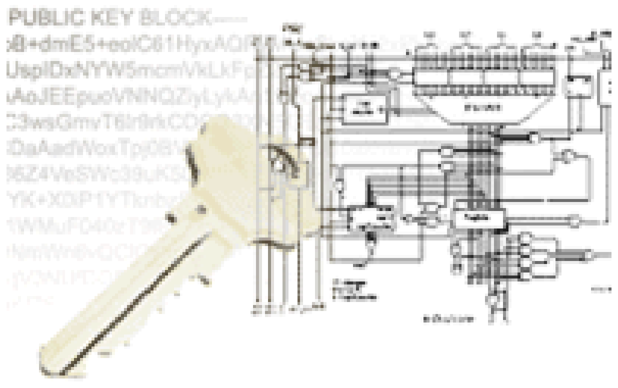
Lecture Notes

Introduction to Information Security



Lecturer: Mario Lamberger

Institute for Applied Information Processing and Communications
Graz University of Technology, Austria

Introduction to Information Security 705.026

Winter Term 2012/2013
Lecture Notes

by
Elisabeth Oswald (extended by Mario Lamberger)
October 22, 2012

Contents

Abbreviations and Standard Notation

Abbreviations

The following abbreviations of standard phrases are used throughout this book:

Abbreviation	Notation
AES	Advanced Encryption Standard
CBC	Cipher Block Chaining mode
CFB	Cipher FeedBack mode
CRL	Certificate Revocation List
DES	Data Encryption Standard
DHP	Diffie–Hellman problem
DLP	Discrete logarithm problem
DN	Distinguished Name
DPA	Differential Power Analysis
DSA	Digital Signature Algorithm
ECB	Electronic Codebook Mode
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
GDLP	Generalized Discrete Logarithm Problem
HW	Hamming Weight
IDS	Intrusion Detection System
IFP	Integer Factorization Problem
LSB	Least Significant Bit
MAC	Message Authentication Code
MDC	Modification Detection Code
MSB	Most Significant Bit
OFB	Output Feedback Mode

to be continued on next page

Abbreviation	Notation
OID	Object Identifier
PGP	Pretty Good Privacy
PKI	Public Key Infrastructure
RSA	Rivest–Shamir–Adleman encryption scheme
SCA	Side-Channel Attack
SPA	Simple Power Analysis
SSCA	Simple Side-Channel Attack
DSCA	Differential Side-Channel Attack

Preface

This book is intended to serve as syllabus for the "Introduction to Information Security" lecture. It should accompany the lecture, but it is not intended as sole means to study for the examination!

This book contains information in a very condensed style. Explanations are brief, and in most cases, examples are not given. Students should consult the slides for examples and more colorful explanations.

After each chapter, a set of so-called review exercises are provided. With these exercises, students can assess their understanding and knowledge of the subject that has been discussed in the chapter.

Several people deserve acknowledgments for proofreading chapters of this book: Udo Payer, Vincent Rijmen and Karl Scheibelhofer.

Chapter 1

Basic Cryptologic Principles

The protection provided by encryption is based on the fact that most people would rather eat liver than do mathematics.
— Bill Neugent

During the last century, digital communication has become a major part of people's every day life. Industrialized countries are evolving towards information societies, where bank cards, mobile phones and wireless Internet access are available to every citizen. Moreover, the value of information keeps growing, while it is being subjected to an increased number of threats such as eavesdropping of communication lines and theft of sensitive data. This clearly demonstrates that there is a strong need for techniques to protect information and information systems. Cryptography is the science of protecting information. A cryptographic algorithm is a mathematical function that uses a key to encipher information. Without knowledge of the key, deciphering is not possible. Contemporary cryptography deals with more issues than plain encryption. Mathematical concepts are known which can be used to construct digital signatures or protocols for entity authentication, for example.

1.1 CIA and Non-repudiation

In the typical scenario, two entities which are called Alice and Bob, want to exchange messages securely over an insecure channel. The adversary, which is called Eve, can have several goals including eavesdropping the communication or altering it, for example. Eavesdropping is a threat to the *confidentiality* of the messages, i.e., no unintended third part can understand the content. Altering is a threat to the *integrity* of the message, i.e., no-one can change the message while it is in transit. In addition, Eve could also try impose one of the communicating parties. Therefore, it is required to *authenticate* (i.e. get assurance of the identity) the communicating parties. The last of the four basic security services deals with *non-repudiation*, i.e., the property, that the sender cannot deny a message was sent. This property cannot be achieved by secret key cryptography alone.

1.2 Secret Key Cryptography

Numerous applications use (software) implementations of cryptographic algorithms to provide security at low cost. The most important requirement for such algorithms, besides their resistance against attacks, is that the performance of the application itself is reduced as little as possible. Three types of secret key cryptographic primitives are discussed in this section. *Block ciphers* are used to encrypt data. If block ciphers are not fast enough for an application, *stream ciphers* can be used as alternative. In order to ensure integrity of data, modification detection codes (MDC) which are also called *(un-keyed) hash functions* are used. For authentication, message authentication codes (MACs) (keyed hash functions) are used.

1.2.1 Block Ciphers

A *block cipher* is defined [DR02] as a set of Boolean permutations operating on n-bit vectors. This set contains a Boolean permutation for each value of a key k. In other words, a block cipher is a length preserving transformation which takes an element (a plaintext) from the set of plaintexts and transforms it to an element (the ciphertext) from the set of ciphertexts under the influence of a key.

Such a block cipher usually consists of several operations (transformations), which form the encryption algorithm. To allow efficient implementations, block ciphers apply the same Boolean transformation several times on a plaintext. The Boolean transformation is then the so-called *round function*, and the block cipher is called an *iterated block cipher*. For each round, a key has to be used. To generate such *round keys* K_i from the cipher key, a *key schedule* algorithm is applied to the cipher key. If the round key is applied in a very simple way, i.e. it is exclusive or-ed (x-ored) to an intermediate value, then the cipher is called *key-alternating block cipher*.

Modern block ciphers are usually based on two different types of designs. *Feistel ciphers* operate on $2m$ input bits. The plaintext is split into a right and a left half, L and R, and the round function f only acts on one of the halves.

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus f(R_{i-1}, K_i) \end{aligned}$$

As a consequence from the Feistel structure, it takes two rounds before all plaintext bits have been subjected to the round transformation. Decryption can be done in the same way as encryption with the round keys supplied in reverse order. Moreover, the round function f does not have to be a permutation. The old DES (Data Encryption Standard) [Nat99] consists of an algorithm of that type.

Another popular construction is called a *substitution permutation network* (short SPN). This construction emphasizes on separating confusion (substitution) and diffusion (permutation) in the cipher. In many recent SPN ciphers, the permutation layer has been replaced by an affine transformation that is chosen in such a way that a high level of diffusion is guaranteed. Rijndael [DR02] which is the algorithm of the AES (Advanced Encryption Standard) [Nat97] is an example for this design. Feistel ciphers can also be considered as a kind of SPN.

Modes of Operation Whenever a message being longer than the block size needs to be encrypted, the block cipher is used in a certain mode of operation. For the DES, four modes have been standardized [Nat80]. The *electronic code book* (ECB) mode corresponds to the usual use of a block cipher; the message is split into blocks and each block is encrypted separately with the same key. In the *cipher block chaining* (CBC) mode, each ciphertext block is x-ored (chained) with the next plaintext block before being encrypted with the key. In the *output feedback* (OFB) mode and the *cipher feedback* (CFB) mode, a keystream is generated and x-ored with the plaintext. Hence, the latter two modes work as a synchronous additive stream cipher.

There is ongoing work to define new modes. In a special publication [Nat01], five confidentiality modes are specified for use with any approved block cipher, such as the AES algorithm.

1.2.2 Stream Ciphers

Stream ciphers encrypt individual characters, which are usually bits, of a plaintext one at a time. They use an encryption transformation which varies with time. Hence, in contrast to block ciphers, the encryption depends not only on the key and the plaintext, but also on the current state. As mentioned before at the end of Section 1.2.1, a block cipher can be turned into a stream cipher by using a certain mode of operation (such as CFB or OFB). Except that, there are no stream ciphers that are standardized today. A de-facto standard however is the RC4 stream cipher [Sch96].

Synchronous stream ciphers generate a keystream independently of the plaintext messages and of the ciphertext. Sender and receiver must therefore be synchronized; they must use the same key and operate at the same state within that key. A ciphertext digit that is corrupted during transmission does not influence any other ciphertext bit.

An *asynchronous* stream cipher is a stream cipher in which the keystream is generated as a function of the key and a fixed number of previous ciphertext bits. Because the keystream is dependent on only a few previous ciphertext bits, self-synchronization is possible even is some of the transmitted ciphertext bits are corrupted.

1.2.3 MDCs

An MDC (modification detection code, un-keyed hash function, or sloppily spoken, a hash function) takes an input of arbitrary length and compresses (digests) it to an output of fixed length, which is called the hash value. Cryptographic hash functions satisfy the following properties in addition:

1. *preimage resistance:* it should be computationally infeasible to find a preimage to a given hash value,
2. *2nd preimage resistance:* it should be computationally infeasible to find a 2nd preimage to a given input,
3. *collision resistance:* it should be computationally infeasible to find two different inputs with the same hash value.

Hash functions are used to ensure the integrity of data. This can be done by using the data as input to the hash function and storing its output. Later on, to verify that the input data has not been altered, the hash value is recomputed using the data at hand and compared with the original hash value. Another application for hash functions is to use them in digital signature schemes. Hash functions of the SHA family have been standardized by NIST [Nat03].

Starting 2004, hash functions have drawn a lot of attention since a Chinese group of researchers under the lead of Prof. Wang, published a series of papers presenting collision attacks on all popular hash functions, including the most frequently deployed algorithms MD5 and SHA-1. For MD5, the technique could also be used to produce meaningful examples of collisions (like colliding .ps-documents, colliding X.509 certificates or even a rogue certificate authority).

For SHA-1, no actual collision has been found yet, but the results lead to the adjustment of a lot of standards and even government laws, which all require a collision resistant hash function. As a result of these developments, the National Institute for Standards and Technology (NIST) has launched a competition to find a new hash standard SHA-3 [Nat07]. 64 hash function proposals were submitted whose analysis lead to many new insights in the understanding of hash functions. On October 2, 2012, the Belgian proposal KECCAK was selected as SHA-3 (`http://keccak.noekeon.org`).

1.2.4 MACs

Hash functions which involve a secret key are called MACs (message authentication codes). The output of such a *keyed hash function* is also called MAC.

In contrast to MDCs they can also be used to guarantee data origin authentication (i.e. corroborate the source of information) and data integrity. Most contemporary MACs are constructed either based upon a block cipher or a hash function.

In order to ensure authenticity of data, an entity computes a MAC on the data by using the private key. In order to verify the authenticity of the data later on, the MAC can be recomputed by anyone who may access the private key. Standardized MACs are the CBC-MAC [Int99], CMAC (based on a block cipher) and HMAC [Nat02] (based on hash functions).

1.3 Public Key Cryptography

In public key cryptography, secret keys are replaced by keypairs consisting of a *private key*, which must be kept confidential, and a *public key*, which is made available to the public by for example publishing it in a directory. Anyone who wishes to send a message to the owner of a certain keypair will take the public key, encrypt the message under this public key and send it to the owner of the keypair. The owner can decrypt the message by using the private key.

This idea works out in practice because the private and the public key are linked in a mathematical way (by a mathematical function) such that knowing the public key does not allow to recover the private key. Several mathematical functions which are useful for public key cryptography are known today. Amongst others, the most important hard mathematical problems are:

- *IFP* (integer factorization problem): given a positive integer n, find its prime factorization $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$.
- *GDLP* (generalized discrete logarithm problem): given a finite cyclic group G of order n, a generator α of G, and an element $\beta \in G$, find the integer $x, 0 \leq x \leq n-1$, such that $\alpha^x = \beta$.
- *DHP* (Diffie-Hellman problem): given a prime p, a generator α of $\mathbb{Z}_p^*$, and elements $\alpha^a \mod p$ and $\alpha^b \mod p$, find $\alpha^{ab} \mod p$.

In the subsequent sections, we discuss a reference cryptosystem for each of the above given hard mathematical problems. Thereafter, we sketch two other important applications for public-key cryptography, namely key agreement and digital signatures.

1.3.1 RSA

The Rivest-Shamir-Adleman (RSA) algorithm [RSA78] was the first public key encryption algorithm invented. Under certain assumptions, the RSA algorithm is based on the IFP. RSA keys are generated as follows: one selects two large secret prime numbers p and q and computes the public RSA modulus $n = pq$. Then one chooses a public encryption exponent e which satisfies $gcd(e, (p-1)(q-1)) = 1$. The private key, i.e. the decryption exponent, d can then be calculated by solving $ed = 1 \pmod{(p-1)(q-1)}$. Hence the public key is the pair (e, n) and the private key is the triple (d, p, q).

Suppose that someone wishes to encrypt a message for the entity which is associated with the public key (e, n). Then, the message needs to be represented as a number $m < n$. The ciphertext is computed by raising m to the power e: $c = m^e \pmod{n}$. This message can be decrypted by exponentiation with the private key $m = c^d \pmod{n}$.

In the following we look at several toy examples that illustrate RSA.

Example 1 (RSA Key Generation) *We show how Bob creates his RSA key pair in this example. Bob chooses two prime numbers $p = 5$ and $q = 11$. Remember that this is just a toy example. In real applications, p and q are supposed to have at least 512 bits!*

From p and q Bob derives $n = pq = 55$ and $(p-1)(q-1) = 40$. Then, he picks a random number $e = 7$ and checks that $gcd(e, (p-1)(q-1)) = 1$. Then, he derives $d = e^{-1} \equiv 23 \pmod{40}$.

The public key of Bob is $(e, n) = (7, 55)$ and the private key of Bob is $(d, n) = (23, 55)$.

Example 2 (RSA Encryption) *Suppose that we want to encrypt a message M for Bob. We assume that the message M is represented as number $m = 25$. We obtain somehow Bob's authentic public key, which is the tuple $(e, n) = (7, 55)$. Then, we compute the ciphertext $c = 25^7 \equiv 20 \pmod{n}$.*

Example 3 (RSA Decryption) *Suppose that Bob has received an encrypted message $c = 20 \pmod{55}$. Bob's private decryption key is the tuple $(d, n) = (23, 55)$. Hence, Bob computes the plaintext $m = 20^{23} \equiv 25 \pmod{55}$.*

1.3.2 El Gamal

The El Gamal encryption algorithm [Gam85] is based on the DLP in the finite group $(\mathbb{Z}_p^*, \cdot)$. An advantage of El Gamal is that some public parameters can be shared by a group of users. These so-called *domain parameters* are a large prime p (such that $p-1$ is divisible by another prime q), and an element $g \in \mathbb{Z}_p^*$ of order p (or that the order is divisible by q). The private key is chosen to be an integer d. The public key e can then be computed by solving $e = g^d \pmod{p}$.

Suppose that someone wishes to encrypt a message for the entity which is associated with the public key (e, p). Then, the message needs to be represented as a number $m < p$. In a first step of the encryption procedure, a random ephemeral key k is generated. The ciphertext c consists of a pair $c = (c_1, c_2) =$

(g^k, me^k). To decrypt the ciphertext, $c_2 \cdot c_1^{-d} \pmod{p}$ is computed. Since each message has a different ephemeral key, encrypting the same message twice will produce two different ciphertexts. Cryptosystems with this property are also called *non-deterministic* or *randomized* cryptosystems.

In the following we look at several toy examples that illustrate El Gamal.

Example 4 (El Gamal Key Generation) *Bob picks a prime number* $p = 41$. *Remember that this is just a toy example. In real applications, p is supposed to have at least 1024 bits!*

Then, Bob picks a so-called generator $g = 6$. *A generator has the property that it can be used to produce all numbers between 1 and 41 by taking the generator to its powers.*

Then Bob picks his private key d as some random number between 2 and $p-1$: $d = 17$. *With it, he derives his public key* $e := g^d = 6^{17} \equiv 26 \pmod{41}$.

Example 5 (El Gamal Encrytpion) *Suppose that we want to send Bob an encrypted message M that is represented as number* $m = 5$. *We obtain his authentic public key, which is the tuple* $(e, p, g) = (26, 41, 6)$.

Then, we generate a random number $k = 11$ *and compute* $c_1 = g^k = 6^{11} \equiv 28 \pmod{41}$ *and* $c_2 = me^k = 526^11 \equiv 19 \pmod{41}$. *We send the encrypted message* $(c_1, c_2) = (28, 41)$ *to Bob.*

Example 6 (El Gamal Decryption) *Suppose that Bob wants to decrypt the ciphertext* $(c_1, c_2) = (28, 41)$. *He takes his private decryption key, which is* $(d, p, g) = (17, 41, 6)$ *and recovers the message* $c_2/c_1^d = 41/28^{17} \equiv 5 \pmod{41}$.

1.3.3 ECC

An *elliptic curve E* over the field $\mathbb{F}$ is a smooth curve in the so called *Weierstrassform* [Kob94]

$$Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6, \quad a_i \in \mathbb{F}. \tag{1.1}$$

We let $E(\mathbb{F})$ denote the set of points $(x, y) \in \mathbb{F}^2$ that satisfy this equation, along with a "point at infinity" denoted $\mathcal{O}$. For finite fields $\mathbb{F}_p$ or $\mathbb{F}_{2^m}$ this formula can be simplified.

The set of points $E(\mathbb{F})$ together with the point at infinity forms an additive group. The group operations are called point addition (two distinct points on the curve are added) and point doubling (a point is added to itself). The GDLP is a hard mathematical problem in this group; it is therefore well suited for cryptography [Kob87]:

If E is an elliptic curve over $\mathbb{F}$ and B is a point of E, then the *elliptic curve discrete log problem* on E (to the base B) is the problem, given a point $P \in E$, of finding an integer $x \in \mathbb{Z}$ such that $xB = P$ if such an integer x exists.

Similar to the El Gamal cryptosystem, several public parameters can be shared amongst a group of users. These parameters are the elliptic curve group itself and the base point. The private key is chosen to be an integer d. The public key is then given by $e = dB$.

Suppose that someone wishes to encrypt a message for the entity which is associated with the public key $(e, B, E(\mathbb{F}))$. Then, the message needs to be represented as a point on the curve $M \in E(\mathbb{F})$. In a first step of the encryption procedure, a random ephemeral key k is generated. The ciphertext c consists of a pair $c = (c_1, c_2) = (kB, M + ke)$. To decrypt the ciphertext, $c_2 - dc_1$ is computed.

1.3.4 Key Agreement

A major disadvantage of symmetric key cryptography is that secret keys need to be distributed securely to the users. However, this problem can be solved efficiently using public key cryptography [DH76].

Suppose that two entities, A and B, wish to agree on a secret key over an insecure channel. A generates a secret random number a and B generates a secret random number b. Then, A sends g^a to B and B sends g^b to A. Now they can both compute g^{ab} which is their common secret key.

An adversary who sees the messages g^a and g^b and must reconstruct g^{ab} from them. This is exactly the DHP considered in the first paragraph of this section. The DHP is a hard mathematical problem in several mathematical structures. In the original version the structure was the finite multiplicative group in the finite field $\mathbb{F}_p$. However a more efficient version can be produced by taking the group $E(\mathbb{F})$.

1.3.5 Digital Signatures

From the cryptographer's point of view, a digital signature is the digital equivalent of a handwritten signature; it binds someone's identity to a piece of information. With a digital signature creation algorithm, one produces a digital signature. With a digital signature verification algorithm one verifies that a digital signature is authentic (i.e., was indeed created by the specified entity). A digital signature scheme consists of a signature creation algorithm and an associated verification algorithm.

There are different types of signature schemes. Digital signatures *with appendix* require the original message as input to the verification algorithm. Digital signatures *with message recovery* do not require the original message as input to the verification algorithm. In this case, the original message is recovered from the signature itself.

The DSS [Nat00] is a standardized signature scheme with appendix. The DSS describes an algorithm, called DSA (Digital Signature Algorithm) which is based on the DLP. It has been extended recently and includes now a version of the same algorithm but on elliptic curves, the ECDSA. In order to sign a message, the owner first computes the hash value of the message. This hash value and the private key are then subjected to the signing algorithm. The output of this algorithm is a signature of the input message. Anyone who wishes to verify the signature can simply take the message, the public key and the signature and compute the verification algorithm with these inputs.

1.4 Security and Attacks

There exist many different types of attacks on cryptographic primitives. The goal of such attacks is typically to deduce the secret key. In this section we characterize attacks based on the adversary's knowledge and on the adversary's (computational) capabilities.

A *passive* attack is one where the adversary only monitors the communication channel or the side-channels (e.g. electromagnetic emanations of power consumption) of the communication devices. The latter attacks are then called *side-channel attacks* [KJJ99]. They do not target the used algorithm itself, but its implementation.

An *active* attack is one where the adversary attempts to alter the transmission on the channel or the computation inside the device. The latter attacks are then called *fault attacks* [BDL00]. They do not target the used algorithm itself, but its implementation.

