Z}_{14}^*)) &= \varphi(6) \\ &= 2.\end{aligned}$$

Solution for Assignment 9:

We will solve the following equations

$$\begin{aligned}x &\equiv 12 \mod 25 \\ x &\equiv 9 \mod 26 \\ x &\equiv 23 \mod 27.\end{aligned}$$

We use

$$\begin{aligned}m_1 &= 25, \qquad a_1 = 12 \\ m_2 &= 26, \qquad a_2 = 9 \\ m_3 &= 27, \qquad a_3 = 23.\end{aligned}$$

Since $\gcd(m_i, m_j) = 1$ for each $1 \leq i \neq j \leq 3$, we can use the Chinese remainder algorithm to find x.

$$\begin{aligned}m &= m_1 \cdot m_2 \cdot m_3 \\ &= 25 \cdot 26 \cdot 27 \\ &= 17550.\end{aligned}$$

$$M_1 = \frac{m}{m_1} = 702$$
$$M_2 = \frac{m}{m_2} = 675$$
$$M_3 = \frac{m}{m_3} = 650.$$

Now we compute y_i with the use of the Extended Euclidean algorithm from the congruence $y_i M_i \equiv 1 \mod m_i$ for $i = 1, 2, 3$.

$$y_1 \cdot 702 \equiv 1 \mod 25 \Rightarrow y_1 = 13$$
$$y_2 \cdot 675 \equiv 1 \mod 26 \Rightarrow y_2 = 25$$
$$y_3 \cdot 650 \equiv 1 \mod 27 \Rightarrow y_3 = 14.$$

The solution is

$$x \equiv (\sum_{i=1}^{n} a_i y_i M_i) \mod m$$
$$x \equiv (a_i y_i M_i + a_2 y_2 M_2 + a_3 y_3 M_3) \mod m$$
$$\equiv (12 \cdot 13 \cdot 702 + 9 \cdot 25 \cdot 675 + 23 \cdot 14 \cdot 650) \mod 17550$$
$$x = 14387.$$

We verify that

$$14387 \equiv 12 \mod 25$$
$$14387 \equiv 23 \mod 27$$
$$14387 \equiv 9 \mod 26.$$

Solution for Assignment 10:

a) GF(2^5) contains the polynomials $a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$, with $a_i \in$ GF$(2) = \{0, 1\}$ for $i = 1, 2, 3, 4$. Arithmetic in GF(2^5) is performed modulo the irreducible polynomial $p(x) = x^5 + x^2 + 1$.

b) We will compute $(x^4 + x^2 + x + 1) \cdot (x^2 + 1)$ in GF(2^5). To compute the product $(x^4 + x^2 + x + 1)$ and $(x^2 + 1)$ in GF(2^5), we multiply the two polynomials, divide the result by $p(x) = x^5 + x^2 + 1$ and then we get the remainder of this division as result. That is

$$(x^4 + x^2 + x + 1) \cdot (x^2 + 1) = x^6 + x^3 + x + 1$$
$$\frac{x^6 + x^3 + x + 1}{x^5 + x^2 + 1} = x + \frac{1}{x^5 + x^2 + 1}.$$

Thus, $(x^4 + x^2 + x + 1) \cdot (x^2 + 1) \equiv 1 \mod p(x)$.

c) We deduce from b) that $x^4 + x^2 + x + 1$ is an inverse of $x^2 + 1$ in GF(2^5).

d) $p(x)$ is a primitive polynomial, since x is a generator of GF$(2^5)^*$. To prove that, note that all elements of $(GF(2^5))^*$ can be obtained as powers of x modulo $p(x)$. This is shown in the next table, where the notation $a_4a_3a_2a_1a_0$ substitutes that of the polynomial $a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$, with $a_i \in$ GF$(2) = \{0, 1\}$ for i=1, 2, 3, 4.

i	$x^i \mod x^4 + x + 1$ ($a_4 a_3 a_2 a_1 a_0$)
0	00001
1	00010
2	00100
3	01000
4	10000
5	00101
6	01010
7	10100
8	01101
9	11010
10	10001
11	00111
12	01110
13	11100
14	11101
15	11111
16	11011
17	10011
18	00011
19	00110
20	01100
21	11000
22	10101
23	01111
24	11110
25	11001
26	10111
27	01011
28	10110
29	01001
30	10010
31	00001
⋮	⋮

Solution for Assignment 11:

a) We can determine whether the number 1729 is a prime or not using the following tests:

Fermat's test:
Recall Theorem 2.6-3

With Fermat's theorem, we can only decide if the number 1729 is composite. That is, for $a > 1$ if $\gcd(a, 1729) = 1$ and $a^{1729-1} \not\equiv 1 \mod 1729$ then 1729 is composite.
For $a = 2, \gcd(2, 1729) = 1$ and $2^{1728} \equiv 1 \mod 1729 = 1$.
For $a = 3, \gcd(3, 1729) = 1$ and $3^{1728} \equiv 1 \mod 1729 = 1$.
For $a = 4, \gcd(4, 1729) = 1$ and $4^{1728} \equiv 1 \mod 1729 = 1$.
For $a = 5, \gcd(5, 1729) = 1$ and $5^{1728} \equiv 1 \mod 1729 = 1$.
For $a = 6, \gcd(6, 1729) = 1$ and $6^{1728} \equiv 1 \mod 1729 = 1$.
For $a = 7, \gcd(7, 1729) = 1$ and $7^{1728} \equiv 742 \mod 1729 \neq 1$.
1729 is pseudoprime to the bases $a = 2, 3, 4, 5, 6$. Only with the use of base 7, we can see that 1729 is composite.

b) Miller-Rabin test:
Recall Theorem 2.6-4 and all notations utilized in the Miller-Rabin test.
$s = max\{r \in \mathbb{N} : \ 2^r$ divides $1729 - 1\} = 6$ and $d = \frac{(1729-1)}{2^6} = 27$.
For $a = 2$ we have

$$\begin{aligned} 2^{27} &\equiv 645 \mod 1729 \\ &\not\equiv 1 \mod 1729. \end{aligned}$$

We can deduct that 1729 is not prime by using Miller-Rabin test.

Solution for Assignment 12:

We want to factorize 221 with Pollard's rho algorithm. By choosing $f(x) = x^2 + 1$ and $x_0 = 1$ we get:

$$\begin{aligned} x_0 &= 1 \\ x_1 &\equiv 1^2 + 1 \mod 221 \\ &= 2 \\ x_2 &\equiv x_1^2 + 1 \mod 221 \\ &= 2^2 + 1 \\ &= 5 \\ x_3 &\equiv x_2^2 + 1 \mod 221 \\ &= 5^2 + 1 \\ &= 26 \\ x_4 &\equiv x_2^2 + 1 \mod 221 \\ &= 26^2 + 1 \mod 221 \\ &= 14 \end{aligned}$$

$\gcd(x_1 - x_0, 221) = \gcd(2 - 1, 221) = 1$.
$\gcd(x_2 - x_1, 221) = \gcd(x_2 - x_0, 221) = 1$.
$\gcd(x_3 - x_2, 221) = \gcd(x_3 - x_1, 221) = \gcd(x_3 - x_0, 221) = 1$.
$\gcd(x_4 - x_0, 221) = \gcd(14 - 1, 221) = 13 > 1$.

Since $13 < 221$, $\gcd(14-1, 221)$ is a divisor of 221.
We verify that $221 = 13 \cdot 17$.

Solution for Assignment 13:

We want to solve the DL problem:

$$7^x = 5$$

in the group $\mathbb{Z}_{17}^*$. The order of the group is $n = 17 - 1 = 16$. We determine $m = \lceil \sqrt{16} \rceil = 4$.
The baby step is

$$B = \{(5 \cdot 7^{-r}, r); 0 \leq r < 4\}.$$

For $r = 0$ we obtain the pair (5.0).
For $r = 1$ the first component of the pair is $5 \cdot 7^{-1} \mod 2017$. That is, we first have to determine the inverse of 7 in $\mathbb{Z}_{17}^*$. We get $7^{-1} \mod 17 = 5$. Therefore,

$$5 \cdot 7^{-1} \mod 17 \equiv 5 \cdot 5 \mod 17 \equiv 8.$$

For $r = 2$ we get

$$\begin{aligned} 5 \cdot 7^{-2} \mod 17 &\equiv 5 \cdot 49^{-1} \mod 17 \\ &\equiv 5 \cdot 8 \mod 17 \\ &\equiv 6. \end{aligned}$$

For $r = 3$ we get

$$\begin{aligned} 5 \cdot 7^{-3} \mod 17 &\equiv 5 \cdot 343^{-1} \mod 17 \\ &\equiv 5 \cdot 6 \mod 17 \\ &\equiv 13. \end{aligned}$$

Thus,

$$B = \{(5.0), (8, 1), (6, 2), (13, 3)\}.$$

The computation of the elements $(7^4)^q \mod 17$ for $q = 1, 2, \ldots$ results in $4, 16, 13$. The calculation is stopped when $q_0 = 3$, because the first component in the pair $(13, 3)$ of the baby step set contains 13. We get $r_0 = 3$ and subsequently

$$\begin{aligned} x &= q_0 m + r_0 \\ &= (3 \cdot 4) + 3 \\ &= 15. \end{aligned}$$

Thus, $7^{15} = 5 \mod 17$

.

