Authentication Course IMT 4721

Dr.ir. Patrick A.H. Bours
patrick.bours@hig.no
phone: 611 35250

version 1.0.0

Gjøvik University College, NISlab

September 27, 2006

Contents

Revision History

This section has been added to aid the students. The version at the start of the Authentication Course is 0.1 in which some of the chapters have not been written and some other chapters still need to be revised. The numbering convention is as follows. The number will be denoted by **0.X.Y** where **X** denotes the major revisions and **Y** denotes the minor revisions. A minor change is for example only a revision of some or more chapters, or some small typing error corrections. A major change is when a new chapter or a section within a chapter is added.

RevNr	**Date**	**Description**
0.1.0	23-08-06	Starting document with Chapters 1-4 already revised.
0.2.0	05-09-06	Revision of Chapters 5 and 7-10, addition of Chapter 6, and an spell check of the entire document.
0.2.1	19-09-06	Inclusion of pictures in Chapters 6 and 8 and correction of small typos.
1.0.0	27-09-06	Revision of Chapters 10 and 11. Final version for Authentication Course of fall 2006.

Chapter 1

General Introduction

Authentication is used in many places today. Well known are passwords to get access to computers, PIN codes to get money from an ATM machine and passports used at border control. The last is an example of "old fashioned" human authentication. Human authentication is used very much, think of identifying friends and family by their faces or the way they walk, recognising people on the phone by their voice, recognising a sender of an email by his email address, or even recognising a blind date by the red rose she carries. Notice that we use both the terms identification and authentication. The meaning of these two terms is almost the same and they will be used for the same purpose by many authors. But there is a small but very distinct difference between the two. We give our definitions of the two terms below:

Definition 1.1 Authentication:
By authentication we mean verifying a claimed identity.

Definition 1.2 Identification:
By identification we mean establishing an identity.

The way we recognise a friend by his[1] face is a form of identification because he did not provide his identity first. An example of authentication is the passport control at an airport. Someone first presents his identity in the form of a passport. The identity is verified by comparing the picture in the passport with the face of the person presenting the passport.

When using the term authentication we imply that a user provides a system with his identity and prove of correctness of this identity. When we use the term identification we imply that a user provides information to a system from which the system can determine the identity of the user.

[1]Everywhere in this document where he/him/his is used you can also read she/her/her.

We also say that authentication is a 1-on-1 verification of an identity and identification is 1-on-N.

For the remainder of this document and the authentication course the focus will be on authentication. In some cases we will still turn to identification, either to clarify things or to describe specific differences between authentication and identification. We will focus on automated authentication, i.e. authentication performed by machines, in contrast with human authentication.

In both authentication and identification users must first enroll in the system. During enrollment the identity of the user is stored in a database together with information that he must provide for authentication or identification purposes. This information is called the authentication data. During the authentication process the provided identity is looked up in the database and the provided authentication data is checked against the stored authentication data. During identification the presented authentication data is compared to (a subset of) the complete database of stored authentication data.

When performing authentication various parties are involved. The first and most important party is the one trying to authenticate himself, called the authenticator. We sometimes refer to “the authenticator” as “the user”. The second, and equally important party is the one being authenticated to, i.e. the one verifying the claimed identity of the authenticator. This party is called the verifier. Finally there can also a a party trying to attack and break the authentication process. This party is called the attacker. The attacker can do his job in various ways. He can be passively listening to all communication in an authentication process or he can actively participate in it. In the last case he can for example try to impersonate a specific person and start an authentication process with the verifier. He can also wait until a genuine authentication process is started and tamper with the messages that are send back and forth to see if he can find out information which should be kept hidden from him.

Chapter 2

Authentication Factors

There are various ways authentication can be achieved. All authentication factors can be classified as being a member of 1 out of 3 classes, being:

1. Something you *know*, e.g. a password.
2. Something you *have*, e.g. a token.
3. Something you *are*, e.g. a biometric property.

Below we will describe the three classes in more detail.

2.1 Know

The oldest and best known way of identifying oneself is proving knowledge of some secret. In ancient times spoken passwords were used among friends[1]. The automated analogy of this is the password used mainly in computer systems to get access to a computer or to specific data on that computer. Another examples of this authentication factor is a PIN codes used to get money out of an ATM machine.

For a long time this way of authentication has been the only one in use. It is easy and cheap to implement and in most cases also very fast. The weakest link in the authentication process is often the authenticator himself. Users often have to use a lot of passwords and PIN codes for various applications. It is for a user very convenient to have the same password for many applications. Also easy to remember passwords are used, like the names of the children, wife or dog, the birthdays or another important date, or a combination of a

[1] *Then Ali Baba climbed down and went to the door concealed among the bushes, and said, "Open, Sesame!" and it flew open.* from Ali Baba and the Forty Thieves

few of these. The disadvantage of these easy to remember passwords is that an attacker can also easily guess it, either by hand or in an automated way. If a user is forced to use multiple passwords or difficult to guess passwords there is always a risk that he writes them down and leaves them in an easy accessible place. For more information on password security see Chapter 4.

Beside guessing a secret another attack could be stealing it. This can be done secretly, where the user doesn't know that he has been robbed, or more open where it is clear for a user that someone else possesses his password or PIN code too. For example, a hidden camera can record the password or PIN code when it is entered. Also a modified keyboard, monitoring all the keystrokes, can be used to get hold of the password of an authenticator.

Another major disadvantage of using passwords is the high handling costs. Users tend to forget passwords (however easy they are) and the costs of restoring passwords is very high.

2.2 Have

In case the authentication factor is have, the authenticator possesses a unique piece of hardware that can be matched to his identity. Examples are keys, tokens, smart cards, bankcards to get money from the ATM, and SIM cards in mobile phones.

Problems with memorising something difficult are no longer an issue when using this authentication factor. Except the fact that the authenticator must now carry with him some piece of hardware, the authentication process itself has become easier for him. It has become more cumbersome for the attacker to pose as the authenticator. In such a case he must copy the hardware item, which is in most cases difficult, if not impossible, or he must physically steal it, which might not go undetected. The disadvantage of using smart cards or other hardware items is that they are expensive. This holds not only for the smart card itself but also the equipment used at the verification side. Loss or theft implies replacing the smart cards and taking precautions that the missing smart card can not be used any longer.

In case of getting money from an ATM machine, the user must provide both the bank card and the PIN code to the ATM. In such a case we say that two factor authentication is used. More on two factor authentication in Section 2.4.

2.3 Are

Since a few years there has been a growing interest in using biometrics for authentication. Most biometric items used are unique per person or chances are very small that 2 different persons posses the same biometric item. For example fingerprints are unique, even with identical twins. Biometric items are always present when the user is present, exceptional cases excluded. Obviously it is for an attacker even harder to steal the biometric item from the legitimate user, although it is not impossible. Depending on which biometric item is used, this can be harder or easier for the attacker. Not only the biometric item itself, but also the hardware checking the biometric features are important for the resistance against fraud.

Biometric properties can be divided into 2 categories, being *physiological*, also called *static* and *behavioural*, also called *dynamic*. Static biometrics are features that will normally not change, e.g. fingerprints, face, iris, hand, and DNA. Dynamic biometrics include for example voice, signature, gait, keystroke and lip motion.

2.4 Two Factor Authentication

Two factor authentication simply means that (at least) two forms of authentication are used. One example is the way a bankcard is used to get money out of the ATM machine. First the card authenticates itself to the ATM machine (and vice versa) and after that the user authenticates himself to the central computer using his PIN code. So we see that *have* and *know* are combined. Both factors are needed to get money out of the ATM.

We find in general the following combinations:

Know and Have: A token is needed and a password or PIN code is needed to open the token. Often also access to computers with classified information is protected in this manner. Another example is access to a building with a wireless token and a PIN code entered when entering.

Have and Are: For example a token that needs a fingerprint to check if it is used by the correct person. In this manner the token will be useless if lost or stolen.

Know and Are: For example access to a building using a fingerprint reader or face recognition, in combination with a PIN code.

Are and Are: Use of multiple biometric features, like prints of 2 different fingers, or fingerprint and face, to enlarge both the security and the reliability of a system.

When two factor authentication is used, it must be made sure that indeed both factors are needed for authentication. Assume for example we have a system with a token and a password or PIN code. In general the password will be needed to *open* the token and the token will in general contain a key for a strong encryption algorithm, needed to authenticate for example to a remote server. At some point the PIN code is entered in the system and checked. If the check is done outside the token, and only the result (true or false) of this check is send to the token, the system is not much stronger then a one-factor authentication system. An attacker will in that case either try to circumvent the PIN code check mechanism and send a “true result” directly to the token, or he will modify the result signal from the mechanism to the token. In any case, without knowledge of the correct PIN code, the attacker can convince the token to open and reveal its secret information.

2.5 Comparisons between Various Authentication Factors

It is very difficult , if not impossible, to compare authentication factors in general. It depends heavily on the specific application. In case of phone based banking, either a PIN code or a voice based authentication might be best. Access to a computer or a network can be implemented easily and cheap using passwords or tokens, or even using fingerprint or face recognition. Access to high security areas might be controlled with retina scan biometrics.

It is very important to get the side conditions clear when trying to find a good (or even the best) authentication possible. Various factors have to be taken into account, like cost, usability, etc.

Chapter 3

Mathematics

When using “Know” or “Have” authentication factors, the outcome of the authentication process is very clear. Either the correct password is used or not, either the correct smart card or a fake one is presented, either the correct PIN code is entered or a false one. In all these cases the decision of whether or not the authentication was successful can be answered with absolute certainty. Note that this does not mean that the correct person has been authenticated! In case a smart card is presented the only check that is performed is whether this is the expected card, not if the person presenting the smart card is the real owner of the smart card.

A presented PIN code at an ATM machine will be sent to a central location to check it. Anybody who is listening in on the communication between the ATM machine and the central location would be able to pick up the PIN code if no additional measures to protect it would have been taken. To protect the confidentiality (i.e. to keep it secret) and the integrity (i.e. to keep it unaltered) of the PIN code cryptography will be used. In Section 3.1 we will give a brief introduction into cryptography.

When using biometric authentication it is not possible to decide with absolute certainty that the person presenting the biometric feature is really who he claims he is. For example in case of fingerprints a finger can be presented in various ways, e.g. shifted or rotated, with more or less pressure, under different light conditions, with dirt on the finger or on the reader and with more or less sweat. Another example is a signature which is different every time it is written down. We need techniques that can compare biometrics features and can decide whether two sets of features are close together (i.e. from the same person) or far apart (i.e. from 2 different persons). More on this can be found in Section 3.3.

In this chapter we will give a brief introduction into various mathematical subjects needed to understand many of the topics related to authentication.

We will give a short introduction into cryptography (see Section 3.1) and statistics (see Section 3.2). Furthermore we will describe *distance functions* which are needed to compare biometric properties in Section 3.3.

3.1 Basic Cryptography

Using authentication is often a first step in a large set of security measures taken to protect people, properties, information or otherwise. Authentication is a sub area of the much larger area of cryptology. The main focus of cryptology is keeping information secret to unauthorised people, but being able to share it in a secure way with those who need to have knowledge of the information. For a comprehensive overview of cryptography see [10] and for a description of the basics of cryptography see [12].

In general we think of authentication as a local process, e.g. someone getting access to the files on his computer or getting access to a physically secured area. But when communicating information to others through automated systems, authentication is no longer local. Sitting at your PC you must prove your identity to someone elsewhere in the world. This process of authentication could be unilateral (you proving your identity to the other party involved <u>or</u> the other way around) or mutual (proving your identity to the other party involved <u>and</u> the other way around). For this purpose protocols are used. In a protocol information is send back and forth several times until the purpose of the protocol is satisfied. The information that is send can be partly open or it could be hidden using a symmetric or an asymmetric cryptographic algorithm. More on protocols can be found in Chapter 5.

A cryptographic algorithm is a mathematical description of how to process data. It takes (in general) 2 parts of input, being the information that needs to be protected and a so called *key*. The main reasons to use cryptography are confidentiality, integrity and authentication/identification. We will explain these terms briefly

Confidentiality: Confidentiality is used to make sure that the secret information will be encrypted into something that is not readable for unauthorized people. Only those who share the correct key are able to turn the encrypted information back into the original information. An example where confidentiality is needed is when your PIN code is transmitted from the ATM machine to the central location where it is checked.

Integrity: Integrity is used to make sure that any change in the information will be detected. In this case the information itself does not need to be secret and can even be transmitted in plain. An example where integrity is needed is for press releases of the government or major companies. The information is public by default but it is very important that nobody can change it.

Authentication/Identification: These terms speak for themselves, but in terms of cryptology often protocols are used to achieve authentication and identification.

In cryptography we distinguish between *symmetric* and *asymmetric* algorithms. Both will be described in greater detail below. In the remainder of this document we will speak of *encryption* of data when we mean the transformation of plain, open, readable information to closed, scrambled data. *Decryption* will be the reverse of this process. An *algorithm* will be the (mathematical) method by which the information is encrypted or decrypted. A *key* is a (relatively) short piece of information that in most (but not all) cases needs to be kept secret. If a secret key (or the secret part of a public key pair) is lost or stolen, the new owner can use it to get access to the data in the same manner as the legitimate owner.

3.1.1 Symmetric Cryptography

The term symmetric cryptography reflects the fact that in this case the same key is used for encryption and decryption of the data. The encryption algorithm is fully reversible, and in general it must be (slightly) adjusted before it can be used for decryption. The value of the key (mostly a string of random bits of a specific length) is the same for encryption and decryption. The way the key is processed before using it in the encryption or decryption algorithm is called *key schedule.* The key schedule for encryption and decryption is in general the same, but the resulting *sub-keys* are numbered differently for the specific purpose.

Symmetric cryptography can again be divided in two categories, called *block ciphers* and *stream ciphers.* These will be described in more detail in the following subsections

Block Cipher

The name block cipher comes from the fact that it takes in blocks of data of a fixed size (the "block size", e.g. 128 bits). The first step, before the real encryption begins, is often a key schedule in which the bits of the key K are mangled and various sub keys $K_1, ..., K_t$ are produced. The number of sub keys and their size depends on the specific algorithm. The data that has to be encrypted is split into blocks of the correct size (the block size) and they are processed by the algorithm one by one in the correct order. The exact way the blocks are processed depends on the mode the algorithm is used in. This will be explained later in this section from page 15 and further. For the moment we will concentrate on the processing of a specific block of data.

A popular method for constructing block ciphers is using Feistel structures and the best known example for this is DES (Data Encryption Standard). We will not describe DES here in great detail but merely try to explain some basics. DES uses input blocks of 64 bits and a key K of 64 bit (being 56 real key bits and 8 check bits). In the key schedule the key K is transformed into 16 sub keys K_i, each 48 bits long. DES transforms the 64 bit input data into 64 bits output data using 16 rounds and in each round a sub key K_i is used. The main feature of DES is that it splits the data into 2 halves, each 32 bits in size (L_i and R_i for left and right). Let (L_{i-1}, R_{i-1}) denote the input to round i (where i runs from 1 to 16), then the output of this round equals $(L_i, R_i) = (R_{i-1}, L_{i-1} \oplus f(R_{i-1}, K_i))$. Figure 3.1 gives a graphical impression of DES.