Side-channel attacks and fault attacks are so called *implementation attacks.* In the last years it turned out that several types of them pose a serious threat against practical implementations of cryptographic systems.

A widely accepted assumption in modern cryptography is that the adversary knows all details about the used algorithms and protocols. Only the secret key is unknown to the adversary. This assumption goes back to Auguste Kerckhoffs, a dutch cryptographer of the 19th century. Based on this assumption, attacks can be classified according to the adversary's knowledge as follows [MvOV97]:

1. A *ciphertext-only* attack is one where the adversary tries to deduce the secret key by only observing ciphertext.

2. A *known-plaintext* attack is one where the adversary has a number of plaintexts and corresponding ciphertexts.

3. A *chosen-plaintext* attack is one where the adversary chooses plaintext and is then given the corresponding ciphertext.

4. An *adaptive chosen-plaintext* attack is a chosen plaintext attack wherein the choice of the plaintext may depend on previous ciphertexts.

5. A *chosen-ciphertext* attack is one where the adversary chooses the ciphertext and is then given the corresponding plaintext.

6. An *adaptive chosen-ciphertext* attack is a chosen ciphertext attack wherein the choice of the ciphertext may depend on previous plaintexts.

Encryption schemes which are vulnerable to ciphertext-only attacks are considered completely insecure.

1.4.1 Security Models

There are several different models under which the security of a cryptographic primitive can be evaluated. Practical models are computational security and ad-hoc security; the confidence in the amount of security of a primitive based on computational or ad-hoc security increases with time and investigation of the primitive. Of course, time alone is not enough if only very few people have tried to break the primitive.

Unconditional security: In this model, the adversary is assumed to have unlimited computational power. Unconditional security for encryption schemes is also called *perfect secrecy.* A necessary condition for a secret key cryptosystem to have perfect secrecy is that the key is at least as long as the message. The one-time pad is an example for an unconditionally secure encryption algorithm. Because the key needs to be at least as long as the message to be encrypted, such systems are impractical. None of the modern primitives offers perfect secrecy.

Computational security: Adversaries are assumed to have polynomial computational power. Thus, an algorithm is considered to be secure if the best algorithm for breaking it requires a superpolynomial or exponential number of steps. State of the art ciphers are supposed to have computational security.

Provable security: A cipher has provable security if the difficulty of breaking it can be shown to be equivalent to solving a well known and supposedly hard mathematical problem (such as IFP, DLP, etc.). As a consequence, the proof of security only holds under certain mathematical assumptions. Public key cryptosystems typically allow to construct such proofs. Proofs of security often imply computational security or unconditional security.

Practical security: Practical security is also related to the computational power of the adversary. However, it is concerned with the computational power which is needed with respect to previously known attacks. A cipher is considered to be practically secure if the best *known* attack requires at least N operations, where N is a sufficiently large number. State of the art ciphers typically offer practical security.

An attack which is applicable to all types of keyed cryptographic primitives is the *exhaustive key search*. As a consequence it is mandatory to choose key sizes to be sufficiently large.

1.5 Summary

Basic security services and related cryptographic primitives were considered in this chapter. We gave a survey on secret key cryptography in Section 1.2. Within this section, we introduced block ciphers, stream ciphers, hash functions and MAC algorithms. Thereafter, in Section 1.3, we provided a survey on state-of-the art public key cryptosystems as well. The RSA, El Gamal and ECC cryptosystems were covered. In the last section of this chapter we discussed the security of modern cryptosystems.

1.6 Review Exercises

1. Name and explain the four security services (properties) cryptography can provide.
2. What is Kerckhoffs' principle?
3. What are the basic components of a block cipher?
4. What is the difference between a block cipher and a stream cipher?
5. What is the difference between a MAC and an MDC?
6. What is the meaning of "pre-image resistance"? For what cryptographic primitive is it relevant?
7. Explain how RSA encryption and decryption work.
8. What is a digital signature with appendix?
9. Describe and explain the Diffie-Hellman Problem. What is it used for?
10. Name and explain at least three different models of security.
11. What does the term "exhaustive key search" mean?

Chapter 2

Electronic Signatures and Public Key Infrastructures

A commercial CA protects you from anyone whose money it refuses to take.
— Matt Blaze, RSA 2000

Electronic signatures and public key infrastructures are important concepts in modern information security and they are essential enablers for electronic processes in e-commerce and e-government.

Electronic signatures are the electronic analogies to handwritten signatures. They can be created by the entity, which owns a private key, only. But everyone can take the entity's corresponding public key and verify the signature. Digital signatures rely on the concept of public key cryptography. The authentic distribution of public keys is therefore a major concern for this technology.

Public key infrastructures (PKI) take on the challenge to provide mechanisms to distribute authentic public keys. For this, they offer different services. The core service of each PKI is the certification authority (CA) which is responsible for managing certificates. With the aid of certificates, the authenticity of public keys is guaranteed.

This chapter aims to introduce the reader to PKIs.

2.1 Digital Signatures

In the language of cryptographers, a *digital signature* is a data string which associates a message in digital form with some originating entity.

With a digital signature *creation* algorithm, one produces a digital signature. With a digital signature *verification* algorithm one verifies that a digital signature is authentic (i.e., was indeed created by the entity belonging to the public key).

A *digital signature scheme* consists of a signature creation algorithm and an associated verification algorithm. There exist several classes of digital signature schemes:

DS with appendix Digital signatures with appendix require the original message as input to the verification algorithm.

DS with message recovery Digital signatures with message recovery do not require the original message as input to the verification algorithm. In this case, the original message is recovered from the signature itself.

Randomized DS A digital signature scheme is said to be a randomized digital signature scheme if there are several valid signatures for a given message; otherwise, the digital signature space is said to be **deterministic**.

Most of the signature schemes which are in use today belong to the class of (randomized) digital signatures with appendix. This is due to the fact that in most schemes, the hash of a message is signed instead of the message. This has several advantages. Firstly, the signatures are much shorter when the hash is signed. Second, several attacks which apply for signature schemes with message recovery can be thwarted by signing the hash.

Also breaks of digital signatures schemes can be classified:

Total break An adversary is either able to compute the private key or find an efficient signing algorithm which is functionally equivalent to the valid signing algorithm.

Selective forgery An adversary is able to create a valid signature for a particular message or class of messages chosen a priori. Creating the signature does not directly involve the legitimate signer.

Existential forgery An adversary is able to forge a signature for at least one message. The adversary has little or no control over the message whose signature is obtained, and the legitimate user may be involved in the deception.

Attacks which lead to the three previously mentioned breaks can be classified according to the capabilities of an attacker.

Key-only attack: The adversary only knows the signer's public key.

Message attacks: The adversary is able to examine signatures corresponding either to known or to chosen messages.

Known-message attack: An adversary has signatures for a set of messages which are known to him.

Chosen-message attack: An adversary obtains valid signatures from a set of messages which were chosen by him.

Adaptive chosen-message attack: An adversary is allowed to use the signer as an oracle; he may request signatures which depend on the signer's public key and he may request signatures which depend on previously obtained signatures.

2.1.1 RSA Signature Scheme

The key setup is the same in RSA encryption 1.3. In our notation, m denotes the input message, n denotes the modulus, d denotes Alice's private key, and e denotes her public key.

Algorithm 1 RSA Signature Creation.

```
INPUT: $m$, $d$, $n$
OUTPUT: $Sig(m)$
1: $M = Encode(m)$.
2: $s = M^d \pmod{n}$.
3: Return s.
```

Algorithm 2 RSA Signature Verification.

```
INPUT: $m$, $s$, $e$
OUTPUT: $Accept/Reject$
1: Check length of $s$, if invalid then Return “Reject”.
2: $M = s^e \pmod{n}$.
3: If $M = Encode(m)$ then Return “Accept” else Return “Reject”.
```

Depending on the particular definition of the encoding function, which is used in both signature creation and signature verification, this scheme is either randomized or not. If the encoding includes processing through a hash function, the scheme belongs to the class of signature schemes with appendix. Otherwise, it constitutes a signature scheme with message recovery.

2.1.2 DSA Signature Standard (DSS)

It describes the Digital Signature Algorithm (DSA). It requires the usage of a hash function. In the original version, SHA-1 was required, the 2009 version of the FIPS 186 standard allows other hash functions, see [Nat09]. Key generation works similar as for an ordinary El Gamal scheme. However, key sizes are required to fulfill certain constraints for security and efficiency.

The numbers created by the key generation algorithm are the two primes p and q, the element α with order q, and the private key x and the corresponding public key y.

Algorithm 3 DSA Key Generation.

```
INPUT:
OUTPUT: $p$, $q$, $\alpha$, $x$, $y$, $g$
1: Choose prime $q$ such that $2^{N-1} < q < 2^N$ ($N$ bit-length of $q$).
2: Select another prime $p$ such that $2^{L-1} < q < 2^L$ and with the property that $q|(p-1)$.
3: Choose $\alpha$ of order $q$ in $\mathbb{Z}_p^*$. It holds then that $\alpha = g^{(p-1)/q}$.
4: Choose a random integer $x$ such that $1 \leq x \leq q-1$.
5: Compute $y = \alpha^x \bmod p$.
6: Return Alice's public key $(p, q, \alpha, y)$ and her private key $x$.
```

It can be seen in Algorithm 3 that in Step 1 the prime q is chosen according to the size of the hash function output.

Signature creation is described in Algorithm 4 and signature verification is described in Algorithm 5.

Algorithm 4 DSA Signature Creation.

```
INPUT: $m$, $x$
OUTPUT: $Sig(m)$
1: Choose a random $k, 0 < k < q$.
2: Compute $r = (\alpha^k \bmod p) \bmod q$.
3: Compute $s = k^{-1}(h(m) + xr) \bmod q$.
4: Return the signature for $m$ which is $(r, s)$.
```

We can conclude from Step 1 that DSA is a randomized signature scheme. Every time the algorithm is executed, a new random number k (the so-called ephemeral key, or nonce) is picked in the first step. Hence, even if the same algorithm is used several times with one and the same message, a different signature will be produced. Because we are using such a random number, and we hash the message, we have to include some information about the random number in the signature to enable verification. This is the reason for having a pair of values (r, s) representing one signature. The value r holds the necessary information about k for verification. Because the hash value of the message is used within the computation of the value s, we have a signature scheme with appendix.

In verification, we first have to check some properties of the received values. Clearly, if r and s are not in the interval, which is imposed by the moduli, a signature should never be accepted as valid.

Algorithm 5 DSA Signature Verification.

```
INPUT: $m$, $(r, x)$, $y$
OUTPUT: Accept/Reject
1: Verify that $0 < r < q$ and $0 < s < q$.
2: Compute $u_1 = wh(m) \bmod q$ and $u_2 = rw \bmod q$ with $w = s^{-1} \bmod q$.
3: Compute $v = (\alpha^{u_1} y^{u_2} \bmod p) \bmod q$.
4: Return the signature for $m$ which is $(r, s)$.
5: Return Accept if $v = r$ otherwise return Reject.
```

2.1.3 ECDSA Signature Scheme

The elliptic curve digital signature algorithm (ECDSA) is the analogue to the DSA. The major difference is that the DLP which gives the algorithm its security it hereby defined over an elliptic curve 1.3.3.

Alice and Bob first choose a finite field $\mathbb{F}_q$, an elliptic curve E, defined over that field and a base point G with order n. Alice's key pair is (d, Q), where d is her private and Q is her public key. Signature creation and signature verification work as described in Algorithm 6 and 7.

Algorithm 6 ECDSA Signature Creation.

```
INPUT: $m$, $d$, elliptic curve with base point $G$ of order $n$.
OUTPUT: Accept/Reject
1: Choose a random number $k$ with $k : 1 \leq k \leq n - 1$
2: Compute $kG = (x_1, y_1)$ and $r = x_1 \bmod n$. If $r = 0$ then go to step 1.
3: Compute $k^{-1} \bmod n$.
4: $e = h(M)$.
5: Compute $s = k^{-1}(e + dr) \bmod n$. If $s = 0$ then go to step 1.
6: Return the signature which is $(r, s)$
```

From Step 1 we can conclude that we have a randomized signature scheme. Every time the algorithm is executed a new random number is picked. Therefore no two signatures for the same message will be identical. As we used the hash value of a message during the signing process, we have a signature scheme with appendix.

In the verification algorithm we check at the beginning if the signature values r and s are in the interval which is imposed by the parameters. If not, then the signature cannot be regarded as valid.

2.2 Public Key Infrastructures

Public key cryptography makes it possible to distribute the public key openly via an insecure network. However, an attacker can still bring down any public key

Algorithm 7 ECDSA Signature Verification.

```
INPUT: $m$, $(r, s)$, $Q$, elliptic curve
OUTPUT: $Accept/Reject$
 1: Verifiy that $r, s$ are integers in the intevall $[1, n-1]$.
 2: Compute $e = h(M)$.
 3: Compute $w = s^{-1} \bmod n$.
 4: Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
 5: Compute $X = u_1G + u_2Q$.
 6: if $X = \mathcal{O}$ then
 7:     Return Reject
 8: else
 9:     Compute $v = x_1 \bmod n$ where $X = (x_1, y_1)$.
10: end if
11: Return Accept if $v = r$ else Return Reject.
```

cryptosystem by modifying published public keys. For example, if Alice's public encryption key has been modified, than no-one can send her confidential messages. Or, if her public signature verification key has been modified, no-one can verify signatures she has created. Consequently, even public keys need protection; we must ensure that they are *authentic*. By authentic we mean that we have ensured that a certain public key belongs indeed to a certain person. In addition we require that this relationship can be verified by another entity. A public key infrastructure (PKI) is the basis of a security infrastructure whose services are implemented and delivered by using public-key concepts and techniques.

2.2.1 Public Key Distribution

We explore the different possibilities of distributing public keys. Suppose Alice wants to deliver her public key to Bob.

The first option is that they simply meet and make the key exchange *face-to-face*. This approach has the advantage that Alice and Bob can first verify their identities, by for example, checking their IDs. Another advantage of this approach is that Bob can be assured of the authenticity of the key. The major disadvantage of this approach is that Alice needs to meet everyone who wishes to obtain her public key in person. This is clearly an impractical solution.

The second option is that Alice publishes her key on a trusted web server (web directory). If Bob wishes to obtain Alice's key, he simply looks at the web server for Alice's key. The advantage of this approach is that Alice and Bob do not have to meet in person. However, the major disadvantages of this approach are that Bob cannot verify Alice's identity and, that he cannot verify that the key has not been modified. This makes this approach clearly impractical as well.

The third option is that Alice publishes her certified public key on a trusted web server (web directory). By certified public key we mean that the trusted web server (which is also called trusted third party, short TTP) provides a service which adds additional information to a key, the so-called certificate. This certificate allows to verify that a key has not been altered. A certificate basically consists of a public key and some user information which has been digitally signed by the issuer of the certificate. This approach has the advantages that Alice and Bob never need to meet in person and that the authenticity of public keys can be verified. However, also in this approach, the public key of the web server needs to be checked for it's authenticity.

2.2.2 PKI in Practice

Amongst the many PKIs that have deployed in practice, we study the concepts of Pretty Good Privacy (PGP) and X.509 certificate based PKIs in more detail. The difference between those two PKIs is the way in which trust is established between different entities (users) of a PKI.

2.2.3 PGP

In a PGP based PKI, each user acts as a third party, who can sign keys. Each user also manages keys herself, hence, there is no need for a central controlling authority. PGP is based on the so-called user-centric trust model. Each user trusts herself, and a limited amount of other users. For example, Alice trusts Bob. Hence, she trusts all the keys which have been signed by Bob. If Bob trusts Clark, then also Alice trusts Clark, because she trusts Bob. In this way, a whole web of trust is spanned around the users.

The problem which arises from the user-centric trust model is that it is implicitly assumed that trust is a transitive: If Alice trusts Bob and Bob trusts Clark, then Alice trusts Clark. This is, however, a questionable approach, with which not everyone is comfortable.

Another problem which arises from the lack of a central authority is that the revocation of public keys is difficult. Even though some key servers may exist, it is not their task to reliably ensure that revocation information is transmitted to users. Consequently, in a PGP-based PKI, revocation is a service which cannot be guaranteed (but is technically possible). For many practical applications (especially when it comes to applications which have to fulfill some legal requirements), revocation checking is mandatory. Hence, PGP-based PKIs are not used for such applications.

2.2.4 X.509

The trust model which is the basis for X.509-based PKIs is the hierarchical trust model. In a hierarchical trust model, there is exists a so-called *root CA* which

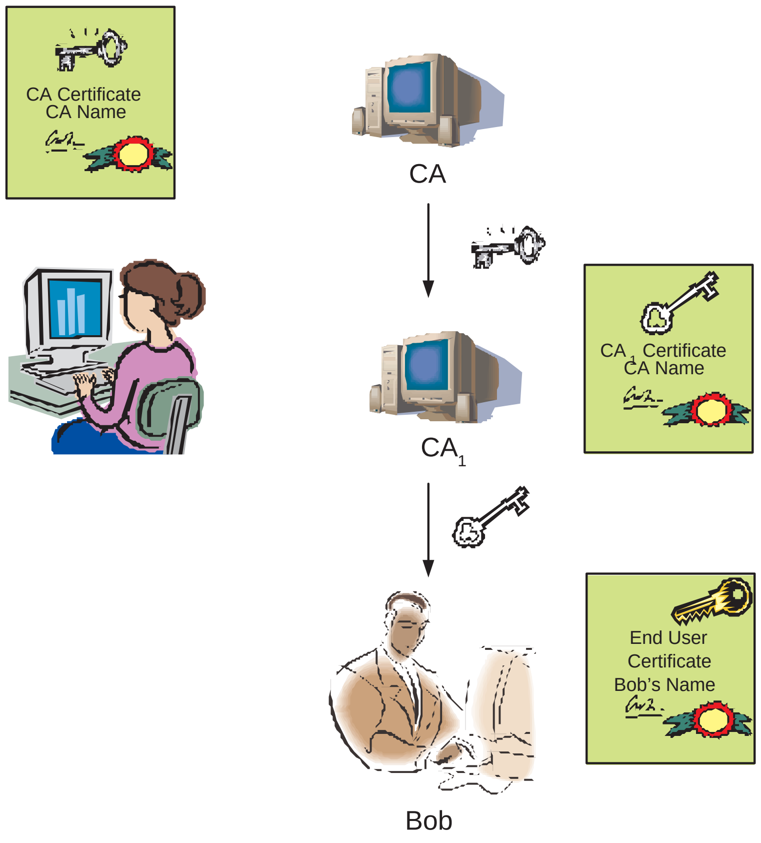


Figure 2.1: Certificate chain: Alice can verify the certificate of Bob by checking the complete chain.

acts as root, or anchor, of trust for the entire domain of entities below it. This root CA owns a self-issued (self-signed) root certificate, which is the basis of trust for all entities that belong to the domain. The root CA certifies zero or more CAs immediately below it. Each of those CAs certifies zero or more CAs immediately below it. At the second to last level, the CAs certify end entities. Each entity in the hierarchy must be supplied with the certificate of the root CA.

Alice, who holds the certificate of the root CA, can verify the certificate of Bob in the following way. Suppose Bob holds a certificate which was issued by CA_1, whose certificate was issued by the root CA. Alice, can verify the certificate of CA_1 by using the root certificate. Then, she can verify Bob's certificate by using the certificate of CA_1. Verifying a certificate essentially consists of verifying the signatures of the certificate (checking the certificate chain, see Figure 2.1) and checking its contents for soundness.

The advantage of X.509-based PKIs is that due to their hierarchical structure, revocation checking mechanisms can be reliably implemented. The disadvantage of X.509-based PKIs is that they tend to be big and complex and that still the

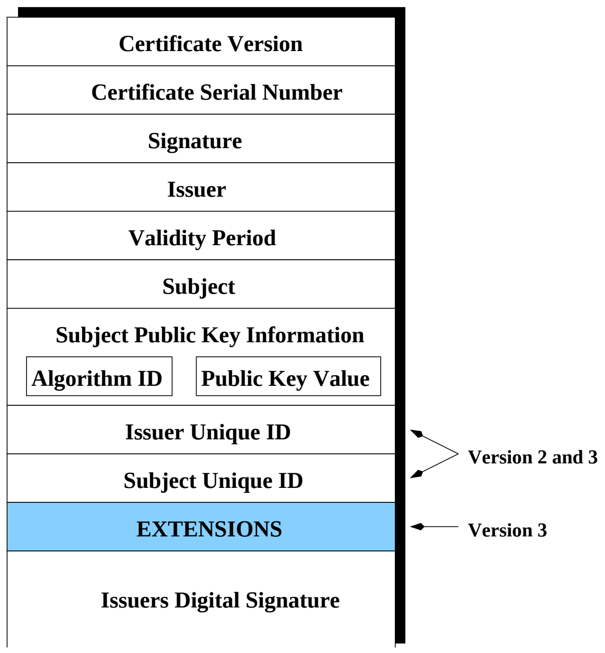


Figure 2.2: X.509 certificate format

root CA certificate must be checked manually.