Solution for Assignment 14:

a) The Crypto-Interpreter has four functions to handle LFSRs:

- `lfsr_init`, functionality: Initialisation of an LFSR with feedback polynomial c and an initial state $\underline{s}_0$. The LFSR, the polynomial c, the initial state $\underline{s}_0$ are represented as integer values. Input type for LFSR, c and $\underline{s}_0$: `INTEGER`, no output, it is a procedure.
  ```
  CALL lfsr_init(LFSR,c,s);
  ```
- `lfsr_state`, functionality: Return the current state of the LFSR, input type for LFSR: `INTEGER`, output type for the state: `INTEGER`.
  ```
  TASK s:= lfsr_state(LFSR);
  ```
- `lfsr_run`, functionality: Return the current output of the LFSR and go to the next state, input type for LFSR: `INTEGER`, output type: `INTEGER`.
  ```
  TASK out:= lfsr_run(LFSR);
  ```
- `lfsr_exit`, functionality: Release the memory of the LFSR, input type for LFSR: `INTEGER`, no output, it is a procedure.
  ```
  CALL lfsr_exit(LFSR);
  ```

In the case of $c(x) = x^5 + x^2 + 1$ the feedback polynomial is represented by the integer $32+4+1 = 37$. The initial state $(1, 0, 0, 0, 0)$ is represented as integer value 1.

b) Table of the current state $\underline{s}_t = (s_t, s_{t+1}, \ldots, s_{t+4})^T$ of the LFSR for $0 \leq t \leq 10$:

t	s_t	s_{t+1}	s_{t+2}	s_{t+3}	s_{t+4}
0	1	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	1	0	0
4	0	1	0	0	1
5	1	0	0	1	0
6	0	0	1	0	1
7	0	1	0	1	1
8	1	0	1	1	0
9	0	1	1	0	0
10	1	1	0	0	1

c) The period of the output sequence of the LFSR can be determined in different ways, i.e.:

- Determine the exponent of the polynomial c.

- Determine the primitivity of the polynomial c.
- Compute the output sequence s_t of the LFSR for $0 \leq t \leq 2 \cdot (2^5 - 1)$.

The period of the sequence s is 31.

d) The feedback polynomial is primitive, because the exponent is 31 or because the period of the output sequence s is 31.

e) The output values of the filter function $f : \mathrm{GF}(2)^3 \rightarrow \mathrm{GF}(2)$, $f(x_1, x_2, x_3) = x_1x_2 + x_3$, are:

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

This table is called the truth table of f.

f) The output sequence z of the nonlinear filter generator is given by

$$z_t = f(s_{t+\gamma_1}, s_{t+\gamma_2}, s_{t+\gamma_3}) = f(s_t, s_{t+2}, s_{t+3})$$

for $t \geq 0$. The values for z_t, $0 \leq t \leq 10$, are illustrated in the following table:

t	s_t	s_{t+2}	s_{t+3}	z_t
0	1	0	0	0
1	0	0	0	0
2	0	0	1	1
3	0	1	0	0
4	0	0	0	0
5	1	0	1	1
6	0	1	0	0
7	0	0	1	1
8	1	1	1	0
9	0	1	0	0
10	1	0	0	0

Solution for Assignment 15:

a) The sequence elements s_t, $t \geq 0$, are called balanced and statistically independent, if the following two conditions hold for any $t \geq 0$:

a. $P(s_t = 0) = P(s_t = 1) = 0.5$ and

b. $P(s_t|s_0, s_1, \ldots, s_{t-1}) = P(s_t)$.

b) The following inputs are needed to produce the output (z_t, z_{t+1}):

$s_t, s_{t+1}, s_{t+2}, s_{t+3}, s_{t+4}$.

s_t, s_{t+2}, s_{t+3} for z_t and $s_{t+1}, s_{t+3}, s_{t+4}$ for z_{t+1}.

c) The truth table for the output (z_t, z_{t+1}) is given by:

s_t	s_{t+1}	s_{t+2}	s_{t+3}	s_{t+4}	z_t	z_{t+1}
0	0	0	0	0	0	0
0	0	0	0	1	0	1
0	0	0	1	0	1	0
0	0	0	1	1	1	1
0	0	1	0	0	0	0
0	0	1	0	1	0	1
0	0	1	1	0	1	0
0	0	1	1	1	1	1
0	1	0	0	0	0	0
0	1	0	0	1	0	1
0	1	0	1	0	1	1
0	1	0	1	1	1	0
0	1	1	0	0	0	0
0	1	1	0	1	0	1
0	1	1	1	0	1	1
0	1	1	1	1	1	0
1	0	0	0	0	0	0
1	0	0	0	1	0	1
1	0	0	1	0	1	0
1	0	0	1	1	1	1
1	0	1	0	0	1	0
1	0	1	0	1	1	1
1	0	1	1	0	0	0
1	0	1	1	1	0	1
1	1	0	0	0	0	0
1	1	0	0	1	0	1
1	1	0	1	0	1	1
1	1	0	1	1	1	0
1	1	1	0	0	1	0
1	1	1	0	1	1	1
1	1	1	1	0	0	1
1	1	1	1	1	0	0

d) The following inputs give the output $(z_t, z_{t+1}) = (0, 0)$:

s_t	s_{t+1}	s_{t+2}	s_{t+3}	s_{t+4}	z_t	z_{t+1}
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	0	0	0
1	0	1	1	0	0	0
1	1	0	0	0	0	0
1	1	1	1	1	0	0

e) In the table above, there are 6 of 8 inputs, for which $s_{t+3} = 0$ is true. So, $P(s_{t+3} = 0 | z_t = 0 \wedge z_{t+1} = 0) = \frac{6}{8} = \frac{3}{4}$.

Solution for Assignment 16:

a) To demonstrate how the different modes of operation work, we have chosen the following for both DES and IDEA:

- Plaintext: This is a demonstration!
- Sender Key: ECBmode
- Receiver Key: any.

The result of the different modes of operation is illustrated in Tab. 1 for DES and in Tab. 2 for IDEA.

Tab. 1: *Result of the different modes of operation for DES.*

Modes of operation	Plaintext (Hex)	Ciphertext (Hex)	IV
ECB	5468697320697320 612064656D6F6E73 74726174696F6E21	0834A10727EBD6E0 B339868E98798B1F DC86FBDA94C8D3C9	
CBC	5468697320697320 612064656D6F6E73 74726174696F6E21	3355DCB3D98ED7CD 3B04384353514BA5 98530AB50DEE337D	1234567890ABCDEF
CFB (r=1)	5468697320697320 612064656D6F6E73 74726174696F6E21	7B10748A7F68B4F9 A866E837760A2F45 500DE5B69BFBE833	1234567890ABCDEF
OFB (r=1)	5468697320697320 612064656D6F6E73 74726174696F6E21	7BFBF53E9323FF3B D2262E7AD465BDF7 70DC27ECDE030517	1234567890ABCDEF

Tab. 2: *Result of the different modes of operation for IDEA.*

Mode of operation	Plaintext (Hex)	Ciphertext (Hex)	IV
ECB	5468697320697320 612064656D6F6E73 74726174696F6E21	CE9D0C619B0DC618 7BF6E17024CF763D E74F9D79E4D3E5FD	
CBC	5468697320697320 612064656D6F6E73 74726174696F6E21	FAF60580695B9CC0 D12B388305034786 94DB4A592F048030	1234567890ABCDEF
CFB (r=1)	5468697320697320 612064656D6F6E73 74726174696F6E21	0B2BAD27DFCE71B4 A94F05275B2870BE CE41D972D10B2D36	1234567890ABCDEF
OFB (r=1)	5468697320697320 612064656D6F6E73 74726174696F6E21	OB7CAFC108394D35 6AF2509006A387D0 AC79CF7777547BA9	1234567890ABCDEF

b) In the ECB mode, one or more bit errors in a single ciphertext block affect the decryption of that block only.

In the CBC mode: a bit error in c_t affects the decryption of c_t and c_{t+1}, whereas the recovered plaintext m'_{t+1} has bit errors precisely where c_t did. When c_{t+1} and c_{t+2} have been transmitted without errors, then c_{t+2} is decrypted correctly.

In the CFB mode: changing one bit in c_t affects the decryption of the next $\lceil n/r \rceil$ ciphertext blocks. Proper decryption of c_t requires the preceding $\lceil n/r \rceil$ blocks to be transmitted correctly.

In the OFB mode: a bit error in the ciphertext c_t exclusively affects the corresponding bit in the plaintext m_t.

Solution for Assignment 17:

a) The effective key size is 56 bit, the plaintext and ciphertext lengths are 64 bit.

b) In the Crypto-Interpreter the following functions for the DES algorithm are available:

- `encrypt_des`, functionality: $c = E_k(m)$, input type for m and k: `INTEGER`, output type for c: `INTEGER`:
  `TASK c=encrypt_des(m,k);`
- `decrypt_des`, functionality: $m = D_k(c)$, input type for c and k: `INTEGER`, output type for m: `INTEGER`:
  `TASK m=decrypt_des(c,k);`

c) In ECB mode the encryption is done according to the following steps:

$$c_0 = E_k(m_0),$$
$$c_1 = E_k(m_1) \text{ and}$$
$$c_2 = E_k(m_2).$$

The ciphertexts are:

$$c_0 = 17648341485071271004,$$
$$c_1 = 17721168555847677831 \text{ and}$$
$$c_2 = 17648341485071271004.$$

In ECB mode the decryption is done according to the following steps:

$$m_0 = D_k(c_0),$$
$$m_1 = D_k(c_1) \text{ and}$$
$$m_2 = D_k(c_2).$$

In Listing 1 a Crypto-Interpreter program with corresponding output in Listing 2 shows the above process.