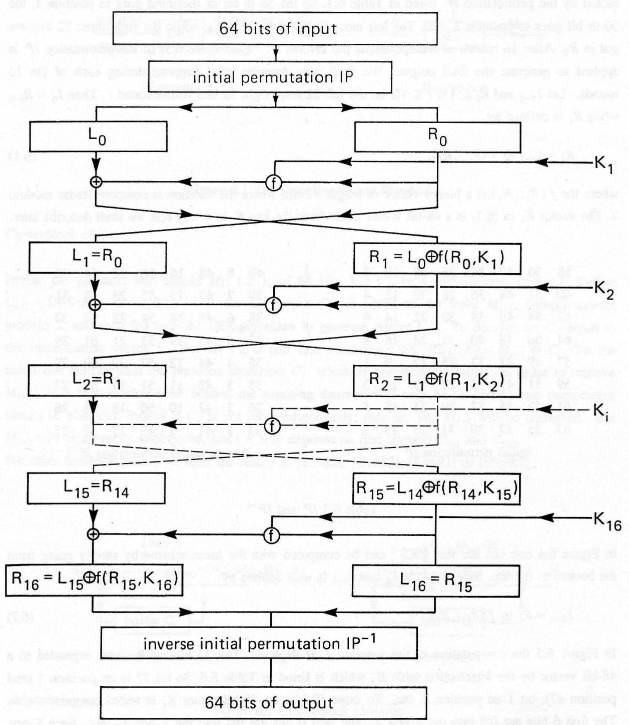


Figure 3.1: DES Global Structure

The f-function is used to make sure that no easy cryptanalysis is possible. The f-function takes as input a 32-bit value R_{i-1} and a 48-bit sub key K_i and returns a 32-bit value. The 32 input bits are first expanded to 48 bits by repeating some of the bits. Then the sub key K_i is XORed to the result. The 48 bits are split up in 8 6-bit values which are reduced to eight 4-bit values by applying so called S-boxes. The outputs of the S-boxes are again concatenated and the resulting 32 bits are permuted before they are outputted as a 32-bit value.

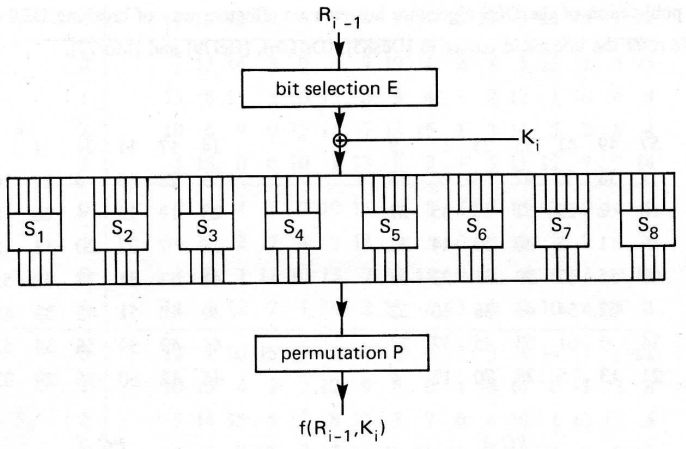


Figure 3.2: DES f-function

Modes of Operation

Using DES as an example we have shown how a block cipher works. But every time just one block (64 bits in case of DES) are encrypted and the size of the information that needs to be encrypted is in general much larger. For this reason the information is divided into blocks of N bits, where N is the block size, and the blocks are encrypted one by one. The input and output of the algorithm can be combined in various ways. The way the input and output are combined is called the *mode* the algorithm is working in. There exist various standard modes, a few of which will be described below.

The easiest mode is the *Electronic Code Book (ECB)* mode. In this case every block is encrypted independent of the other blocks and the resulting encrypted data blocks are concatenated again. The ECB mode is graphically depicted in Figure 3.3. We only show the encryption of the ECB mode because it is obvious how the decryption works.

Another mode that is used a lot is the *Cipher Block Chaining (CBC)* mode. In this case the output of the encryption algorithm is first XORed

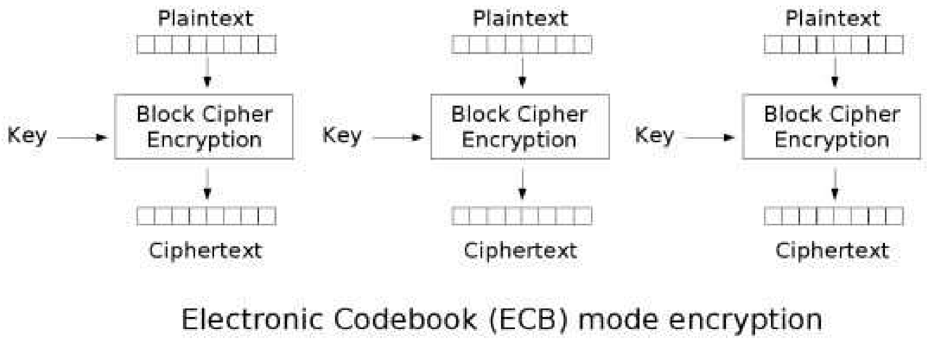


Figure 3.3: Electronic Code Book Encryption Mode

to the next block of information before the new encryption takes place. In Figure 3.4 we can again see the encryption mode of CBC. It is easy to derive the decryption mode of CBC from this, but for the sake of completeness this is given in Figure 3.5. When encrypting the first block of plain data, no output block is available yet. For this reason a so-called *Initialization Vector (IV)* is used. This IV is a randomly chosen string of the same length as the block size. It can be kept secret (for example it can be a part of the result of a key schedule protocol) or it can be transmitted in plain from the sender to the receiver. In most cases the IV is randomly generated by the sender and send in plain to the receiver.

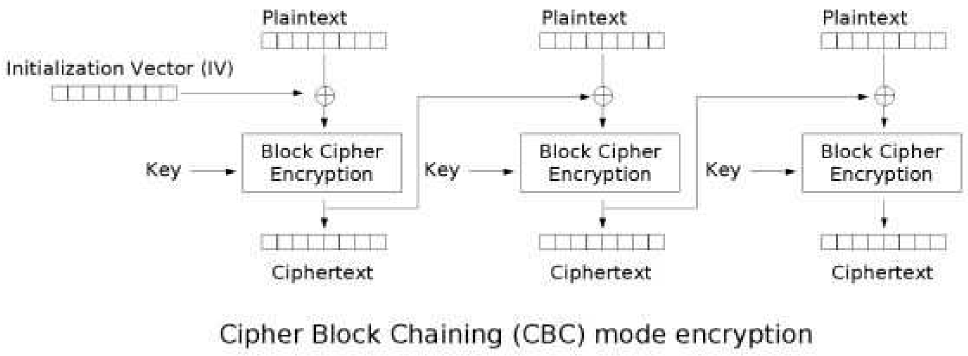


Figure 3.4: Cipher Block Chaining Encryption Mode

Finally we will show one more well know mode of operation called *Output Feed Back Mode (OFB)*. This mode slightly differs from the CBC mode. The

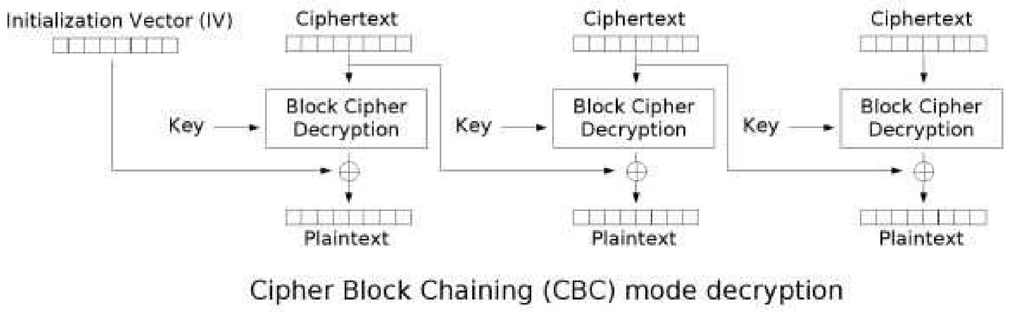


Figure 3.5: Cipher Block Chaining Decryption Mode

output of each round is again XORed to a block of plain text. But now this result is directly used as output of the mode and only the output of the algorithm is fed back into the encryption algorithm again. We see in Figure 3.6 how the CFB encryption mode works. In this case encryption and decryption are actually the same, as long as for decryption also the encryption algorithm is used!

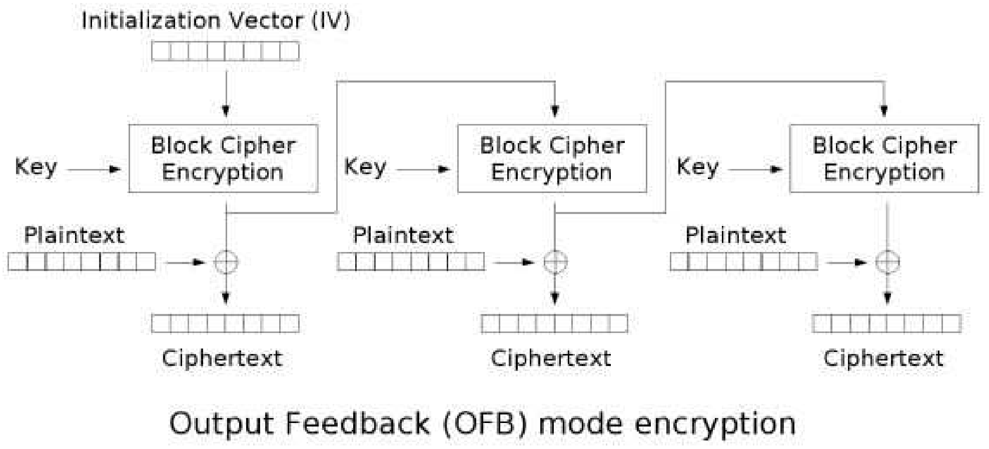


Figure 3.6: Output Feed Back encryption mode

Stream Ciphers

In the OFB mode we see that actually the data itself is not used in the encryption algorithm. The encryption algorithm is initialized with some

Initial Value and then the output is used again for encryption. We can actually see the OFB mode as a form of a *stream cipher*. In a stream cipher the bits of the plain text are XORed with key bits that are generated in a deterministic way. A common way to generate the key bits is to use a *Linear Feedback Shift Register (LFSR)* that is initialised with a secret key shared by the sender and the receiver. A very simple LFSR is displayed in Figure 3.7. In this case the LFSR has 4 cells and the contents of each cell is a single bit. The contents of the right most cell (S_1) is used as key bit. Every time the contents of the 4 cells shift 1 position to the right and the new contents of the left most cell (S_4) is equal to the modulo 2 addition of the old contents of cells S_1 and S_2. In every step an LFSR produces one key bit, so one plain text bit is encrypted.

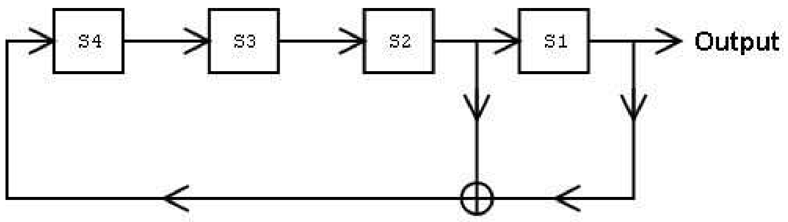


Figure 3.7: Example of LFSR

Obviously more complicated structures can be build using LFSRs as shown in Figure 3.8. Once the LFSR has been initialized with a random value, all the output bits can be determined unambiguously. The security of a stream cipher lies in the secrecy of the initial value. To a lesser extend also the secrecy of the design of an LFSR adds to the security but only moderately. In many practical cases people have been able to retrieve the design of the LFSR from the finished product. The LFSR in Figure 3.8 is actually the LFSR that is used in the A5.1 algorithm in GSM.

3.1.2 Asymmetric Cryptography

Asymmetric cryptography is relatively new and is based on the difficulty of some mathematical problems. For example it is very easy to multiply 2 numbers together, but it is a hard problem to find the two factors from the product, as long as the factors are large enough.

Asymmetric cryptography is also called *public key cryptography*. The reason for this is that the key actually consists of two parts, one of which is made public (P) and the other one is kept secret (S). The keys P and S are related so that one is the inverse of the other. The trick is that from P it is

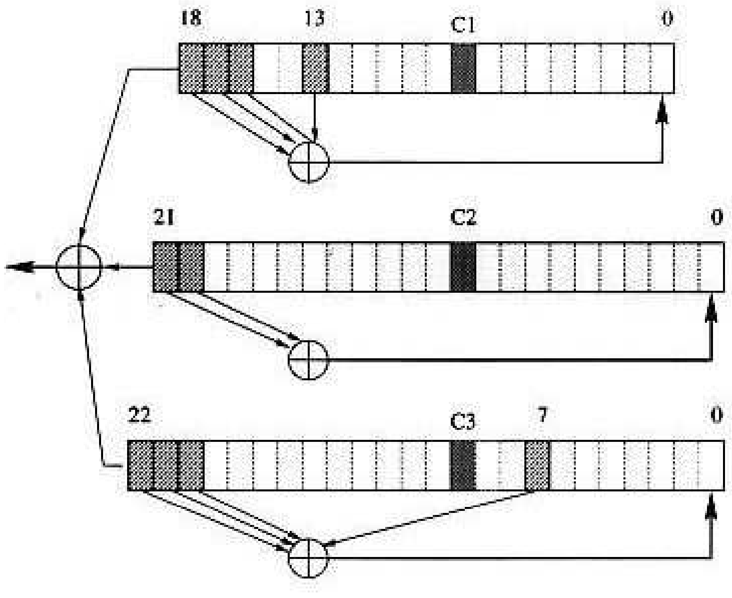


Figure 3.8: Combining various LFSRs

not possible to find S because of the difficulty of the mathematical problem, except when some extra (secret) information is known.

The general disadvantage of asymmetric cryptographic systems is that they are slow because of all the mathematical calculations involved. So they can not really be used to encrypt large amounts of data. Often they are only used to encrypt for example a key used in a symmetric cryptographic system, i.e. they are used in the *key exchange*. Another way to use them is to encrypt a hash of the message. In this case integrity is protected. If the message is changed, the encrypted hash value will show this.

We will describe 2 public key systems, being RSA and DH, the first based on the difficulty of factoring large numbers and the second based on the difficulty of taking discrete logarithms in a finite field.

RSA

RSA is based on the difficulty of factoring the product of two large prime numbers. Let p and q be these prime numbers. Their size ranges from 512 to 1024 bits in nowadays commercial applications. Let n be their product: $n = p \cdot q$, then, because p and q are prime we find that $\phi(n) = (p-1) \cdot (q-1)$. Now let e be a random number satisfying the following two conditions. $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$. Now finally let d be the unique number satisfying:

$$e \cdot d \equiv 1 \bmod (\phi(n)) \tag{3.1}$$

Now we are ready to define the public and the secret key. The public key P equals (e, n) and the secret key S equals (d, p, q). Notice that from S we can also find the values of e and n easily, i.e. the public key P. This does not hold the other way around!

Now we will describe the encryption and decryption processes. When encrypting a message M the sender uses the public key P. He calculates the cipher text C as

$$C = M^e \bmod n. \tag{3.2}$$

The receiver now calculates M' as

$$M' = C^d mod n. \tag{3.3}$$

It is now easy to show that indeed M' equals the original message M:

$$\begin{aligned} M' &= C^d \bmod n = \\ &= M^{d \cdot e} \bmod n = \\ &= M^{1+k\cdot\phi(n)} \bmod n = \\ &= M^1 \cdot \left(M^{\phi(n)}\right)^k \bmod n = \\ &= M^1 \bmod n = M \end{aligned}$$

RSA can very well be used to encrypt information as we can see above. But RSA can also be used to produce digital signatures. This is something different from what will be discussed in Chapter 10! A digital signature on an electronic document can be seen as the electronic equivalent of the normal signature on a paper document. To sign a document the sender does not use the receiver's public key, but instead uses his own secret key. The receiver must then use the public key of the sender to check the electronic signature. Calculations are exactly the same as above, except that the roles of e and d are interchanged.

It is obvious that both things can be combined. A sender can both electronically sign the document with his own secret key and after that encrypt it with the receivers public key to protect the confidentiality of the document. Care must be taken that the receiver performs the operations in the reversed order.

Diffie-Hellman

Diffie Hellman (DH) is based on the difficulty of calculating discrete logarithms, that is logarithms in a finite field. All operations are performed module p, where p is a large prime number. In other words we work in the finite field $\mathbb{F}_p$. Diffie Hellman is mainly used to arrange a common key without having a shared secret. Let $\mathbb{F}_p$ and a generator α of $\mathbb{F}_p$ be shared between the communication partners. These parameters can be open without effecting the security of the system.