X.509 Certificates

A certificate is a data structure, which essentially consists of the public key of an entity plus some information about the entity, and which is digitally signed by the certificate issuer. A certificate which complies to the X.509 standard includes several fields, see Figure 2.2.

The field *Version* identifies the version of the certificate. Version 3 is the common version at the time of writing this book. The field *Certificate Serial Number* contains a unique integer identifier for the certificate relative to the certificate issuer. The field *Signature* holds the algorithm identifier (the object identifier, short OID) of the algorithm which was used to calculate the digital signature on the certificate. The *Validity Period* indicates the window of time that this certificate is valid unless otherwise revoked. This field is composed of *Not Valid Before* and *Not Valid After* time limits. *Subject* indicates the so-called distinguished name (DN) of the certificate owner and must be non-null except if an alternative name form is given in the *Extensions*. *Subject Public Key Info* holds the public key and its algorithm identifier. This field must be always present. The *Issuer Unique ID* is an optional unique identifier of the certificate issuer. This field may be present in version 2 or 3 only. However, this field is rarely used in practice. It is not recommended for use by RFC 3280 [HPFS02]. The *Subject Unique ID* is an optional unique identifier of the certificate owner.

This field may be present in version 2 or 3 only. This field is also rarely used in practice and it is also not recommended for use by RFC 3280.

Certificate extensions were introduced in version 3. Each extension can be marked with a criticality flag. An extension that is marked as critical must be processed and understood. Otherwise the certificate is not to be used. Extensions which are marked as non-critical are to be processed if possible, but may be ignored if they are not recognized. We discuss several useful extensions in the following paragraphs.

The *Authority Key Identifier* is a unique identifier of the key that should be used to verify the digital signature on the certificate. It helps to distinguish between multiple keys that apply to the same certificate owner. RFC 3280 requires to include this field in all but self-signed certificates. The *Subject Key Identifier* is a unique identifier, which is associated with the public key which is contained in the certificate. RFC 3280 recommends to include this field for end-entity certificates and requires to include this field for CA certificates. The field *Key Usage* is used to identify and restrict the functions or services that can be supported by the public key in the certificate. Examples for key usage are support for digital signature, key encipherment, data encipherment, certificate signature, CRL signature, etc. *Extended Key Usage* is used for identifying the specific usage of the public key in the certificate. For example, TLS server authentication, TLS client authentication, code signing, time stamping, etc. In *CRL Distribution Point* the location of the CRL partition is given where revocation information, which is associated with this certificate, can be found. In *Certificate Policies* one or more policy OIDs can be found. The *Subject Alternative Name* indicates alternative name forms for the owner of the certificate. Examples are the e-mail address, IP address, etc. In the *Basic Constraints* field it is indicated whether the certificate is a CA certificate or not. For a CA certificate the value of this field must be set to *true*. If the field is present in an end-entity certificate then it must be set to *false*. The basic constraints field in a CA certificate can also include a *Path length Constraint*. This constraint indicates the maximum number of CA certificates that may follow this certificate in a certification path. If this constrain is not present, then no restriction is placed on the length of the certification path.

OID An OID is a unique representation for a given object, for example an algorithm or a standard. As such, an OID is represented as a sequence of integers that are separated by decimal points. OIDs are hierarchical and they are registered with international, national or organizational authorities to ensure their uniqueness.

DN A DN is a hierarchical naming convention which is defined in the X.500 recommendations. DNs are supposed to ensure the unique name of an entity. DNs are expressed as a concatenation of relative DNs from the top-level node down to the last node of a DN.

X.509 PKI Components

We have already implicitly identified several basic services that we expect from a practical PKI. Firstly, we expect in a practical realization of a PKI that the identity of users is verified in a more or less reliable manner. Clearly, the less effort is spent in verifying the identity of entities, the less trusted signatures created by these entities will be. And the less trust in certificates issued by the PKI will be. Secondly, we expect that certificates are somehow available in a practical manner. Thirdly, we expect that revocation information is broadcasted in a reliable manner. Therefore, in a practical PKI, these services will be handled by different components (services).

Registration Authority (RA) Often, a separate registration authority is used to verify the identity of entities. For highly trusted certificates the verification of the identity of an entity requires the entity to be physically present at the registration authority.

Certification Authority (CA) The certification authority issues certificate to entities which have already been identified.

Certificate Repository The certificates are stored in the certificate repository. A repository is a database which stores certificates. In many practical realizations, such a repository is implemented according to the lightweight directory access protocol (LDAP) version 3 [WHK97].

Certificate Revocation The revocation of issued certificates is hardly implemented in practice. Instead, most mechanisms are focused on the distribution of revocation information. The distribution of revocation information can be implemented in several ways. One possibility is to use a periodic publication mechanism such as *certificate revocation lists* (CRLs). On-line query mechanisms are the other possibility. The *online certificate status protocol* (OCSP) is an example of this type of mechanism.

2.3 Legal Aspects of Electronic Signatures

From the European point of view, the most important framework which regulates electronic signatures is the Directive 1999/93/EC, of the European Parliament and of the Council of 13 December 1999, on a Community framework for electronic signatures. This directive has been implemented already by several EU member states (Austria included). In this directive the legal definition of an electronic signature is given as follows:

Data in electronic form which are attached to or logically associated with other electronic data and which serve as a method of authentication.

This definition, is not targeted only towards cryptographic digital signatures. Anything, which fulfills the above definition is recognized as electronic signature. For example, a paper document, which contains a conventional signature can be scanned in. The corresponding electronic document, which contains the signature in electronic form, is then a document with a legally recognized electronic signature. According to the above mentioned Directive, the validity of this electronic signature cannot be denied solely based on the fact that it is in electronic form. However, such a signature is not considered as equal in a legal sense to a handwritten signature. In order to be considered equal an electronic signature must comply to certain rules.

2.3.1 Advanced Electronic Signatures

An *advanced electronic signature* needs to fulfill certain rules. It must be uniquely linked to the signatory. It must be capable of identifying the signatory. It must be created using means that the signatory can maintain under his sole control. It must be linked to the data to which it relates in such a manner that any subsequent change of the data is detectable.

These four requirements are strongly related to what can be achieved by digital signatures which are based on public key principles. We have discussed this type of electronic signatures in some detail in this chapter.

However, also this type of electronic signatures is not considered as equal to handwritten signatures from a legal point of view.

2.3.2 Qualified (Secure) Electronic Signatures

A *qualified (secure) electronic signature* is an advanced electronic signature which was produced on a secure signature creation device and which is based on a qualified certificate. It must be assured that the signature creation data (*i.e.* the private key) exists only once and is protected by the user from the use by others.

A *qualified certificate* must contain an indication that the certificate is issued as a qualified certificate. It must also contain the identity of the certificate service provider (the company or entity which runs the PKI) and the state. Furthermore it must contain the name of signatory or pseudonym which shall be identified as such. It must also contain signature-verification data, the validity period, the identity code of the certificate, and the advanced electronic signature of the CSP. In addition, limitations on the scope of use (if applicable), and limits on the value of transactions (if applicable) must be included.

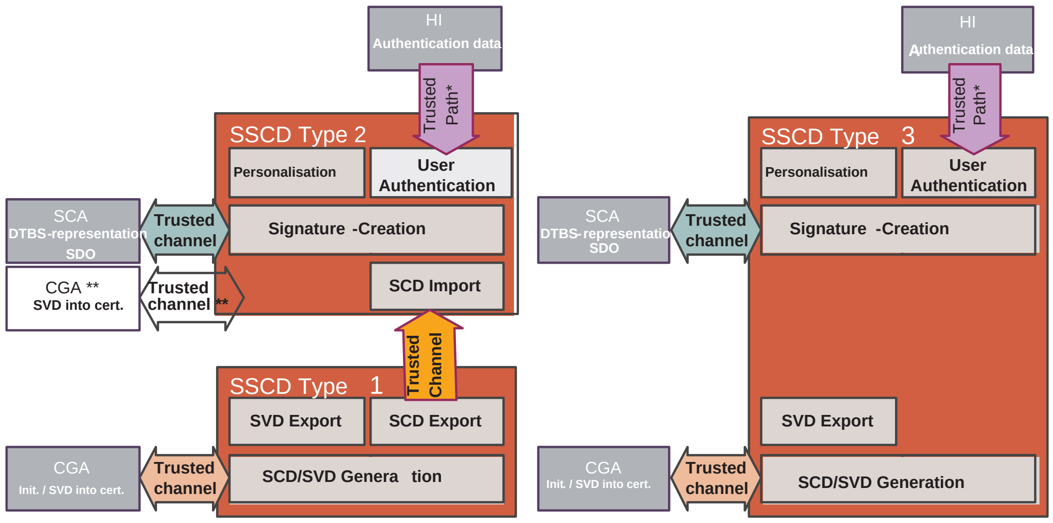


Figure 2.3: SSCD types and modes of operations

A *secure signature creation device* (SSCD) must ensure the secrecy of private keys, protection against forgery of signatures or key material. Different types of signature creation devices are distinguished, see Figure 2.3.

A SSCD of type 1 is only able to generate the signature creation data (SCD) and the signature verification data (SVD), *i.e.*, the key pair. Consequently, it must allow to export this key pair over a trusted channel to a device that can create signatures. A SSCD of type 2 is able to generate digital signatures (and verify them), but needs to import the key material. A type 3 device is a combination of a type 1 and a type 2 device. Here some more additional explanations, which are taken from the Common Criteria protection profile that covers SSCDs. *The signatory must be authenticated to create signatures that he sends his authentication data (e.g., a PIN) to the SSCD Type 2 or Type 3 (e.g., a smart card). If the human interface (HI) for such signatory authentication is not provided by the SSCD, a trusted path (e.g., an encrypted channel) between the SSCD and the SCA implementing the HI is to be provided. The data to be signed (DTBS) representation (i.e., the DTBS itself, a hash value of the DTBS, or a pre hashed value of the DTBS) shall be transferred by the SCA to the SSCD only over a trusted channel. The same shall apply to the signed data object (SDO) returned from a SSCD to the SCA. SSCD Type 1 is not a personalized component in the sense that it may be used by a specific user only, but the SCD/SVD generation and export shall be initiated by authorized persons only (e.g., system administrator). SSCD Type 2 and Type 3 are personalized components which means that they can be used for signature creation by one specific user - the signatory - only.* Furthermore, a SSCD may also provide reliable display of data.

Also the *certificate service provider* must fulfill certain criteria. It must provide a secure directory and immediate revocation service. It must provide verification of the identity of users. It must employ personnel who possess the necessary qualifications. It must use trustworthy systems and products and maintain sufficient financial resources. It must record all relevant information. But it must not store or copy signature creation data. It must inform users by durable means of the terms and conditions regarding the use of the certificate.

The Austrian Signature Law (SigG) specifies that, apart from a few exceptions (like notary acts), a qualified digital signature is legally equivalent to a handwritten signature.

2.4 Electronic Signatures in Practice

Applications that profit from electronic signatures are primarily available in the e-government sector but also the private sector may profit from the concept.

The Austrian initiative is known as the concept "Bürgerkarte" (`www.buergerkarte.at`). This concept defines general minimum requirements:

- qualified electronic signatures i.e., legal equivalence to handwritten signatures,
- additional key-pairs "general signatures", encryption,
- info-boxes to store data persona binding, certificates, power of attorney, access control to info-boxes, and
- DH key exchange session key certificates.

Consequently, any device that meets these minimum requirements is a "Bürgerkarte". In the concept Bürgerkarte, the so-called *Bürgerkartenumgebung* is the combination of the security-relevant components with respect to electronic signatures. Applications communicate with the Bürgerkartenumgebung over the so-called *security layer*. The security layer is an interface that provides a logical view to the Bürgerkartenumgebung, see Figure 2.4.

The security layer protocol is a simple request/response scheme, *i.e.*, an application sends a request and the Bürgerkartenumgebung responds (result or error code). The protocol data elements are encoded in XML.

2.4.1 State of the Art

- The security layer is available for every Austrian citizen due to a general license for the Bürgerkarten-Software provided by the government.
- Realizations of the Austrian citizen card are readliy available:
  - A-TRUST (Austrian CSP for qualified certificates)

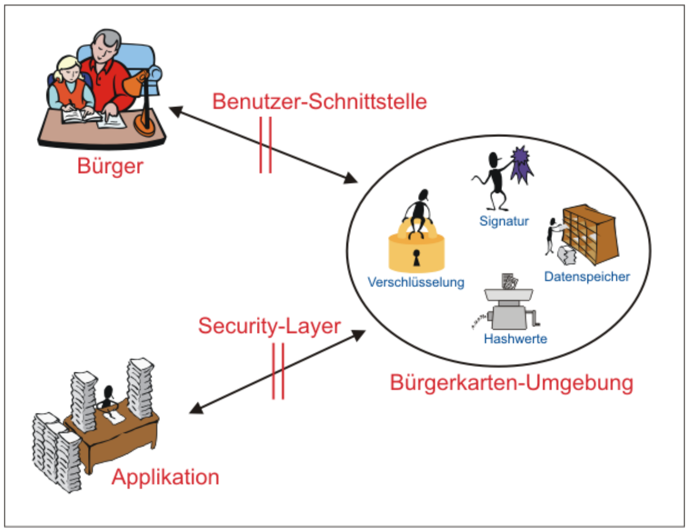


Figure 2.4: Communication between application and Bürgerkartenumgebung via security layer

  - E-Card (Austrian health card)
  - Maestro (Bank card)
  - Cell phones (new initiative)

- E-Government Applications (see also: `https://www.buergerkarte.at/anwendungen.de.php`)
  - Versicherungszeitenauszug
  - Registrierung eines Gewerbes in Wien
  - Online tax declaration
  - Signed petitions to the government
  - Extract from the police records
  - Electronic delivery of legal documents (e.g. RSa-Briefe, `www.meinbrief.at`, `www.brz-zustelldienst.at`)
  - Signed pdf-documents (`https://www.buergerkarte.at/pdf-signatur.de.php`)
- Tools for secure storage of highly sensible data (`https://www.e-tresor.at`)
- Tools for private use (e.g. CCE-encryption of private documents with the citizen card, `http://www.a-sit.at/de/dokumente_publikationen/flyer/cce_en.php`)

2.5 Summary

In this chapter we have discussed digital signatures including some legal aspects and public key infrastructures. Digital signature based on public key principles were introduced already in Section 1.3 and has been refined in Section 2.1. We have pointed out that there legal definitions for several types of digital signatures in Section 2.3.

We have also identified that the distribution of authentic public keys is important. Two practical realizations of PKIs have been discussed is Section 2.2.3 and Section 2.2.4. The mechanism which ensures the authenticity of public keys for X.509-based infrastructures has been introduced in Section 2.2.4.

2.6 Review Exercises

1. What is a digital signature?
2. What classes of digital signatures can you distinguish?
3. What types of breaks of digital signatures schemes can you distinguish?
4. Describe the working principle of a digital signature scheme with appendix?
5. What is a PKI used for?
6. What is the difference between a PGP-based and a X.509-based PKI?
7. What is a CRL?
8. What is an advanced electronic signature?
9. What type of electronic signature is equivalent (from a legal point of view) to a handwritten signature?

Chapter 3

Electronic Commerce

We are not retreating -
we are advancing in another direction.
— General Douglas MacArthur (1880-1964)

Electronic commerce (short e-commerce) is the business of buying and selling products, information, or services electronically. The typical environment in which e-commerce takes place is the internet.

Nowadays, most e-commerce transactions are conducted via a credit or debit card. Typically, e-commerce merchants store cardholder information in databases to streamline the process of purchase. Hence, web merchants compile databases containing thousands of payment card accounts. For hackers, these databases represent a tremendous opportunity for theft and fraud.

In this chapter, we survey the predominant methods used in e-commerce applications. After we sketch the different phases of e-shopping, we take a closer look into the payment phase. We discuss e-cash, e-purse and other payment methods. Thereafter we look into other payment systems such as PayPal, which have become popular over the last years.

3.1 E-shopping

E-shopping is a multi-phase process. In the first phase, the customer browses through the products and makes a decision about what to buy. The selected goods are then typically put into the shopping cart and at the end of the selection process, the customer is asked to proceed to the check out. In the second phase, the customer selects the payment and delivery options. In the third phase, the goods are delivered.

This chapter is only concerned with the second of the three phases. In particular, we focus on the different payment methods, which are used today.

Statistics show that there are several reasons why customers are reluctant to buy online. Many customers are concerned about the lack of decent security mechanisms. Other reasons for not buying electronically include the lack of trust in the merchant, the lack of reliability, and the difficulty of the e-shopping process itself. Consequently, one of the most important factors for the success of e-business is having a secure and reliable system. However, what secure means is heavily depending on who is asking this question, what you are trying to protect, on the perceived threats, on resulting business impacts, and on how much risk you are willing to take. To make a long story short, the definition of security is depending on many factors and can vary from one e-commerce application (system) to the other.

3.2 Types of Payments

As electronic payments are simply the electronic versions of ordinary payment methods, we discuss the different types of ordinary payment methods first.

Cash is a floating claim on a bank or other financial institution that is not linked to any particular account. Cash is a *direct payment method*; the customer pays for the goods and the merchant delivers them virtually at the same time. Direct payment methods are well suited for occasional purchasing and for buying physical products. It is less suited for buying services that are composed of several separate steps, which have to be paid for; repeated payments simply annoy customers. Other types of direct payment systems are *checks* and *bank cards*. Checks are suitable for any not too small value. The payee can be remote from the payer, and the payer gets a few days of grace before the actual payment takes place. A disadvantage is that rejected checks are costly. Bankcards are also suitable for any not too small value. The payee can be remote from the payer. Many bankcard systems require online connections for verification nowadays.

Credit systems work differently. In a credit system, the customer registers with a bank and gets an account. Whenever goods are purchased, this account is debited. The customer is billed only after a certain amount has been reached or after a certain period of time.

Debit systems are the last type of payment systems. In a debit system, the customer pays before the purchase. Such systems are suitable for subscriptions or any regular purchases.

3.3 E-payments

As mentioned before, e-payment methods are simply the electronic versions of traditional payment methods. Consequently, e-cash (or e-money) is the replacement for cash, e-check is the replacement for check, bankcards, credit and debit systems mostly involve electronic transactions anyway. In the subsequent sections, we focus on e-money and bankcards only.

3.3.1 E-money

E-money or *digital currency or currency* is simply an encoded string of digits and essentially consists of a serial number and the amount. As traditional money, it is a floating claim on a bank or other financial institution that is not linked to any particular account. E-cash payments should be possible from one person/entity to another without bank involvement. The expected advantages from e-cash are that it is anonymous, and involved parties do not have to trust each other. The expected disadvantage follows directly from the anonymity; it is not traceable.

An important aspect for e-money is the size of a transaction. We distinguish *micropayments* and *macropayments*. Micropayments are tiny value transactions below \$1. They are supposed to be cheap and a little cheating can be tolerated. Macropayments are medium value transactions (between \$1 and \$1000), and large value transactions (above \$1000).

Several aspects contribute to the acceptance of digital currency.

- **Security:** It is mandatory to prevent forgery, double spending, and collusion of other parties (multi-party security).
- **Acceptability:** A wide range of parties needs to accept the payments.
- **Convenience:** Factors that increase the convenience are speed, reliability, divisibility, and transferability. The latter two aspects are discussed below.
- **Cost:** For ordinary money transactions virtually no costs (of course the banks always try to get more than their fair share) do arise for the involved parties, even if the transaction involves only a very small amount of money. This property is also desirable for digital currency.
- **Privacy:** Cash is the most anonymous payment form. Consequently, e-money should be anonymous as well.
- **Durability:** E-money should not be easily lost. For example, in case of a system crash, you would expect that your money is not automatically lost.
- **Independence:** Cash can be stored in a purse, in the pockets of your trousers, etc. Consequently, for e-money it might be convenient if it is storable on different devices.
- **Transferability:** Cash can be exchanged by different entities. It can go from customer to customer, from customer to merchant (and vice versa), from customer to bank (and vice verse), etc. This type of peer-to-peer payments might be desirable for e-money as well.
- **Divisibility:** If a customer buys goods with a certain amount but has only cash of a higher amount it is easily possible for the merchant to pay back the difference. The same behaviour is expected also from e-money.

- **Immediate Control:** In case of a security breach, such as forgery of money, in the system, it must be possible to identify the security breach immediately.
- **Traceability:** Especially for large transactions, anonymous money is a problem. With ordinary money, as soon as a large transaction is taking place, the merchants require name and address to produce a bill. In addition to counteract money laundering, some sort of tracing mechanism might be an advantage.
- **Spread of Encryption Mechanisms:** We have seen that for many security services encryption mechanisms are required. However, in many countries, encryption is under governmental control. E-money systems which involve encryption heavily might be difficult to export.