Listing 1: Crypto-Interpreter program for the ECB encryption

```
PROCESS des_ecb;
  DCL m0,m1,m2,k,c0,c1,c2,m0_,m1_,m2_ integer;
START; TASK
  k := 4534;
  TASK m0 := 456;
  TASK m1 := 312;
  TASK m2 := 456;
  TASK c0 := encrypt_des(m0,k);
  TASK c1 := encrypt_des(m1,k);
  TASK c2 := encrypt_des(m2,k);
  CALL writeln('Encrypt with ecb mode');
  CALL writeln('key k: ',k);
  CALL writeln('plaintext m0: ',m0);
  CALL writeln('plaintext m1: ',m1);
  CALL writeln('plaintext m2: ',m2);
  CALL writeln('ciphertext c0: ',c0);
  CALL writeln('ciphertext c1: ',c1);
  CALL writeln('ciphertext c2: ',c2);

  TASK m0_ := decrypt_des(c0,k);
  TASK m1_ := decrypt_des(c1,k);
  TASK m2_ := decrypt_des(c2,k);
  CALL writeln('Decrypt with ecb mode');
  CALL writeln('key k: ',k);
  CALL writeln('ciphertext c0: ',c0);
  CALL writeln('ciphertext c1: ',c1);
  CALL writeln('ciphertext c2: ',c2);
  CALL writeln('plaintext m0_: ',m0_);
  CALL writeln('plaintext m1_: ',m1_);
```

```
  CALL writeln('plaintext m2_: ',m2_);
STOP;
ENDPROCESS;
```

Listing 2: Output of the Crypto-Interpreter for the ECB encryption

```
Encrypt with ecb mode
key k: 4534
plaintext m0:456
plaintext m1: 312
plaintext m2: 456
ciphertext c0: 17648341485071271004
ciphertext c1: 17721168555847677831
ciphertext c2: 17648341485071271004
Decrypt with ecb mode
key k: 4534
ciphertext c0: 17648341485071271004
ciphertext c1: 17721168555847677831
ciphertext c2: 17648341485071271004
plaintext m0_: 456
plaintext m1_: 312
plaintext m2_: 456
Prozess beendet.
```

d) In CBC mode the encryption is done according to the following steps:

$$c_0 = E_k(m_0 + \mathrm{IV}),$$
$$c_1 = E_k(m_1 + c_0) \text{ and}$$
$$c_2 = E_k(m_2 + c_1).$$

The ciphertexts are:

$$c_0 = 12426018224378480581,$$
$$c_1 = 95123789879080796 32 \text{ and}$$
$$c_2 = 10433574674834059951.$$

In CBC mode the decryption is done according the following steps:

$$m_0 = \mathrm{IV} + D_k(m_0),$$
$$m_1 = c_0 + D_k(m_1 + c_0) \text{ and}$$
$$m_2 = c_1 + D_k(m_2 + c_1).$$

In Listing 3 a Crypto-Interpreter program with corresponding output in Listing 4 shows the above process.

Listing 3: Crypto-Interpreter program for the CBC encryption

```
PROCESS des_cbc;
    DCL i0,i1,i2,o0,o1,o2,m0,m1,m2,m0_,m1_,m2_,k,iv,c0,c1,c2 integer;
START;
    TASK k := 4534;
    TASK iv := 3241;
    TASK m0 := 456; TASK m1 := 312;
    TASK m2 := 456;
    TASK i0 := iv xor m0;
    TASK c0 := encrypt_des(i0,k);
    TASK i1 := c0 xor m1;
    TASK c1 := encrypt_des(i1,k);
    TASK i2 := c1 xor m2;
    TASK c2 := encrypt_des(i2,k);
    CALL writeln('Encrypt with cbc mode:');
    CALL writeln('key k: ',k);
    CALL writeln('initial value: ',iv);
    CALL writeln('plaintext m0: ',m0);
    CALL writeln('plaintext m1: ',m1);
    CALL writeln('plaintext m2: ',m2);
    CALL writeln('input i0: ',i0);
    CALL writeln('input i1: ',i1);
    CALL writeln('input i2: ',i2);
    CALL writeln('ciphertext c0: ',c0);
    CALL writeln('ciphertext c1: ',c1);
    CALL writeln('ciphertext c2: ',c2);

    TASK o0 := decrypt_des(c0,k);
    TASK m0_ := o0 xor iv; TASK o1 := decrypt_des(c1,k);
    TASK m1_ := o1 xor c0; TASK o2 := decrypt_des(c2,k);
    TASK m2_ := o2 xor c1; CALL writeln('Decrypt with cbc mode:');
    CALL writeln('key k: ',k);
    CALL writeln('initial value: ',iv);
    CALL writeln('ciphertext c0: ',c0);
    CALL writeln('ciphertext c1: ',c1);
    CALL writeln('ciphertext c2: ',c2);
    CALL writeln('output o0: ',o0);
    CALL writeln('output o1: ',o1);
    CALL writeln('output o2: ',o2);
    CALL writeln('plaintext m0_: ',m0_);
    CALL writeln('plaintext m1_: ',m1_);
    CALL writeln('plaintext m2_: ',m2_);
STOP;
ENDPROCESS;
```

Listing 4: Output od the Crypto-Interpreter program for the CBC encryption

```
Encrypt with cbc mode:
key k: 4534
initial value: 3241
plaintext m0: 456
plaintext m1: 312
plaintext m2: 456
```

```
input i0: 3425
input i1: 12426018224378480381
input i2: 9512378987908080088
ciphertext c0: 12426018224378480581
ciphertext c1: 9512378987908079632
ciphertext c2: 10433574674834059951
Decrypt with cbc mode:
key k: 4534
initial value: 3241
ciphertext c0: 12426018224378480581
ciphertext c1: 9512378987908079632
ciphertext c2: 10433574674834059951
output o0: 3425 output o1: 12426018224378480381
output o2: 9512378987908080088
plaintext m0_: 456
plaintext m1_: 312
plaintext m2_: 456
Prozess beendet.
```

Solution for Assignment 18:

An example for the derivation of subkeys from the IDEA key schedule is shown in the table below.

Round j	$k_1^{(j)}$	$k_2^{(j)}$	$k_3^{(j)}$	$k_4^{(j)}$	$k_5^{(j)}$	$k_6^{(j)}$
1	0001	0002	0003	0004	0005	0006
2	0007	0008	0400	0600	0800	0a00
3	0c00	0e00	1000	0200	0010	0014
4	0018	001c	0020	0004	0008	000c
5	2800	3000	3800	4000	0800	1000
6	1800	2000	0070	0080	0010	0020
7	0030	0040	0050	0060	0000	2000
8	4000	6000	8000	a000	c000	e001
9	0080	00c0	0100	0140	-	-

Solution for Assignment 19:

We will briefly describe the encryption process in AES (see Fig. 1, see also [Fer03]). Hereby we will only describe a single round of AES because other rounds are similar.

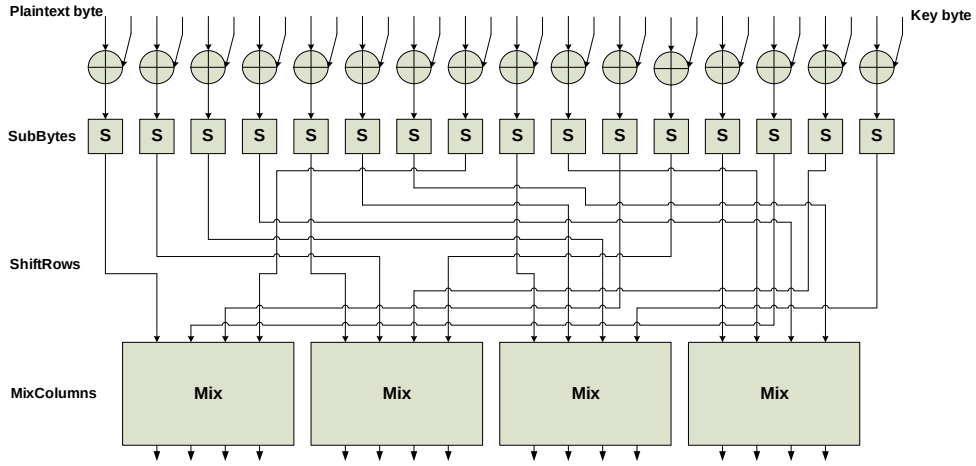


Fig. 1: A single round of AES.

In AES, each round consists of four steps:

- SubBytes,
- ShiftRows,
- MixColumns,
- AddRoundKey.

In the first step (SubBytes) the plaintext comes in as 16 bytes at the very top. The first operation is to XOR the plaintext with 16 bytes (128 bits) of round key. Each of the 16 bytes is then used as an index into an S-box table that maps 8-bit inputs into 8-bit outputs. The S-boxes are all identical. In the second step (ShiftRows) the bytes are then rearranged in a specific order that looks a bit complicated but has a simple structure. Finally, the bytes are mixed into groups of four (MixColumns). The last step of the encryption process is AddRoundKey, a simple bitwise XOR operation of round key and a state to output the new state.

Solution for Assignment 20:

a) **Statement:** When an attacker can factorize the modulus n of the RSA-scheme, then he can compute the secret key $K_{d,A}$ of the user A.
Proof: The RSA modulus n is publicly known. Suppose that an attacker can factorize n, i.e. he can compute the primes p and q from n. He also knows that the secret key $k_{d,A} = d$ has been computed from the equation $d \cdot e \equiv 1 \bmod \varphi(n)$. Because he knows p and q, he can compute $\varphi(n) = (p-1)(q-1)$. In the equation

$$d \cdot e \equiv 1 \bmod (p-1)(q-1)$$

all variables except d are known. So, he can get the secret key $d = k_{d,A}$ by solving this congruency (e.g. with the Extended Euclidean algorithm).

b) Speeding up the decryption operation of RSA using the Chinese Reminder theorem:
Suppose that Alice has the public key $k_{e,A} = (e, n)$ and the private key $k_{d,A} = d$.

When Bob sends a confidential message m to Alice, he computes the ciphertext $c = m^e \bmod n$. To decrypt, Alice computes: $m = c^d \bmod n$.
Because Alice knows p and q (factorization of n), she can use the Chinese Reminder theorem to speed up this exponentiation. She uses her private key d to compute the values m_p and m_q as:

$$m_p = c^d \bmod p$$
$$m_q = c^d \bmod q.$$

Then she can set up the following system of equations:

$$m = m_p \bmod p$$
$$m = m_q \bmod q.$$

The original plaintext m can be found from this simultaneous congruency using the Chinese Reminder theorem (see Chapter 2, Mathematical Background). With the Extended Euclidean algorithm she computes two numbers y_p and y_q which satisfy the equation

$$y_p p + y_q q = 1.$$

Using y_p and y_q the message m can be computed as:

$$m = (m_p y_q q + m_q y_p p) \bmod n.$$

Note that the expressions $y_p p \bmod n$ and $y_q q \bmod n$ are not dependent on the message m to be decrypted and can be precomputed.