Let A and B be the two communication partners and let A chose a random, secret value x and B a random and secret value y. Now A sends

$$X = \alpha^x \bmod p$$

to B and he receives

$$Y = \alpha^y \bmod p$$

from B. Now both A and B can calculate

$$k = \alpha^{x \cdot y} \bmod p.$$

Because no third party can deduce a from x or y from y, they are not able to calculate k.

There exist various variations to this basic DH algorithm of which we will briefly describe one here. Assume that A and B use a DH system with parameters p and α. A has chosen a "long term" secret value r and published the triple $(p, \alpha, R = \alpha^r \bmod p)$ as his public key P_A, keeping as his secret key S_A the triple (p, α, r). Similarly B has chosen a "long term" secret value s and found $P_B = (p, \alpha, S = \alpha^s \bmod p)$ and $S_B = (p, \alpha, s)$. Now assume that A sends the same value X as above, and also B sends Y, obviously again with newly generated random values x and y. Now both A and B can find as a secret key:

$$K = \alpha^{(x \cdot s + y \cdot r)} \bmod p.$$

To do this A calculates:

$$K = (S^x \cdot Y^r) \bmod p,$$

which he can do because he knows the long term secret S of B, he received Y from B, and he knows the secret values x and r. B can perform similar operations.

3.2 Basic Statistics

A lot of probabilities and statistics is based upon counting. For example when we consider the security of passwords we need to count the number of possible passwords. We will first go into a little more detail of counting before we advance to statistics

3.2.1 Counting

Some things are easy to calculate, like the number of possible outcomes of a throw with a single dice. It is more difficult to count the number of possible outcomes of a throw with two dices. We cannot even give one correct answer. On the one hand can we say that the outcome is in the set $\{2, 3, 4, ..., 12\}$ if we count the total number of dots. Another option is to see the outcome as a pair $\{x, y\}$ and now we find 21 possible outcomes, being $\{\{1, 1\}, \{1, 2\}, \{2, 2\}, \{1, 3\}, \{2, 3\}, ..., \{6, 6\}\}$. In this case we see the two dices as similar and we do not mind if we throw a 1 with dice 1 and a 3 with dice 2 or vice versa. The third counting option is to see the two dices as different and report their values as an ordered pair (x, y). In this case we find the 36 options $\{(1, 1), (1, 2), ..., (1, 6), (2, 1), ..., (6, 6)\}$. So we need to be careful in describing what we are going to count and how we are going to count it.

When composing a password, we can use various characters. First of all we can use the lower-case letters (26) and upper-case letters (26). Next the numbers (10) are available. But on a keyboard also other, special, characters are available. We will assume that we can use 32 special, although this may vary per keyboard. The total number of characters that can be used is now equal to 26+26+10+32 = 94. Now let us see how to find the number M of 6 character passwords with at least 1 number in it.

Obviously the total number of passwords is $N = 94^6$, because for every character we can choose 1 out of 94 options. Passwords like 'aaaaaa' are also included in this. So obviously M is smaller then N. The difference between N and M is that we counted also all the passwords with *no* numbers in it. How many passwords have *no* number in it? To see this we note that there exists 84 characters that are not a number. So the number of passwords with *no* number in it is $K = 84^6$. Now it is easy to see that $M = N - K = 94^6 - 84^6$. The trick we applied was to count the number of *incorrect* passwords and subtract that from the total number of passwords to find the number of correct passwords.

A bit more complicated is counting the number of passwords with exactly 1 number in it. First we note that this number can occur in 6 different places and that for every possible place we can use one of the 10 numbers. If we would remove the number from the password we are actually left with a 5 character password with no numbers in it. And we already know that there exist $N = 84^5$ of such passwords. Picking the value of the number and its position can be done in $K = 10 \cdot 6 = 60$ possible ways. So we find that for each of these K possibilities we find N passwords, thus the total number of passwords with exactly 1 number in it equals $M = N \cdot K$. The trick here was to see that we have two independent options. The choice of the value and the position of the number is independent of the choice of the rest of the password, with the restriction that the remainder could not contain a number.

In similar ways we can now count the number of passwords that have to fulfil some specific properties. We will not go into more detail here but leave it as an exercise for the reader.

From the number of possibilities to a probability is only a small step. The probability that a random chosen password of 6 characters contains no number equals $(84/94)^6$. To find a probability we divide the number of correct options by the number of all possible options. Let us see what the possibility is that a 4-digit PIN code has exactly 3 different numbers. We know that the total number of possible 4-digit PIN codes equals $10^4 = 10.000$. So we need to count the number N of PIN codes with exactly 3 different values, i.e. one of the numbers occurs twice. First we forget about the double number and we choose a 3-digit value with 3 different numbers. There are $10 \cdot 9 \cdot 8 = 720$ options for that. Now we choose which of these 3 digits will appear twice (3 possibilities). Furthermore we choose where this digit must appear. Remember we already have a 3-digit value. So we can either put it in front, between the first and second digit, between the second and the third digit, or at the end. We see that this gives 4 options. It is now tempting to count the number N as $720 \cdot 3 \cdot 4 = 8640$, but now we counted too much. We illustrate this as follows. Let 123 be the 3-digit value and assume we want to insert an extra 1. So we now find the 4 possibilities 1123, 1123, 1213 and 1231. We immediately see that 1123 appears two times. But also 1213 can also result from inserting a 1 in front of 213 and 1231 can result from inserting a 1 in front of 231. So every possibility is counted twice and we find that $N = 720 \cdot 3 \cdot 4/2 = 4320$.

So we find the probability that a random 4-digit consists of exactly 3 digits equals 0,432. Actually it is not difficult to check the values in Table 3.1, where n is the number of different digits and p_n is the probability of occurrence.

n	$\mathbf{p_n}$
1	0,001
2	0,063
3	0,432
4	0,504

Table 3.1: PIN code probabilities

Things can often be counted in more then 1 way. To see this we will again have a look at the number of PIN codes with exactly 3 different numbers. But now we will start with looking at the four positions in the PIN code and first choosing which 2 positions will contain the same number. The number of ways we can choose these 2 positions out of the 4 possible positions equals $\binom{4}{2} = 6$. Next we choose which number will appear twice, which can be done in 10 different ways. The 2 remaining open positions can be filled in $9 \cdot 8 = 72$ different ways. In total we find again $6 \cdot 10 \cdot 72 = 4320$ possibilities.

3.2.2 Statistics

We will here define the most important concepts, needed to understand most of what is needed. Let us first assume that $\mathbf{x} = (x_1, x_2, ..., x_n)$ is a sequence of numbers. This can be anything, but in general it will be a set of measurement of some biometric feature in this document. For example for the values of a dice we could have $\mathbf{x} = (1, 2, 3, 4, 5, 6)$ but it could also be a set of acceleration values of the ankle when using gait authentication.

Before we will define *mean* and *variance* of $\mathbf{x}$ we must first define probabilities $\mathbf{p} = (p_1, p_2, ..., p_n)$ where p_i is the probability of occurrence of x_i. In the example of the dices, it is clear that $\mathbf{p} = (1/6, 1/6, ..., 1/6)$.

Definition 3.1 ***Mean:*** *The mean $\mu_\mathbf{x}$ of $\mathbf{x}$ is the weighted average of its values, where the weights equal the probabilities, i.e.:*

$$\mu_\mathbf{x} = \sum_{i=1}^{n} p_i \cdot x_i.$$

The mean $\mu_\mathbf{x}$ is also denoted by $E(\mathbf{x})$ and within formulas we will mostly use the $E(.)$ notation. It is not difficult to calculate that in case of the dice we find $\mu_\mathbf{x} = 3\frac{1}{2}$. Note that if every item of $\mathbf{x}$ is multiplied by a constant c, the mean also will be multiplied by that constant c.

Definition 3.2 ***Variance:*** *The variance $\sigma^2_\mathbf{x}$ of $\mathbf{x}$ is a measure of how much the members of $\mathbf{x}$ are scattered around its mean $\mu_\mathbf{x}$. It is mathematically defined as:*

$$\sigma^2_\mathbf{x} = E(\mathbf{x} - \mu_\mathbf{x})^2.$$

The variance is also denoted by $V(\mathbf{x})$. Note that, because $\mu_\mathbf{x}$ is constant, the variance is equal to

$$\begin{aligned} \sigma^2_\mathbf{x} &= E(\mathbf{x}^2 - 2 \cdot \mu_\mathbf{x} \cdot \mathbf{x} + \mu^2_\mathbf{x}) = \\ &= E(\mathbf{x}^2) - 2 \cdot \mu_\mathbf{x} \cdot E(\mathbf{x}) + \mu^2_\mathbf{x} = \\ &= E(\mathbf{x}^2) - 2 \cdot \mu_\mathbf{x} \cdot \mu_\mathbf{x} + \mu^2_\mathbf{x} = \end{aligned}$$

$$
\begin{aligned}
&= E(\mathbf{x}^2) - \mu_{\mathbf{x}}^2 = \\
&= \sum_{i=1}^{n} p_i \cdot x_i^2 - \mu_{\mathbf{x}}^2.
\end{aligned}
$$

When all the values x_i are close to their mean, the variance will be small, but if they are far away, the variance will be large. The value $\sigma_{\mathbf{x}}$ is called the *standard deviation.*

We say that $\mathbf{x}$ is *uniformly distributed* if every x_i has the same probability of occurrence, i.e. if $p_i = 1/n$ for all i. For example the outcome of dice is uniformly distributed. Another possible *distribution* is called *binomial distribution.* To explain this, think of the following experiment. Let us flip a coin n times and let us count the number of times we see heads. This is a number between 0 and n, both included. Obviously we expect to see heads around $n/2$ times if the coin is unbiased. But let us assume for the moment that the probability for heads is p, so the probability for tails is $1-p$. The unbiased coin has $p = \frac{1}{2}$, but we will use p in this discussion for generality.

The probability p_i that we see i times heads equals:

$$
p_i = \binom{n}{i} \cdot p^i \cdot (1-p)^{n-i}, \tag{3.4}
$$

where

$$
\binom{n}{i} = \frac{i! \cdot (n-i)!}{n!}.
$$

If $\mathbf{x}$ has a probability distribution as above, we say that it is a *binomial variable.*

Now let $\mathbf{x}$ be as before and assume that $\mathbf{y} = (y_1, y_2, ..., y_n)$. We are interested to know what the relation between $\mathbf{x}$ and $\mathbf{y}$ is. The most common way to measure this is their *covariance.* We already know what the variance of $\mathbf{x}$ is and the covariance is defined in a similar manner:

Definition 3.3 ***Covariance:*** $\sigma_{\mathbf{xy}} = E\left[(\mathbf{x} - \mu_{\mathbf{x}}) \cdot (\mathbf{y} - \mu_{\mathbf{y}})\right]$.

It is again easy to derive that $\sigma_{\mathbf{xy}} = E(\mathbf{xy}) - \mu_{\mathbf{x}} \cdot \mu_{\mathbf{y}}$, where

$$
E(\mathbf{xy}) = \sum_{i=1}^{n} \sum_{j=1}^{n} p_{i,j} \cdot x_i \cdot y_j,
$$

and $p_{i,j}$ is the probability that x_i and y_j occur simultaneously. Once the covariance is defined we can conclude with the definition of the correlation between $\mathbf{x}$ and $\mathbf{y}$ as:

Definition 3.4 ***Correlation:*** $\rho_{\mathbf{xy}} = \frac{\sigma_{\mathbf{xy}}}{\sigma_{\mathbf{x}} \cdot \sigma_{\mathbf{y}}}$.

It can be shown that the value of the correlation lies between -1 and +1. If it equals 0, $\mathbf{x}$ and $\mathbf{y}$ are said to be uncorrelated. The further away from 0, the stronger the correlation between $\mathbf{x}$ and $\mathbf{y}$ is. If the correlation is positive, then an increase in $\mathbf{x}$ implies an increase in $\mathbf{y}$. If the correlation is negative, then $\mathbf{y}$ will decrease when $\mathbf{x}$ increases.

3.3 Distance Metrics

As mentioned before, and described in more detail in Chapter 7, when using biometrics it is not possible to have 100% certainty when authenticating a person. This is due to the fact that the captured biometric feature is different every time. The biometric feature, e.g. a fingerprint, is examined and distinct features are extracted. These features are represented as a sequence of values and this sequence has to be compared to other stored sequences. The stored sequence is referred to as *template*. In this section we will describe how sequences of values can be compared to each other. The methods for this are endless and it is the task of the researcher to find the method that gives the best results. What is the best method depends on various things like biometric feature used (e.g. fingerprint, voice or gait), the extracted features and even the group of people for that are going to use the specific authentication method.

In this section we will briefly discuss methods that can be used without going into too many details. As explained, the best method depends on the extracted features and there is a lot of variation in that. Therefore there is no one way of even describing the best way of finding the best distance metric.

A distance metric is no more or less then a method of defining "how far apart or how close together" 2 sequences of values are. In general these sequences are of equal length and corresponding items can indeed be compared. By this we mean that if the values in one sequence represent speed and the values in the other sequence represent acceleration, these sequences are not comparable, or at least the distance between them can not be interpreted in a useful manner. For the remainder of this section we will assume that the 2 input sequences to a distance function (i.e. the function defining the distance metric) are of equal length and contain comparable features. The output of the distance function is a value and is called the distance between the two input sequences.

We will now assume that the common length of the two sequences is n and the sequences are denoted by $\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n})$ and $\mathbf{y} = (\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_n})$. One very easy way of comparing these two sequences is:

Definition 3.5 Absolute distance:

$$d_1(\mathbf{x}, \mathbf{y}) = \sum_{\mathbf{i=1}}^{\mathbf{n}} |\mathbf{x_i} - \mathbf{y_i}|$$

In this case the absolute differences of the elements of the sequences are summed to get a total distance. Note that $d_1(\mathbf{x}, \mathbf{y}) = \mathbf{d_1}(\mathbf{y}, \mathbf{x})$, i.e. in this case the distance function is symmetric in its two input values. The distance between $\mathbf{x}$ and $\mathbf{y}$ is the same as the distance between $\mathbf{y}$ and $\mathbf{x}$. In most cases this will hold, but certainly not in all cases. In a biometric system a number of templates of enrolled persons is compared with the biometric features of the person trying to authenticate himself. It is not hard to see that the role of the sequences of the templates can be different from the role of the sequence of the person trying to authenticate.

We will define some more commonly used distance functions:

Definition 3.6 Euclidean distance:

$$d_2(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{\mathbf{i=1}}^{\mathbf{n}} (\mathbf{x_i} - \mathbf{y_i})^{\mathbf{2}}}$$

Definition 3.7 Maximum difference distance:

$$d_3(\mathbf{x}, \mathbf{y}) = \max_{\mathbf{i=1..n}} |\mathbf{x_i} - \mathbf{y_i}|$$

In general a lot of distance functions can be defined. The distance between two sequences is by itself not relevant. It relevance comes from the comparison with distances between other pairs of sequences. Let us assume we have a biometric system where N users are enrolled. Let us assume also that each of these N users has presented its biometric M times, i.e. from every user we have M sequences representing his biometric feature. Now we can have a look at the *inter-person* and the *intra-person* distances. The intra-person distance of person i is defined as the maximum of the distance between the M sequences of person i. The inter-person distance between person i and person j is the minimum of the distances between a sequence of person i and person j. More mathematically, let $\mathbf{x}_i^k$ denote the k^{th} sequence of person i, then the intra-person distance D_i^{intra} of person i is defined as:

$$D_i^{intra} = \max_{m,n,m \neq n} d(\mathbf{x}_i^m, \mathbf{x}_i^n),$$

where $d(.,.)$ is the distance metric used to measure distances between sequences. The inter-person distance $D_{i,j}^{inter}$ between person i and person j is defined as

$$D_{i,j}^{inter} = \min_{m,n} d(\mathbf{x}_i^m, \mathbf{x}_j^n).$$

A good distance metric $d(.,.)$ has the property that the intra-person distances are small while the inter-person distances are large. This has the following consequence. Let us have a look at our biometric system with our N users again. Assume that each user has M templates stored in a database, where M is in general 1 or at least small. Now a users tries to log on to the system, providing his identity i and his biometric feature which is transformed into a sequence $\mathbf{y}$. This sequence $\mathbf{y}$ is compared to the templates $\mathbf{x}_i^j$ of the claimed identity. If the distance to these templates is indeed small, we may conclude that, because of the small intra-person and large inter-person distance, it is likely that the claimed identity is true. In order to be able to make such a decision a threshold t needs to be used. This threshold is a value based on all the inter- and intra-person distances. Now a claimed identity is accepted as true if the distance between the captured biometric sequence $\mathbf{y}$ and the claimed templates $\mathbf{x}_i^j$ is below this threshold. The claimed identity is regarded as false if the distance is above the threshold.