Apparently, some of the before mentioned aspects contradict each other. Trade-offs between these aspects will have to be made in real-world systems:

Privacy vs Traceability: Tiny value payments deserve to be anonymous, while for large value payments traceability is probably favourable.

On-line vs Off-line: On-line payments allow to prevent double spending easily and simplify tracing transactions. Off-line payments often use special purpose hardware to prevent double spending.

Hardware vs Software: Hardware solutions can also prevent double spending, but often hardware decreases the customer acceptance.

Transparency vs Explicitness: Users want to have control over all their transactions but they also expect not to be bothered with any (technical) details.

Because of the highly demanding properties of e-money most vendors offering such systems went bust quickly. In the following paragraph, we discuss the most prominent example of an e-money solution, which was a highly sophisticated system, but had a low user acceptance. Consequently, also the company behind the system went bust as well.

E-cash In this section, we discuss the basic principles behind the famous DigiCash system. In the DigiCash system, coins are anonymous, *i.e.*, the bank does not know which coin belongs to which customer. Despite the anonymity, the system prevents multiple spending of coins. Furthermore, it allows even to determine with a high probability who tried to double-spend money in case such a fraud occurred.

We keep the explanation as short and simple as possible. There are three entities involved in the system: Alice the customer, Bob the bank, and Martin the merchant. In the DigiCash system, Alice generates the money together with Bob. The protocol describing the DigiCash system works as follows:

Money Generation Phase:

1. Alice prepares n anonymous money orders for a given amount. Each money order consists of a random uniqueness string X (a serial number), and m pairs of identity bit strings

$$I_1 = (I_{1_L}, I_{1_R}), I_2 = (I_{2_L}, I_{2_R}), \ldots, I_m = (I_{m_L}, I_{m_R})$$

   to which Alice commits. These identity strings have three important properties. Firstly, they cannot be changed later on by Alice. Secondly, each pair of strings allows to identify Alice, and thirdly, the strings are sealed by default.
2. Alice puts each of the n money orders into an envelope, seals all envelopes, and sends all n envelopes to the bank Bob.
3. Bob asks Alice to unseal and open $n - 1$ envelopes of his choice. Bob checks the opened money orders for correctness and asks Alice to reveal all identity strings.
4. If Bob is satisfied that Alice did not make any attempts to cheat, Bob signs the one remaining sealed money order, gives it back to Alice and deducts the proper amount from Alice's bank account.
5. Alice unseals the money order to extract the coin.

Money Spending Phase:

1. Alice unseals the money order and spends the coin with the merchant Martin.
2. Martin checks the validity of the coin by verifying Bob's signature.
3. Martin asks Alice to reveal either the left or the right half of each identity string on the coin. He does that by giving Alice a random selector string that tells her which half to reveal.
4. Alice complies.

Double-spending Checking Phase:

1. Martin takes the coin to Bob.
2. Bob verifies the signature and checks his database for the uniqueness string X to make sure that the coin has not been previously deposited. If it has not, then Bob credits the amount to the merchant's account and records the uniqueness string in the database.

3. If the uniqueness string is in the database, Bob does not accept the coin. Then, Bob compares the identity string on the coin with the one stored in the database. If the strings match Bob knows that the merchant cheated. Bob knows this because only with a probability of $1/2^m$ the same parts of the identity strings are revealed without cheating. If the strings do not match, then Alice must have copied the coin. As the second merchant who accepted the coin gave Alice a different random selector string, Bob finds a position where one merchant has opened the left half and the other merchant has opened the right half of the identity string. Both halves together allow Bob to reveal Alice's identity.

Remark: In order to fully understand the ideas, we introduce two cryptographic concepts: *commitment schemes* and *blind signatures.*

In the first step of the money generation phase Alice is said to commit to the identity strings. This commitment ensures that she cannot change them later on. A simple way of realizing a commitment scheme is to use a cryptographic hash functions h. In the money generation phase, Alice produces n money orders. That is, she needs n serial numbers X_i, and n identity string sequences for $1 \leq i \leq n$:

$$I_{i1} = (I_{i1_L}, I_{i1_R}), I_2 = (I_{i2_L}, I_{i2_R}), \ldots, I_{im} = (I_{im_L}, I_{im_R})$$

Then she computes $y_{ij_L} = h(I_{ij_L})$ and $y_{ij_R} = h(I_{ij_R})$ for all $1 \leq i \leq n$ and $1 \leq j \leq m$. Then, the (commited) money orders for the ammount A have the form:

$$\begin{aligned} O_1 &= (X_1, (y_{11_L}, y_{11_R}), \ldots, (y_{1m_L}, y_{1m_R}), A) \\ &\cdots \\ O_n &= (X_n, (y_{n1_L}, y_{n1_R}), \ldots, (y_{nm_L}, y_{nm_R}), A) \end{aligned} \tag{3.1}$$

Next, Alice "blinds" the money orders, so that the bank doesn't see the serial numbers. (Otherwise, the bank could link the serial numbers with Alice). Assume (e, N) is the RSA public key of the bank. For every money O_i order Alice produces a random number r_i and then computes $B_i \equiv r_i^e \cdot O_i \bmod N$. That is, the contents of B_i is blinded, it looks like garbage for anyone who doesn't know r_i.

The next step is that the bank asks Alice to reveal the contents of $n-1$ money orders, that is, the bank asks Alice to send the blinding values r_i for $n-1$ orders of the bank's choice. With these values, the bank "unblinds" the money orders and checks them for consistency (do they conform to the standard, and especially, do all request the *same* ammount of money?). If this is correct, then the bank signs the last remaining blinded money order with its private key d without seeing what is inside:

$$B^d \equiv (r^e \cdot O)^d \equiv r^{ed} O^d \equiv r \cdot O^d \bmod N,$$

and sends the result back to Alice. Since Alice knows the blinding value r, she computes $r^{-1} \cdot r \cdot O^d$ mod N and now possesses an anonymous coin signed by the bank.

In the money spending phase, Martin the merchant checks the validity of the signature on the coin. Furthermore, Martin sends to Alice a sequence of $L - R$ values of length m. According to this sequence, e. g. $(L, R, L, L, \ldots, R, R, L)$, Alice has to open either the left or the right half of the commitment. That is, Alice sends to Martin the string $(I_{1_L}, I_{2_R}, I_{3_L}, I_{4_L}, \ldots, I_{(m-2)_R}, I_{(m-1)_R}, I_{m_L})$ and he tests if $h(I_{1_L}) = y_{1_L}$ etc. If all the commitments were correct, then Martin stores the coin together with the identity string sent by ALice.

When Martin now deposits the coin at the bank, the signature is again validated. Furthermore, the database of the bank is checked if the serial number X is already present in the database. If this is the case, the bank checks the stored identity strings. If the merchant wanted to collect money for the same coin twice, then these identity strings will be equal, since the merchant is not able to compute valid identity strings I from the commitments y, he has to use Alice's result twice.

If, on the other hand, Alice wanted to spend the coin at two merchants, the probability that both of them asked her for the same sequence of $L - R$-values is negligible small. In that case, there exists an index j such that both I_{j_L} and I_{j_R} are present. The identity strings were chosen in a way, that if both halves are present, the owner (Alice) can be identified.

3.3.2 Bank cards

In several European countries, bankcards are indeed smart cards – they make use of the integrated circuit and only use the magnetic stripe as a fallback. In Austria, bankcards have two different functionalities: Maestro and Quick. Both allow making payments, but only Quick provides a certain degree of anonymity. Similar systems to Quick do also exist in other countries. For example, in Germany the *Geldkarte* is used, or in Belgium, *Proton* is a well-known system.

Information about the details of the Quick system is hard to obtain. The official Quick website `www.quick.at` does not reveal any technical detail. Of course, no specification can be obtained from there either. The following information is based on an email from Europay Austria.

Whenever money is being put on a Quick card, the PIN must be entered and is checked by an online system. Hence, all such transactions are authorized by this central system. In all Quick terminals, there is a smart card, the so-called *terminal card* included which is used for "offline" payments. This card communicates with the customer card, saves information about the turnover, and can provide additional authentication mechanisms.

Whenever a Quick card is loaded, the bank account of the owner of the card is debited and the amount of money is put on a so-called pool-account of Europay

Austria. Whenever a payment is made with a Quick card, the Quick card and the terminal card perform an authentication. Only after the authentication succeeds, a transaction may take place. The amount of money for the payment is deducted from the customer's Quick card and booked to the terminal card. Hence, each payment increases the amount of money that is loaded up to a terminal card. Whenever the merchant collects the cash, the balance is supplied with an authenticator. The data is then transferred to Europay Austria and checked for correctness. If everything works out, the merchant receives the money and the pool-account is debited.

Europay Austria is the only party responsible for this system.

3.3.3 SET and its Successors

SET stands for *Secure Electronic Transactions* and is a specification for authentication for paying with credit cards within online networks, including the internet. Visa and MasterCard have jointly developed SET. SET relies heavily on PKI. The security goals targeted by SET include ensuring the privacy of the customer-data (order information, credit card information) and asserting the authenticity of the involved party.

In SET, the cardholder uses a software called an *electronic wallet*, in which the credit card numbers and digital certificate are stored. The merchant acquires a digital certificate from a financial institution. Both the cardholder and the merchant use their digital certificates for authentication before a transaction. During a SET transaction, the cardholder's credit card number is not seen by the merchant. Only the encrypted credit card number is sent to the credit card issuer, which approves the transaction for the merchant. In this way, disclosure of private data is prohibited during the transaction.

The cryptographic means to ensure the privacy of customer data is the so-called *dual signature*. It allows the cardholder to communicate with both the merchant and the bank in a single message. The message contains an order section plus a payment section:

- The merchant does not need to see the payment section.
- The bank does not need to see the order section.
- But, both sections need to be bound together.

Alice produces hashes of the order section and the payment section. She signs the concatenation of these both hashes (*i.e.*, she produces the dual signature). The bank and the merchant can verify the signature. However, neither of the two hashes allows reconstructing the input.

In order to send the message confidentially to, for example, the bank, Alice generates a session key and encrypts the payment section, the dual signature, the digest of her order section and her certificate. Alice encrypts the session key under the bank's public key (i.e. she produces a so-called *digital envelop*) and sends the encrypted message and the encrypted key to the bank. The bank can decrypt the key and with the key the message. In addition, the bank can verify the dual signature. However, as only the hash of the order section has been included, the bank has no information about the ordered goods, but only about the amount of money, Alice spent.

The disadvantages of SET were the significant investment required for cardholder, merchant, and bank. Cardholders in particular need a wallet along with public/private key pair and digital certificate. Therefore, take-up was very slow. In the meantime, SET has been replaced by proprietary systems from VISA (3D-Secure) and MasterCard (MasterCard SecureCode). Both systems are incompatible, but do require less effort for the customer. 3D-Secure is a system, which requires no software on the customer's side but only on the merchant's side. In a 3D-Secure transaction, the customer sends payment details to the merchant who checks the registration of the customer with VISA. If the customer is registered with VISA, the merchant sends an authentication request to the customer. The customer authenticates herself against her bank but sends the authentication result back to the merchant. After successful authentication, the merchant requests payment from the bank.

3.4 Alternative E-Payment Methods

As all sophisticated e-payment methods have more or less vanished, the less sophisticated but more practical ones are discussed in this section.

3.4.1 PayPal

Founded in 1998, PayPal enables any individual or business with an email address to send and receive payments online. PayPal's service builds on the existing financial infrastructure of bank accounts and credit cards. PayPal has over 25 million registered users, including more than 3 million business accounts at the time of writing this lecture notes.

To conduct some business, the customer and the merchant need to have a PayPal account. Such a PayPal account can be created by providing some personal data, such as credit card and/or bank information, and by verifying the customer's e-mail address.

To send money, one just goes to the PayPal site, enters information, such as the name, e-mail address of the person (merchant, customer), and amount to be transferred. The payee receives the cash in his/her PayPal account at the speed of e-mail, billed to the payer's PayPal account (which is tied electronically to the payers credit card, debit card or checking account).

The payee can ship the bought item, with no need to wait for the mail to arrive and for the check to clear. Furthermore, the payee never sees details such as credit card or bank account number of the payer. It is also possible that the payee can deduct money from the payers account.

PayPal automatically encrypts confidential information in transit using the Secure Sockets Layer protocol (SSL). All information stored by PayPal, resides on a server that is protected both physically and electronically.

Although PayPal takes an effort to guarantee the security of the system, the system shows some considerable problems that are illustrated best by a real world example (taken from `http://www.relfe.com/paypal_problem.html`).

On this website, a user describes his experience with PayPal. The user gave away his VISA details at an online shop to buy some goods (the payment was not via PayPal). The VISA card was hooked to his bank account. This user also had a PayPal account hooked to the same bank account. The user had expected the online shop to deduct some amount from his VISA (bank account). However, as it turned out, the money was deducted by PayPal and not by the merchant. How was this possible? It turned out that the merchant had opened a PayPal account in the name of the user. Therefore, PayPal would deduct the money and transfer it to the merchant. That some merchant could open a PayPal account for a customer without needing confirmation from the customer was already bad. However, it became even worse because the customer was not able to close this second PayPal account. The reasoning for this was that he did not create the account. Consequently, he cannot provide the necessary credentials (password, e-mail address) to close the account. Even an e-mail to PayPal did not really help. The account was only blocked. It will be deactivated per default after three years of inactivity.

There are several website on which users describe their problems with the PayPal system (`www.paypalsucks.com`, `www.endpaypal2002.netfirms.com`). It appears to be wise to use a special bank account for PayPal with very little money on it to limit a possible loss.

In the end of 2002, PayPal has been fully taken over by eBay. Some additional security features were introduced later on. In 2006, a security key was introduced to provide an additional authentication layer. In the login process, the username/pwd combination has to be provided and additionally, one has to be in the possession of the security key, to produce a six-digit number. Analogous to banking accounts, also a Mobile TAN system is available for PayPal.

3.4.2 PaySafe Card

PaySafe card is a payment system which was invented in Austria. It is a pre-paid card and can be used in many web-shops. The advantage of the system is that it protects the privacy of users: a user buys a paysafecard, which can be used to buy goods up to a certain amount of money. Different paysafecard types do exist.

Standard paysafecards are worth up to 100 Euros. Other variants (intended for teens) are worth up to 25 Euros. Cards can be bought from several dealers.

The usage of the card is quite simple. The buyer rubs with a coin to uncover a 16-digit PIN code on the card. In the Web shop, he or she enters this code and the sum to be paid. The entered pin number is routed to the Web server of paysafecard.com and encrypted per SSL. Here, the input is checked by the application and routed to various databases. The amount is debited from the card account.

Before the cards are produced and delivered, the system creates unique 16-digit numbers. Before the cards go to the distributor, they are loaded in the system. However, they are not activated yet. It is only when the distributor, for example, passes a small number of them to a dealer that the cards are activated. This means that if the cards are stolen beforehand, they are worthless.

The entire system is hosted in a Vienna IBM data centre, physically secured towards the outside and with a shared firewall to two Internet service providers. There are also firewalls to the client PCs at `www.paysafecard.com` itself, to SAP Server Support, and to the IBM service network for maintenance.

The PaySafe card has won the prestigeous Paybefore Awards 2009 in the category "Best Prepaidcard /Prepaidprogram Outside the US".

3.4.3 FirstGate Click&Buy

This payment system has been developed from the German company FirstGate. It can be used for micropayments and works very simple. The customer registers with FirstGate by providing information such as her name, address, e-mail and bank account details. At the merchant's side, content is linked to Click&Buy and can only be downloaded after the user has confirmed the price by providing username and password. All transmissions are secured by SSL. The payments/purchases a customer has made are not directly deducted from her account. Instead FirstGate collects them and only bills them only once per month from the customer. This reduces the overall cost of the billing.

3.4.4 Google Wallet and NFC

Google Wallet is a mobile payment system developed by Google^{TM} that allows its users to store debit cards, credit cards, loyalty cards, and gift cards `http://www.google.com/wallet/`. Google Wallet is based on the near field communication technology (NFC) `http://www.nfc-forum.org/aboutnfc/`. NFC is a set of standards covering data exchange and communications of smartphons and similar devices. The concept is to make secure payments by simply tapping the phone on the terminal at the time of payment. The NFC technology is also used in several AFC-schemes (automatic fare collection) with contact-less smartcards.

3.5 Summary

Electronic commerce has many aspects of which we discussed only one, namely digital money (or digital currency, or e-cash). We have discussed properties of interest for digital money. Then, we have sketched a system that allows the generation of anonymous digital money and that at the same time, allows preventing double spending and that allows finding the cheating party. Bankcard systems such as Quick, and Internet and credit card based systems such as SET were also briefly investigated. Finally, we showed that all successful systems have one common property: they are easy to use and their security concept is simple. Apparently, sophisticated systems are too complex and inconvenient to succeed in practice.

3.6 Review Exercises

1. What types of payment systems can you distinguish?
2. What is the difference between micropayments and macropayments?
3. Discuss 4 aspects of digital currency.
4. Discuss the trade-offs that are typically made in electronic payment systems.
5. Explain in short how money is generated in the DigiCash system.
6. What means SET and what was/is the SET system used for?
7. Explain the purpose of the dual signature in the SET system.

Chapter 4

Network Security

Using encryption on the internet is the equivalent of arranging an armored car to deliver credit card information from someone living in a cardboard box to someone living on a park bench.
— Gene Spafford

Network security addresses all security issues which come into play when two parties, Alice and Bob, attempt to communicate over a network. Typically, it is assumed that the network is not secure and certainly not trusted. This scenario is actually the same as we have introduced in the first chapter of this book, which introduced basic cryptographic principles.

However, cryptography alone does not solve the network security problem. For example an attack, in which the attacker infiltrates your PC by installing malicious software, is not counteracted by encryption (or any other cryptographic procedure).

In this chapter, we focus on network security, *i.e.*, on attacks and countermeasures, which are not prevented by cryptographic methods.

4.1 Social Engineering

Software, which shows weaknesses against attacks, can typically be patched, upgraded or in the worst case, replaced. Humans however, cannot be patched or upgraded. Humans are always the weakest link in every security infrastructure.

Social engineering attacks target humans. In such attacks, the attacker tries to persuade a person (with access to some valuable information) to give the attacker access to this information. The less security-aware users are, the easier social engineering attacks are. Even worse, the best firewall and the best intrusion detection system will not defend you against an attacker who got a password from one of the users.

In some social engineering attacks, users are tricked into opening backdoors into their system by offering free downloads (music, movies, porns, etc.).

Social engineering attacks typically try to achieve one of the following four goals [PC04]:

1. Physical access (to violate any of the three CIA services)

2. Remote access credentials (login and password)

3. Confidential information (data, source code)

4. Violation of other security controls (run malicious code)

Social engineering attacks are further subdivided into *active* and *passive* attacks. In active attacks, the attacker actively interacts with the attacked person. In a passive attack, the attacker acquires information with stealth.

Various samples of engineering attacks can be found in the literature. For example, an attacker can pretend to be a mailman in order to obtain access to the office of some company. In this case, the attacker uses an *impersonation* technique to get privileged physical access. Alternatively, in another scenario, the attacker can use information, which he obtained in the past, to *blackmail* a person and to force the person to give her the desired information. Another strategy of an attacker can be to ask for advice or guidance from an employee. Such an attacker, which requires the attacker to show and obtain the *sympathy* of the employee requires some acting skills.

4.2 OS Fingerprinting

OS Fingerprinting is a technique in which the attacker tries to determine the operating system, which is running on the target machine. As much vulnerabilities, which can be exploited in an attack, are OS dependent, OS fingerprinting is typically the first step in a network attack.

4.2.1 Telnet

Telnet session negotiation is probably the simplest way to determine the OS. You simply have to open a telnet session to the remote server. Surprisingly, many servers have a telnet server running although telnet is known to have no security features and should be deactivated per default. Even worse, telnet servers often respond with a whole bunch of information about the operating system.

4.2.2 TCP Stack Fingerprinting

Another way of gathering information about the OS running on a remote host is to look at how the TCP stack is implemented by the OS. Some operating systems, in particular of the Windows family, have very distinct implementations of the TCP stack. Consequently, they can be identified by sending special TCP packets to the server and see, how the server reacts on these packets.

A very good tool, which implements this strategy, is *Nmap*. Nmap can be found at `www.insecure.org/nmap`. Nmap uses several interesting techniques including:

FIN probe: Sends a FIN package to the remote host. According to RFC 793, the server should not respond, but Windows implementations do not follow this RFC.

ACK value: Different operating systems use different values in the ACK field.

ICMP error message quenching: Operating systems that follow RFC 1812 limit the rate at which various error messages are sent. Just sent a bunch of packets and see how the implementation behaves.

TCP options: The TCP options vary from OS to OS.

TCP ISN Sampling: The idea here is to find patterns in the initial sequence numbers (ISNs) chosen by TCP implementations when responding to a connection request. TCP implementations can be categorized into many groups. These groups can further be divided into subclasses by computing variances, greatest common divisors, etc.

The above list is non-exhaustive.