We now show that decryption using the Chinese Reminder theorem is faster. We assume that the modulus n has the bit length k and its factors, p and q, have the bit lengths $k/2$. Multiplication of two numbers modulo n needs time Ck^2 (C is a constant). If we denote with l the number of ones in the binary representation of d, the computation of $m = c^d \bmod n$ ("classical encryption") has computational cost of $C(k + l)k^2$. The computational cost for m_p and m_q is $2(k + l)Ck^2/4 = C(k + l)k^2/2$. After that we apply the Chinese Reminder theorem and we have two more modular exponentiations. If we ignore the cost for computation of the Chinese Reminder (we can ignore it because d is very long and the time for computing the Chinese Reminder is negligible compared to the time for exponentiation), we see that in this way we achieve an acceleration of the decryption operation of a factor of two. So, decryption with the Chinese Reminder theorem is twice as fast as the "classical" encryption. This is very important since the decryption operation is very often performed on a smart card which has limited storage capacity and processing power and is very slow.

Example 1:
Suppose n= 253 ($p = 11$ and $q = 23$), $e = 3$, $d = 147$, and $c = 119$. We can reconstruct the message m "classicaly" ($m = c^d \bmod n = 119^{147} \bmod 253 = 26$) or using the Chinese Reminder Theorem:

$$
\begin{aligned}
m_p &= c^d \bmod p = 119^{147} \bmod 11 = 4 \\
m_q &= c^d \bmod q = 119^{147} \bmod 23 = 3 \\
y_p &= -2 \\
y_q &= 1 \\
m &= (m_p y_q q + m_q y_p p) \bmod n = (4 \cdot 23 - 3 \cdot 2 \cdot 11) \bmod 253 = 26.
\end{aligned}
$$

Solution for Assignment 21:

We have the pair $(e = 1123, n = 117739)$. The modulus n should be product of two primes. We use the function `factor(117739);` from the Crypto-Calculator to find one possible factor of n. This delivers $p = 281$ as a first factor of n. We find the second factor: $q = n/p = 419$. In order to check whether p and q are prime, we use the function `is_prime(281);` resp. `is_prime(419);` because 419 and 281 are prime, this condition is fulfilled.

The exponent e should satisfy $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$. In this case, $\varphi(n) = \varphi(419 \cdot 281) = (419 - 1)(281 - 1) = 418 \cdot 280 = 117040$. For $e = 1123$ we have $1 < 1123 < 117040$ and $\texttt{gcd}(1123, 117040) = 1$. So, the requirements are fulfilled and the pair $(1123, 117739)$ can be an RSA encryption key (public key).

Computing of the decryption key d:
The decryption key d should satisfy $1 < d < \varphi(n)$ and $d \cdot e \equiv 1 \bmod \varphi(n)$. Because $\gcd(e, \varphi(n)) = 1$, such a number d exists . The decryption exponent d can be computed from the equation $d \cdot 1123 \equiv 1 \bmod 117040$. This can be done with the Extended Euclidean algorithm, which has also been used to implement the `rcp` function of the Crypto-Calculator. The output is $d =$`rcp`$(1123, 117040) = 92027$. So, the corresponding private key of the public key $(e = 1123, n = 117739)$ is $d = 92027$.

Solution for Assignment 22:

The question is, if the element $g = 9$ from $\mathbb{Z}_{23}^*$ can be used as a public element of ElGamal encryption scheme. Suitable numbers g which can be used as public elements in encryption schemes based on the discrete logarithm problem are those which are generators (prime elements) of the group in which we operate. Only in this case, the existence of the discrete logarithm of each element a of the group is guaranteed. So, we should verify whether the element $g = 9$ is a generator of

the group $\mathbb{Z}_{23}^*$, i.e. if the powers of g generate all elements of the group $\mathbb{Z}_{23}^* = \{1, 2, 3, ..., 22\}$

$$9^1 = 9$$
$$9^2 = 12$$
$$9^3 = 16$$
$$9^4 = 6$$
$$9^5 = 8$$
$$9^6 = 3$$
$$9^7 = 4$$
$$9^8 = 13$$
$$9^9 = 2$$
$$9^{10} = 18$$
$$9^{11} = 1$$
$$-- - - - - - - --$$
$$9^{12} = 9$$
$$9^{13} = 12$$
$$.....$$

We can see that from the power 9^{12} all elements are repeated. The powers of 9 generate the set $\{9, 12, 16, 6, 8, 3, 4, 13, 2, 18, 1\}$, i.e. a subgroup of order 11 is generated and not the whole group $\mathbb{Z}_{23}^*$ of order 22. So, the element $g = 9$ should not be used as a public element of the ElGamal encryption scheme.

Solution for Assignment 23:

a) Computation of all points on the curve $E_{11}(7, 6): \quad y^2 = x^3 + 7x + 6$.

We should find all pairs (x, y) which satisfy the equation $y^2 = x^3 + 7x + 6$ in $\mathbb{Z}_{11}$. So, for each x in $\mathbb{Z}_{11}$ we calculate $x^3 + 7x + 6$ and then try to find the square root. This process is shown in the following table:

x	$x^3 + 7x + 6$	$sqrt$ in Z_{11} exists?	$y = sqrt(x^3 + 7x + 6)$
0	6	no	-
1	3	yes	$y = 5,\ y = 6$
2	6	no	-
3	10	no	-
4	10	no	-
5	1	yes	$y = 1,\ y = 10$
6	0	yes	$y = 0$
7	2	no	-
8	2	no	-
9	6	no	-
10	9	yes	$y = 3,\ y = 8$

Because in the book we have not dealt with the algorithms for finding a square root of element in Z_n, in order to find the required square roots (last column of the table above), we squared all elements of Z_{11} and took the square roots from the following table:

y	0	1	2	3	4	5	6	7	8	9	10
y^2 in Z_{11}	0	1	4	9	5	3	3	5	9	4	1

So, the curve $E_{11}(7, 6)$ has 7 points and they are: (1,5), (1,6), (5,1), (5,10), (6,0), (10,3), and (10,8).

b) The point $P(5, 1) \in E_{11}(7, 6)$ is given. We should compute the point $Q = 3P$.

We can do this both as $Q = 3P = 2P + P$ (one doubling and one addition) or as $Q = 3P = P + P + P$ (two additions). Independent of whether we chose the first or the second way, we have to perform two point operations in an elliptic curve group. But note that if we had to compute for example $Q = 9P$, the first way of computation is $Q = 9P = 2{\cdot}(2{\cdot}(2{\cdot}P))+P$ (three doublings and one addition = four operations) and the second way is $Q = 9P = P+P+P+P+P+P+P+P+P$ (nine additions). So, the first way is in general much faster. That's why we chose this way in our computing.

So, we first double the point $P(x_p, y_p)$ and then add it to the result. If $R = 2P$, the doubling formula is:

$$x_r = s^2 - 2x_p$$
$$y_r = s(x_p - x_r) - y_p$$
where $s = (3x_p^2 + a)(2y_p)^{-1}$ in Z_{11}.

So, we have:

$$\begin{aligned} s &= (3x_p^2 + a)(2y_p)^{-1} \bmod 11 = (3 \cdot 5^2 + 7)(2 \cdot 1)^{-1} \bmod 11 \\ &= (75 + 7) \cdot 2^{-1} \bmod 11 = 82 \cdot 6 \bmod 11 = 492 \bmod 11 \\ &= 8. \end{aligned}$$

$$\begin{aligned} x_r &= s^2 - 2x_p \bmod 11 = 8^2 - 2 \cdot 5 \bmod 11 \\ &= 64 - 10 \bmod 11 = 54 \bmod 11 \\ &= 10. \end{aligned}$$

$$\begin{aligned} y_r &= s(x_p - x_r) - y_p \bmod 11 = 8\,(5 - 10) - 1 \bmod 11 = 8\,(-5) - 1 \bmod 11 \\ &= 8(-5 + 11) - 1 \bmod 11 = 8 \cdot 6 - 1 \bmod 11 = 47 \bmod 11 \\ &= 3. \end{aligned}$$

Thus, we get $R = 2P = (10, 3)$. We now compute $Q = 3P = 2P + P = R + P$, i.e. we have to add the points $P(x_p, y_p) = (5, 1)$ and $R(x_r, y_r) = (10, 3)$. We do this according to the following equations:

$$x_q = s^2 - x_p - x_r$$
$$y_q = s(x_p - x_q) - y_p$$
where $s = (y_r - y_p)(x_r - x_p)^{-1}$ in Z_{11}.

So, we have:

$$\begin{aligned} s &= (y_r - y_p)(x_r - x_p)^{-1} \bmod 11 = (3-1)(10-5)^{-1} \bmod 11 \\ &= 2 \cdot 5^{-1} \bmod 11 = 2 \cdot 9 \bmod 11 = 18 \bmod 11 \\ &= 7. \end{aligned}$$

$$\begin{aligned} x_q &= s^2 - x_p - x_r \bmod 11 = 7^2 - 5 - 10 \bmod 11 \\ &= 49 - 15 \bmod 11 = 34 \bmod 11 \\ &= 1. \end{aligned}$$

$$\begin{aligned} y_q &= s(x_p - x_q) - y_p \bmod 11 = 7\,(5-1) - 1 \bmod 11 \\ &= 7 \cdot 4 - 1 \bmod 11 = 27 \bmod 11 \\ &= 5. \end{aligned}$$

The result is $3P = Q(x_q, y_q) = (1, 5)$.

Solution for Assignment 24:

a) The public elements of the system are the curve $E_{17}(7, 8)$ and the generator $G(2, 8)$. Alice wants to send an encrypted message to Bob.