Chapter 4

Passwords

4.1 Password Introduction

When we speak of a password we actually refer to a large variety of methods. Beside the well known concept of a password we can distinguish PIN codes, pass phrases, associative and cognitive passwords, pass faces ([3])and pass images. All of these are either explained briefly or examples are given below:

Password: Well know concept where a number of characters is entered to get for example access to a computer or to a program on the computer. Mostly completely software based.

PIN code: Best known in combination with a bankcard to get money out of an ATM machine. In contrast to a password does a PIN code only consist of numbers and is mostly limited in length to 4 or 5 digits.

Pass phrase: Very similar to a password except that it is much longer and can also contain the "space" character.

Associative passwords: A number of words is presented as a challenge and a response must follow with associative words, for example black-white, hill-mountain, pc-laptop, etc.

Cognitive passwords: This is more or less similar to associative passwords, except that the challenge must be responded based on autobiographical data, for example "What is the name of your cat", "What is your favourite brand of cars", etc.

Pass faces and pass images: In this method a person must recognise pictures of faces or random images from a pre-chosen set. During enrollment the user is presented with (or can select) a number of pictures. During authentication he must recall the pictures of his designated set.

In this chapter we will look at security offered by using passwords. The term password will be used for a string of characters from 4 different sets being:

1. The set of capital letters: A,B,...,Z;
2. The set of lower case letters: a,b,...,z;
3. The set of numbers: 0,1,...,9;
4. The set of special characters: e.g. !, ", @, $, %, &, /, { , }, \, and ".

In general there can be rules about the minimum or maximum number of characters that can be used and also there can be restrictions on combinations that need to be used (e.g. at least one special character or characters from multiple sets). In general no guidelines are provided how to come up with a password that is both hard to guess and on the other hand easy to remember. Later on we will give some rules on how to do this.

4.2 Password Security

In this section we will investigate the security offered by passwords. First of all we must define the size s of the space of all possible passwords denoted by $S = \{\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_s\}$. In case of 4 digit PIN codes we have $s = 10^4$ and in case of 6 letter words we have $s = 52^6$. Note that by a word we do not mean something that makes sense in English, Norwegian, Dutch or any other language, just a collection of lower and upper case letters. From the size s we define the entropy as the number of bits describing the password space. If all passwords are equally likely (as in the case of PIN codes for bankcards), the entropy is just equal to $\log_2(s)$. In case that not all passwords in the password space are equally likely, then the entropy can be calculated using the following formula:

$$h = -\sum_i p_i \cdot \log_2(p_i)$$

where the summation runs over all passwords in the space and p_i denotes the probability that password $\mathbf{s}_i$ is chosen. In case all passwords are equally likely we find:

$$h = -\sum_i \frac{1}{s} \cdot \log_2(\frac{1}{s}) = -s \cdot \frac{1}{s} \cdot \log_2(\frac{1}{s}) = \log_2(s),$$

just as was to be expected. Now let us see what happens when people can change there PIN codes for bankcards. The set S of all PIN codes is $S = \{0000, 0001, ..., 9999\}$ and the size of S is $s = 10.000$. When generated by the bank we must assume that all PIN codes are equally likely, so the entropy is will be equal to $\log_2(10^4) = 13.3$ bits. Now let us next assume that people can change their PIN code to something they can easily remember. Assume for example that half of the people will change the code to a date, e.g. 1606 for the 16^th of June. So half of the people will have a PIN code that looks like a date, i.e. 1 out of 366 possibilities. The other half of the people have a PIN code that is more or less random, i.e. one of the remaining 9634 possibilities. So we see that 366 possible PIN codes each have a probability of $p_i = \frac{1}{2 \cdot 366}$ while the remaining 9634 PIN codes each have a probability of $p_i = \frac{1}{2 \cdot 9634}$. The entropy now becomes:

$$\begin{aligned} h &= -\sum_i p_i \cdot \log_2(p_i) = \\ &= 366 \cdot \frac{1}{2 \cdot 366} \cdot \log_2 2 \cdot 366 + 9634 \cdot \frac{1}{2 \cdot 9634} \cdot \log_2(2 \cdot 9634) = \\ &= \frac{1}{2} \cdot (\log_2(2 \cdot 366) + \log_2(2 \cdot 9634)) = 11.9 \end{aligned}$$

We see that the entropy has reduced by 1.4 bit. But there is more. An attacker will not try all possible PIN codes when he has access to a bankcard. He will restrict his attention to date-like PIN codes. If we look at only those cards the entropy equals $h = log_2(366) = 8.5$, almost 5 bits lower then the full entropy. For every 2 cards he gets access to, on average one of them will have a date-like PIN code and can thus be broken with less then 9 bits of work.

When looking at the keyboard, we see that there are many special characters. If we count the 3 extra Norwegian letters as special characters, then there are at least 32 of them. We will assume that the set of special characters is restricted to 32, so in total we can use $26+26+10+32 = 94$ characters to form a password. If we were to make a completely random password of 6 characters we have $94^6 = 6.898.781.056$ which looks enormous. But expressed in bits we have $\log_2(94^6) = 39.3$ bits[1] which is not that big. But although this number is not very big, it should provide a reasonable amount of security. For 8 character passwords the entropy increases to over 50 bits, 52.4 to be precise. So why are so many passwords cracked?

[1]Back in the time when crypto got its first serious commercial attention many products used 40 bit keys, due to export regulations, mainly imposed by the USA.

The major problem when choosing passwords is that they are not chosen at random. Random passwords are difficult to remember and will therefore be written down (which is in many cases not very secure) or changed to an easy to remember password. Suppose there is a rule that the password must be 6 characters long and it should contain at least characters from 3 out of the 4 sets mentioned above. Many people will then use a name that starts with a capital letter, followed by a number of lower case letters and will end with a number or a special character. Let us now see how many passwords we find in this way. We will be gently and not only look at words, but just assume the first 5 characters to be letters of which only the first is a capital. We now have $S = 26 \cdot 26^4 \cdot 42 = 499.017.792$ possible passwords and $\log_2(S) = 28.9$ which is a reduction of over 10 bits. Restricting ourselves to real words probably will reduce this number again by 10 bits.

4.3 Password composition

As described in the previous section we must be careful how we select a password. Long passwords with many restrictions that have to be changed frequently do not necessarily result in a secure system. These passwords will in general either be written down or still be easy to guess. Many people will choose an easy to remember password and extend that with some kind of counter. Whenever the password has to be changed the counter will be increased, thereby resulting in more or less the same password. Because passwords are mostly stored in a hashed form such a small difference will result in a totally different hash value and a system administrator will not detect this form of "fraud".

Using totally random generated passwords will have the disadvantage that they are hard to remember. People will most likely write them down in order to not forget them. Or share them with colleagues. In both cases the purpose of authentication is lost. One such password can probably still be remembered but it is not unusual that many different applications have their own password and if a lot or even all of these passwords are randomly generated this will present problems. If someone would write down a password and store it in a secure place, this does not necessarily present a problem. If someone would have sole access to a vault in which he stores a written copy of all his passwords, then it is still considered secure.

We must first see what properties a good password has before we can tell how to compose good passwords. We list the following properties:

Hard to guess: Obviously the password must be hard to guess. An (educated) attacker will first try the name of the wife, dog and children with some variations in that, like a number or a date.

Easy to remember: The legitimate user should not have a problem of remembering the password.

Private: The owner of the password should be the only one to know it. The password authentication system only tests for knowledge of the password and not for the person having this knowledge.

Secret: It should not be written down in paper or stored in a computer in an unencrypted form where somebody might have access to it.

If no vault is present a user can still use smart tricks to get more or less random looking passwords that are difficult or even impossible to guess. A trick that can easily be applied by users is changing lower case letters into upper-case, changing letters into numbers (e.g. change 'l' into '1'), reversing or otherwise permuting the order of the characters within a word, adding numbers to the end (commonly used when forced to change a password frequently), or randomly placing a single special character in the password or replacing one letter by that character. Unfortunately all of this will not aid the user. Dictionary attacks on passwords are resistant against all of these elementary changes. The basic word will still come from a dictionary and an attacker who uses a dictionary attack is well aware of all the tricks users can apply and implements these also into the dictionary attack.

So what should be done to prevent a user from choosing a password that is vulnerable for a dictionary attack? In [8] the author advices to check the strength of password at the time it is entered. This makes sure that no easy guessable passwords can be used while the user is still able to determine his own password, i.e. choose something he can easily remember. He gives the following list of bad characteristics of a password and passwords should be rejected based upon them:

- Passwords based on user's account name;
- Passwords which exactly match a word (or a reversed word) in a dictionary, regardless if some or all of the letters are capitalized;
- Passwords which match a word in a dictionary with an arbitrary letter turned into a control character;
- Passwords which are simple conjugations of a dictionary word (i.e. plurals, adding "ing" or "ed" to the end of a word, etc.);

- Passwords which do *not* use mixed upper and lower case, or mixed letters and numbers, or mixed letters and punctuation;
- Passwords based on the user's initials or given name;
- Passwords which match a dictionary word with letters replaced by numbers (e.g. 'e' by '3' or 'o' by '0');
- Passwords which are patterns from the keyboard (e.g. "aaaaaa" or "qwerty");
- Passwords which only consist of numbers (e.g. a phone number, D-number, or bank account number).

But the question remains how the user must choose his easy to remember password, satisfying the above conditions. All of the before mentioned tricks can still work, as long as the starting point is not a word from a dictionary. A way to start is by choosing a sentence like "Yesterday I watched a nice program on television". Taking the first letters we get "yiwanpot", a word that most certainly will not appear in a dictionary. Of course there is no reason to use the first letter of each of the words. Using for example the second, the last, or even alternating between the first and the last letter of each word are all valid options. Remembering a sentence is much easier then remembering a random looking password. Probably some restrictions apply to the choice of the password like using numbers and special characters. We can either choose an appropriate sentence ("For breakfast I like to have at least 3 eggs and 1 toast!" becomes "FbIlthal3e&1t!") or adapt the password resulting from the chosen sentence (e.g. "yiwanpot" becomes "YIwanp0t!"). Now the only thing is to remember the chosen sentence. It is (relatively) easy to come up with a sentence that is easy to remember, certainly if it is used regularly, in which case the brain is more trained to remember it. But what if the password needs to be changed regularly? Or if many (different) passwords are needed. One trick for changing passwords is looking at the news and choosing a sentence related to a major news item. Or looking at the personal situation and choosing a sentence from that. For example "The authentication course I teach will take 6 weeks and will end on September 27th" becomes "TacItwt6w&weo9-27". A bit long, so probably a shorter sentence would be easier, but the only thing is if the user can remember the sentence.

There are more manners to get more secure passwords. In general passwords are stored in encrypted form so it is not possible for someone who has access to the stored password to retrieve the password itself. Obviously a user cannot use an elaborate encryption scheme to encrypt his password, but simple techniques are also usable. At “http://geodsoft.com/cgi-bin/pwcheck.pl” you can check the strength of a password. A simple password “testing1” was reported as a dictionary word. Shifting every character by one position (so ‘a’ becomes ‘b’, ‘b’ becomes ‘c’, etc.) the password is changed into “uftujoh3”, which looks random but apparently contains 1 dictionary word (“joh”) and 2 reversed dictionary words (“jut” and “hoju”). But doing the same trick again, getting the password “vguvkpi3”, we still were told that it contained the reversed dictionary word “vug”. Despite this the password passed the tests and the strength was given to be equal to 2.

A similar trick is to shift the first character by one position, the next by two positions, then three, etc. Our sample password transforms into “ugvxntn9”, which receives no comments at all and also has a strength of 2. By simply adding a ‘.’ behind it the strength goes up to 4, which means it is 100 times stronger then without the point at the end. Every time the strength goes up with 1 point, it is 10 times as strong. Obviously a longer starting password will give better results. For example the before mentioned password “FbIlthal3e&1t!” has strength 16 and “TacItwt6w&weo9-27” even has strength 20.

4.4 Pass faces and images

The idea behind pass faces and pass images is the same. The central idea is that it is easier to recognise then to recall. The passface product is described in [3]. During enrollment the user selects first whether he wants to use male or female faces. Then he selects 4 faces of his choice. During the authentication he is presented with 4 3x3 grids of faces. Each grid contains 1 of his selects faces. A recent visit of the demo on the website `www.id-arts.com` revealed that their product has changed slightly. Now 5 faces (combinations of male and female) are presented to the user and no choices can be made by the user. During authentication five 4x4 grids are presented. When deliberately making an error on the first grid no information was revealed about the incorrectness of this choice. Therefore the user (or attacker) only finds out after the fifth grid that (at least one of) his choices was wrong. From this we see that when an attacker must guess the pass faces he has $16^5 = 2^{20} \approx 1.000.000$ options. This is obviously an improvement from the previous version which had $9^4 = 6561$ options. In the first version the user could also select the face images, so knowing the specific preferences of the user could help in reducing this already small number of options.

Another application is called “Déjà Vu” and uses “random” images. The initial reason behind this was the before mentioned preference for specific faces or other kind of images which might decrease the security of the system. The images are based on mathematical functions and are rather abstract. During the enrollment the user can select a number p of images of the total set of images. During the authentication phase n images are presented of which m are from the selected set. To authenticate these m images must be identified correctly. The remaining $n - m$ images are called *decoy* images. This is called "m-out-of-n" authentication.

Now let us see how secure these two systems are in general or how this can be improved. From the articles it looks like all parameters are fixed. For the Passfaces we get 5 4x4 grids and from the Déjà Vu paper it looks like the parameters p, n, and m are also fixed. Let us assume the p is large enough and $m = 5$ and $n = 20$. So a user must recognise 5 pictures out of 20. The probability that a random choice is correct is 1 in $\binom{20}{5}$, i.e. 1 in 15504. Although this is a little bit better then a 4-digit PIN code, there are some drawbacks. It is in the advantage of the attacker to know that he has to choose exactly 5 images, no more and no less. Of course this also aids the legitimate user! Another drawback of the system is that the m images come from a set of p images. The system should be careful in selecting the images that are presented to the user. If the total database of images is very large and p, m, and n are relatively small the following could happen. An attacker could observe the pictures presented to the user every time he has to authenticate himself. Because the decoy images are drawn from a large set they are likely to be different every time. The correct images appear regularly and from this the attacker could start recognising them. So the set of possible decoy images must also be limited given specific values of p, m, and n.

For Passfaces the security per mxn grid is $\log_2(m \cdot n)$, so if k grids are used and the size of grid i is m_ixn_i, the entropy of the system is

$$\sum_{i=1}^{k} \log_2(m_i \cdot n_i).$$

For a single authentication we can say that the security offered by Déjà Vu is $\log_2(\binom{n}{m})$ is, but a more careful analysis must be made. As already mentioned before, it must be made sure that decoy images appear with the same probability as the chosen images. We assume the total number of images, from which the user can choose his p images, to be equal to N. For every authentication m chosen and $n-m$ decoy images are presented. So we find that the probability that a specific image is chosen is m/p for the chosen images and $(n - m)/(N - p)$ for the decoy images. If these probabilities are equal we find that

$$N = p + \frac{n-m}{m} \cdot p = \frac{n}{m} \cdot p,$$

so for $m = 5$ and $n = 20$, as before, we find that $N = 4 \cdot p$, as was to be expected.

Chapter 5

Protocols

In this chapter we will consider some authentication protocols. When using passwords, smart cards, biometrics or other means of authentication, there is an underlying protocol handling the actual authentication. A protocol is a process where information is send from the authenticator to the verifier and vice versa. This received information is processed at both ends and a decision is made whether authentication has been established or not. An introduction into the subjects of protocols and an easy example of a protocol is given in Section 5.1. In Section 5.2 we will see what different types of attacks can be launched against a protocol and in Sections 5.3 and 5.4 we will describe two examples of protocols.