Nmap is an active fingerprinting tool which sends certain packages to the remote host. Passive fingerprinting tools only sniff packages without sending any packets. Such tools work because some TCP/IP flag settings vary from implementation to implementation. A passive fingerprinting tool is for example *p0f*, which can be found at `www.stearns.org/p0f`. An extensive collection of fingerprinting tools can be found at `http://www.l0t3k.org/security/tools/fingerprinting/`.

There are attempts to defeat mechanisms such as Nmap. There exists, for example, a Linux netfilter module (IP Personality, `ippersonality.sourcefourge.net`, which attempts to slightly alter the behavior of the IP stack. Even without dedicated countermeasures, some obstacles can make fingerprinting difficult in practice. For example, a packet can be modified while it is in transit. Packet filtering devices might change one or several field values of a packet.

4.3 Attacking Internet Protocols

In this section, we discuss attacks on the TCP Protocol. Especially the initial handshake in TCP, where the communicating parties perform a three-way handshake (see Figure 4.1) and set up sequence numbers, can be attacked in surprisingly many ways.

$$\begin{array}{ccc} \textit{Alice} & \xrightarrow{SYN, X} & \textit{Bob} \\ & \xleftarrow{ACK, X+1, SYN, Y} & \\ & \xrightarrow{ACK, Y+1} & \end{array}$$

Figure 4.1: TCP Handshake

4.3.1 SYN Flooding

In a *SYN flooding* attack, the adversary sends a large number of SYN packets (*i.e.* the message in Step 1 of the the TCP handshake protocol) to the remote host and never acknowledges any of the replies. In this way, the remote host receives accumulates more SYN packages than it can handle. This is a classical denial-of-service attack and has been known since the 1980s.

A technical solution to thwart such attacks is to use the SYNcookie and is incorporated into several systems. Instead of keeping a copy of an incoming SYN packet, Bob uses as Y an encrypted (hashed) version of X. In this way, sessions do not need to be kept half-open.

4.3.2 Smurfing

In a *smurfing* attack, a vulnerability of the Internet Control Message Protocol (ICMP) is exploited. In the ICMP, a user can send an echo packet to a remote host in order to check if the host is alive. In some implementations, the host even responds to pings to the broadcast address that is often shared by a number of hosts.

In a smurfing attack, the adversary constructs a packet with the source address forged to be that of the attacked computer and to send it to a number of other hosts. The other hosts will respond by sending a packet to the target. If enough hosts respond, the target computer will receive more packets than it can handle.

A technical solution to this problem was introduced in 1999. It consists of a change in the ICMP standard such that ping packets sent to a broadcast address are unanswered.

4.3.3 Distributed Denial-of-Service Attacks

In *distributed denial-of-service* (DDOS) attacks, an adversary takes over a large number of machines and installs custom attack software on them. At a pre-determined time, or on a given signal, the software activates and starts bombarding the target machine with messages.

An attempt to counteract such attacks is to add ICMP trace back messages to the infrastructure. This means that for some messages, a router sends an ICMP packet to the destination that includes information about the previous hop and the next hop.

4.3.4 Spoofing

Assume that the evil web server Eve wants to impersonate the web server Alice and establishs a connection as Alice with Bob.

Spoofing attacks combine some of the ideas, which are used in the previously described attacks. An adversary can use a denial-of-service attack to take down the web server with the name Alice. Then Eve can attempt to initiate a connection with Bob. Eve can send messages to Bob as Alice. However, Bob's response will be directed to Alice. Consequently, Eve cannot necessarily read those messages and therefore, needs to guess the sequence number Y which Bob uses. The best way to detect a spoofing attack is to monitor the reply to the SYN+ACK packet (sent by Alice). If the spoofed host is still alive it will return a NACK (RESET flag set), which can be used to identify this type of attack. Another method is to check the acknowledgement number arriving right after the SYN+ACK was sent. If the acknowledgment number does not comply with the initial sequence number, the attacker Eve is trying to guess the ISN.

4.4 Trojans, Viruses, Worms and other Malicious Code

A trojan, virus or a worm is essentially a piece of malicious code or software. Roughly spoken, a *Trojan* is a program doing something malicious such as stealing passwords; a *worm* is a program that replicates and a *virus* is a worm that replicates by attaching itself to other programs [And01]. *Polymorphic* viruses change their code whenever they replicate.

4.4.1 Taxonomy of Malicious Programms

Malicious Programs can be divided into two categories: those that need a host program and those that are independent. Into the first category fall *trap doors*, *logic bombs*, *Trojan horses* and *viruses*. Into the second category fall *worms* and *zombies*. Viruses, worms and zombies are furthermore able to replicate themselves.

4.4.2 Trap doors

A trap door is a secret entry point into a program. This technique has been used legitimately for many years by programmers to debug and test programs. Such a trap door can be a part of the code that recognizes some special (input) sequence or is triggered by being run from a certain user ID. A trap door is always then a problem when unintended users discover and abuse it.

4.4.3 Logic Bombs

A logic bomb is some code in a program that is set to detonate when certain conditions are met. Examples of such conditions are the presence or absence of certain files, a certain date, or a particular user. Once triggered, a logic bomb may cause a machine halt, delete files or do other things.

4.4.4 Zombies

A zombie is a program that takes over a computer, which is attached to the internet, to launch attacks that are difficult to trace to the creator of the zombie. Often, zombies are used in denial-of-service attacks against web sites.

4.4.5 Viruses and Worms

A virus is a program that infects other programs by modifying them. A virus or a worm typically consists of two components: a replications mechanism and a payload. A simple *replication mechanism* is for example when a worm makes a copy of itself wherever it runs. In former days, DOS viruses replicated themselves by attaching themselves to an executable file, and then patch themselves in such that the execution path jumped to the virus code and then back to the original program. It was popular to infect `.com`. Such files had their code always starting at address $0x100$. Hence it was simple to replace the instruction at $0x100$ with one that would point to the virus location. A trigger such as a date typically activates the *payload*. Upon activation, the payload can perform several (bad) things such as deleting or modifying user data, bring down the network, abuse the network by for example opening a dial-up connection to an extremely expensive number, or simply open a backdoor to the system.

Viruses and worms typically go through four phases during their lifetime: in the *dormant phase* the virus is idle. In the *propagation phase* the virus infects other programs. In the *triggering phase* the virus is activated and prepares to perform its function. In the *execution phase* the virus performs its function.

Different types of viruses are distinguished [Sta03]:

- **Parasitic viruses** attach themselves to executables and replicate by finding other executables to infect.
- **Memory-resident viruses** reside in main memory as part of the host system program; then they infect every program that executes.
- **Boot sector viruses** infect the master boot record (or boot record) and spread when a system is booted from the disk containing the virus.
- **Stealth viruses** are viruses that are explicitly designed to hide themselves from detection by antivirus software.
- **Polymorphic viruses** mutate with every infection.

Worms replicate themselves actively. Typically, network worms use network services for that purpose. Examples include

- **Electronic Mail:** Worms e-mail a copy of themselves to other systems.
- **Remote Execution:** Worms execute a copy of themselves on another system.
- **Remote Login:** Worms log onto remote systems as a user then copy themselves from one system to the other.

Code Red is one of the most prominent examples of worms. It exploits a security hole in the MS Internet Information Server to penetrate systems and spread. It probes random IP addresses to spread to other hosts. After a certain period, it initiates a denial-of-service attack against a web site (SYN flooding). The worm suspends activities and re-activates periodically. In the second wave of attack, Code Red infected nearly 360.000 servers in 14 hours.

Countermeasures against viruses and worms consist mainly of using antivirus software. Such software investigates programs (running or not) for strings which are known to be part of viruses. Another technique, which is used, is to keep checksums of original executables on the system and to check whether the checksum change. However, a virus writer can easily fool a simple checksum. Cryptographic checksums (maybe together with digital signatures) are becoming more and more important in this application. Today, four generations of antivirus software are distinguished. *First-generation* scanners require a virus signature to identify a virus. *Second-generation* scanners use heuristics to search for a probable virus infection. Such a heuristic often consist of searching for code fragments that are often associated with viruses. *Third-generation* products are memory resident programs that identify viruses by their actions rather than their structure. *Fourth-generation* products are packages of programs that consist of all known techniques.

4.5 Intrusion Detection and Firewalls

In principle, a *firewall* tries to actively prevent attacks by using fixed policies, whereas an *intrusion detection system* (short IDS) tries detect attacks. Modern IDSs go a step further, play a more active role, and try to block attacks as they happen. Network IDSs look at the network traffic for evidence. Host IDSs look at various hosts, OS and application activities.

4.5.1 Firewalls

A firewall is the last line of defense of a protected (internal) network against an untrusted (external) network. As such, a firewall is a process that filters all traffic between a protected network and a less trustworthy network. The filtering is done according to a security policy. A security policy determines which internal services (and information) may be accessed from the outside. Thus, a firewall does not provide more security as it is defined by the security policy! Furthermore, firewalls may be penetrated by new attacks and firewalls provide no protection against attacks inside the internal network. A short introduction to firewalls can be found at `http://csrc.nist.gov/publications/nistpubs/800-10/`. Firewalls consist of different components. These components are described in the following paragraphs.

Packet Filtering Firewalls, which are based on *packet filtering*, apply the security policy's rules by inspecting the header information of each packet. Header information from IP packets can be used for *source/destination level filtering.* For example, only a particular server of the protected domain may be accessed from the outside. TCP/UDP port information can be used for *service level filtering.* For instance, most services use well-known TCP/UDP ports and can thus filtering those works well. Advanced filtering rules, such as checking IP options, fragment offset, etc. can prevent attacks that are more sophisticated.

Application Gateways In order to counter some of the weaknesses of pure packet filtering, firewalls need to use software applications to filter connections for services such as TELNET and FTP. Such an application is referred to as a *proxy service*, while the host running the proxy service is referred to as an *application gateway.* Traffic control on the application level instead of IP level, allows observing and filtering the contents of particular services (context-sensitive). Advantages of this approach are the higher security than pure packet filters, the simplicity of logging and controlling all activities in detail and that supervised authentication is possible. Disadvantages of this approach are the need for specialized proxies and proxy-capable clients for most services; normally only most important services supported and this approach is harder to adapt to new technologies.

Circuit-Level Gateways A *circuit-level gateway* relays TCP connections but does no extra processing or filtering of the protocol. As an example of how circuit level gateways work, say computer Alice is in a network protected by a circuit level gateway firewall, and wants to view a web page on computer Bob outside the firewall. Alice sends the request for the web page to Bob, which is intercepted and recorded by the firewall before being passed on. Bob receives the request, which as far as it is concerned came from the address of the firewall, and starts sending the web-page data back across the Internet. When it reaches the firewall, it is compared to Alice's request to see if the IP address and the port match up, then the data is either allowed or dropped. There are two advantages of this approach. Firstly, no direct communication is allowed between outside sources and computers behind the firewall, since everything must first pass through a proxy. Secondly, filtering can now be done using the actual content of the data, as opposed to just where it came from and where it is going. *SOCKS* is a standard for circuit level gateways

4.5.2 Intrusion Detection Systems

Intrusion detection is because the behavior of an intruder is likely to differ from that of an legitimate user. Of course, there is not an exact distinction between an attack and a normal user but there is a way in which abnormal behavior can be quantified (detected). One approach to detect abnormal behavior is based on statistical methods. Such methods require collecting data relating to normal (expected) behavior of users. Statistical methods are used subsequently to determine whether a behavior encountered later on is not normal. Another approach is rule-based anomaly detection. It involves defining a set of rules that define proper behavior. Modern approaches also involve neural networks to detect abnormal behavior. IDSs can be roughly classified in three layers:

- host-based vs. network
- anomaly vs. misuse
- supervised vs. unsupervised.

Logfile Monitors The simplest example of a host-based IDS is a *log file monitor*. It attempts to detect and investigate intrusions by inspecting the log files of a system. This technique is rather limited; only afterwards an incident happened the IDS can investigate it. As log files are typically held only for high-level system events, intrusions occurring on a low level will not be noticed. Furthermore, an attacker can modify the log file and thus fool the IDS. A well-known log file monitor is *swatch* (`www.oit.ucsb.edu/~eta/swatch/`). Swatch scans log files in real time.

Integrity Monitors *Integrity monitors* watch key system structures such as system files or registry keys for change. A popular integrity monitor is called *Tripwire* (`www.tripwire.com`). It can monitor events such as file additions (deletions, or modifications), file flags changes, last access time, last write time, change time, file size, etc. When using an integrity monitor, it is important to start from a known safe baseline. Such a baseline can only be accomplished with a fresh system before it has been connected to the network. This baseline is also best recorded on a read-only memory.

Signature Matchers Somewhat like anti-virus software, *signature-based* IDSs (the so-called *signature matchers*) try to detect attacks based on known attack signatures. Hence, when an attack takes place and the attacker tries to use a certain exploit, the IDS attempts to match the exploit against its database. A prominent example of network-based IDS is *Snort* (`www.Snort.org`). Snort consists of a packet decoder, a detection engine and a logging and alerting subsystem. Snort is a stateful IDS. This means that Snort does not deal with isolated packets, instead, with the state of each connection. This concept is similar to stateful firewalls.

Anomaly Detectors *Anomaly detectors* are IDSs establish a baseline of a normal system or network activity and produce an alert when a deviation from the normal system occurs. If a network behaves not uniform (as it might be the case in many open networks) such an approach typically leads to host-based IDSs. However, in closed environments where the traffic is expected to be rather uniform, this approach may lead to network-based IDS.

General Problems with IDSs One of the problems of IDSs is that new exploits, which are not yet in the database of the IDS, are difficult to handle for the IDS. Network IDSs have another problem; they have to deal with a huge amount of data. Moreover, in switched networks, the IDS's sensors are curtailed. One can try to compensate for this by embedding the IDS in the switch. However, monitoring gigabit links with IDSs is also not trivial.

Quality of an IDS The quality of IDS can be measured by its sensitivity, its specificity and its accuracy. The *sensitivity* is defined as the fraction of intrusions that are detected by the IDS. The more sensitive an IDS is, the less likely it is to miss intrusions. The *specificity* is defined as the fraction of no intrusions that are detected by the IDS. The more specific an IDS is, the less false alarms it will produce. The *accuracy* is defined as the proportion of all IDS results which are correct. Therefore, it is a trade-off between sensitivity and specificity.

Circumventing IDSs Fragmentation (packet splitting) is a typical problem of network IDSs. Of course, stateful IDS will try to reassemble all fragmented packets for analysis. However, with a large amount of network traffic, this process requires more and more resources and becomes less and less accurate.

Another technique to fool IDS is to spoof the sequence number that the network IDS sees. For instance, a post-connection SYN packet with a forged sequence number can de-synchronize the IDS from the host. This is because the host will simply drop the unexpected SYN, whereas the IDS resets itself to the new sequence number. As it becomes clear from this example, a major disadvantage of a network IDS is that it does not know how a host will interpret the incoming traffic.

Whisker (`www.wiretrip.net`) is a tool, which sends carefully deformed HTTP requests to pass IDSs. It is one of the most prominent anti-IDS (AIDS) tools.

IPSec is the standard for securing (encrypting) data over the network on the network layer. Unfortunately, encrypted packets cannot be analyzed by a network IDS. Only a host IDS, which resides on each level of the TCP stack can analyze such packets.

4.6 Summary

Network security is an integral part of communication security. Several aspects make it difficult to achieve real network security. Firstly, social engineering can render all technological attempts to secure a system useless. A lessen learned from social engineering attacks is that user education is mandatory for a secure system. Secondly, we have seen that OS fingerprinting is already a well-developed technique. OS fingerprinting is often the first step in an attacker over the network. Due to the different implementations of the TCP stack, OS fingerprinting is feasible. Thirdly, because of TCP/IP was designed without having any security concerns in mind the current version of IP (Ipv4) opens to door to a large number of attacks. In addition, malicious software exploits weaknesses in the network and in the OS. Mechanisms to prevent network attacks, such as IDS and firewalls were discussed as well. We have seen that although a manifold of approaches do exist, secure networks are still far in the future.

4.7 Review Exercises

1. What is social engineering?
2. What are the goals of social engineering?
3. What is OS fingerprinting?
4. What is TCP stack fingerprinting?
5. What is accomplished by a smurfing attack?
6. What is a trap door?
7. What are the four phases of a virus?
8. What types of viruses can you distinguish?
9. What is a firewall?
10. What types of intrusion detection systems can you distinguish?

Chapter 5

Implementation Security

C makes it easy to shoot
yourself in the foot;
C++ makes it harder,
but when you do,
it blows away your whole leg.
— Bjarne Stroustrup

Most systems which have been broken in practice have been broken because of their insecure implementation. Implementation errors have always been a pain for both developers and end users. However, in security related applications, they are not only annoying. In security related applications, implementation errors can cause a complete break of the system. In this chapter, we discuss several types of such attacks.

5.1 Implementation Attacks and Countermeasures

Ever since cryptography became part of our everyday life, not only the cryptographic algorithms have been subject to attacks, but also their implementations in hard- or software. The traditional cryptographic model, however, does not take the aspect of implementation attacks into account. In the traditional scenario, Alice and Bob secure their communication by some mathematical function, namely the cryptographic algorithm. The adversary, Eve, is only assumed to have knowledge about this mathematical function and some plain- and ciphertext pairs. Consequently, security proofs only involve these components. Despite the given proofs of security for any cryptographic algorithm in any theoretical model, a communication system based on this algorithm can still be vulnerable to quite a number of other attacks. A very dangerous class of such attacks is commonly referred to as side-channel attacks.

In Section 5.2 we deal with the passive types of implementation attacks, which are known as side-channel or information-leakage attacks. Such attacks do not require the adversary to actively manipulate the computation, but only to monitor the side-channel leakage during the computation.

In Section 5.3 we deal with active attacks. As can be deduced from their name, this type of implementation attacks assume an attacker actively manipulating the execution of a cryptographic algorithm.

Each of the sections list some attacks and some algorithm specific countermeasures. Hence, algorithm independent countermeasures are completely omitted in this chapter.

Errors in (software) implementations also lead to serious compromises of security. In Section 5.4, we discuss the most prominent flaw, which are buffer overflows, and how they can be used for attacks.

In the last section of this chapter, Section 5.5 we explain why Java is a good choice for implementing security sensitive applications.

5.2 Passive Implementation Attacks

Passive attacks were published by Kocher in [Koc96] for the first time. In this article, timing information was used to gain knowledge about the secret key from implementations of the RSA, DSS, and other cryptosystems.

The attack described in this article required an attacker to be able to simulate or predict the timing behavior of the attacked device rather accurately. The second article by Kocher *et al.* [KJJ99] presented a similar, but far more dangerous attack. This second article introduced the usage of power consumption information to determine the secret key. Because of its statistical nature, one of the attacks in [KJJ99] is called *differential* power analysis. Another type of side-channel information was introduced only recently. The first articles, [QS01] and [GMO01], about the usage of electromagnetic emanations were presented in 2000 and published in 2001.

5.2.1 Types of Information Leakage

We briefly discuss the different types of information leakage that have been exploited by attacks published in the open literature so far.

Execution Time Leakage Often, a device takes slightly different amounts of time to execute an algorithm. Explanations for this behavior include differing input data which might cause some instructions to take different amounts of time for their executions, performance optimizations and branching instructions. Practical implementations of attacks using this kind of information leakage, such as [DKL+98], indicate that such attacks are difficult to realize in practice owing to the difficulty of measuring the real execution time. In many modern processors, even on smart cards, instructions can be cached and so the execution time is more and more related to other influences.

Countermeasures appear to be easy to implement, and to work efficiently in practice. Since their first introduction, most work in the area of side-channel attacks has been dedicated to the exploitation of side-channels with a higher amount of information.

Power Consumption Leakage Most commonly used cryptographic devices are implemented in CMOS (complementary metal-oxide semiconductor) logic. The power consumption characteristics of CMOS circuits can be summarized as follows. Whenever a circuit is clocked, the circuit gates change their states. This leads to a charging and discharging of the internal capacitors, and this in turn results in a current flow which is measurable at the outside of the device.

The measurements can be acquired easily using either a data acquisition card or a digital oscilloscope. The current flow can be measured directly with a current probe, or by putting a small resistor in series with the device's ground-input or power-input. Power analysis attacks are the most popular attacks at the time of writing owing to their effectiveness and simplicity. While the first publication [KJJ99] was mainly concerned with power-analysis attacks on secret-key cryptosystems, Messerges *et al.* [MDS99] presented such an attack for public-key cryptosystems.

Electromagnetic Radiation Leakage The same charging and discharging which occurs whenever a circuit is clocked creates, besides the current flow, also a certain electromagnetic (short EM) field. *Direct emanations* are caused by intentional current flow which is caused by the execution of an algorithm. *Unintentional emanations* are caused by the miniaturization and complexity of modern CMOS devices. This miniaturization and complexity results in coupling effects between components in close proximity. EM attacks are becoming more and more popular at the time of writing because of the high amount of information this side-channel can leak and because the information can be exploited at some distance (cm up to several meters depending on the environment) from the attacked device [AARR03].