To encrypt the message "Christmas" = $M(13, 1)$, Alice uses Bob's public key $K_b = (12, 16)$ and the random integer $k = 3$.
She computes the values
$C_1 = kG = 3 \cdot (2, 8) = (13, 1)$ and
$C_2 = M + kK_b = (13, 1) + 3 \cdot (12, 16) = (13, 1) + (1, 13) = (4, 7)$.
The ciphertext is the pair $C(C_1, C_2)$. So, the ciphertext Alice sends to Bob is $C = (13, 1, 4, 7)$.

b) Bob decrypts the received ciphertext C with his private key. His private key is some random integer b he has chosen and then he has computed his public key as $K_b = bG$. Since we know his public key and the generator G, we can try to compute his private key, i.e. we can try to solve the equation $(12, 16) = b \cdot (2, 8)$ in the curve $E_{17}(7, 8)$ by finding the discrete logarithm of $K_b(12, 16)$ to the base $G(2, 8)$. Since all parameters in this example are artificially small (and thus insecure), we can try to find the discrete logarithm by testing all possible values $2G, 3G, 4G, ..., bG$ and see which of these values matches to $K_b(12, 16)$.
Using the Java-Applet we get $2G = (2, 8)$, $3G = (13, 1)$, $4G = (1, 13)$, ...,$16G = (12, 16)$, i.e. $b = 16$. Thus, Bob's private key is $b = 16$. In real systems these parameters are of the magnitude of $2^{150} - 2^{300}$ and it would be impossible to get the discrete logarithm by testing all possible values or by employing some other algorithms for finding the discrete logarithms.

c) Bob receives the ciphertext $C(C_1, C_2) = (13, 1, 4, 7)$ and uses his private key $b = 16$ to decrypt the message. He gets the plaintext message M by computing $C_2 - bC_1$, i.e.

$$\begin{aligned} M = C_2 - bC_1 &= (4,7) - 16 \cdot (13,1) = (4,7) - (1,13) \\ &= (4,7) + (-(1,13)) = (4,7) + (1,-13) = (4,7) + (1,-13+17) \\ &= (4,7) + (1,4) \\ &= (13,1). \end{aligned}$$

This is the original message M that Alice sent to Bob.

Solution for Assignment 25:

The properties of an electronically signed document are data integrity, authentication and non-repudiation.

- Data integrity: The assurance that the data received was exactly the data sent.
- Authentication: The guarantee that the individual sending the message really is the one he or she claims to be.
- Non-repudiation: Prevents a user from denying having signed a message after having done so.

Solution for Assignment 26:

$n = 127 \cdot 127$, $e = 5$ and $d = 22781$. The resulting signature is $s = 7003$.

Solution for Assignment 27:

a) Computation of Alice's private key d:

$$\begin{aligned} p &= 13 \\ q &= 19 \\ n &= p \cdot q = 247 \\ \rho(n) &= (p-1)(q-1) = 12 \cdot 18 = 216 \\ e &= 25. \end{aligned}$$

Public key:

$$(e, n) = (25, 247).$$

Alice's private key d:

$$\begin{aligned} de &\equiv 1 \, mod \, \rho(n) \\ d \cdot 25 &\equiv 1 \, mod \, 216 \\ d &= 121. \end{aligned}$$

b) Creation of the digital signature of message m:

$$m = (11001011)_2 = 203$$
$$s = m^d \, mod \, n = 203^{121} \, mod \, 247 = 34.$$

c) Alice has sent the message $m = 203 = 11001011$ and the signature $s = 34$ to Bob. To verify Alice's signature, Bob does the following: In order to authenticate the signature he uses Alice's public key $(e, n) = (25, 247)$ to compute $s^e \, mod \, n$ and compares the result with m. If $s^e \, mod \, n = m$ he has successfully authenticated the signature.

$$s^e \, mod \, n = 34^{25} \, mod \, 247 = 203 = m.$$

Since no one but Alice possesses her private key, the signature is authentic and the message has not been altered since Alice signed it and no error occurred during transmission.

d) Bob receives the altered message $m' = (11010011)_2 = 211$ and the signature $s = 34$. He uses Alice's public key to verify the signature:

$$s^e \, mod \, n = 34^{25} \, mod \, 24 = 203 \neq 211.$$

Since $s^e \, mod \, n \neq m'$ Bob knows that the message has been altered since Alice signed it.

Solution for Assignment 28:

a) This property means that for a given message m it is computationally infeasible to find a message m' such that $H(m') = H(m)$ for $m' \neq m$. If this property is not fulfilled, an attacker can intercept a valid message-signature pair (m, sig) on the communication chanal and change it to (m', sig). Because $H(m') = H(m)$, sig would be a valid signature for m', i.e. a forgery.

b) This property means that for *any* message m it is computationally infeasible to find a message m' such that $H(m') = H(m)$ for $m' \neq m$. If this property is not fulfilled, an attacker can first find any two messages $m' \neq m$ such that $H(m') = H(m)$. Then he gives m to Alice and persuades her to sign the message digest $H(m)$, obtaining sig. This signature would also be valid for the message m' and the attacker can use the pair (m', sig) as a valid forgery.

c) This property means that when hash a value h is given, it is computationally infeasible to compute the origin message m, i.e. given message m the hash value $h = H(m)$ is easy to compute, but given hash h, the message $m = H^{-1}(h)$ can not be computed. Using hash functions with the one-way property, it is impossible

to forge signatures on random hash values h. If it would be possible to invert the hash function, an attacker could compute signature sig on a random hash value h and then find a message m such that $h = H(m)$. Then the pair (m, sig) would be a valid forgery.

Solution for Assignment 29:

If $P(n,k)$ denotes the probability that we have at least one duplicate in k items, where each item is able to take one of n equally likely values between 1 and n (general birthday paradox), then this can be computed as:

$$P(n,k) = 1 - \frac{n!}{(n-k)!\, n^k}.$$

Using some transformations, we can get the simplified relation for this probability:

$$P(n,k) > 1 - e^{-\frac{k(k-1)}{2n}}.$$

From this relation we can deduce the value of k for a given probability. To achieve the probability of $0.5\,(50\%)$, we should take $k \approx \sqrt{n}$ values.

We can use this result in order to determine a "secure" length of hash values, i.e. to determine an appropriate length which minimizes the probability of collision (i.e. that two messages m and m' with variable length have the same hash value). The result of the equation above implies that hashing just over $\sqrt{n}$ random values yields a collision with the probability of 0.5. If we have a hash function with an m-bit output (i.e. 2^m possible outputs), then with hashing

$$k = \sqrt{2^m} = 2^{\frac{m}{2}}$$

random inputs, the probability of getting a duplicate hash values is 0.5. Thus, the length of the output m must be large enough, so that the computing of $2^{\frac{m}{2}}$ hash values in order to find a collision would be computationally infeasible. The 128-bit hash value guarantees enough security, because computing of $2^{\frac{128}{2}} = 2^{64} > 10^{19}$ values for finding a collision is infeasible with respect to the computational power available today.

Solution for Assignment 30:

a) Hash functions are usually computed by a sequence of similar compression steps (iterations) through which a given message m is processed block-wise to a hash value $h(m)$. An input message m of arbitrary finite length is divided into fixed-length n-bit blocks m_i. Before the iterations begin, a preprocessing initialisation step is carried out. This typically involves appending extra bits (padding) as necessary to attain an overall bit length which is a multiple of the block length n, and often includes for security reasons a block indicating the bit length of the unpadded input. Then the loop for processing the blocks m_i begins. Each block m_i serves as an input to an internal fixed-size compression function, which computes

a new intermedia result of bit length k (k is fixed), as a function of the previous k-bit intermediate result and the input block m_i. After all message blocks m_i are processed, some optional output transformation is possible, before the hash value is given as output of the algorithm.

b) Regarding the design of the compression function, the preprocessing and the output transformation, we distinguish between four categories of iterated hash functions:

1. Hash functions based on symmetric block ciphers. Hash functions based on symmetric block ciphers make it possible to use cryptographic techniques which are already implemented and they use the know-how which already exists for designing block ciphers. The idea is that if the block algorithm is secure, then the one-way hash function will also be secure. The block cipher is iteratively used as an internal compression function. The general scheme is as follows:

$$\begin{aligned} h_0 &= IV, \, where \, IV \, is \, a \, random \, initial \, value \\ h_i &= E_A(B) \oplus C(for \, all \, blocks \, i) \\ h &= h_b \end{aligned}$$

   where A, B and C can be either m_i, h_{i-1}, ($m_i \oplus h_{i-1}$), or a constant.

2. Hash functions based on modular arithmetic. The basic idea is to construct an iterated hash function using $mod\, n$ arithmetic as basis of the compression function. Motivating factors are the re-use of existing software or hardware for modular arithmetic and scalability to match required security levels. A significant disadvantage however is that the computation of such functions is very slow. A well known and wide spread example is the "squaring modulo n" function.

3. Dedicated hash functions. Dedicated or customized hash functions are ones which are specially designed for the explicit purpose of hashing, with optimized performance in mind, and without being constrained to re-use existing system components such as block ciphers or modular arithmetic. Dedicated hash functions have become more and more important in recent years. Those who have received the greatest attention in practice are based on the MD4 hash function.

4. Provable secure hash functions. These are designed using some of the hard solvable number theoretic problems as kernel of the compression function. A challenge in this approach is that all possible attacks lead to the ability to solve the referenced problem, which is considered infeasible, given current knowledge and an opponent with bounded resources. (A cryptographic method is said to be provably secure if it can be shown that breaking the method is essentially as difficult as solving a well-known problem, such as integer factorisation or the computation of discrete logarithms). A well known example is the Chaum-van

Heijst-Pfitzmann (CHP) hash function, whose security is based on the discrete logarithm problem.

Solution for Assignment 31:

a) The most commonly used MAC algorithms based on block cipher make use of the CBC-mode. Input of the algorithm (see Fig. 7.6-1) is the message m and the secret MAC-key K. The message m is first padded if necessary (preprocessing), and then devided into b blocks $m_1, ..., m_b$, each of size l which is the processing size of the block cipher. If E_K denotes encryption using the algorithm E with the key K, then the block H_b (which is the MAC value) is computed as follows:

$$H_1 = E_K(m_1)$$
$$H_i = E_K(H_{i-1} \oplus m_i) for\, 2 \leq i \leq b$$
$$H = H_b.$$

b) A common approach is to construct a MAC from a hash algorithm by simply including the secret key as part of the hash input. There are several constructions:

1. Secret prefix method. In this method, the MAC of the message m is $MAC = H(K||m)$, i.e. the key K is appended at the beginning of the message, and then the hash value is computed (security concerns arise!).