5.1 Introduction into Protocols

As mentioned before, a protocol is no more or less then a set of communications between sender and receiver, following a strict rule of design. The content of each message is well described and deviation of this description should be regarded as a potential attack on the protocol. It is possible that more parties, beside sender and receiver, are involved in the protocol. In Protocol 5.1 also a trusted server S is involved. The two parties are always denoted by A and B, where A, in general, initiates the protocol. The notation in a protocol description is as follows. Each line consists first of a sequence number because a protocol has to be executed in the correct order. Next we find a notation like $X \rightarrow Y$:, indicating that X sends information to Y. Finally, the last part of each line is the actual information sent from the sender to the receiver. Furthermore by K_{XY} we denote a common key between X and Y. This can be a preshared key, a key being transmitted from X to Y or the other way around, or a key that only X and Y can calculate from previously transmitted information. A key K is used for encryption and by $\{M\}_K$ we denote the encryption of message M with key K. No assumptions are made on the encryption algorithm used, not even on the type (symmetric or asymmetric) of algorithm.

1.	$A \rightarrow S :$	A, B
2.	$S \rightarrow A :$	$\{B, K_{AB}\}_{K_{AS}}, \{A, K_{AB}\}_{K_{BS}}$
3.	$A \rightarrow B :$	$\{A, K_{AB}\}_{K_{BS}}$

Protocol 5.1: Simple Protocol

Protocol 5.1 shows a few things that need to be taken into account. First of all we must assume that any attacker C is capable of listening in on the messages that are being sent. Therefore we must encrypt all sensitive data like common keys that are transmitted. Furthermore we must assume that the attacker is capable of altering transmitted information. Any attacker should do this in a smart way, i.e. it should not interfere with the protocol description. In the described protocol an attacker should not for example add extra information to the second message for this would be detected by A. He could however change the identity of B in the first message to his own identity C. The trusted server S would not detect this because it looks like a normal first message to him. This would however be detected by A after receiving the second message. In the part that he can decrypt he would see an incorrect identity C, i.e. he would detect again that the rules of the protocol have not been followed and he would abort the protocol.

An attacker can play several roles within a protocol. He could *passively* listen to messages being transmitted and later try to retrieve secret information from these. He could more *actively* store them and replay them at a later stage with one of the two parties involved in the original protocol or even with a third party. An attacker could also play an active role and impersonate one (or more) of the true parties in the protocol. Suppose Protocol 5.1 is being used and an attacker C has at some point been able to get access to the key K_{AB}. The next time the protocol is initiated by A or B, the trusted server S will generate a new key K_{AB} and C would not be able to listen to the encrypted transmissions between A and B any more. He could at this point pose as S and retransmit the previous version of $\{B, K_{AB}\}_{K_{AS}}$ and $\{A, K_{AB}\}_{K_{BS}}$ with the known value of K_{AB}. Even if B would start the protocol he could simply change the order of the two parts in the second transmission and still fool B in believing that he is the trusted server S.

To make sure a *fresh* key K_{AB} is used, often a counter or a so called *nonce* is used. A nonce N_X is a random value generated by the user X. Making sure that he generates a new value every time he can make sure that the resulting information cannot be subject to a replay attack. The nonce must at a later stage return in a secure manner to the party that generated it so he can check that indeed it is unchanged. Protocol 5.2 has the same goal as Protocol 5.1 (using a trusted server S to obtain a secret key K_{AB} shared between A and B) but is resistant against the previously described replay attack. Both A and B find in the decrypted part their own nonce back, thereby having proof that the key K_{AB} is indeed fresh.

1.	$A \rightarrow B :$	A, N_A
2.	$B \rightarrow S :$	A, B, N_A, N_B
3.	$S \rightarrow B :$	$\{B, N_A, K_{AB}\}_{K_{AS}}, \{A, N_B, K_{AB}\}_{K_{BS}}$
4.	$B \rightarrow A :$	$\{B, N_A, K_{AB}\}_{K_{AS}}$

Protocol 5.2: More Complicated Protocol

5.2 Types of Attacks on Protocols

There are various different types of attacks on protocols. We will give a short description of a number of them here.

Eavesdropping: This is the most basic attack, where the attacker captures the information sent in the protocol. In this type of attack the attacker is said to be *passive*. A possible way to make sure that secret information is not disclosed is to use encryption of the sensitive parts of the information.

Modification: An attacker can also modify information transmitted in the protocol. The attacker will change parts of the transmitted information by, for example, replacing them with previously recorded parts of messages of the protocol. This attack can be prevented by using a cryptographic integrity check over all parts of the message simultaneously.

Replay: In this attack the attacker replays complete messages or parts of messages from a previous execution of the protocol. A way to avoid this thread is to add *freshness* to the messages from the protocol, e.g. by using randomly chosen nonces. The replay attack is a special form of the modification attack.

Reflection: This is a special form of a replay attack. In this case two simultaneous instances of the protocol are started and challenges from A to B are reversed by the attacker in the second protocol. The attacker reflects the challenge back to the challenger. To explain this attack see Protocol 5.3 and Attack 5.1 which is the attack on Protocol 5.3. In this case all the cryptographic calculations are performed by A while C poses as both B.

In the original protocol we see that A and B mutually and implicitly identify each other by checking the possession of the secret and shared key K. In Attack 5.1 attacker C poses as B and returns every challenge A thinks he is sending to B in the first protocol back to A self in the second instance of the protocol. In this case C will get a correct response to the challenge in the second instance, which he can then use in the first instance of the protocol to return to A.

Denial of Service: In this case the attacker prevents the legitimate users to engage in or complete a started protocol. The Denial of Service (DoS) attack is often aimed against a server who is used to interact with many clients. The DoS attack will either aim at consuming all the resources of the server or exhaust the maximal number of connections allowed to the server.

Typing Attacks: In this kind of attack a part of the message is replaced by another with a different meaning. The reason this will go unnoticed is because each message just a string of bits. The computer will interpret the strings according to the definition of the protocol, but an attacker can replace some parts of a message by something that looks like it. For example assume that in a protocol $\{N_A, M\}_{K_{AS}}$ has been transmitted from A to a trusted server S, where M is a part known to the attacker. Further on in the protocol A might expect to receive $\{N_A, K_{AB}\}_{K_{AS}}$ from either the server S or from B. The attacker can replace this encrypted part by the part that was first transmitted from A to S. In this case the replaced part looks correct to A if the length of M equals the length of K_{AB}. Because the message looks correct for the receiver A, he will accept, after decryption, the second part as the session key between A and B. But the attacker made sure that $K_{AB} = M$, which is known to him.

Cryptanalysis: Eavesdropping on a protocol might help an attacker to crypt-analyse the used cryptographic algorithms. Although in general the amount of data protected within each run of the protocol is small there are some things that make it vulnerable. For example in a protocol often information is send both encrypted and in plain. This will give the receiver some way of checking the integrity of the received message, but it will also give the attacker known plaintext-ciphertext pairs. If the key used for encryption is weak, a small number of such pairs might be enough to crypt-analyse the algorithm. A weak key might arise from the fact that it is derived from a password, which in general is not as strong as needed.

There are more attacks possible, like *preplay attack* and *certificate manipulation attack*, but we will not go into details here.

1.	$A \to B:$	$\{N_A\}_K$
2.	$B \to A:$	$\{N_B\}_K, N_A$
3.	$A \to B:$	N_B

Protocol 5.3: Protocol Vulnerable to a Reflection Attack

1.	$A \to C_B:$	$\{N_A\}_K$
1'.	$C_B \to A:$	$\{N_A\}_K$
2'.	$A \to C_B:$	$\{N_{A'}\}_K, N_A$
2.	$C_B \to A:$	$\{N_{A'}\}_K, N_A$
3.	$A \to C_B:$	$N_{A'}$
3'.	$C_B \to C:$	$N_{A'}$

Attack 5.1: Attack against Protocol 5.3

Above we described a number of attacks on protocols. In [1] the authors give a set of "rules of thumb" for designing a cryptographic protocol. These principles are derived from observing commonly made mistakes. Following these rules might help in designing a protocol that is not vulnerable for the above mentioned attacks, but possibly in the further new attacks or flaws in protocols will be found that call for an extension of the existing set of rules.

5.3 STS protocol

In [2] the authors describe a mutual authentication protocol using asymmetric cryptography. This protocol is referred to as *STS-protocol* which stands for Station-To-Station protocol. The STS protocol is an extension of the ordinary Diffie Hellman (DH) protocol. Its goal is to establish a common key in an authenticated way. The key $K_{AB} = g^{xy}$ is identical in form to the one established by DH, but the suers are authenticated within the protocol. Signatures are added to the protocol to provide for the authentication. By

1.	$A \rightarrow B:$	g^x
2.	$B \rightarrow A:$	$g^y, \{Sig_B(g^y, g^x)\}_{K_{AB}}$
3.	$A \rightarrow B:$	$\{Sig_A(g^x, g^y)\}_{K_{AB}}$

Protocol 5.4: STS Protocol

$Sig_X(M)$ we denote the signature by person X on the message M. The total protocol is given in Protocol 5.4.

It is easy to check that both A and B can construct the key K_{AB} before they need it to encrypt or decrypt information. Furthermore, because A and B are they only ones capable of finding K_{AB}, both parties are the only ones that can decrypt the signed messages and after that check the validity of the signatures. Moreover, the protocol also checks that both parties have the same (and correct) key K_{AB}, for otherwise, the decryption would result in rubbish instead of a true signature. Finally we see that a replay attack is not possible. Both parties know that their random value (x and y) are fresh and therefore this implies that the key K_{AB} is not the same as during an earlier execution of the protocol.

Despite all this, it is possible to attack the protocol. To describe the attack more clearly the STS protocol has been rewritten in Protocol 5.5. We see that the only addition to the original protocol is that the sender always includes the identification of both sender and receiver to a message.

1.	$A \rightarrow B:$	A, B, g^x
2.	$B \rightarrow A:$	$B, A, g^y, \{Sig_B(g^y, g^x)\}_{K_{AB}}$
3.	$A \rightarrow B:$	$A, B, \{Sig_A(g^x, g^y)\}_{K_{AB}}$

Protocol 5.5: STS Protocol with Identifiers

Now let us see what an attacker C can do. C will pose as B towards A, but will be himself towards B. Attacker C does little more then relaying the message received from A to B and from B to A. The complete protocol is described in Attack 5.2. At the end of the protocol A believes that he has participated in an STS protocol with B. C has started an STS protocol with B, but this was aborted after the second message. Because B never receives the third message from C, the protocol was, as far as he is concerned, not completed and therefore unsuccessful. The trick of C was to make A believe he has completed the whole protocol and has accepted that B is his communication partner sharing both the secret key K_{AB}. Note that the identifiers of sender and receiver at the beginning of each transmission are not necessary for attacking the protocol but are only there for clarity. Because C has no access to K_{AB}, all information protected by A with this key will not be disclosed to C. But A is the only person having this key, so all encrypted information can only be decrypted by himself. Attacker C has used B as an oracle to get the correct responses to the challenges posed by A, but C has gained no knowledge of any secret information.

1.	$A \rightarrow C_B :$	A, B, g^x
1'.	$C \rightarrow B :$	C, B, g^x
2'.	$B \rightarrow C :$	$B, C, g^y, \{Sig_B(g^y, g^x)\}_{K_{AB}}$
2.	$C_B \rightarrow A :$	$B, A, g^y, \{Sig_B(g^y, g^x)\}_{K_{AB}}$
3.	$A \rightarrow C_B :$	$A, B, \{Sig_A(g^x, g^y)\}_{K_{AB}}$

Attack 5.2: Attack on modified STS Protocol

5.4 ISO/IEC 9798

We are interested in protocols for the purpose of authentication. More specifically we are looking at *entity authentication*, as opposed to *data authentication*. ISO/IEC 9798-3 takes care of this entity authentication. Actually ISO/IEC 9798-3 is not one protocol but a set of 5 protocols, two of which provide unilateral authentication and the other 3 provide mutual authentication. The first protocol (see Protocol 5.6) is the easiest one. It is just one message with which A proves his identity to B. The message consists of a time stamp T_A, the identity of B, and added to this the signature of A on the previous information. The timestamp assures that the message is not replayed by an attacker and the included identity shows that A is aware of who he is sending the message to. This protocol can well be used for example in email or other means of one-way communication.

1.	$A \rightarrow B :$	$T_A, B, Sig_A(T_A, B)$

Protocol 5.6: ISO/IEC 9798-3 One-Pass Unilateral Authentication Protocol

Protocol 5.7 also provides unilateral authentication of A to B, but this time two messages are used. The only difference between Protocols 5.6 and 5.7 is that in the new protocol the timestamp is replaced by a nonce N_B generated by B. In the second message of the protocol A also includes his own nonce N_A. The reason for this is that he must not sign information that is purely based on information from B. If he would not include his own nonce B could misuse this protocol to have A sign messages of his choice with A's signature.

1.	$B \to A:$	N_B
2.	$A \to B:$	$N_A, N_B, B, Sig_A(N_A, N_B, B)$

Protocol 5.7: ISO/IEC 9798-3 Two-Pass Unilateral Protocol

The third protocol provides a two-pass mutual authentication. Actually it is not a new protocol, but merely two versions of Protocol 5.6, one initiated by A and a similar initiated by B. So naturally we need two passes instead of one and also we have mutual authentication instead of unilateral.

1.	$A \to B:$	$T_A, B, Sig_A(T_A, B)$
2.	$B \to A:$	$T_B, A, Sig_B(T_B, A)$

Protocol 5.8: ISO/IEC 9798-3 Two-Pass Mutual Authentication Protocol

In a more or less similar way we can extend Protocol 5.7 to a three-pass mutual authentication protocol as described in Protocol 5.9. But unlike the two-pass mutual authentication protocol, which was just a doubling of the one-pass unilateral authentication protocol, we can save one message here. So we have a three-pass protocol instead of a four-pass. The reason for this is that the nonce N_A need not be send as a separate message because it is already included in the response from A to B.

1.	$B \to A:$	N_B
2.	$A \to B:$	$N_A, N_B, B, Sig_A(N_A, N_B, B)$
3.	$B \to A:$	$N_B, N_A, A, Sig_B(N_B, N_A, A)$

Protocol 5.9: ISO/IEC 9798-3 Three-Pass Mutual Authentication Protocol

Originally there was another version of the fifth protocol which as also a three-pass mutual authentication protocol. The original fifth protocol is described in Protocol 5.10

1.	$B \to A:$	N_B
2.	$A \to B:$	$N_A, N_B, B, Sig_A(N_A, N_B, B)$
3.	$B \to A:$	$N'_B, N_A, A, Sig_B(N'_B, N_A, A)$

Protocol 5.10: Original fifth ISO/IEC 9798-3 Three-Pass Mutual Authentication Protocol

The difference between Protocols 5.10 and 5.9 is that in the third message B does not reuse the old version of the nonce N_B from the first message, but he generates a new nonce N'_B. But this original fifth protocol is vulnerable for an attack as described in Attack 5.3. The attacker C would pose as B when initiating this protocol. But after the second message he would initiate Protocol 5.7 with B, posing as A and sending the nonce N_A he just received from A in their second message. The response received from B would C use to send back to A, again posing as B. So C uses B as an oracle to get the correct response that can be send to A.

1.	$C_B \rightarrow A :$	N_C
2.	$A \rightarrow C_B :$	$N_A, N_C, B, Sig_A(N_A, N_C, B)$
1'.	$C_A \rightarrow B :$	N_A
2'.	$B \rightarrow C_A :$	$N_B, N_A, A, Sig_B(N_B, N_A, A)$
3.	$C_B \rightarrow A :$	$N_B, N_A, A, Sig_B(N_B, N_A, A)$

Attack 5.3: Attack On Original Fifth ISO/IEC 9798-3 Protocol

The final fifth protocol of the ISO/IEC 9798-3 standard is just two copies of Protocol 5.7 executed in parallel, once initiated by A and once by B.