Error Message Leakage An error message attack usually targets a device implementing a decryption scheme. In the standard model, there is a feedback from the device to tell whether or not the message has been successfully decrypted. If the attacker can somehow know the reason why the decryption operation failed, he might gain some secret information by sending well chosen ciphertexts to the device, or by observing others do the same.

Combining Side-Channels Not much research has been done on this topic so far. However, the following simple observation has been used for attacks. Timing attacks can suffer from the difficulty of obtaining precise measurements. Attacks are even more difficult when only one intermediate operation is targeted. In such a case, power measurements lead directly and more precisely to timing information about the intermediate operation if this intermediate operation is visible in the power consumption trace.

5.2.2 Simple Side-Channel Attacks

Most attacks presented so far have been performed with power consumption leakage information.

A *trace* refers to a measurement taken for one execution of the attacked cryptographic operation. In a simple side-channel attack, only one measurement is used to gain information about the device's secret key. Obviously, for such an attack to work, the side-channel information needs to be strong enough to be directly visible. Additionally, the secret key or hidden message needs to have some simple, exploitable relationship with the operations visible in the side-channel trace. Such an attack typically targets implementations which use key dependent branching.

Symmetric ciphers

A special class of simple power-analysis attacks, the so called *Hamming weight attacks*, exploit a strong relationship between the Hamming weight of the secret key and the power-consumption trace. For this type of attack it is vital that the implementation is based on relatively small data-words, as happens for example in an 8-bit implementation. Usually, this type of attack is applied to implementations of ciphers with a simple key schedule.

To counteract the type of simple power-analysis attack that uses Hamming weight information, a designer has to assure that the Hamming weight information which is leaked is not correlated with the intermediate values that are processed. In dedicated hardware implementations this can be achieved by using a special logic-style or by masking intermediate values (this can be achieved by bus encryption as well as by masking the operations of the algorithm in general). In software implementations the intermediate values have to be masked.

Asymmetric Ciphers

Scenarios in which simple side-channel attacks are a possible threat can be summarized as follows. If a *multiplication of a known and a secret value* has to be calculated, then a simple side-channel attack is theoretically possible, but unlikely to work in practice. An *exponentiation of a known with a secret value* is also in principle vulnerable to simple side-channel attacks. The practical feasibility of the attack is heavily dependent on the implementation. Unprotected *Scalar multiplications* of a known elliptic curve point by an unknown scalar are highly vulnerable to this kind of attack, regardless of the underlying hardware.

Attacks against a multiplication can be counteracted by switching multiplier and multiplicand. To protect implementations of modular exponentiations, a square-and-always-multiply approach can be helpful. The same is valid for implementations of the scalar point-multiplication on elliptic curves.

5.2.3 Differential Side-Channel Attacks

Differential side-channel attacks exploit the correlation between the processed data and the instantaneous side-channel leakage of the attacked cryptographic device. Because this correlation is usually very small, statistical methods must be used to exploit it efficiently. In a differential side-channel attack the output(s) of the real physical device and the output of a hypothetical model of the device (working with a hypothetical key) are compared. Only if the hypothetical key equals the real key the output of the hypothetical model is correlated with the output of the real device. By comparing the two outputs, the attacker can determine the secret key.

Symmetric Ciphers

The strength of an attack depends largely on the quality of the hypothetical model used by the attacker. Dedicated hardware implementations of Feistel ciphers without an initial bit-wise addition of the key allow the implementation of a very powerful hypothetical model. The statistical qualities of the S-boxes also influence the strength of an attack.

As we pointed out in the previous sections, software countermeasures are usually based on masking the data and the key during a computation. Ciphers which allow the cheap implementation of masking schemes are certainly preferable. For hardware countermeasures, their cheap realization is also a priority. If a special logic style is used then it is certainly an advantage if only a few different types of gates have to be designed in this logic style. The simpler a cipher's description, the more suited it is for such an implementation.

Asymmetric Ciphers

Typical targets for an attack are again implementations of the modular exponentiation and implementations of scalar point-multiplication on an elliptic curve.

Countermeasures for elliptic curve scalar point-multiplication include randomizing points, randomizing curves, randomizing the scalar and randomizing the algorithms for the scalar point-multiplication. Since practical realizations of elliptic curve cryptosystems are software implementations (which probably make use of some accelerator unit anyway), most of these countermeasures are cheap to implement and to combine with one another. Countermeasures for RSA do also randomize parameters and are typically referred to as blinding.

5.3 Active Implementation Attacks

In a passive attack, the attacker only eavesdrops on some side-channel information, which is analysed afterwards to reveal some secret information. An active attack involves an attacker that takes *active* part in the attack: we make the assumption that the attacker is able to somehow deviate the device from its normal behaviour, and tries to gain additional information by analysing its reactions. Some passive techniques seen in the previous section can be used to determine these reactions. This can be done, for instance, by modifying some internal data used by the device.

In the following we describe the most popular type of active attack: fault attacks. We will give examples of how successfully they have been applied, and see how they can be avoided.

5.3.1 Fault Attacks

When an attacker has physical access to a cryptographic device, he may try to force it to malfunction. A fault attack is an attack in which information about the message or the secret key is leaked from the output of erroneous computations. This kind of attack can be applied to both symmetric and asymmetric cryptosystems, and was first introduced in [BDL00].

There are several ways to introduce an error during the computation performed by the cryptographic device. Though the description of these practical means is beyond the scope of this introduction, we cite some non-invasive methods:

- spike attacks work by deviating the external power supply more than can be tolerated by the device. This will surely lead to a wrong computation.
- glitch attacks are similar to spike attacks, but target the clock contact of the integrated circuit.
- optical attacks work by focusing flash-light on the device in order to set or reset bits.

Introducing faults in the secret key of asymmetric schemes This kind of attack has been applied to the RSA decryption (or signature) scheme, and to the El Gamal, and DSA signature schemes. It works efficiently in the following model: bit flip fault model, complete control on the number of faulty bits induced, complete control on the location. For the timing, the only requirement is that the fault occurs before the critical computation, so we need only loose control on it.

The idea is to flip one bit of the secret key and to use the erroneous computation of the device to get the value of this bit. This is particularly easy for discrete logarithm based schemes because we can use the simple relation between the secret and the public keys to successively guess the bits of the secret key.

Introducing faults in registers Here, faults are introduced in an intermediate value stored in the registers of the device. This kind of attack has been applied to different identification schemes, where the random value r chosen by the prover is the target of the fault introduction. The prover has to store this value r, as he is going to use it again when responding to the verifier's challenge. The idea is to swap exactly one bit of r while the prover is waiting for the challenge. Several such erroneous computations are used to recover the secret key.

Random faults This is the most powerful attack, as the fault model and the number of faulty bits induced are random, and controls on the location and the timing are loose. The well known Bellcore attack [BDL00] against RSA using the Chinese Remainder Theorem belongs to this category. Here, a random error occurs during some (more or less time consuming) step of the computation, and the erroneous output completely reveals the secret key.

Fault attacks against elliptic curve cryptosystems The use of elliptic curves can lead to specific fault attacks. The idea is to somehow modify the point involved in the scalar multiplication step in such a way that the resulting point is on a cryptographically weak curve, where we can solve the discrete logarithm problem, and thus find out the secret key.

Fault attacks against symmetric cryptosystems Fault attacks as introduced in [BDL00] use algebraic properties of the asymmetric cryptosystems. With respect to symmetric cryptosystems, researchers introduced *differential fault analysis*, which uses statistical methods. Various fault models are considered, and several cryptosystems are attacked, among them the full DES.

5.3.2 Other attacks

The problem of public keys validation can also be exploited. It has been demonstrated that attacks can be mounted against some elliptic curve based schemes if the receiver of an elliptic curve point does not check that the point lies on the right curve. This shows the importance of validating values received from the outside.

5.3.3 Preventing fault attacks

As we have seen, fault attacks are very powerful attacks that may permit the cryptanalysis of theoretically secure schemes. Several software countermeasures have been proposed, among them:

- Double computation: for encryption schemes, this could be a solution. However, it doubles the computational time, and does not protect against permanent faults.
- Checking the output: this can be done quite efficiently with signature and identification schemes. However, it assumes that the device contains the whole public key, and this is not always the case.
- Randomization: here random bits are introduced in the computation. They are either XOR-ed to sensitive data to blind them, or appended to the message.

5.4 Software Security

Whereas implementation attacks are based on the fact that a careless implementation can lead to side-channels and to vulnerability against fault attacks, software security problems typically arise from simple programming errors. In this section, we discuss some of the common problems that arise from unskilled software implementations.

5.4.1 Buffer Overflow

A *buffer* is simply a contiguous block of computer memory that holds information. Hence, a stack is a simple buffer. A *buffer overflow* attack deliberately enters more data into the buffer than the buffer is supposed to handle.

Buffer overflows typically result from an inherent weakness in the C/C++ programming language. The problem is that C does not perform any bounds checking when passing data. The following sample code illustrates this problem [One96].

```
void function(char *str) {
   char buffer[16];

   strcpy(buffer,str);
}

void main() {
  char large_string[256];
  int i;

  for( i = 0; i < 255; i++)
    large_string[i] = 'A';

  function(large_string);
}
```

This sample code shows a typical programming error where a large string is copied in a too small string. This program will compile correct, but will cause an error during runtime.

5.4.2 Smashing the Stack

Because reading and writing data directly from RAM takes a lot of time, processors have a small amount of internal memory. This memory is split up into *registers* of a specific size. The most common size is 32 bits nowadays. The amount of data a processor can hold within its registers is extremely limited. Consequently, some external memory is used to hold pieces of information. The *stack* is simply a chunk of RAM that stores such data. If a program needs to store information, the information is *pushed* on the stack. If a programm needs to recall information, the information is *popped* of the stack. This is also called *last in, first out* (LIFO) principle.

A stack consists of logical stack frames. A stack frame contains the parameters to a function, its local variables, and the data necessary to recover the previous stack frame. The *stack pointer* points to the top of the stack. The *frame pointer* points to a fixed location within a frame.

When an attacker overflows a buffer on the stack, the buffer will grow toward the return address. The goal of an attacker is to change the return address. When the function is executed, the return address is popped off the stack and the new address is executed. If malicious code is located at that new address, it is executed with the same privileges as the application [PC04]. Partially, buffer overflow problems are due to the von Neumann architecture where data and code are stored in the same memory.

5.4.3 Heap Overflows

The *heap* is memory that is dynamically allocated for storing variables. In a heap overflow, the attacker tries to overwrite critical variables such as passwords, filenames etc.

The difference between a stack overflow and a heap overflow is that with a heap overflow attack we try to increase the level of system privilege.

5.4.4 Preventing Buffer Overflows

As a programmer, you should always check the the bounds of an array before writing the array into a buffer. Furthermore, you should use functions that limit the number (format) of input characters. Dangerous functions, such as strcpy() should not be used. Instead, you should use alternative functions such as strncpy(), which copies only the first n bytes of the source string into the buffer.

5.5 Java Security

In languages like C, it is solely up to the programmer to take care of malicious input data that can cause buffer overflows or other problems. Other languages, like Java, do at least aid the programmer with this task. In Java, there are two major concepts, which are the *sandbox* and the *JCA* that are important for implementing security related applications.

5.5.1 Java Sandbox

Java provides a customizable "sandbox" in which Java programs run. A Java program must play only inside its sandbox. It can do anything within the boundaries of its sandbox, but it can't take any action outside its sandbox.

The sandbox for untrusted Java applets, for example, is supposed to prohibit:

- Reading or writing to the local disk.
- Making a network connection to any host, except the host from which the applet came.
- Creating a new process.
- Loading a new dynamic library and directly calling a native method.

Hence, the sandbox restricts code from untrusted sources from taking actions that could possibly harm your system.

The security (or strength) of the sandbox heavily depends on the security features which are delivered by the Java Virtual Machine (JVM). The JVM ensures:

- Type-safe reference casting.
- Structured memory access (no pointer arithmetic).
- Automatic garbage collection (can't explicitly free allocated memory).
- Array bounds checking.
- Checking references for null.

Consequently, many drawbacks of languages like C, are not present in the Java programming language provided that the JVM works correctly.

5.5.2 Java Cryptographic Architecture

The *Java Cryptography Architecture* (JCA) [Mic02] is a framework for accessing and developing cryptographic functionality for the Java platform. The Java Cryptography Architecture (JCA) was designed around two principles:

1. Implementation independence and interoperability
2. Algorithm independence and extensibility

Algorithm independence is achieved by defining types of cryptographic *engines* (services), and defining classes that provide the functionality of these cryptographic engines. These classes are called *engine classes*, and examples are the `MessageDigest`, `Signature`, `KeyFactory`, and `KeyPairGenerator` classes.

Implementation independence in the JCA is achieved by using a *provider*-based architecture. The term *Cryptographic Service Provider* (short provider) refers to a package or set of packages that implement one or more cryptographic services. Examples of such services are digital signature algorithms and message digest algorithms. A program may simply request a particular type of object (such as a Signature object) implementing a particular service (such as the DSA signature algorithm) and get an implementation from one of the installed providers. If desired, a program may instead request an implementation from a specific provider.

The `SUN` provider for example includes:

- DSS
- A pseudo-random RNG according to IEEE 1363.
- A certificate path builder according to X.509
- A certificate store implementation according to LDAP v2.
- A certificate parser for X.509

An engine class defines a cryptographic service in an abstract fashion without a concrete implementation. A cryptographic service is always associated with a particular algorithm or type. It either provides cryptographic operations (like those for digital signatures or message digests), generates or supplies the cryptographic material (keys or parameters) required for cryptographic operations, or generates data objects (keystores or certificates) that encapsulate cryptographic keys (which can be used in a cryptographic operation) in a secure fashion. Examples for engine classes are:

- `MessageDigest`: used to calculate the message digest (hash) of specified data.
- `Signature`: used to sign data and verify digital signatures.
- `KeyPairGenerator`: used to generate a pair of public and private keys suitable for a specified algorithm.
- `KeyFactory`: used to convert opaque cryptographic keys of type Key into key specifications (transparent representations of the underlying key material), and vice versa.
- `CertificateFactory`: used to parse public key certificates and Certificate Revocation Lists (CRLs).
- `KeyStore`: used to create and manage a keystore. A keystore is a database of keys. Private keys in a keystore have a certificate chain associated with them, which authenticates the corresponding public key. A keystore also contains certificates from trusted entities.
- `AlgorithmParameters`: used to manage the parameters for a particular algorithm, including parameter encoding and decoding.
- `AlgorithmParameterGenerator`: used to generate a set of parameters suitable for a specified algorithm.
- `SecureRandom`: used to generate random or pseudo-random numbers.
- `CertPathBuilder`: used to build certificate chains (also known as certification paths).
- `CertPathValidator`: used to validate certificate chains.
- `CertStore`: used to retrieve Certificates and CRLs from a repository.

An engine class provides the interface to the functionality of a specific type of cryptographic service (independent of a particular cryptographic algorithm).

It defines Application Programming Interface (API) methods that allow applications to access the specific type of cryptographic service it provides. The actual implementations (from one or more providers) are those for specific algorithms.

The application interfaces supplied by an engine class are implemented in terms of a Service Provider Interface (SPI). That is, for each engine class, there is a corresponding abstract SPI class, which defines the SPI methods that cryptographic service providers must implement.

An instance of an engine class, the API object, encapsulates (as a private field) an instance of the corresponding SPI class, the SPI object. All API methods of an API object are declared final and their implementations invoke the corresponding SPI methods of the encapsulated SPI object. An instance of an engine class (and of its corresponding SPI class) is created by a call to the `getInstance` factory method of the engine class.

For each engine class in the API, a particular implementation is requested and instantiated by calling a factory method on the engine class. A factory method is a static method that returns an instance of a class.

The basic mechanism for obtaining an appropriate Hash function object, for example, is as follows: A user requests such an object by calling the `getInstance` method in the `MessageDigest` class, specifying the name of a signature algorithm (such as "MD5"), and, optionally, the name of the provider or the Provider class.

```
try {
  MessageDigest md5 = MessageDigest.getInstance("MD5","IAIK");
} catch (NoSuchProviderException ex) {
  System.out.println("Provider IAIK not found");
} catch (NoSuchAlgorithmException ex) {
  System.out.println("MD5 is not supported by IAIK");
}
```

In order to compute the hash value of some input data methods associated with the hash function object have to be called:

```
byte[] data_1 = {01,02,03,04,05,06,07,08,09};
byte[] data_2 = {09,08,07,06,05,04,03,02,01};

md.update(data_1);
MessageDigest firstPart = md.clone();
byte[] firstPartDigest = firstPart.digest();
md.update(data_2);
byte[] digest1 = md.digest();
```

In this example, two chunks of data are hashed. In order to compute the hash value of some data, the data needs to be put into the *MessageDigest* object. This is accomplished by calling the method $update(data)$. In this example, an intermediate hash value, which only hashes $data\_1$ is calculated. This is done by first cloning the $MessageDigest$ object and then applying the method $digest()$ onto the new object. Afterwards, more data is put into the (original) $MessageDigest$ object. In the last line, the hash value over both data chunks is calculated.

5.6 Summary

Attacks on implementations are mostly devastating for the security of a system. Implementation attacks can be either active or passive, both types require an attacker with some knowledge about security and implementations. Often, security breaches are due to insecure programming languages such as C. We have seen that Java might be a better language to implement security critical applications.

5.7 Review Exercises

1. What are active and passive implementation attacks, resp.? Give an example for such attacks.
2. What types of side-channels that have been exploited in attacks do you know?
3. What is a simple side-channel attack?
4. What is a differential side-channel attack?
5. Name some methods to insert faults in a cryptographic device.
6. What can be done to counteract fault attacks?
7. How does a buffer overflow attack typically work?
8. What does it mean to smash the stack?
9. What is the purpose of a sandbox?
10. What are the goals of the Java Cryptographic Architecture?

Chapter 6

Operating System Security

Your password must be at least 18770
characters and cannot repeat any
of your previous 30689 passwords.
Please type a different password.
Type a password that meets these
requirements in both text boxes.
— Microsoft Knowledge Base Article Q276304.

Nowadays, valuable information is typically stored on computers. Protecting this information from unauthorized usage is therefore becoming increasingly important. We have already discussed protective measures such as cryptography and threats such as network (in)security. However, we still miss another important component of each information system, which is the computer. Naturally, a computer is a piece of hardware which is operated by some software, the so-called operating system (OS). The OS's job is to manage all devices present in the PC and to provide the user with a simple interface (programs) to the hardware.

With the advent of multi-user systems, the first step in using a computer is to log into the computer. At this login the system tries to determine who the user is. This process is called *user authentication* or, more general *entity authentication*, and is the first of all security mechanisms incorporated in an operating system.

Based on the identity, the user is granted access to different services or resources. The management of user (and process) privileges is the second important security mechanism of an operating system.

Several more security mechanisms can be identified in different operating systems, such as encrypted file systems, (personal) firewalls, virtual machines, etc.

6.1 Authentication and Access Control

Entity authentication (or *identification*) is the process whereby one party is assured (through acquisition of corroborative evidence) of the identity of a second party involved in a protocol, and that the second has actually participated (i.e., is active at, or immediately prior to, the time the evidence is acquired)[MvOV97].

An entity authentication (or identification) protocol is a real-time process. It assures that the party being authenticated is operational at the time of protocol execution. For example, the party that is taking part needs to carry out some action since the start of the protocol execution. Authentication protocols are typically based on one (or several) of the three principles:

- Something known: like a password or a PIN.
- Something possessed: like a smart card or a password generator.
- Something inherent: like the fingerprint or voice.

The most commonly used identification principle in practice is based on something known. In particular, all modern operating systems use this method (with passwords) as standard authentication.

6.1.1 Password based Authentication

A *password* is typically a string of at least eight characters, which is associated with one user. The password serves as a shared secret between the user and the system.

Whenever a user attempt to access the system, he is required to state his identity (user id) and to reveal his password. The system matches the data delivered by the user against its own data. The demonstration of knowledge of this secret (by revealing the password itself) is accepted by the system as corroboration of the entity's identity.

Naturally, the system should not display the password while it is being entered. Sending a password in clear over an insecure line is also not being advised (but occurs in many standard applications as telnet, ftp, etc.).

Storing passwords A naive approach is to store user passwords cleartext in a *password file*. Naturally, the file should be read (and write) protected to secure it against ordinary users. When a user enters the password, the system compares it to the password stored in the password file corresponding to the user. An obvious drawback of this method is that there is no protection against privileged users (superusers, administrators). Furthermore, system backups including the password file are insecure as well.

A cryptographic solution to this problem are *encrypted password files*. In this approach, not the cleartext passwords are stored but the result of a one-way function of each user password is stored instead. To verify a password (entered by a user) the system computes the hash of the entered password and compares it to the stored entry of this user. Also in this approach, the file needs to be writing protected.