2. Secret suffix method. In this method, the MAC of the message m is $MAC = H(m||K)$, i.e. the key K is appended at the end of the message, and then the hash value is computed (security concerns arise!).

3. Constructing MAC using the key K as initial value IV of some iterated hash function (security concerns arise!).

4. Envelope method with padding. In this method, the key K is appended at both the start and the end of the hash computation: $MAC = H(K||p||m||K)$. Here p is a string used to pad K to the length of one block. For example, if H is MD5 and K is 128 bits, p is a 384-bit string.

5. Hash-based MAC: In this method, the MAC of the message m is $MAC = H(K||p_1||H(K||p_2||m))$, where p_1 and p_2 are distinct strings of sufficient length to pad K out to the full block for the compression function.

Solution for Assignment 32:

a) hash md5('Die Pruefung werde ich bestehen') =
108539661692143697175881964544052334954

b) hash md5('die Pruefung werde ich bestehen') =
258964952933001188505809377677471075 15

c) hash sha1('Die Pruefung werde ich bestehen') =
950425923872302811263129281384413934060031762536

d) hash sha1('die pruefung werde ich bestehen') =
108561878832503001441850528497942532785064678 6652

e) hash squaremod('Die Pruefung werde ich bestehen',10552043297) = 6967079270

f) hash squaremod('die pruefung werde ich bestehen',10552043297) = 6967079270

Solution for Assignment 33:

One possible example of the Feige-Fiat-Shamir identification protocol:

1. Selection of system parameters. After selecting two secret primes $p = 683$ and $q = 811$, the trusted third party T publishes $n = pq = 553913$. Furthermore, $k = 3$ and $t = 1$ are selected.

2. Selection of per-entity secrets. Entity *A* does the following:

   a. *A* selects 3 random integers $s_1 = 157$, $s_2 = 43215$, and $s_3 = 4646$. The three random bits are $b_1 = 1$, $b_2 = 0$, and $b_3 = 1$.

   b. *A* computes $v_1 = 441845$, $v_2 = 338402$, and $v_3 = 124423$.

   c. *A*'s public key is (441845; 338402; 124423; 553913). *A*'s private key is (157; 43215; 4646).

3. Protocol actions:

   a. *A* chooses a random integer $r = 1279$ and a random bit $b = 1$. A then computes $x = 25898$ and sends it to *B*.

   b. *B* sends the 3-bit vector (0; 0; 1) to *A*.

   c. *A* computes the response $y = r \cdot s_3 \mod n = 403104$ and sends it to *B*.

   d. *B* computes $z = y^2 \cdot v_3 \mod n = 25898$ and accepts *A*'s identity since $z = \pm x$ and $z \neq 0$.

Solution for Assignment 34:

One possible example of the GQ identification protocol:

1. Selection of system parameters.

   a. The trusted third party *T* selects random primes $p = 569$ and $q = 739$ yielding $n = pq = 420491$.

b. T computes $\phi = (p-1)(q-1) = 419184$ and selects a public exponent $v = 54955$. *T* then computes $s = v^{-1} \mod \phi = 233875$.

c. The system parameters $(v, n) = (54955, 420491)$ are made available for all users.

2. Selection of per-entity secrets. Entity *A* does the following:

a. Provided that *A*'s redundant identity $J_A = 34579$.

b. *T* gives the secret $s_a = (J_A)^{-s} \mod n = 403154$ to *A*.

3. Protocol actions:

a. *A* selects a random integer $r = 65446$, i.e. the commitment, $1 \leq r \leq n-1$, and computes the witness $x = r^v \mod n = 89525$.

b. *A* sends the pair of integers $(I_A, x) = (I_A, 89525)$ to *B*.

c. *B* selects the challenge $e = 38980, 1 \leq e \leq v$, and sends it to *A*.

d. *A* computes the response $y = r \cdot s_{A^e} \mod n = 83551$ and sends it to *B*.

e. *B* computes $z = J_{A^e} \cdot y^v \mod n = 89525$ and accepts *A*'s proof since $z = x$ and $z \neq 0$.

Solution for Assignment 35:

- Storage of keying material refers to a key storage facility which provides secure storage of keys for future use, e.g. confidentiality and integrity for secret keying material, or integrity for public keys. Secret keys must be protected by physical security (e.g. by storing it within a cryptographic device) or enciphered by keys that have physical security. For all keying material, unauthorized modification must be detectable by suitable authentication mechanisms.
- Assuming an encrypted data link where users want to change keys daily, the effort to distribute a new key every day is laborious. An easier solution is to generate a new key from the old key, and this process is called key updating. This can be done using a one-way function. Two entities sharing the same key and both operating on it using the same one-way function, they will get the same result. This result can be used to create a new key, but it is obvious that the new key is only as secure as the old one. If an adversary get access to the old key.
- Key destruction refers to procedures by which parties are assured of the secure destruction of keys that are no longer needed. Destroying keys eliminating all records of this key, such that no information remaining after the deletion provides any usable information about the destroyed key. A key may be destroyed by overwriting it with a new key or by zeroizing it. Keying material stored on magnetic media should either be zeroized or the media itself should be destroyed.

Solution for Assignment 36:

Diffie-Hellman key exchange with three parties is as follows:

1. A chooses a large random integer *x* and sends
   $X = gx \mod n$ to B.
2. B chooses a large random integer *y* and sends
   $Y = g^y \mod n$ to C.
3. C chooses a large random integer *z* and sends
   $Z = g^z \mod n.$ to A.
4. A sends to B
   $Z' = Z^x \mod n.$
5. B sends to C
   $X' = X^y \mod n.$
6. C sends to A
   $Y' = Y^z \mod n.$
7. A computes
   $k = Y'^x \mod n.$
8. B computes
   $k = Z'^y \mod n.$
9. C computes
   $k = X'^z \mod n.$

The secret key *k* is equal to $g^{xyz} \mod n$ and no one else listening in on communications can compute that value.

Solution for Assignment 37:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 3846 (0xf06)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=DE, ST=NRW, L=Hagen, O=FernUni in Hagen, OU=URZ,
                CN=Certification Authority (CA)
                /emailAddress=caadmin@fernuni-hagen.de
        Validity
            Not Before: Dec  9 10:02:27 2003 GMT
            Not After : Dec  8 10:02:27 2005 GMT
        Subject: C=DE, O=FernUni Hagen, OU=Mitarbeiter,
                 CN=Thorsten Kisner -- kisner
                 /emailAddress=thorsten.kisner@fernuni-hagen.de
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:b7:45:b0:2e:7c:20:fb:06:09:ab:41:17:d5:8c:
                    65:c4:d2:c2:53:81:55:fe:f4:1c:9b:c9:5e:dd:cc:
                    bc:54:ba:b7:4d:2e:6a:08:20:fa:46:e8:7d:c6:f7:
                    f1:1e:01:4b:47:82:4c:fd:45:1f:d8:c4:8a:fc:dc:
```

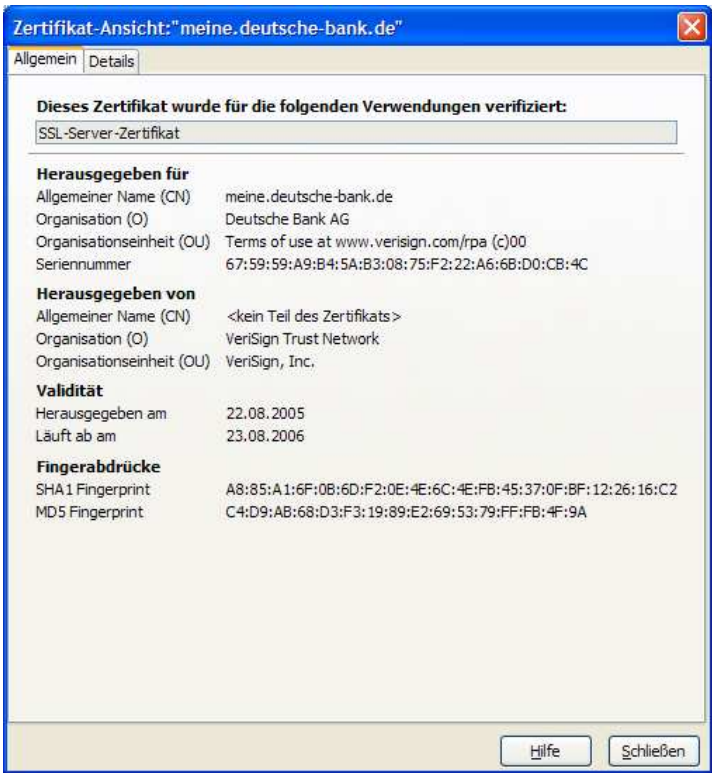
```
                    7d:68:9c:49:05:c3:cf:3f:73:72:2e:87:60:70:63:
                    1c:55:86:3b:f9:89:05:a0:b0:2f:cf:ad:c7:d8:16:
                    e3:0b:68:02:45:23:86:15:03:c8:f7:2d:e8:21:4c:
                    f3:91:b0:b9:ec:32:30:0a:9d:42:2f:a5:37:59:92:
                    ca:93:55:b5:a5:dc:f8:74:2b
                Exponent: 65537 (0x10001)
            X509v3 extensions:
                X509v3 Basic Constraints:critical
                    CA:FALSE
                Netscape Cert Type:
                    SSL Client, S/MIME
                X509v3 Key Usage:
                    Digital Signature, Non Repudiation, Key Encipherment,
                    Data Encipherment, Key Agreement
                X509v3 Extended Key Usage:
                    TLS Web Client Authentication, E-mail Protection,
                    TLS Web Client Authentication
                Netscape Comment:
                    Issued by: FernUni-Hagen, Certification Authority (CA)
                X509v3 Subject Key Identifier:
              EC:2A:44:EA:A8:33:9D:44:3C:41:06:E6:EF:F2:60:6B:25:20:C1:8D
                X509v3 Authority Key Identifier:
              keyid:D2:CE:51:43:8B:C8:1B:1C:AA:BC:2D:B0:C1:2F:D2:F8:5C:3B:45:54
                    DirName:/C=DE/ST=NRW/L=Hagen/O=FernUni in Hagen
                            /OU=URZ/CN=Certification Authority (CA)
                            /emailAddress=caadmin@fernuni-hagen.de
                    serial:00