1.	$A \rightarrow B :$	N_A
1'.	$B \rightarrow A :$	N_B
2.	$A \rightarrow B :$	$N_A, N_B, B, Sig_A(N_A, N_B, B)$
2'.	$B \rightarrow A :$	$N_B, N_A, A, Sig_B(N_B, N_A, A)$

Protocol 5.11: ISO/IEC 9798-3 Two-Pass Parallel Authentication Protocol

Chapter 6

Tokens and Smart Cards

A token is some piece of hardware you posses and use to authenticate yourself. Obviously, to do so, you need to keep the token with you at all times and make sure that nobody else can have access to it. If a token is lost, even if it is recovered afterwards, the token must be considered compromised and should be replaced. In Section 6.1 we will describe a number of possible tokens we can encounter and describe some general properties of tokens. In Section 6.2 we will describe how authentication using tokens can happen.

6.1 Different Kinds of Tokens

A token is a piece of hardware containing some secret information. The term secret should be considered loosely here and in some occasions be read as "unique" instead of secret. This secret information acts in a similar way as a password. It is "remembered" by the token and presented when authentication is needed. The obvious advantage of a secret on a token compared to a password is that a token in general will not forget the secret. Therefore it can be as long as needed and completely random. In some cases the token can also process the secret, e.g. use it as a key in an encryption algorithm. In such a case we say that the token is *active*. Examples of active types of cards are smart cards and one-time-password generators. The opposite of active is *passive* and a token is said to be passive if it is only used at a storage device, e.g. a mechanical key or an access card.

There are also some disadvantages connected to the use of tokens. They are expensive and a user must not forget to bring it. If a card is lost or stolen, it could be copied, thus whenever it is displaced for a shorter or longer time, the secret on the token must be considered compromised and the token must be blocked and replaced. There is also the aspect of costs. Tokens by themselves are expensive, but additional costs come from the readers of the tokens.

In the remainder of this section we will describe a few samples of tokens in more detail.

6.1.1 Internet Banking

When using internet banking in general one of two methods of authentication is used. In the first method a user is asked to copy a specific number from a list of (say) 100 (random) numbers. This list of numbers should be kept secret by the user and each number will be used only once. The list of numbers is also a token: a peace of hardware with a secret "inside". Only in this case the user must take care that it is stored securely to keep the numbers indeed secret.

The other method uses a so-called *one-time-password* generator. This is a small device that can communicate with the chip on the banking card and can check the validity of that card. The user is then given a challenge being a 6 digit number. This number is entered into the one-time-password generator and its response will be an 8 digit number. This response is then returned to the internet application and send to the bank where it is checked for correctness. These devices can be time dependant, meaning that the response depends on both the time and information on the banking card. This is however more complicated because the password generator and the server at the bank should have synchronized time which cannot be synchronized because there is no physical connection between the two. In case the response is time dependant it should be returned within a certain timeslot to make sure that the authentication goes well.

6.1.2 Dongles and Soft Tokens

A dongle is piece of hardware that can be connected to an IO-port of the PC, e.g. to a USB-port. Stored on the dongle is a secret key needed for authentication. In this case either authentication of a piece of software or of the owner of the dongle. Other terms used for a dongle are hardware key, hardware token or even security device. A soft token is just another name for a floppy disk that contains a secret key, needed for authentication, in software.

The difference between a dongle and a soft token that it is very easy to get the key out of the floppy disk but it is supposed to be hard to get it out of the dongle. The access to the dongle should be protected and the key on the dongle will never leave the dongle in the ideal situation. The key will be used to encrypt or decrypt information inside the dongle for the authentication protocol. The key on the floppy is either unprotected or protected using an easy mechanism, e.g. a password. Of course the floppy itself must be protected physically and the mechanism to protect the key should not be weak. Nowadays floppies are often replaced by secure servers.

6.1.3 Smart Cards

There are 2 main types of smart cards. In general is a smart card just an electronic storage system. The difference between the two types of smart cards comes from the fact that some of them are capable of computations while others just store information. A smart card has (in general) the size of a credit card and can be recognised from the galvanic connectors on the outside of the card. One of these connectors is used to transmit information to and from the smart card. Other connectors are used for example for powering the smart card or to provide a clock to the smart card.

The typical architecture of a memory card consist of an EEPROM and an sequential logic (state machine) that is used to access the EEPROM. Possibly also a simple security logic is incorporated to protect the contents of the EEPROM a little better. Such a logic will in general be a stream cipher logic. There is not much flexibility in the applications of these kinds of cards due to the lack of computational power. On the positive side, they are very cost effective and for that reason are they used a lot in large-scale applications where security is not the major issue.

The typical architecture of a smart card that has a microprocessor incorporated is more complicated. Beside the EEPROM we can find a ROM memory with an operating system that controls the access to the EEPROM. The smart card contains also RAM memory and a CPU. The RAM memory is just temporary working memory for the CPU and will be lost when the power is disconnected. This type of smart card has much more flexibility in the choice of the application. The EEPROM contains application specific software and data. It is even possible that more then one application is present on the smart card. Examples of these kinds of smart cards can be found in mobile phones and banking cards.

6.1.4 Wireless Token

Wireless tokens are closely related to smart cards. The difference is that a wireless token has no connectors on the outside of the card. Instead it has on the inside a transponder that, when close to a card reader, will be powered using electromagnetic fields. When the transponder is not close to the reader it will be completely passive.

We can distinguish two types of transponders, the transponder with a antenna coil (left in Figure 6.1) or with a bipolar antenna (right in Figure 6.1). The reader uses a magnetic field to induce a current in the coil/antenna. The wireless token uses a weaker magnetic field for its replies. Other information like timing pulses and data is also transmitted via these magnetic fields.

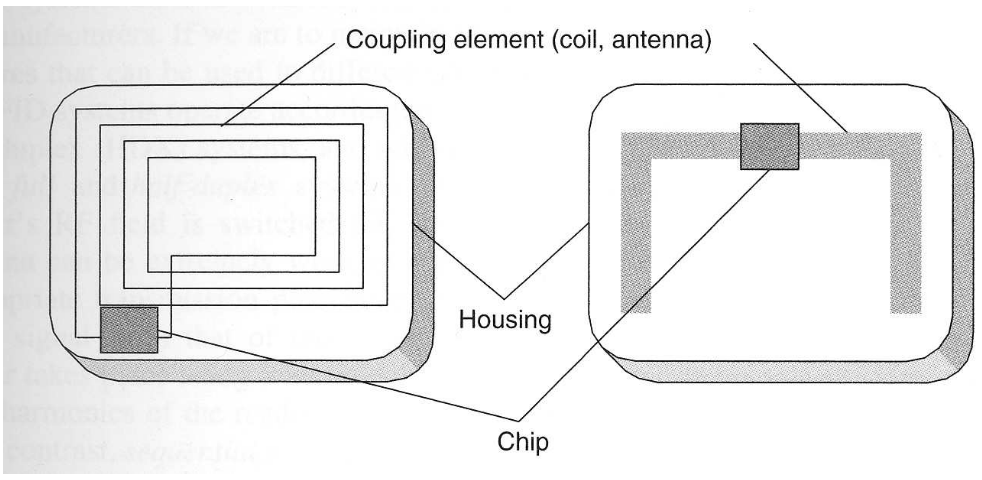


Figure 6.1: Wireless Token Transponders

6.2 Token Authentication

When using smart cards and smart card readers, the goal of authentication is to assure that the smart card is communicating with a genuine smart card reader and vice versa. For this purpose the smart card and the reader both possess a common secret key and the opposite party must be convinced of the knowledge of this secret key. Of course this should be achieved without disclosing the secret key, even towards the other party. To protect against replay attacks, this procedure should not be static (i.e. always based on the same information) but dynamic.

There is a fundamental difference between unilateral and mutual authentication. In the first case only the smart card authenticates itself towards the reader or the other way around. Mutual authentication means that both parties authenticate each other.

The authentication process used in smart cards is based on cryptographic protocols as described in Chapter 5. One of the protocols that can be used for mutual authentication is similar to the ISO/IEC 9798-3 three pass mutual authentication protocol as described in Protocol 5.9. In this ISO/IEC 9798-3 protocol asymmetric crypto is used to have signatures on data that is being transmitted. Most smart cards are however not capable of performing asymmetric crypto. But when using symmetric crypto we need a shared and secret key K.

The adapted protocol can be found in 6.1. We used T for terminal and C for card. We see that not only the signature has been replaced by symmetric encryption but that also the plain information has been removed. One of the reasons for this is that we want to minimize the transmission time. Another reason is that otherwise an attacker could get access to multiple plaintext-ciphertext pairs which could help him crypt-analyse the data and possibly find information on the key K.

1.	$C \rightarrow T:$	N_C
2.	$T \rightarrow C:$	$E_K(N_T, N_C)$
3.	$C \rightarrow T:$	$E_K(N_C, N_T)$

Protocol 6.1: First Version of Token Authentication Protocol

We see that the card C and the terminal T use the same symmetric key K. The terminal must be able to communicate with all possible smart cards. This could be possible by sharing the same key K amongst all cards and terminals. Naturally this implies that if one of these keys gets compromised, the whole system is compromised. On the other hand if every card C would have its own key K_C, the terminal needs to store many different keys which might not be practical. For this reason a mechanism has been found that makes sure that every card has its own unique key K_C but that each terminal also only has one key. Beside the unique key K_C every smart card also stores its identity C. The key K_C is actually derived from this identity C, e.g. by encrypting it using a master key K_M. This master key is now stored in the terminals. A terminal can now find the card specific key from the cards identity and the master key, e.g $K_C = Enc_{K_M}(C)$. The first step in the authentication protocol will then be that the terminal requests the identity of the card and computes the card unique key from that information. The total description can now be found in Protocol 6.2. The transmissions in step 1 and 3 in this protocol are of course not the texts written in this description but codes representing these messages. After step 2 the terminal is capable of computing K_C from the received information C and the secret master key K_M.

1.	$T \rightarrow C:$	Get card ID
2.	$C \rightarrow T:$	C
3.	$T \rightarrow C:$	Get random nonce
4.	$C \rightarrow T:$	N_C
5.	$T \rightarrow C:$	$E_{K_C}(N_T, N_C)$
6.	$C \rightarrow T:$	$E_{K_C}(N_C, N_T)$

Protocol 6.2: Token Authentication Protocol

Chapter 7

Biometrics

Biometric identification has a long history. People have always recognized others by their face, voice, and gait and more recently also signatures. Shakespeare gave various hints towards gait or other biometric recognition, e.g. "Great Juno comes; I know her by her gait" from "The Tempest". The first scientific literature (see for example [11]) dates back to the 1870's where Alphonse Bertillon describes a system of body measurements for identifying people. This system was used until the 1920's in the USA to identify prisoners. Work on fingerprint recognition started in the 1880's by Henry Faulds, William Herschel and Sir Francis Galton. Latent fingerprints were collected at criminal sites and manually compared to cards with fingerprints from known criminals. This was a very labour intensive task with the every increasing number of fingerprint cards. Other biometric features are less long used for authentication purposes. Hand geometry, voice, signature and retina recognition have been used since the 1980's and commercial face and iris recognition has been around since the 1990's.

A brief introduction into the general aspects of biometrics is given in this chapter. In the subsequent chapters we will go into more detail of biometric authentication using specific characteristics. In Chapter 8 we will describe fingerprint recognition in more detail. Face recognition will be described in Chapter 9. Fingerprints and faces are called *physiological* or *static* biometric features. Another type of biometric features is called *behavioural* or *dynamic* and authentication is this case based on the fact that people perform specific tasks in a similar manner every time. Examples are writing a signature (see Chapter 9) and the way of typing at a keyboard (see Chapter 11).

The two sources used for this chapter are [6] and Chapter 1 of [13]. Although the approach in both sources differs slightly, we try to make one consistent part out of it.

7.1 Introduction

There are many biometrics features (also called biometric characteristics) that can be used to authenticate humans. The best known are fingerprint, face, iris and voice recognition. But also less known biometrics can be used like odor and footprint. According to [6], any biometric characteristic needs to satisfy some properties in order to be practical:

Universality: Each person should have the characteristic.

Distinctiveness: Any two persons should be sufficiently different in terms of the characteristic.

Permanence: The characteristic should be sufficiently invariant over a period of time.

Collectability: The characteristic can be measured quantatively.

Performance: The achievable recognition accuracy and speed, the resources required to achieve the desired recognition accuracy and speed, as well as the operational and environmental factors that effect the accuracy and speed.

Acceptability: The extent to which people are willing to accept the use of a particular biometric identifier in their daily lives.

Circumvention: Wow easily can the system be fooled using fraudulent methods.

The first four properties are really needed for everybody to be able to use the system and to make sure that the system can indeed distinguish between various persons. The last 3 properties are needed to make the system practical. Biometrics can, like passwords and tokens, be used for both authentication and identification. In general authentication and identification is used for *positive recognition*. People want to prove their identity to someone else or to a system. The aim of positive recognition is to prevent that multiple people use the same identity. On the other hand could we also aim at making sure that one person is not using multiple identities. This is exactly the aim of *negative recognition*, where we try to bind an identity to a person who claims not to be that identity. This is mostly used in forensics, e.g. binding latent fingerprints to specific persons. This property of negative recognition is specific for biometrics. Other methods of authentication cannot achieve this aim (it is not possible to prove *not* knowing something).

Biometric identifiers can be divided into two categories, being *physiological* and *behavioural*. The first category, also called *static*, contains the physical features that are "unchangeably" connected to a person, like a fingerprint or his/her DNA. A behavioural biometric, also called *dynamic*, consists of the way a particular action is carried out, e.g. writing a signature, speaking or walking. Every time a person performs on of these actions, his performance is a little bit different from all of the other times the action was performed. On the other hand the way one person performs the action is (in the ideal case) very different from the way another person performs the same task. Behavioural biometrics can, like physiological biometrics, change gradually or drastically over time. The writing of ones signature might "mature" over time. The way somebody walks might drastically change due to an accident.

7.2 Biometric system

Any biometric system consists of two subsystems. One is the enrollment system where new users are added to the system. The other is the authentication system where the identity of the user is checked against the data obtained during enrollment. This is depicted in Figure 7.1.

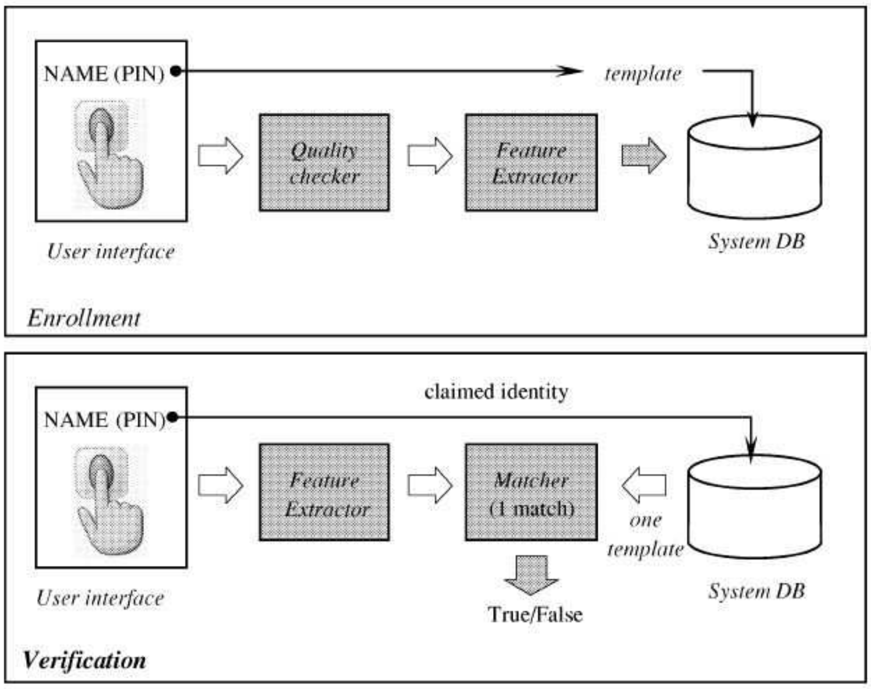


Figure 7.1: Biometric Subsystems: Enrollment and Authentication

The first subsystem is the enrollment subsystem. Here the user is enrolled and his biometric features are measured and transformed into a template. Before the features are transformed, the quality of the measured features is checked. The quality of the biometric characteristic, i.e. the quality of the template, should be as high as possible because it is used every time during authentication. For example, when using gait authentication, it is not a smart idea to store a template of a walk where the user almost stumbled. This is obviously not normal walking behaviour and the quality check should prevent such walks from being entered into the system.