Guessing passwords If the password file is not the weakest part of the system, then guessing the password is the next logic approach of an attacker. An *exhaustive* search requires an attacker to try all possible sequences of admissible characters of a certain length until the correct one has been found. Such an attack, if it is on-line, can be thwarted by restricting the number of logins.

It is possible to improve the chance of success of an exhaustive search by searching the space in order of decreasing probability. Simply spoken, the attacker will attempt to try the most likely passwords first. Studies indicate that a large fraction of user-selected passwords is found in typical (intermediate) dictionaries of only 150.000 words, while even a large dictionary of 250.000 words represents only a tiny fraction of all possible passwords. Such passwords may be revealed by an attacker who tries all the entries in the dictionary. Attacks of this types are called *dictionary attacks*.

If password guessing attacks are performed off-line by using an encrypted password file a countermeasure can be to use *salted* password files. Each password, upon initial entry, is augmented with a t-bit random string called the salt before applying the one-way function. Both the hashed and the salt are recorded in the password file. This trick does not increase the difficulty of exhaustive search on any particular password because the password itself is unaffected by the salting. However, salting increases the complexity of a dictionary attack by requiring the dictionary to contain all 2^t variations of the hash of each trial password.

Weak vs. strong authentication A weakness of schemes using fixed, reusable passwords is the possibility that an attacker learns a user's password by observing somehow. For example, as it it is typed in, or if they are transmitted to somewhere in cleartext, or when they are stored in cleartext temporarily during system verification. An eavesdropping adversary can record this data, allowing subsequent impersonation of the legitimate user. Therefore we call such fixed password schemes as *weak* authentication schemes.

One-time passwords schemes, such as schemes where each user is given a hash, are schemes, which go towards strong authentication.

So called *challenge-response protocols* belong to the class of strong authentication schemes [MvOV97]. In this type of authentication mechanism, one entity (the claimant) proves its identity to another entity (the verifier) by demonstrating knowledge of a secret known to be associated with that entity, without revealing the secret itself to the verifier during the protocol. This is done by providing a response to a time-variant challenge, where the response depends on both the entity's secret and the challenge. The challenge is typically a number chosen by the verifier (randomly and secretly) at the beginning of the protocol.

6.2 Windows Security

In this section, we focus on security features, as they are present in MS Windows 2000 and its successors. Some of the discussed security features were present already in Windows NT in slightly different versions.

6.2.1 User Authentication

Windows 2000 and it successors (we simply call this family Windows from now on) provides a secure login with anti-spoofing mechanism. Spoofing refers to an attacker writing a program that displays a login prompt on screen and records the user name and password when they are entered by an innocent user. The attacker would attempt to install such a program and use it to gather *credentials* (*i.e.*authentication information such as username and password) of innocent users. Windows prevents such attacks by instructing users to hit CTRL-ALT-DELETE (known as the Secure Attention Sequence (SAS)) to log in. SAS is a special sequence always captured by the keyboard driver, which invokes a system program that starts the genuine login screen.

The component of Windows that handles authentication is called the authentication authority. If you login locally to a machine, the authentication authority it is the Local Security Authority (LSA); but if you login to a domain, it is the LSA of a domain controller (DC). A DC is a kind of central server of every Windows domain (network of computers). Besides its central role during login, it manages network resources and policies, and stores all user profiles. A Windows authentication process can use different authentication protocols: NTLM (NT LAN Manager), Kerberos, SSL (TLS), Distributed Password Authentication (DPA), etc. The default protocol is Kerberos. In Windows, there are four basic login types: interactive login, non-interactive login, batch, and service login. Interactive login (locally, or on the network) is the typical user login. We will focus on it in the following paragraph.

Winlogon An interactive authentication starts whenever a user initiates an SAS sequence. Subsequently, the Winlogon service, which is the component responsible for interactive logins, calls the GINA (Graphical Identification and Authentication) module. GINA is responsible for displaying the login interface, extracting the user's credentials, and passing them to the LSA. The LSA interacts with the local security database and the authentication packages and handles user authentication.

Authentication packages are software packages that implement different authentication protocols. Windows comes with two authentication packages: MSV1_0 and Kerberos. The available authentication package DLLs can be found in the registry. The Kerberos authentication package cannot handle local login requests on workstations. Kerberos requires the presence of a Key Distribution Center (KDC) service, which is only available on a Windows DC.

The authentication database stores the credentials needed during the authentication process. A Windows DC stores credentials in the Active Directory.

Summarizing, in an interactive local login, after the user presses CTRL-ALT-DELETE, Gina displays the login prompt. GINA also passes the username and password to the LSA, which chooses an authentication package and contacts the authentication database.

Kerberos The Kerberos protocol allows two communicating partners, Alice and Bob, to communicate privately with the help of a trusted third party (TTP) acting as a key distribution center. Kerberos is based on symmetric-key cryptography.

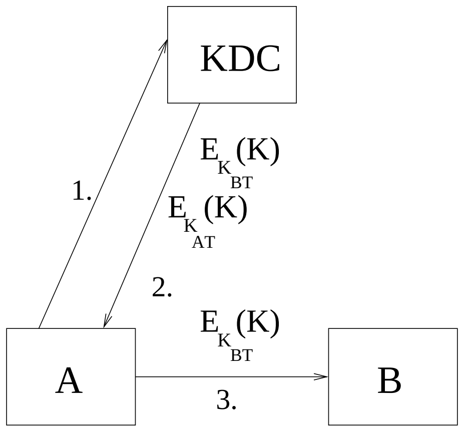


Figure 6.1: Key Management with Kerberos.

Figure 6.1 shows the basic architecture of Kerberos. In Figure 6.1, Alice attempts to communicate privately with Bob. Therefore, she needs to share a secret key with Bob. However, Alice and Bob do not possess a long-term shared secret. Consequently, Alice uses a KDC to establish a shared secret with Bob. The KDC is a trusted party and shares keys with both Alice K_{AT} and Bob K_{BT}. The protocol consists of several steps. Firstly, in Step 1, Alice contacts the KDC (she authenticates herself against the KDC) and requests a shared secret with Bob. The KDC verifies her identity and if authentication succeeds, Alice receives credentials to contact Bob (Step 2). These credentials consists of a shared secret K which is encrypted under the secret key that Alice shares with the KDC and of a so-called *ticket* which is encrypted under the secret key that Bob shares with the KDC. The ticket is a data structure that contains the secret key shared between Alice and Bob K and some additional information about Alice, the lifetime of the ticket, etc.

Integrating Kerberos authentication to Windows brings several advantages. Firstly, Kerberos is a rather scalable architecture. Secondly, many network resources should be protected from not-legitimate users. This requires authentication for such resources. By using Kerberos, so-called *single sign-on* can be supported; the user only needs to be authenticated once. The user gets during login tickets for resources and can further on access those network resources without needing to supply a password. Of course, in practice, the Kerberos protocol is more involved than the simple 3-step protocol depicted and discussed before. However, the basic principle of Kerberos is indeed that simple.

6.2.2 Access Control

Access control mechanisms mediate the access of users (processes) to files, processes, etc. A simple way of managing access rights are *access control lists* (ACLs). For each file (or resource) a list of users and their privileges is stored.

In Windows, every user (and group) is identified by a *secure ID* (SID), which is an (almost random) number that is intended to be unique worldwide. Furthermore, each process has a so-called *access token* that contains its SID and additional information such as privileges. The SID (the access control token) is mainly used for discretionary access control (DACL). DACL allows the owner of a file (or other object) to define who can use it and in what way.

Another concept which is used in Windows is the security descriptor. Every object has such a descriptor that defines who may perform which operations on it. A security descriptor consists of DACL and a SACL (system access control list). A SACL specifies which operations on the object are logged in the system-wide security event log.

Naturally, Kerberos is also used for access control. All applications which support Kerberos can therefore base access control on it.

6.2.3 Encrypted File System

Windows has an option to encrypt files; it can be invoked by marking certain directories as encrypted which causes all files in them to be encrypted, and new files moved to them or created in them to be encrypted as well.

The encryption is performed by driver called *encrypting file system* (EFS) which resides between the actual file system and the user process. When a user asks to perform this encryption, a random key is generated and used to encrypt a file block-wise. Currently, CBC-DES (128-bit key) is used as encryption algorithm. For each file, a separate key is chosen. These secret keys are stored encrypted on the harddisk. For key-encryption, a public-key algorithm is used. Of course, also for this algorithm, the private key needs to be stored confidentially somewhere. Consequently, this key is encrypted under a secret key, which is either stored on a tamper resistant device (such as a smart card) – this option is not implemented yet, or is derived from the user's password. The problem of where to store private or secret keys confidentially becomes apparent from this application.

6.2.4 Windows Attacks

We briefly sketch some prominent attacks on Windows clients (in particular on XP) in this section.

SMB Attack The *Service Message Block* (SMB) protocol is used for sharing files, printers and communication methods between (Windows) computers. SMB is a simple client/server protocol and typically runs over NETBIOS over TCP/IP. It is set up to run automatically under a default Windows installation. Remote clients can check for SMB by performing an ordinary port scan.

The weakness, which is exploited in SMB attacks, is located in a certain SMB command (`SMB_COM_TRANSACTION`). Two of the parameters used are the so-called `MAX DATA COUNT` field and the `MAX PARAM COUNT` field. They are supposed to limit the amount of data sent to the server. If these fields are set to zero, the implementation crashes. The program *smbnuke* uses this erratic behavior to crash XP systems.

The best way to avoid SMB attacks is to disable NETBIOS.

Help Center Attack The Windows Help Center (as it is implemented in XP) is more than a cute looking source of information. It allows a user to perform hardware and software tests, configure OS tools, and sends out requests for assistance. The management of this collection of tools is done by a large collection of web pages and XML files. One important file is called *uplddrvinfo.htm* and appears to be fundamental to processing information related to hardware and drivers. There is a particularly problematic code section in this file, which allows an attacker to delete any number of files on the attacked system:

```
var sFile = unescape( sThisURL.subString( sThisURL.indexOf( '?'
) + 1));
sFile = sFile.replace('file://', '').replace(/\&.*/,'');
var oFSO = new ActiveXObject( 'Scripting.FileSystemObject');
try{
oFSO.DeleteFile( sFile );}
```

The first line assigns the value of the parameter to a variable. The second line removes irrelevant data. In the third line, a new filesystem object is created. In the fourth and fifth line the file with the name given as parameter is deleted.

This piece of code can readily be exploited in an attack by using the *Help Center Protocol* (HCP). The file is loaded with a rather high level of permission.

This allows even a remote attacker to exploit this weakness. For example, it is sufficient to create a link to the file and tell it what to delete:

hcp://system/DFS/uplddrvinfo.htm?file://c: (to delete everything on C :).

To avoid this attack it is best to remove the questionable code from the file, or to remove the file itself.

6.3 Unix Security

Unix, and all Linux and other variations, use similar security mechanisms. In the following, we focus on the general approaches which are used for authentication and access control.

6.3.1 User Authentication

When a user logs in, the *login* program requests user name and password. The *login* program also hashes the password and matches it to the entry corresponding to the user. The one-way function which is used is computed as follows: each user password serves as the key to encrypt a known plaintext (64 zero-bits). This yields a one-way function of the key, since only the user (aside from the system, temporarily during password verification) knows the password. For the encryption algorithm, a minor modification of DES is used; variations may appear in different UNIX and Linux versions. A salting mechanism is incorporated as well. The *login* process starts as process which is SETUID root (for a discussion of access control read the following section). After the password has been successfully verified, *login* changes its UID (user id) and GID (group id) to the user's UID and GID, opens the keyboard for standard input and the screen for standard output and executes a shell, thus terminating itself.

6.3.2 Access Control

Each user has a unique *user id*. User can be organized into groups, which have group ids. File, processes and other resources are marked inherit the UID and the GID of their owners.

Each process carries a UID and a GID. In addition, each process also gets a set of permissions determined by the owner. The permissions specify what access the owner, group members and other users have. Access categories include read, write and execute. As there are three categories of users and three access categories, 9-bit numbers are used to represent access rights. For directories the execute bit makes no sense and is instead used to indicate whether a user may search the directory or not.

A special user in a Unix system is the *superuser* (or *root*). The superuser has all rights to all files (processes or other resources) in the system no matter of their access rights.

Special files corresponding to I/O devices have the same protection bits as ordinary files. Consider for example */dev/lp*, which is the default printer. It is certainly not advisable to allow everyone to directly access it because it can only print one job at a time anyway. A more sensible approach would be to only allow a distinct user, such as root or the printer daemon, to directly access it. However, only the superuser or the printer daemon could print in this approach but nobody else. Therefore, another protection bit, the SETUID bit was introduced. When for an executable the SETUID bit is set, the resulting process will run with the owner's permission and not with the permission of the user launching it.

In case of the sketched printer scenario the program that accesses the printer should be owned by the printer daemon but with the SETUID bit set. Then, any user can execute it, but it runs with the permissions of the printer daemon and thus, can indeed print. This works because each process has a real UID that identifies the owner of the process, an effective UID that is used in most access control decisions, and the saved UID that stores a previous user ID so that it can be restored later. When a process is created, it inherits the three user IDs from its parent process. When a process executes a new file, it keeps its three user IDs unless the SETUID bit of the new file is set, in which case the effective UID and saved UID are assigned the user ID of the owner of the new file.

Since access control is based on the effective user ID, a process gains privilege by assigning a privileged user ID to its effective UID, and drops privilege by removing the privileged user ID from its effective UID. Privilege may be dropped either temporarily or permanently.

The SETUID mechanism is at first sight an elegant way to facilitate access control. However, badly written SETUID-programs allow gaining root privileges easily. It is advisable to unset the SETUID bit for all programs that do not absolutely require it.

6.3.3 Unix Attacks

The SysAdmin, Audit, Network, Security (SANS) institute (`http://www.sans.org`) compiles a list of the twenty most critical internet security vulnerabilities once a year. The top ten Unix vulnerabilities at the time of writing these notes are:

- BIND Domain Name System
- Remote Procedure Calls (RPC)
- Apache Web Server
- General UNIX Authentication Accounts with No Passwords or Weak Passwords

- Clear Text Services
- Sendmail
- Simple Network Management Protocol (SNMP)
- Secure Shell (SSH)
- Misconfiguration of Enterprise Services NIS/NFS
- Open Secure Sockets Layer (SSL)

In the following paragraphs, we quote parts of the information given on the SANS website for the four most prominent vulnerabilities. Thereafter, we briefly go through other well-known UNIX problems.

BIND problems The Berkeley Internet Name Domain (BIND) package is the most widely used implementation of the Domain Name Service (DNS), a critical system that allows the conversion of hostnames (e.g. www.sans.org) into the registered IP address. The critical nature of BIND has made it a frequent target, especially in Denial of Service (DoS) attacks. Whilst BIND developers have historically been quick to repair vulnerabilities, an inordinate number of outdated, misconfigured and/or vulnerable servers remain in place.

Among the most recently discovered BIND weaknesses was a denial of service discussed in CERT Advisory CA-2002-15 (`http://www.cert.org/advisories/CA-2002-15.html`). In this case, an attacker can send specific DNS packets to force an internal consistency check which itself is vulnerable and will cause the BIND daemon to shut down. Another was a buffer overflow attack, discussed in CERT Advisory CA-2002-19 (`http://www.cert.org/advisories/CA-2002-19.html`), in which an attacker utilizes vulnerable implementations of the DNS resolver libraries. By sending malicious DNS responses, the attacker can explore this vulnerability and execute arbitrary code or even cause a denial of service.

A further risk is posed by a vulnerable BIND server, which may be compromised and used as a repository for illicit material without the administrator's knowledge or attacks which use the server as a platform for further malicious activity.

RPC problems Remote procedure calls (RPCs) allow programs on one computer to execute procedures on a second computer by passing data and retrieving the results. RPC is therefore widely used for many distributed network services. However there are numerous flaws in RPC which are being actively exploited. Many RPC services execute with elevated privileges that can provide an attacker unauthorized remote root access to vulnerable systems.

There is compelling evidence that the majority of the distributed denial of service attacks launched during 1999 and early 2000 were executed by systems that had been victimized through these RPC vulnerabilities. Recently, an MS Windows DCOM Remote Procedure Call vulnerability has played a role in one of the most significant worm propagation events to this date.

RPC services are typically exploited through buffer overflow attacks which are successful because the RPC programs do not perform sufficient error checking or input validation. Buffer overflow vulnerabilities allow an attacker to send unexpected data (often in the form of malicious code) into the program memory space. Due to poor error checking and input validation, the data overwrite key memory locations that are in line to be executed by the processor. In a successful overflow attack, this malicious code is then executed by the operating system. Since many RPC services execute with elevated privileges, successful exploitation of these vulnerabilities can provide unauthorized remote root access to the system.

Apache problems Apache has historically been the most popular web server on the Internet. In comparison to the Microsoft Internet Information Server, Apache may have a cleaner record in regards to security, but it still has its fair share of vulnerabilities.

In addition to exploits in Apaches core and modules (CA-2002-27, CA-2002-17) (`http://www.cert.org/advisories/CA-2002-27.html`,`http://www.cert.org/advisories/CA-2002-17.html`), SQL, databases, CGI, PHP vulnerabilities are all potentially exposed through the web server.

If left unsecured, vulnerabilities in the Apache web server implementation and associated components can result in denial of service, information disclosure, web site defacement, remote root access, or countless other unfavorable results.

Password problems In general, a compromised password is an opportunity to explore a system virtually undetected. An attacker in possession of a valid user password would have complete access to any resources available to that user, and would be significantly closer to being able to access other accounts, nearby machines, and perhaps even obtain root level access on this system. Despite this threat, user and administrator level accounts with poor or non-existent passwords are still very common. As well, organizations with a well-developed and enforced password policy are still uncommon.

The most common password vulnerabilities are:

- user accounts that have weak or nonexistent passwords;
- users accounts with widely known or openly displayed passwords;
- system or software created administrative level accounts with widely known, weak, or nonexistent passwords; and

- weak or well known password hashing algorithms and/or user password hashes that are stored with weak security and are visible to anyone.

The best defense against all of these vulnerabilities is a well developed password policy that includes: detailed instructions for users to create strong passwords; explicit rules for users to ensure their passwords remain secure; a process in place for IT staff to promptly replace weak/insecure/default or widely known passwords and to promptly lock down inactive or close down unused accounts; and a proactive and regular process of checking all passwords for strength and complexity.

Other problems The *ping of death* can be regarded as the most famous ancestor of network denial-of-service attacks. The ping of death is an oversized ping packet that used to crash very old Unix TCP stack implementations (a buffer overflow occurs thereby in the stack). In the *land attack*, a packet with identical source and destination address is used. It used to crash many TCP/IP stacks (however, the majority of them were found in Windows). Distributed denial-of-service attacks have become popular in 1999. Several tools to perform such attacks can be found on the web. The most famous ones are *Stacheldraht*, which allows to send TCP ACK floods and allows ICMP flooding. TFN and TFN2K are similar tools, which also allow different types of floods.

6.4 Improving Security

There are several approaches possible to deal with security problems. The most natural one is of course to avoid programming flaws that most often lead to exploits. However, it seems to be infeasible at the moment to produce flawless software (or hardware). Therefore it seems to be wise to follow a second approach in parallel, which consists of some mechanism to update (exchange) programs that have turned out to be problematic. A third reasonable approach is to try to limit the damage a flawed program can make.

In the following paragraphs we discuss several methods that follow one of the mentioned approaches.

6.4.1 Windows Update

Especially for Windows operating systems it has become clear that due to their complexity flawless code cannot be expected. Therefore, Windows comes with a nice update mechanism that allows even uneducated users to obtain the relevant (security) updates in time.

6.4.2 Sandboxing

Sandboxing is a popular technique for creating restricted execution environments, which can be used for running non-trusted programs. A sandbox limits, or reduces, the level of access its applications have.

Virtual machines (VMs) can be used to provide runtime sandboxes, which are helpful in enhancing users' level of confidence in the associated computing platform. The Java virtual machine is an example, although it can only be used to run Java programs. For more general sandboxing systems, the sandboxing layer could be implemented within the operating system kernel.

6.4.3 Code Signing

How can users trust code that is published on the internet or that they get from somewhere else? Additionally, even if users can trust the source of information, there is no guarantee that the code hasn't been altered while being downloaded.

This problem can be solved by using cryptography. The authenticity and integrity of code can be guaranteed by using standard cryptographic methods such as hash functions, and digital signatures. In particular, by signing a piece of code, both the authenticity and the integrity of this piece of code are protected. The user can verify the authenticity and integrity by verifying the digital signature. Of course, the major issue here is the trust the user can have in the underlying PKI. If the issuer of the digital certificates is not trustworthy, the whole system is rendered useless.

6.4.4 Trusted Computing

The trusted computing base is the set of all hardware, software and procedural components in a computer that enforce a security policy. This means that in order to break security, an attacker must subvert one or more of them.