             Netscape CA Revocation Url:
             https://ca.fernuni-hagen.de/certserver/certs/ca-crl2003.pem
             Netscape Base Url:
             https://ca.fernuni-hagen.de/certserver
             Netscape CA Policy Url:
               https://ca.fernuni-hagen.de/certserver/help/policy.pdf
             X509v3 CRL Distribution Points:
             URI:https://ca.fernuni-hagen.de/certserver/certs/crl2003.crl

        Signature Algorithm: md5WithRSAEncryption
            91:eb:9a:07:32:54:be:58:63:59:39:ab:50:d3:75:e4:3b:ba:
            90:0e:71:2d:a8:e1:9b:c9:61:3a:5f:70:c6:e7:48:79:64:b3:
            6a:8b:3f:ce:f4:ac:4a:de:56:2b:98:b1:cb:58:03:0c:66:56:
            88:28:41:49:ca:5a:24:a0:6f:d0:f8:32:4a:73:0f:0a:7a:83:
            50:7b:d4:3b:31:bb:09:3a:b0:7c:84:b5:5f:62:d5:93:ae:8d:
            67:43:c1:ba:cc:b5:fe:1c:6b:80:e6:dd:0b:ad:b6:70:64:97:
            d7:0f:e3:b1:a7:81:41:fa:cd:44:af:1d:22:9f:98:fc:59:15:
            13:18:d6:70:e3:f8:41:e9:da:42:3a:7d:79:c0:0f:14:34:7a:
            41:ba:d9:b9:f8:03:03:55:62:16:cc:8a:d6:54:12:ec:47:e8:
            5c:03:f0:3e:bb:c3:93:5e:a5:7b:f9:f2:df:bd:12:5b:e2:e0:
            46:90:c1:c2:78:76:79:a5:ce:52:c4:d0:35:24:e3:e1:c3:59:
            9d:d3:95:b0:32:05:d6:63:aa:41:7d:71:8b:e0:dc:7d:a2:c9:
            32:bf:e9:d3:4b:ff:9b:bb:94:6b:8c:ac:c0:f6:b6:c5:30:31:
            39:d8:89:a9:d4:56:8b:92:7b:11:48:56:fb:dd:30:01:00:b6:
            15:f8:d4:7a
```

Solution for Assignment 38:

The certificate is signed by *VeriSign, Inc*. The browser explicitly trusts all certificates signed by this TTP.



Solution for Assignment 39:

No, the correct fingerprint is `d2 a8 f0 24 9e 2f 5a 62 db 5d 9c 14 3e b4 a7 d4 3f 77 1f 32`

Solution for Assignment 40:

a) An OpenPGP key should carry a self-certificate to guarantee the integrity of the key-User ID binding. If the binding was not signed, an attacker could replace the user ID (including the e-mail address). This can be regarded as a kind of denial-of-service attack.

b) 1. This method is very insecure. E-mails can easily be intercepted and altered. This method makes man-in-the-middle attacks very easy.

2. If Alice and Bob know each other this method is very secure. Alice can authenticate Bob by his voice. The hash value of the public key is for all practical purposes unique.

3. Since Alice does not know the CA she cannot trust it. She even does not know if the binding of public key to user ID is correct. Anybody can generate a key pair and bind an arbitrary user ID to it. Alice has two problems: firstly, she

cannot authenticate the CA and secondly she does not know and trust the CA. Consequently, Alice should not rely on the certificate from the alleged CA.

4. This method is as insecure as the unencrypted e-mail case above. Alice's public key is public (as the name suggests). Anybody can use this key to send an encrypted e-mail to Alice. Encrypting an e-mail with Alice's public key does not authenticate the sender.
5. This method is reasonably secure. Alice received Bob's public key (again, for all practical purposes the hash value of the key is unique) in a strong context. She could authenticate the public key via this context. If a different person than Bob had sent the e-mails, Alice normally should have noticed. Nevertheless, whether Alice is willing to depend on this evidence depends on her own personal security policy.
6. This method is rather secure. Again, the public key is submitted in a strong context. The context is a letter with Bob's handwriting on it and an attached floppy disk. It is much harder to intercept and exchange traditional mails than it is to modify e-mails.

Solution for Assignment 41:

a) The certificate may look as follows (without signature):