Feature extraction is used to get some relevant information out of the biometric feature that will be also extracted from the input biometric during authentication. The two sets of features can then be compared. For example when using fingerprints the positions of the minutiae (ridge endings and ridge bifurcations) can be extracted, but also the directions of the specific minutiae and their types can be recorded. The extracted information depends on the way the system tries to match the enrolled data with the extracted data during the authentication. The extracted features are stored in what is called a *template*. In many systems not just one template is stored, but a number of them, depending on the system. For example for signature authentication it is useful to store a small number of template signatures (say 5 or 10), because the differences in signatures might be rather big. For fingerprint authentication one template of high quality might be more then enough.

During authentication more or less the same happens as during enrollment. The user again presents his biometric characteristic to a reader. From this the biometric properties are extracted and then they are matched against one or more templates in the database. When performing authentication, it will only be matched against the templates that correspond to the claimed identity of the user. A decision rule must be specified if more then 1 template is stored in the database. It could be enough that the input matches with at least one template or it must match at least a given number. It is even conceivable that a rule could be that it matches at least 1 template when a high threshold is used, while it also matches almost all templates with a lower threshold. Many rules are possible for this.

There are some specific differences between enrollment and authentication. Use of the equipment during enrollment will in many cases be supervised to ensure that new users know how to operate the equipment. Also the supervisor can control the process and see if the quality of the captured biometric feature is good enough. Another difference is that this quality is not checked during authentication. The check is needed during enrollment to ensure the quality of the templates in the database. During authentication it could happen that the quality of the feature is not good enough. This will simply result in a rejection and the user needs to present the biometric characteristic again. So implicitly there is a check on the quality, but not one implemented in the system.

7.3 Errors

As mentioned before do biometric systems have the advantage that, unlike any other means of authentication, they can be used for negative recognition. But there is also a specific drawback attached to biometric authentication systems. When checking a password or token, the outcome of the authentication is clear and unambiguous. Either the entered password is correct or it is false. Either the PIN code is correct or it is wrong. Either the token is authentic or it is false. Either a physical key fits the keyhole, or it does not. There is no in-between. A PIN code can not be partially correct. But biometric features never match for exactly 100%. When looking at the details of a fingerprint, we always find some matches and some non matches, due to various factors. But the more matches we find, the more convinced we are that the correct person is trying to authenticate himself.

We express the similarity between the extracted features in a template and the extracted features during the authentication phase in a *matching score*. This matching score is compared to a threshold value and the matching score should exceed the threshold value to authenticate a person positively. This threshold value is a fixed value and during the matching some errors might occur. Obviously an impostor could get a high matching score and thereby authenticating himself falsely. We say that a so called *false match* or *type I error* has occurred. On the other hand a legitimate user could also be rejected when trying to authenticate himself due to a low matching score. In such case we have a *false non match* or *type II error*. The *False Match Rate (FMR)* and *False Non Match Rate (FNMR)* are defined as the probability that an authentication suffers from a type I or II error. r could get a high matching score and thereby authenticating himself falsely. We say that a so called *false match* or *type I error* has occurred. On the other hand a legitimate user could also be rejected when trying to authenticate himself due to a low matching score. In such case we have a *false non match* or *type II error*. The *False Match Rate (FMR)* and *False Non Match Rate (FNMR)* are defined as the probability that an authentication suffers from a type I or II error.

The FMR and FNMR are naturally dependant on the threshold that is used. In order to decrease the FMR the threshold needs to be increased, thereby also increasing the FNMR. On the other hand does lowering a threshold mean that the FNMR decreases while the FMR increases. The appropriate threshold is dependant on the specific application. In case of access to a high security area the FMR must be as low as possible, at the cost of inconvenience for legitimate users, who more often need to re-enter their biometric feature due to the higher FNMR. On the other hand we could imagine paid access to internet at an internet cafe. In order to keep customers satisfied the FNMR must be low, implying that the FMR increases and some loss of money might come from the fact that impostors mimic other peoples biometric feature. The distribution of probabilities of an impostor and a legitimate user for a correct authentication are presented in the left part of Figure 7.2. We see that the FMR equals the area under the *genuine distribution* curve to the left of the threshold. The FMR equals the area under the *impostor distribution* to the right of the threshold.

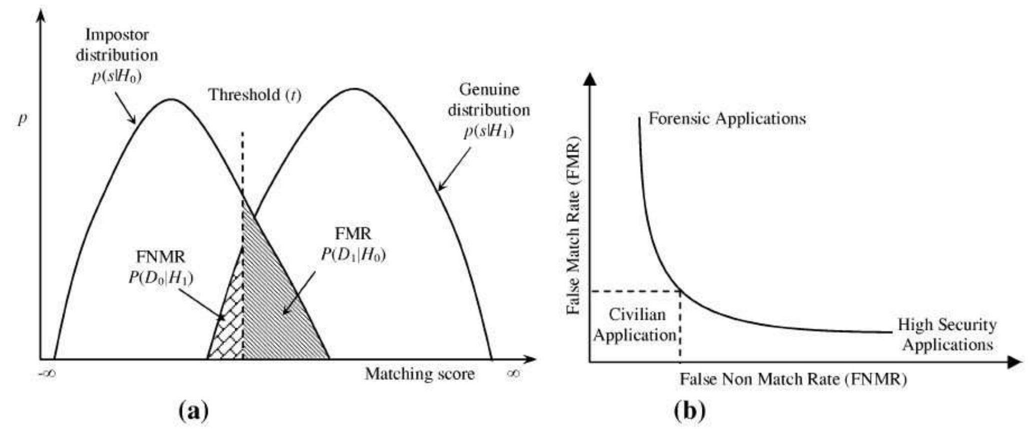


Figure 7.2: Biometric Errors and ROC curve

The graph on the right hand side of Figure 7.2 is a so-called *Receiver Operating Characteristics (ROC)* curve. Here the FMR is plotted against the FNMR for varying values of the threshold t. A ROC curve is often used to compare various biometric products. The curves for the various products are plotted in one ROC curve and are compared in the area of interest of the end user. Also we can find the so-called *Equal Error Rate (EER)* which is defined as the error rate for which the FMR and FNMR are equal. This can be found from the curve by intersecting it with the $x = y$ line.

Beside the terms FMR and FNMR also the terms *False Acceptance Rate (FAR)* and *False Rejection Rate (FRR)* are used. The relation between FMR and FNMR on the one hand and FAR and FRR on the other hand is very simple. In a positive recognition system we simply have FAR=FMR and

FRR=FNMR. In a negative recognition system these are precisely reversed, i.e. FAR=FNMR and FRR=FMR. As we are more interested in authentication then in identification, i.e. in positive recognition systems, we will use the terms FAR and FRR as described for such systems.

Beside false acceptances and false rejections also other errors can occur when a biometric system is used. One of the desired properties of a biometric system is "Universality", i.e. everybody should be able to use the system. Obviously this cannot be enforced all of the time. Just think of a system based on gait used within a company to allow people to enter the facilities. People in wheelchairs would not be able to use this system. Also people without arms or hands cannot enroll in a fingerprint system. The fraction of people that cannot enroll in the system is measured with the *Failure to Enroll Rate (FER)*. As explained in Section 7.2, the quality of the extracted biometric feature will be checked during enrollment. Depending on the chosen level of quality needed to ensure good performance during authentication at a later stage, it might be possible that people cannot enroll. If, for reasons of performance during authentication, the quality of the template at a fingerprint system should be of very high quality, people might get rejected from entering the system. People who do a lot of hard physical labour with their hands can have very poor quality fingerprints and could therefore be rejected.

There is a trade-off between FAR and FRR on the one side and FER on the other side. By setting high quality standards during enrollment the FER might increase, but as a result the quality of the templates in the database is very high. This at least eliminates authentication errors due to bad quality of the template. Given a fixed threshold this implies that the FAR will go down and the FRR will go down. So also the EER will go down because of this.

A final error rate that we briefly discuss here is the *Failure to Capture Rate (FCR)*. The FCR is the percentage of unsuccessful tries when capturing the biometric feature. This type of error occurs when the device is not able to locate the biometric feature when presented with it. For example it is not able to locate the face due to bad light conditions or it is not able to capture a fingerprint because of a dirty reader. Obviously the performance of a biometric system is related to the FCR.

7.4 Examples of Biometric Features

In this section we will give a short description of a large number of possible biometrics.

DNA: DNA stands for Deoxyribonucleic Acid and is the ultimate building block for human beings. DNA is unique for every person, except that identical twins have identical DNA patterns. It is used mainly in forensic applications for negative recognition.

Ear: It is suggested that the shape of the ear can be used for recognition. This can be achieved by matching the distances between salient points on the pinna of the ear.

Face: For more information on face recognition see Chapter 9.

Facial, Hand and Hand Vein Infra-red Thermogram: The heat radiation pattern of the human body is unique and can be captured using infra-red cameras. Capturing a clear image is difficult due to noise caused by other humans present or by other heat sources near the persons. Infra-red sensors are too expensive to make this technology commercially attractive on a large scale in the near future.

Fingerprint: For more information on the way fingerprints are used for authentication see Chapter 8.

Gait: For more information on gait recognition will be included in a newer version of this document.

Hand and Finger Geometry: A number of measurements are taken from the hand of the user, including shape, size of palm, and lengths and widths of fingers. This information is used to authenticate people. Many commercial hand geometry based systems are installed already around the world. The method is simple, easy to use and inexpensive. The method is not very distinctive so it cannot scale to large user groups. During growth period of children it is not sure if geometry information stays invariant. Although the method does not seem to effected negatively by factors like weather or dry skin, it might be vulnerable to other things like jewellery or illnesses like arthritis.

Iris: The iris is the area of the eye bounded by the pupil and the sclera (white of the eye). Apart from the first few years of childhood and apart from eye diseases like cataract, the iris remains unchanged. Its texture contains very distinctive features that can very well be used for authentication. Although early days iris based systems were very intrusive, new systems have become more user friendly and also cost effective. Irises of identical twins are different and systems are not fooled by artificial irises like designer contact lenses.

Keystroke: For information on keystroke authentication see Chapter 11.

Odor: Peoples odor is very characteristic and decomposition of its chemical components could be used for human recognition. It is not yet known whether it is invariant over time or if deodorant use can interfere with the human odor in a negative way for authentication.

Palmprint: Palmprints can be used in a similar way as fingerprints. Palms also contain ridges and values with all their distinctive minutiae. Palms are much bigger and contain therefore more information which can be used for authentication. On the other hand is this specifically the reason palm readers need to be much bigger making their usage less convenient.

Retinal Scan: The retinal vasculature is supposed to be the most secure biometric. It is supposed to be very characteristic and at the same time very hard to change or replicate. Acquisition however is not very user friendly because the user has to look into a special eye piece and focus on a particular spot. This is the main reason against public acceptance of this system.

Signature: For more information on signature verification see Chapter 10.

Voice: Voice is both a physiological and a behavioural biometric. Voice based recognition can be text-dependent or text-independent. Voice is not very distinctive and may therefore not be suitable for large scale applications. Furthermore someone's voice may change over time and can be very sensitive for changes due to medical conditions, for example a cold. An additional problem can be background noise. A well suited application for voice authentication is in phone based applications. Here the quality of the voice signal is further reduced due to the quality of the equipment (like the microphone) and the communication channel.

7.5 Comparing Metrics

As explained before, when comparing extracted biometric features, we need a distance metric to measure how close together or how far apart two samples are. The distance metric that will be used will very much depend on the biometric feature that is used. For example for gait we have acceleration measurements in 3 perpendicular directions, sampled at a rate of 100 samples per second, and each sample a 12-bit value. A step will, depending on the specific person, takes about 1 second, or equivalently 3*100*12/8 = 450 bytes. More accurately we might say that we have 3 times 150 bytes per step because we can use each of the 3 perpendicular directions independently. In a fingerprint system, most commonly minutiae will be extracted, and each minutia will be described using 27 bits,i.e. almost 4 bytes. The number of extracted minutiae lies generally in the range 10-100, so the extracted data will be approximately 400 bytes, but templates of up to 1024 are not uncommon.

But not so much the difference in size of the templates is the main reason for not being able to use a single distance metric. More the interpretation of the information in the template. For a fingerprint the template is likely to consist of several positions of minutiae. A good way to compare the positions of two minutiae might be to use the Euclidean Distance (metric d_2 in Section 3.3). When looking at gait we see that steps are never exactly equally long, so we have to deal with comparing sequences of unequal length. A technique called *Dynamic Time Warping (DTW)* is suitable to compare sequences of unequal length and might be best used in gait authentication. The technique is briefly explained in [14] where it is used for online signature verification.

This diversity makes it not possible to come up with a single useful distance metric. The general metrics, described in Section 3.3, can be used as building blocks to more elaborate distance metrics, tailored to their specific use. We will have a closer look, when applicable, at specific distance metrics for every biometric feature that we describe in more detail in the next chapters.

7.6 Good biometrics

It is not possible to classify one specific biometric as the best solution in all situations. Situations and user demands differ too much to make this possible. In some cases even a combination of two biometrics will have preference over a system with only one. Table 7.1 is copied from [6].

Biometric Feature	Univ	Dist	Perm	Coll	Perf	Acce	Circ
DNA	H	H	H	L	H	L	L
Ear	M	M	H	M	M	H	M
Face	H	L	M	H	L	H	H
Facial Thermogram	H	H	L	H	M	H	L
Fingerprint	M	H	H	M	H	M	M
Gait	M	L	L	H	L	H	M
Hand Geometry	M	M	M	H	M	M	M
Hand Vein	M	M	M	M	M	M	L
Iris	H	H	H	M	H	L	L
Keystroke	L	L	L	M	L	M	M
Odor	H	H	H	L	L	M	L
Palmprint	M	H	H	M	H	M	M
Retina	H	H	M	L	H	L	L
Signature	L	L	L	H	L	H	H
Voice	M	L	L	M	L	H	H

Table 7.1: Comparison of Various Biometric Features

Chapter 8

Fingerprint

In this chapter we will describe the processes involved in fingerprint recognition. There are various processes involved, being:

Acquisition: The process of capturing a fingerprint and representing it in an appropriate format.

Authentication: The process to determine whether or not 2 fingerprints are from the same finger.

Identification: The process to determine, given an input fingerprint, its matching template fingerprint in a database, if any.

Classification: The process to determine to which pre specified category a fingerprint belongs.

For the remainder of this document we will only be interested in the first two processes: acquisition and authentication. Both these processes must first capture a fingerprint to extract from it the information that can be used further on. The term *minutiae* will be used to denote the interesting points on a fingerprint. As a part of the acquisition process, the extracted information is stored in a database as a template for future use in one of the other 3 processes. During authentication, the extracted information is matched with a template from the database. In practise both minutiae extraction and minutiae matching are complex tasks.

In the remainder of this chapter we will describe the steps involved in the processes of fingerprint acquisition and fingerprint authentication. These steps are *fingerprint capturing*, *minutiae extraction*, and *minutiae matching*. But before we will describe these steps we will in the next section first describe general ideas on fingerprint recognition, needed to understand the other sections.

8.1 Fingerprint Features

In this first section we will describe features of the fingerprint and the process of fingerprint acquisition and recognition, needed to understand the remainder of this chapter. It is well known that a fingerprint is unique for every person. Even identical twins have different fingerprints. What we call a fingerprint is actually a pattern of *ridges* and *valleys*. When we see a picture of a fingerprint the dark lines are the ridges and the white spaces in between are the valleys.