Trusted computing is often associated only with the activities of the Trusted Computing Group (TCG) which has members such as Microsoft, Intel, IBM, HP and AMD. This group is working on specifications that should ultimately lead to a trustworthy computer. Although the goal of this project is certainly a desirable one, the activities of this group are watched with suspicion. This is mainly due to the fact that trustworthy systems are supposed to counteract attacks against user as well as attacks against vendors. For example, a trustworthy computer should block playing a copied DVD.

6.4.5 Personal Firewalls

Firewalls have been security measures that were installed mainly on a network server in the past. However, modern operating systems such as Windows XP do incorporate so-called personal firewalls even on their clients. In principle, this should be an advantage because every user can limit the incoming data from the internet. However, personal firewalls are software and are therefore often flawed themselves.

6.4.6 Data Execution Prevention

Data execution prevention (DEP) is a set of hardware and software technologies that perform additional checks on memory to help protect against malicious code exploits. In Windows XP SP2, DEP is enforced by both hardware and software. Both Intel and AMD have processors, which support data execution prevention in hardware. In this way, buffer overflow attacks might be successfully counteracted.

6.5 Summary

Operating systems are at the core of every computer system. Insecure operating systems consequently lead to insecure computer systems. Therefore, it is mandatory to understand the security features and the most common insecurity features of state-of-the-art operating systems. We have discussed the logon and authentication features of Windows and UNIX. We have also looked into attacks on Windows and UNIX. Furthermore, we have investigated which mechanisms have been introduced to improve security in both systems.

6.6 Review Exercises

1. What is entity authentication or identification?
2. What is meant by password based authentication?
3. What is a dictionary attack?
4. What is the challenge-response principle?
5. How does user authentication in Windows (2000,XP, etc.) typically work?
6. What is the purpose of Kerberos?
7. What is an ACL, and what is it used for?
8. How does the EFS (Windows' encrypted file system) work?
9. How does authentication in Unix systems typically work?
10. Name some attempts to improve operating security.

Chapter 7

Privacy

Government is like a baby.
An alimentary canal with a big
appetite at one end and no sense
of responsibility at the other.
— Ronald Reagan

The Internet has become one of the major platforms to exchange information. The term “information” is rather unspecific. For example, information can be personal data, personal opinions, or other information that is not intended for the broad public. However, all communication over the Internet can be potentially logged, archived and searched later on. For example, the WayBack Machine `http://www.archive.org/web/web.php` has archived 86 billion web pages from 1996 to now. Such archive tools are not only a source of interesting information. They can be abused and information acquired with them can be used against ordinary users.

So-called *privacy enhancing technologies* (PET) have been developed to provide some protection for users. These PET allow users to keep their identities hidden when for example, sending email, posting to newsgroups, or browsing the web. In this chapter we first discuss the different meanings of “privacy”. Then, we sketch basic mechanisms for PET. And last, we survey what privacy enhancing technologies are available for the Internet today.

7.1 Definitions of Privacy

The definition of privacy, which is used most often, has been given by Alan Westin:

Privacy is the claim of individuals, groups and institutions to determine for themselves, when, how and to what extent information about them is communicated to others.

Privacy however, has several more aspects. For example, there is *territorial privacy*, which refers to protecting the close physical area surrounding a person; or the *privacy of the person*, which refers to protecting a person against undue interferences, such as physical searches or information violating his moral sense; and there is *informational privacy*, which refers to controlling whether and how personal data can be gathered, stored, processed or selectively disseminated.

Data protection laws of mostly western states as well as international privacy guidelines or directives (such as the EU-Directive 95/46/EC on Data Protection), require basic privacy principles to be guaranteed when personal data are collected or processed:

Purpose specification and purpose binding: personal data must be obtained for specified and legitimate purposes and should not be used for other purposes (see also Art. 6 EU Directive);

Necessity of data collection and processing: the collection and processing of personal data shall only be allowed, if it is necessary for the tasks falling within the responsibility of the data processing agency (see also Art. 7 EU Directive);

Transparency: The data subject's right to information, notification, objection and the right to correction, erasure or blocking of incorrect or illegally stored data (see Art. 10–12, 14 EU Directive);

Requirement of Security Mechanisms Requirement of adequate technical and organizational security mechanisms to guarantee the confidentiality, integrity, and availability of personal data (see Art. 17 EU-Directive).

7.2 Privacy Enhancing Technologies

PETs have been defined as a coherent system of measures that protects privacy by eliminating or reducing personal data or by preventing unnecessary and/or undesired processing of personal data; all without losing the functionality of the data system. (Borking 2001)

Or, to bring it to the point, PETs are technologies to protect individuals and communities from casual surveillance and disruption. Several techniques, which can be used to achieve this goal have emerged in the last years. The basic technique, which underlying idea is frequently used in different applications, is the so-called mix network, and has been introduced already in 1981 by David Chaum.

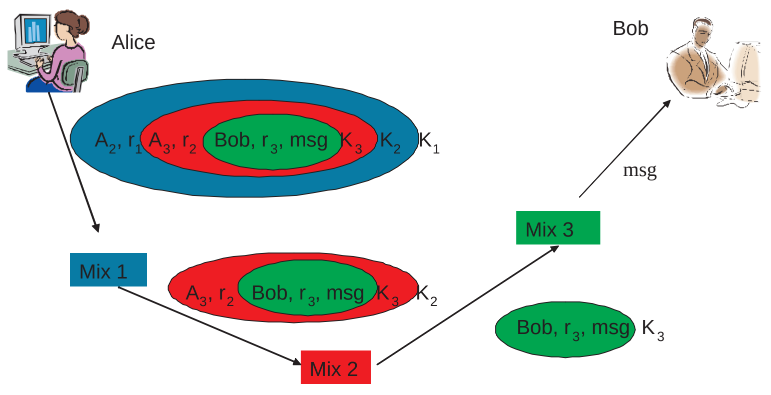


Figure 7.1: Communication from Alice to Bob with a *mix network* that consists of three mixes.

7.2.1 Mix Network

In his pioneering article from 1981 "Untraceable Electronic Mail, Return addresses, and Digital Pseudonyms", David Chaum proposed so-called mix to provide anonymity for electronic mail. The purpose of a mix is to hide the correspondences between the items in its input and those in its output. The order of arrival is hidden by outputting the uniformly sized items in lexicographically ordered batches.

To ensure that the mix server cannot read the contents of incoming emails they need to be encrypted. Therefore, Alice prepares a message *msg* for delivery to Bob by sealing it with the addressee's public key K_B, appending the address of Bob, and then sealing the result with the mix's public key K_i. The left-hand side of the following expression denotes this item which is input to the mix:

$$K_i(r_i, K_B(r_j, msg), Bob) \longrightarrow K_B(r_j, M), Bob. \tag{7.1}$$

In Figure 7.1, this idea is illustrated based on an example that involves two parties Alice and Bob, and three mixes. Alice sends a private (encrypted) message *msg* to Bob. Therefore, she first encrypts her message, together with a random number r_3 under the public key of the last mix (Mix 3) involved: $(r_3, msg)_{K_3}$. This message and the address of the third mix is encrypted under the public key of the second mix: $(r_2, (r_3, msg)_{K_3}, A_3)_{K_2}$. To close the path between Alice and Bob, this message and the address of the second mix is lastly encrypted under the public key of the first mix: $((r_1, (r_2, (r_3, msg)_{K_3}, A_3)_{K_2}, A_2)_{K_1}, A_1$. As can be seen from the last equation, also the address of the first mix needs to be added. When the message is sent through the mix network, encryption layer after encryption layer is removed by the mixes. The encrypted message *msg* reaches Bob although no mix is able to read the email. No outsider, and also not Alice or Bob, can trace which path the message took through the mix. This is due to the fact that the mixes batch messages and the addition of random numbers.

In order to be able to answer to Alice's message (without knowing Alice's identity), Bob needs some kind of untraceable return address from Alice. Such an untraceable return address can be formed again as encrypted string consisting of Alice's real address: $(r_i, Alice)_{K_1}$. Alice can send this return address with the message to Bob. The following sketches how the last mix in the row can use the untraceable return address to deliver the reply to Alice:

$$K_1(r_1, Alice), K_A(r_0, msg) \longrightarrow Alice, r_1(K_A(r_0, msg)). \tag{7.2}$$

This mix uses the string of bits r_1 that it finds after decrypting the address part $K_1(r_1, Alice)$ as a key to re-encrypt the message part $K_A(r_0, msg)$. Only Alice can decrypt the resulting output because Alice created both r_1 and K_A.

7.2.2 Crowds

An approach for privacy on the internet is to use so-called *crowds* [RR98]. In a crowd, users can hide their actions within the actions of many other users. In order to execute web transactions in this model, a user joins a crowd of other users first. Then, the user's request to a web server is first sent to a random member of the crowd. That member can either submit the request directly to the destination server or forward it to another randomly chosen member. In the latter case, the next member chooses to submit or forward independently from all the previous members. When the request is eventually submitted to the destination server, it is submitted by a random member. Therefore, it prevents the destination server from identifying the true initiator of the request. Even crowd members cannot identify the initiator of the request, since the initiator is indistinguishable from a member that simply forwards a request from another. Of course, crowds cannot protect a user's anonymity if the content of her web transactions reveals her identity to the web server.

In a crowd, every user is represented by a process on her computer called a *jondo* (pronounced "John Doe"). When the jondo is started, it contacts a server called the *blender* to request admittance to the crowd. If admitted, the blender reports to this jondo the current membership of the crowd and information that enables this jondo to participate in the crowd. The user selects the jondo as her web proxy; any request coming from the browser is sent directly to the jondo. As explained in the previous paragraph, for every user request from the browser, the jondo creates a random path of jondos to the intended web server: the jondo picks a random jondo from the crowd (possibly itself), and forwards the request to it. When this jondo receives the request, it forwards the request to another jondo with probability $p < 1/2$. If the result is to forward, then the jondo selects a random jondo and forwards the request to it, and otherwise the jondo submits the request to the end server.

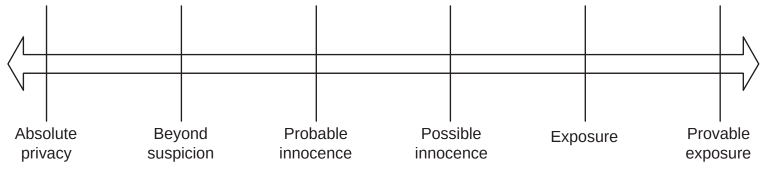


Figure 7.2: Degrees of Anonymity

7.2.3 Degrees of Anonymity

In any communication there are two parties: the sender and the receiver. *Sender anonymity* means that the identity of the party who sent a message is hidden, while its receiver (and the message itself) might not be. *Receiver anonymity* means that the identity of the receiver is hidden, but the message and the sender might not be. *Unlinkability* of sender and receiver means that though the sender and receiver can each be identified as participating in some communication, they cannot be identified as communicating with each other.

In real life, we can identify several degrees of anonymity (or exposure) as pointed out in Figure 7.2. We describe the degrees in the case of sender anonymity (the discussion is similar for the other two cases) according to [RR98].

Absolute privacy : absolute sender privacy that an attacker can in no way distinguish the situations in which a potential sender actually sent messages and those in which it did not.

Beyond suspicion : A sender's anonymity is beyond suspicion if though the attacker can see evidence of a sent message, the sender appears no more likely to be the originator of that message than any other potential sender in the system.

Probable innocence : A sender is probably innocent if, from the attacker's point of view, the sender appears no more likely to be the originator than to not be the originator. This is weaker than beyond suspicion in that the attacker may have reason to expect that the sender is more likely to be responsible than any other potential sender, but it still appears at least as likely that the sender is not responsible.

Possible innocence : A sender is possibly innocent if, from the attacker's point of view, there is a nontrivial probability that the real sender is someone else.

Provably exposed : the identity of a sender is provably exposed if the attacker cannot only identify the sender of a message, but can also prove the identity of the sender to others.

Crowds provide sender anonymity (i.e. probable innocence)against collaborating crowd members. In a group of collaborating mix servers, mixes do not provide sender anonymity but do ensure sender and receiver unlinkability. Mixes provide sender and receiver unlinkability against a global eavesdropper, whereas crowds do not provide anonymity against global eavesdroppers.

7.3 PET for the Internet

The discussion in this section follows closely the article by Ian Goldberg "Privacy-Enhancing Technologies for the Internet, II: Five Years Later" [Gol03].

7.3.1 Anonymous Remailers

Anonymous remailers are also known as "strip-headers-and-resend" remailers or type-0 remailers. A well known example was anon.penet.fi. The penet remailer even allowed for replies to anonymous posts. When a user sent her first message through the system, she was assigned a fake address at the anon.penet.fi domain as pseudonym. The headers on original emails would then be removed and replaced by the fake header. Replies to the fake address would go to anon.penet.fi where the real address would be looked up and the reply message would be forwarded back to the sender.

Due to legal pressure, the operator of this system was forced to give away the real user addresses. As a consequence, the service was shut down.

7.3.2 Cypherpunk Remailers

"Cypherpunk-style" remailers (or type-1 remailers) were used after the disaster with anon.penet.fi. In a type-1 remailer system, a user sends his message not via a single remailer, as with type-0. Instead, the user selects a chain of remailers and arranges that her message be successively delivered to each remailer in the chain before finally arriving at the mail's intended destination. Type-1 remailers also supported PGP encryption. Each remailer in the chain could only see the address of the next remailer, but not the ones further down, or that of the final recipient. Only the last remailer in the chain (called the exit node) could see the address of the recipient.

7.3.3 Mixmaster Remailers

Type-2, or "Mixmaster" remailers were developed because of security problems with type-1 remailers. Mixmaster remailers always use chaining and encryption, and break each message into a number of fixed-size packets, which are transmitted separately through the Mixmaster chain. The exit node recombines the pieces, and sends the result to the intended recipient.

7.3.4 Eternity Service

The eternity service was proposed by Ross Anderson and was later implemented in a simpler form by Adam Back as “Usenet Eternity”. This service promised the ability to publish documents that were uncensorable. In the implementation of Back, the distributed nature of Usenet was used to provide the redundancy and resiliency required to defend against attempts to censor information.

7.3.5 PipeNet and Onion Routing

Wei Dai proposal for “PipeNet”: a service similar to remailer networks, but designed to provide anonymity protection for real-time communication. The additional difficulties imposed by the real-time requirement required significant additions in complexity over the remailer network. Unfortunately, PipeNet was never developed past the initial design stages. Onion Routing was another project that was starting to get deployed in 1997, and which was attempting to accomplish similar goals as PipeNet.

7.3.6 Free Haven

The aforementioned Eternity service was a first attempt provide a cencorship free zone on the internet. More recently, projects such as Free Haven, FreeNet and Publius aimed for similar goals. With Publius, documents are encrypted and replicated across many servers. The decryption keys are split using a secret-sharing scheme +and distributed to the servers. A special URL is constructed that contains the information to retrieve the encrypted document, find the shares of the key, reconstruct the decryption key, and decrypt the document. Publius cryptographically protects documents from modification, and the distributed nature attempts to ensure long-term availability. In addition, the encrypted nature of the documents provides for deniability, making it less likely that the operators of the Publius servers would be held responsible for providing information they have no way to read. With Free Haven, the aim is also to provide content in such a manner that adversaries would find it difficult to remove. Free Haven also provided better anonymity features to publishers than did Publius.

7.3.7 Anonymizer

So far, almost every commercial privacy technology venture has failed, with `Anonymizer.com` being a notable exception. Anonymizer.com offers services including email and newsgroup access, as well as dial-up Internet access. Compared to other infrastructure-heavy attempts, Anonymizer.com has a relatively simple architecture, at the expense of protecting against a weaker threat model. But it seems that that weaker threat model is sufficient for most consumers.

7.4 Summary

The Internet has become more than just a place to exchange scientific information or to do business. Therefore, privacy concern have arisen and the need for privacy enhancing technologies is there. Several of such techniques have been developed and deployed over the last years. Some of them had to be shut down but several have evolved and are still in place.

7.5 Review Exercises

1. Name three aspect of privacy.
2. What are the four basic principles that EU Directive on Data Protection is based on?
3. What is the purpose of a mix network?
4. Illustrate the basic idea of a mix network.
5. What is a crowd?
6. What is the difference between anonymity and unlinkability?
7. Name the five degrees of anonymity.
8. What types of remailers do you know?

Bibliography

[AARR03] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM Side-channel(s). In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2535 of *Lecture Notes in Computer Science*, pages 29–45. Springer, 2003.

[And01] Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 2001. ISBN 0-471-38922-6.

[BDL00] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of checking cryptographic protocols for faults. In *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233, pages 37–51. Springer, 2000.

[DH76] Whitfield Diffie and Martin Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[DKL$^+$98] Jean-François Dhem, François Koeune, Philippe-Alexandre Leroux, Patrick Mestré, Jean-Jacques Quisquater, and Jean-Louis Willems. A Practical Implementation of the Timing Attack. In Jean-Jacques Quisquater and Bruce Schneier, editors, *Smart Card Research and Applications, This International Conference, CARDIS '98, Louvain-la-Neuve, Belgium, September 14-16, 1998, Proceedings*, number 1820 in Lecture Notes in Computer Science, pages 167–182. Springer, 1998. Available online at `http://www.dice.ucl.ac.be/crypto/techreports.html`.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Information Security and Cryptography. Springer, 2002. ISBN 3-540-42580-2.

[Gam85] Taher El Gamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1985.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic Analysis: Concrete Results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.

[Gol03] Ian Goldberg. Privacy-Enhancing Technologies for the Internet, II: Five Years Later. In *PET 2002*, volume 2482, pages 1–512. Springer, 2003.

[HPFS02] R. Housley, W. Polk, W. Ford, and D. Solo. RFC 3280: Internet X.509 Public Key Inrastructure: Certificate and Certificate Revocation List (CRL) Profile, 2002. Available online at `http://www.ietf.org/rfc/rfc3280.txt`.

[Int99] International Organisation for Standardization (ISO). Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher, 1999.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[Kob94] Neal Koblitz. *A Course in Number Theory and Cryptography.* Springer-Verlag, 1994. ISBN 0-387-94293-9.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996*, number 1109 in Lecture Notes in Computer Science, pages 104–113. Springer, 1996.

[MDS99] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Power Analysis Attacks of Modular Exponentiation in Smartcards. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 144–157. Springer, 1999.

[Mic02] Sun Microsystems. Java Cryptography Architecture, API Specification & Reference, 2002.

[MvOV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997. Available online at `http://www.cacr.math.uwaterloo.ca/hac/`.

[Nat80] National Institute of Standards and Technology (NIST). FIPS-81: DES Modes of Operation, 1980. Available online at `http://www.itl.nist.gov/fipspubs/`.

[Nat97] National Institute of Standards and Technology (NIST). FIPS-196: Entity Authentication Using Public Key Cryptography, 1997. Available online at `http://www.itl.nist.gov/fipspubs/`.

[Nat99] National Institute of Standards and Technology (NIST). FIPS-46-3: Data Encryption Standard, October 1999. Available online at `http://www.itl.nist.gov/fipspubs/`.

[Nat00] National Institute of Standards and Technology (NIST). FIPS-186-2: Digital Signature Standard (DSS), January 2000. Available online at `http://www.itl.nist.gov/fipspubs/`.

[Nat01] National Institute of Standards and Technology (NIST). Special Publication 800-38A 2001 ED, Recommendation for Block Cipher Modes of Operation - Methods and Techniques, December 2001. Available online at `http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf`.

[Nat02] National Institute of Standards and Technology (NIST). FIPS-198: The Keyed-Hash Message Authentication Code, March 2002. Available online at `http://www.itl.nist.gov/fipspubs/`.

[Nat03] National Institute of Standards and Technology (NIST). FIPS 180-2: Secure Hash Standard, 2003. Available online at `http://www.itl.nist.gov/fipspubs/`.

[Nat07] National Institute of Standards and Technology (NIST). Cryptographic Hash Project, 2007. `http://www.nist.gov/hash-competition`.

[Nat09] National Institute of Standards and Technology (NIST). FIPS-186-3: Digital Signature Standard (DSS), June 2009. Available online at `http://www.itl.nist.gov/fipspubs/`.

[One96] Aleph One. Smashing the Stack for Fun and Profit, 1996. Available online at `http://www.insecure.org/smashstack.txt`.

[PC04] Cyrus Peikari and Anton Chuvakin. *Security Warrior*. O'Reilly, 2004.

[QS01] Jean-Jacques Quisquater and David Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards. In Isabelle Attali and Thomas P. Jensen, editors, *Smart Card Programming and Security, International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Proceedings*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2001.

[RR98] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, November 1998.

[RSA78] Ron Rivest, Adi Shamir, and Leonard Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[Sch96] Bruce Schneier. *Applied Cryptography - Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., 1996. ISBN 0-471-11709-9.

[Sta03] William Stallings. *Cryptography and Network Security, Principles and Practices*. Pearson Education, 2003.

[WHK97] M. Wahl, T. Howes, and S. Kille. RFC 2251: Lightweight Directory Access Protocol (v3), 1997. Available online at `http://www.ietf.org/rfc/rfc2251.txt`.

Index