```
(cert
 (issuer
  (public-key
   (rsa-md5
    (e #42682163418151075735531010069604871163#)
    (n #19993932509812921571414962404567286697 3#))))
 (subject
  (public-key
   (rsa-md5
    (e #30103164899134424358636143543737677#)
    (n #13374236066348748665557369890605911200 7#))))
 (propagate)
 (tag (support BigBucks)
 (not-before "2001-05-01_00:00:00")
 (not-after "2001-06-01_00:00:00")
```

The signed certificate is contained in the solution as a separate file.

b) DSA public keys have four parameters: (p, q, g, y). The DSA signature consists of two parts, *r* and *s*. Furthermore, the algortihm identifier changes, e.g. into `(public-key(dsa-sha1(...)))`.

References

References for Chapter Introduction

[Buchmann99a] Johannes Buchmann. *Einführung in die Kryptographie.* Springer Verlag, 1999.

[Diffie76a] W. Diffie and Martin E. Hellman. *New directions in cryptography.* IEEE Transactions on Information Theory, 22(6):644-654, November 1976.

[Ibraham92] A. Ibraham. *Al-Kindi: The origins of cryptology: The Arab contributions.* Cryptologia, vol. 16, no. 2 (April 1992) pp. 97-126.

[Kahn67a] David Kahn. *The Codebreakers.* Macmillan Publishing Company, New York, 1967.

[Menezes96a] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[Rivest78a] R. L. Rivest, A. Shamir and L. Adleman. *A method for obtaining digital signatures and public-key cryptosystems.* Communication of the ACM, 21(2):120-126, February 1978.

[Schneier96a] Bruce Schneier. *Applied Cryptography: protocols, algorithms, and source code in C.* John Wiley and Sons, 1996.

[Schneier04] Bruce Schneier. *Secrets and Lies - Digital Security in a Networked World.* Wiley Publishing, Inc., Indianapolis, 2004.

[Shannon49a] Claude Shannon. *Communication Theory of Secrecy Systems.* Bell Systems Technical Journal, 28:656-715, 1949.

[Shannon49b] W. Weaver, C. Shannon. *The Mathematical Theory of Communication.* University of Illinois Press, 1949.

[Stinson95b] Douglas R. Stinson. *Cryptography: Theory and Practice.* CRC Press, 1995.

References for Chapter Mathematical Background

[Buchmann99] Johannes Buchmann. *Einführung in die Kryptographie.* Springer Verlag, 1999.

[Cormen89] Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest. *Introduction to algorithms.* MIT Press, 1989.

[Herstein86] I. N. Herstein. *Abstract Algebra.* Macmillan Publishing Company, 1986.

[Jackson87] T. H. Jackson. *From Number Theory to secret Codes.* IP Publishing Ltd., 1987.

[Koblitz94] Neal Koblitz. *A Course in Number Theory and Cryptography.* Springer Verlag, 1994.

[Koblitz98] Neal Koblitz. *Algebraic Aspects of Cryptography.* Springer Verlag, 1998.

[Lidl94] Rudolf Lidl, and Harald Niederreiter. *Introduction to finite fields and their applications.* Cambridge University Press, 1994.

[Menezes96] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[Schneier96] Bruce Schneier. *Applied Cryptography: protocols, algorithms, and source code in C.* John Wiley and Sons, 1996.

References for Chapter Stream Ciphers

[Blum84] Manuel Blum and Silvio Micali. *How to generate cryptographically strong sequences of pseudo-random bits.* SIAM Journal on Computing, 13(4):850-864, November 1984.

[Fumy94] Walter Fumy and Hans Peter Rieß. *Kryptographie: Entwurf, Einsatz und Analyse symmetrischer Kryptoverfahren*. Oldenbourg Verlag, 1994.

[Kaukonen99] Kalle Kaukonen and Rodney Thayer. *A stream cipher encryption algorithm arcfour.* Technical report, SSH Communications Security, July 1999. Internet Draft, http://ftp.ietf.org/internet-drafts/draft-kaukonencipher- arcfour-03.txt.

[Knudsen98] Lars. R. Knudsen, Willi Meier, Bart Preneel, Vincent Rijmen, and Sven Verdollaege. *Analysis methods for (alleged) rc4*, pp. 327–341. Springer Verlag, 1998.

[Lidl94] Rudolf Lidl and Harald Niederreiter. *Introduction to finite fields and their applications.* Cambridge University Press, 1994.

[Massey69] James L. Massey. *Shift register synthesis and BCH decoding.* IEEE Transactions on Information Theory, 15(1):122–127, January 1969.

[Mister98] S. Mister and Stafford E. Tavares. *Cryptanalysis of rc4-like ciphers.* pp. 136–148. Springer Verlag, 1998.

[Rivest92] Ronald L. Rivest. *The rc4 encryption algorithm.* Technical report, RSA Data Security, Inc., 1992, not published.

[Rueppel86] Rainer A. Rueppel. *Analysis and Design of Stream Ciphers.* Springer-Verlag, 1986.

[Stinson95] Douglas R. Stinson. *Cryptography: Theory and Practice.* CRC Press, 1995.

[Yao82] Andrew C. Yao. *Theory and applications of trapdoor functions.* In Proc. of the 23th Annu. IEEE Symp. on Foundations of Computer Science, pp. 80–91, 1982.

References for Chapter Block Ciphers

[Biham93] Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard.* Springer Verlag, 1993.

[Biham93a] Eli Biham. *New types of cryptanalytic attacks using related keys.* Advances in Cryptology, Proceedings Eurocrypt'93, LNCS 765, 1993.

[Daemen94] Joan Daemen, René Govaerts and Joos Vandewalle. *Weak keys for idea,* pp. 224–231.

[Daemen97] Joan Daemen, Lars R. Knudsen and Vincent Rijmen. *The Block Cipher Square.* Proceedings of Fast Software Encryption Workshop 1997, 1997.

[Daemen99] Joan Daemen and Vincent Rijmen. *AES Proposal: Rijndael.* Technical report, NIST, 1999.

[Fer03] Niels Ferguson and Bruce Schneier. *Practical Cryptography.* Wiley Publishing, 2003.

[Fips81] FIPS. *Des modes of operation.* Technical report, Federal Processing Standards Publications 46-1, U.S. Department of Commerce/N.I.S.T. National Technical Information Service, Springfield Virginia, 1980. FIPS 81.

[Fips01] FIPS. *Advanced encryption standard (aes).* Technical report, Federal Processing Standards Publications 197, U.S. Department of Commerce/N.I.S.T. National Technical Information Service, Springfield Virginia, 2001. FIPS 197.

[Fumy94] Walter Fumy and Hans Peter Rieß. *Kryptographie: Entwurf, Einsatz und Analyse symmetrischer Kryptoverfahren.* Oldenbourg Verlag, 1994.

[Gordon82] J. A. Gordon and H. Retkin. *Are big s-boxes best?* pp. 257–262. Springer Verlag, 1982.

[ISO91] ISO. *Information processing - modes of operation for an n-bit block cipher algorithm.* Technical report, International Organization for Standarization, 1991. ISO/IEC 10116.

[Jakobsen97] Thomas Jakobsen and Lars R. Knudsen. *The Interpolation Attack on Block Ciphers.* Proceedings of Fast Software Encryption Workshop 1997, 1997.

[Kad97] Firoz Kaderali, Hagen Hagemann, Heino Hirsekorn, Heinz Müller and Andreas Rieke. *Kurs 02553 Technischer Datenschutz in Kommunikationsnetzen, FernUniversität in Hagen (Interaktiver CD-ROM-Kurs, inklusive Interpreter für kryptologische Protokolle)*. Addison-Wesley, 1997.

[Knudsen95] Lars R. Knudsen. *Truncated and higher order differentials.* Fast Software Encryption, LNCS 1008, B. Preneel, 1995.

[Lai91] Lai X. and J.L. Massey. *A proposal for new block encryption standard.* Advances in Cryptology, Proceedings Eurocrypt'90, LNCS 437, 1991.

[Lai92] Lai X. *On the Design and Security of Block Ciphers.* ETH Series in Information Processing, v.1, Konstanz: Hartung-Gorre Verlag, 1992.

[Matsui94] Mitsuru Matsui. *The First Experimental Cryptanalysis of the Data Encryption Standard.* Advances in Cryptology - CRYPT0 94, LNCS 839, 1994.

[Menezes96] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

Online Version available from: http://www.cacr.math.uwaterloo.ca/hac/

[Schneier96] Bruce Schneier. *Applied Cryptography: protocols, algorithms, and source code in C.* John Wiley and Sons, 1996.

References for Chapter Public-Key Encryption

[Diffie76] W. Diffie and Martin E. Hellman. *New directions in cryptography.* IEEE Transactions on Information Theory, 22(6):644–654, November 1976.

[Elgamal85] Taher ElGamal. *A public key cryptosystem and a signature scheme based on discrete logarithms.* pp. 10–18. Springer Verlag, 1985.

[Koblitz87] Neal Koblitz. *Elliptic curve cryptosystems.* Mathematics of Computation, 48(177):203–209, January 1987.

[Menezes93] Alfred Menezes. *Elliptic Curve Public Key Cryptosystems.* Kluwer Academic Publishers, 1993.

[Menezes96] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[Miller85] Victor S. Miller. *Use of elliptic curves in cryptography.* pp. 417– 426. Springer Verlag, 1986.

[Rivest78] R. L. Rivest, A. Shamir and L. Adleman. *A method for obtaining digital signatures and public-key cryptosystems.* Communications of the ACM, 21(2):120–126, February 1978.

References for Chapter Digital Signatures

[CRS83] David Chaum, Ronald L. Rivest and A.T. Sherman, editors. *Advances in Cryptology,* CRYPTO'82. Springer Verlag, 1983.

[JM99] D.B. Johnson and A.J. Menezes, editors. *Elliptic Curve DSA (ECDSA): An Enhanced DSA,* Technical Report CORR 99-34, Department of C&O, University of Waterloo, 1999.

[MOV96] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[Pfi96] B. Pfitzmann, editor. *Digital Signature Schemes - General Framework and Fail-Stop Signatures,* LNCS 1100. Springer Verlag, 1996.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. *A method for obtaining digital signatures and public-key cryptosystems.* Communications of the ACM, 21(2):120–126, February 1978.

References for Chapter Hash Functions and Authentication Codes

[Chaum92] D. Chaum, E. van Heijst and B. Pfitzmann. *Cryptographically strong undeniable signatures, unconditionally secure for the signer.* In Proceedings of CRYPTO'91, pp. 470– 484.

[Lai92] X. Lai and J. Massey. *Hash functions based on block ciphers.* In Rainer A. Rueppel, editor, Proceedings of EUROCRYPT' 92, pp. 55–70. Springer Verlag, 1993.

[Matyas85] S.M. Matyas, C.H. Meyer, and J. Oseas. *Generating strong one-way functions with cryptographic algorithm.* IBM Technical Disclosure Bulletin, 27(10), 1989.

[Menezes96a] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[Merkle90] R.C. Merkle. *A fast software one-way hash function.* Journal of Cryptology, 3(1):43–58, 1990.

[Meyer88] C.H. Meyer and M. Schilling. *Secure program load with manipulation detection code.* Proceedings of Securicom, pp. 111–130, 1988.

[NIST92] National Institute for Standards and Technology (NIST). *Proposed federal information processing standard for secure hash standard.* Federal Register, Jan. 1992.

[RACE92] Research and Development in Advanced Communication Technologies in Europe. *Ripe integrity primitives.* Final Report of RACE Integrity Primitives Evaluation (R1040), June 1992.

[Rivest90] Ronald Rivest. *The md4 message digest algorithm.* In Proceedings of CRYPTO'90, pp. 303–311.

[Rivest92] Ronald Rivest. *The md5 message digest algorithm.* RFC 1321, Apr. 1992.

[Schneier96a] Bruce Schneier. *Applied Cryptography: protocols, algorithms, and source code in C.* John Wiley and Sons, 1996.

[Stallings99a] William Stallings. *Cryptography and Network Security: Principles and Practice.* Prentice Hall, 1999.

[Stinson95b] Douglas R. Stinson. *Cryptography: Theory and Practice.* CRC Press, 1995.

[Winternitz84] R.S. Winternitz. *Producing one-way hash functions from des.* In David Chaum, editor, Proceedings of CRYPTO'83, pp. 203–207. Springer Verlag, 1984.

[Zheng93] Y. Zheng, Pieprzyk J., and Seberry J. *Haval – a one-way hashing algorithm with variable length of output.* In Jennifer Seberry and Yuliang Zheng, editors, Proceedings of AUSCRYPT'92, pages 83–104. Springer Verlag, 1993.

References for Chapter Entity Authentication

[Jain99a] Anil K. Jain, R. Bolle and Sharath Pankanti. *Biometrics - Personal Identification in Networked Society.* Kluwer Academic Publishers, 1999.

[Menezes96a] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[Monrose99a] F.N. Monrose, M.K. Reiter, and S. Wetzel. *Password hardening based on keystroke dynamics.* pp. 73–82. ACM, 1999.

References for Chapter Key Management Techniques

[ISO96] ISO/IEC. *Information technology - security techniques - key management.* Technical report, International Organization for Standarization, ISO/IEC 11170, 1996.

References for Chapter Public Key Infrastructure

[Callas98] J. Callas, L. Donnerhacke, H. Finney and R. Thayer. OpenPGP Message Format. Technical report, RFC (Request for Comments) 2440, Status: Proposed Standard, IETF (Internet Engineering Task Force), 1998. http://www.ietf.org/rfc/rfc2440.txt.

[Chokhani99] S. Chokhani and W. Ford. *Internet x.509 public key infrastructure certificate policy and certification practices framework.* Technical report, RFC (Request for Comments) 2527, Status: Informational, IETF (Internet Engineering Task Force), 1999. http://www.ietf.org/rfc/rfc2527.txt.

[Daniel00] P. McDaniel and S. Jamin. *Windowed certificate revocation.* pp. 1406–1414.

[IT97a] ITU-T. *ITU-T Recommendation X.500.* Technical report, ITU-T, 1997.

[IT97b] ITU-T. *ITU-T Recommendation X.509.* Technical report, ITU-T, 1997.

[Kaderali00b] Firoz Kaderali, Biljana Cubaleska, Bernhard Löhlein, Sonja Schaup and Oliver Stutzke. *Course - Foundations of Cryptology.* Department of Communication Systems, University of Hagen 2000.

[Kohnfelder76] L. Kohnfelder. *Towards a practical public-key cryptosystem.* Technical report, MIT, 1978.

[Kohnfelder78] L. Kohnfelder. *Towards a practical public-key cryptosystem.* Technical report, MIT, 1978.

[RFC2440] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer. *OpenPGP Message Format.* Technical report, RFC (Request for Comments) 2440, Status: Proposed Standard, IETF (Internet Engineering Task Force), 1998. http://www.ietf.org/rfc/rfc2440.txt.

[Rivest98] R. Rivest. *Can we eliminate certificate revocation lists?* Financial Cryptography, pp. 178-183, 1998.

Index