Fingerprints were first used in the second half of the 19th century by the police in criminal investigations. In those days fingerprints were collected in ink on 10 finger cards. These were later compared to fingerprints found at a crime scene. Fingerprints were classified according to their global structure and the process of matching was done by human experts who visually compared the prints. On a global level we can distinguish for example *arches*, *loops*, and *whorls*. These are depicted in the Figure 8.1.

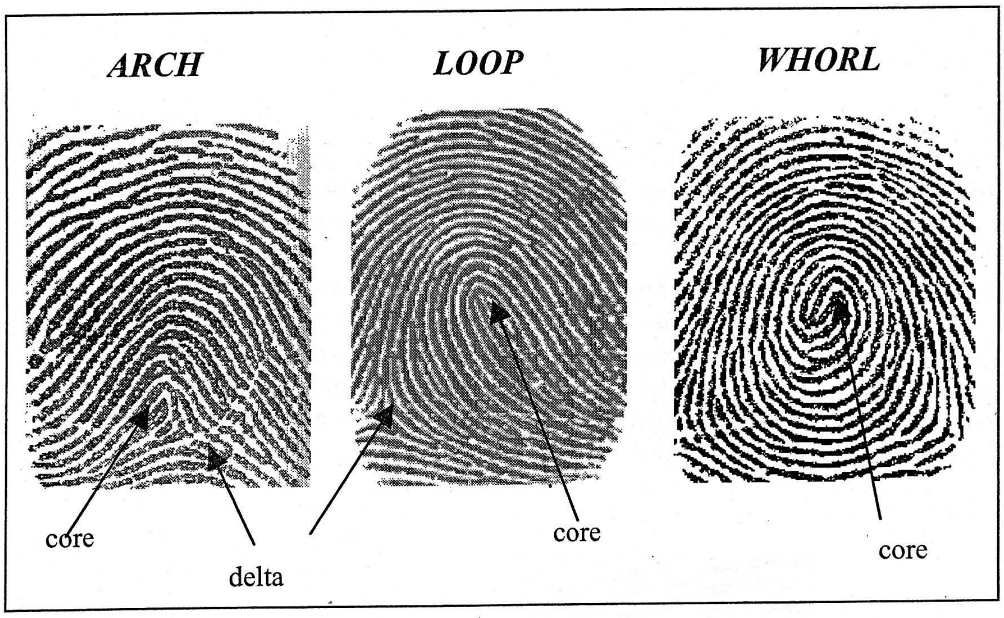


Figure 8.1: Global Fingerprint Patterns: Arch, Loop and Whorl

On a local level also distinct features can be seen on a fingerprint. We take a closer look at the ridges and see that at some points they end or split or show other specific patterns. These patterns are called minutiae and the most common types of minutiae are displayed in Figure 8.2.

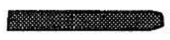
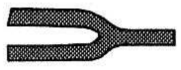
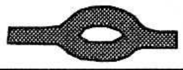
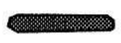
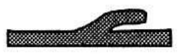
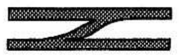
	Termination
	Bifurcation
	Lake
	Independent ridge
	Point or island
	Spur
	Crossover

Figure 8.2: Most common types of Minutiae

All of these distinct types of minutiae can be used to describe a fingerprint, but only 2 types are used in general, being the *ridge termination* (or *ending*) and the *ridge bifurcation*. These two minutiae are very closely related and are in fact each others duals. The termination of a ridge is also the bifurcation of a valley and vice versa. An example of this is shown if Figure 8.3.

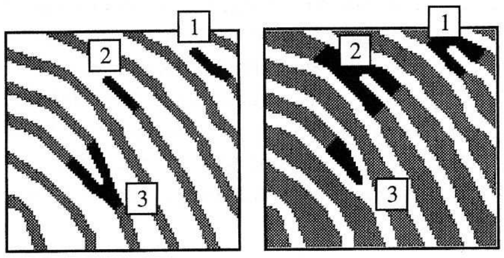


Figure 8.3: Duality between termination and bifurcation

The tasks of minutiae extraction is now to determine where the ridge terminations and bifurcations are located in the image of the fingerprint. But not only the position is determined, also the direction is determined. Looking at Figure 8.4, we see that the direction of a ridge ending is the angle that the minutia tangent makes with the horizontal axis. The direction of a ridge bifurcation is defined as the angle with the horizontal axis of the corresponding valley ending.

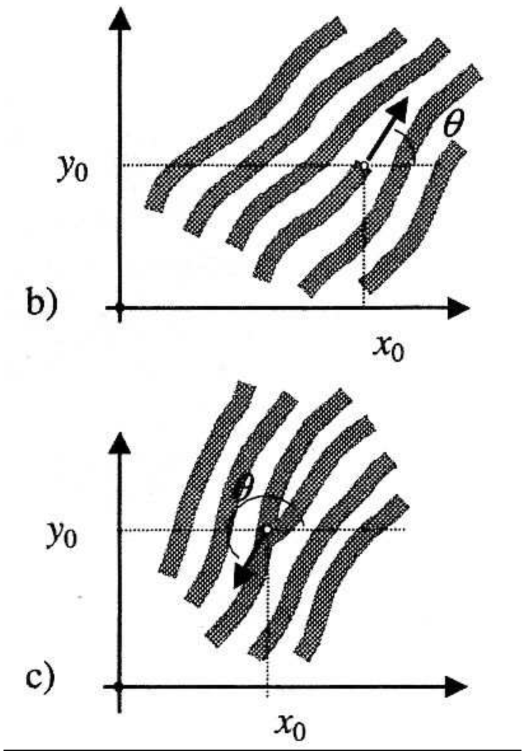


Figure 8.4: Orientation of termination and bifurcation

Now that we know what we are looking for, we can detect and extract the minutiae (see Section 8.3) and having done that we can try to match the minutiae of two fingerprints to see if the prints are of the same finger (see Section 8.4).

8.2 Fingerprint Capturing

Before minutiae can be extracted from a fingerprint image, this image must first be captured! Various different techniques are implemented to make an image of the fingerprint. After the image has been captured, it has to be analysed and the minutiae need to be extracted from it.

When looking at automated systems we can distinguish between online and offline fingerprint capturing. Offline capturing only consists of one technique: using ink. In this case the fingerprints are captured using ink on a card and the prints are later scanned into a computer for further processing.

One of the obvious disadvantages of this method is that it is labour intensive. Furthermore does a certain stigma exist for this way of capturing.

The opposite of offline is online capturing, also called live-scan. Various techniques are used. We can distinguish three different categories, being *optical*, *solid state*, and *other*.

8.2.1 Optical Capturing Devices

The oldest technique in fingerprint capturing, used since the 1970's, are optical devices. In this case a laser light illuminates the fingerprint that is placed on a glass surface. The reflected light is captured by a CCD array, see Figure 8.5. The light that passes through the glass into the valleys of the fingerprint is not reflected. The reflected light comes from the places where the ridges touch the glass. This basic principle is used in many devices but improvements have been made to reduce the size of the capturing devices.

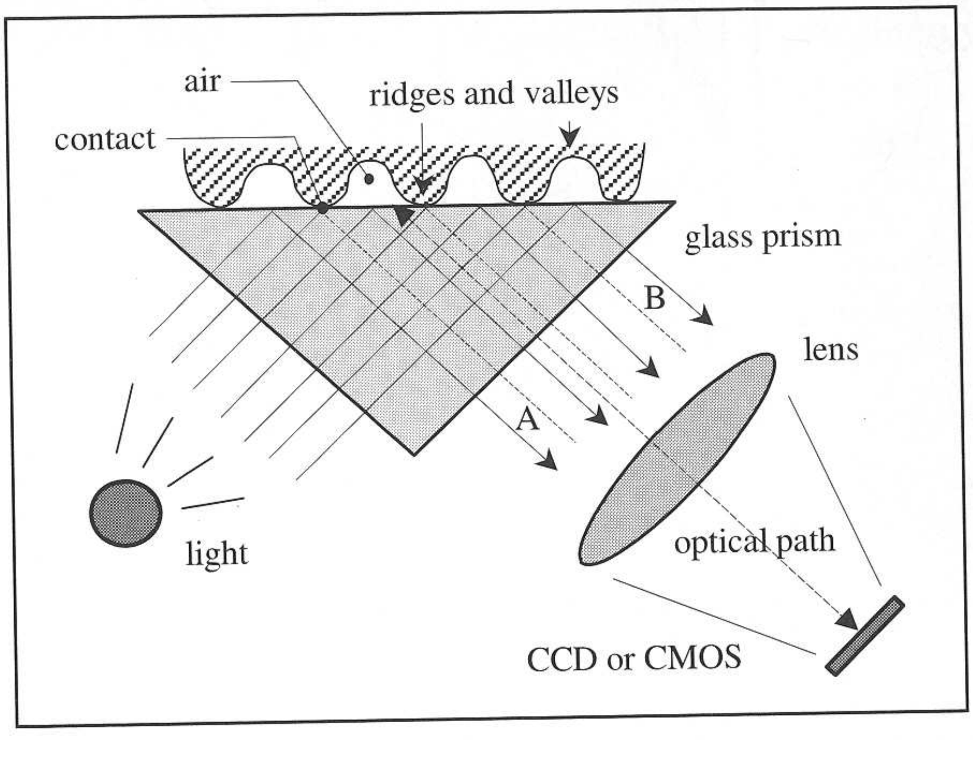


Figure 8.5: FTIR-based fingerprint sensing

Another way of using optics is to make use of fibre optics (see Figure 8.6). In this case a large bundle of optical fibres is placed perpendicular to the glass where the fingerprint should be captured. The fibres are used to illuminate the fingerprint and to detect its reflection. From these reflections the fingerprint image can be constructed.

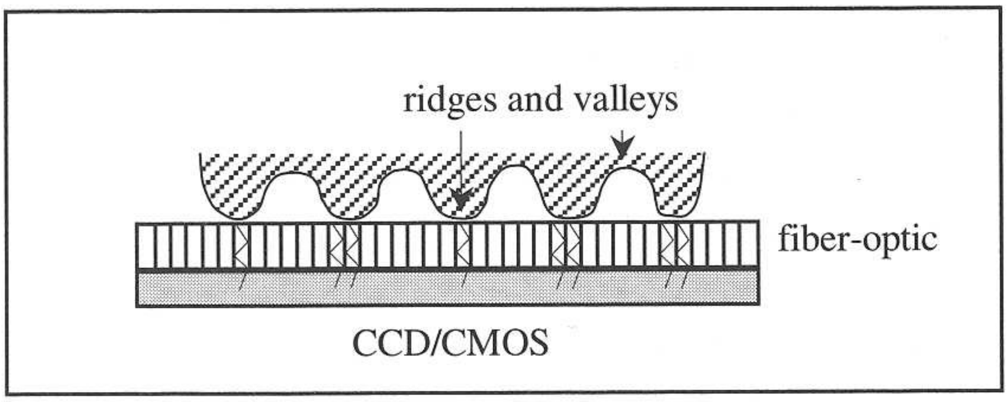


Figure 8.6: A sensor based on optical fibers

Finally an array of micro prisms, mounted on an elastic surface can be used for optical fingerprint capturing. Due to the difference in pressure from the valleys and the ridges the micro-prisms are shifted slightly. When the micro-prisms are lighted they might or might not reflect the light onto the CCD due to their slight shifts. Again the fingerprint image can be constructed from the captured reflections.

8.2.2 Solid State Capturing Devices

The idea behind solid state capturing devices has been known since the early 1980's, but they have only recently appeared on the market. Again various technologies are used.

A capacitive sensor (see Figure 8.7) uses electrical measures to capture the fingerprint. When a finger is placed on a sensing surface, composed of an array of conductive plates, the finger works as the other side of the array of capacitators. The drop in voltage of each capacitator is related to the distance between the finger and the conductive plate. Measuring the voltages of the array of capacitators gives the information to construct the fingerprint.

Another way to capture the fingerprint using a solid state device is using a pressure sensitive surface. In this case the top layer of the device is an elastic piezoelectric material. This material conforms to the topographical

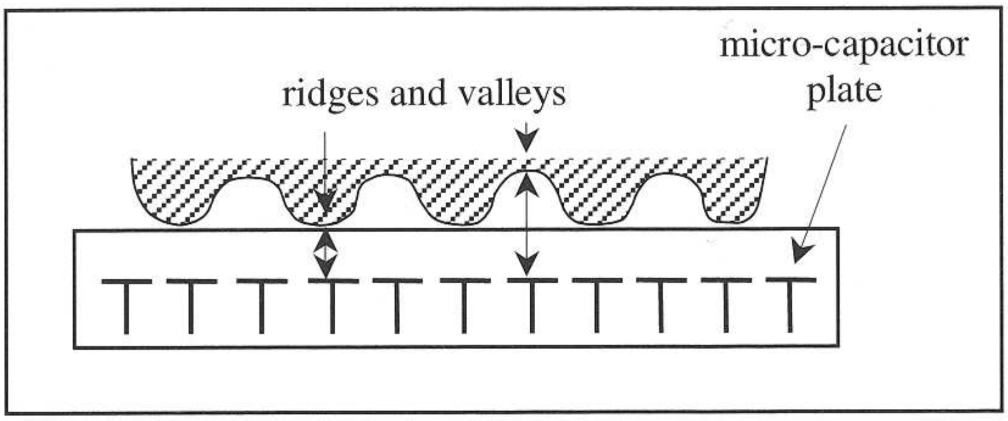


Figure 8.7: Capacative sensing of fingerprint

relief of the fingerprint and the device transforms this information into an electronic signal that in turn will be used to construct the fingerprint.

A final type of solid state capture device that is described here is a temperature sensitive sensor. Because of the difference in distance from the valleys and the ridges to the sensor, there is a slight difference in temperature measured and this information can again be turned into a fingerprint image.

8.2.3 Other Capturing Devices

Here we will only describe one capturing device: the ultrasonic scanner. This works in much the same way as the optical scanner, but now an ultrasonic beam is used instead of a laser light. The echo of the transmitted signal is captured at the receiver end of the device. The characteristics of the received signal depend on the travelled distance, which is different for ridges and valleys, see Figure 8.8. The receiver measures distance, i.e. the distance from the transmitter to the receiver and can use this information to build up a three dimensional image of the fingerprint. The major advantage of the ultrasonic scanner over an optical scanner is that it is less affected by dirt and skin oil accumulation. Therefore an ultrasonic scanner will give a better representation of the fingerprint. The technique however is not mature yet and ultrasonic scanners are still slow, large and expensive.

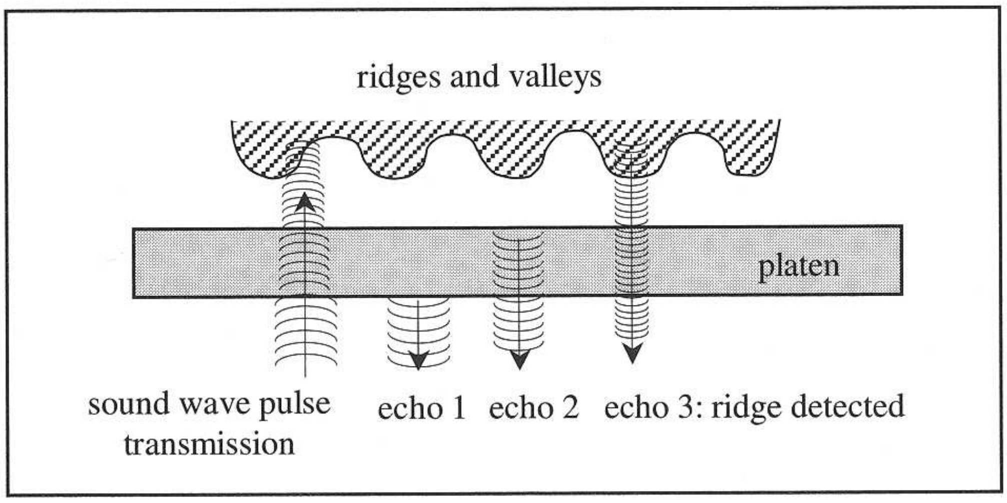


Figure 8.8: Basic principle of ultrasound technique

8.2.4 Problems with Fingerprint Authentication

Whatever device is used to capture the fingerprint, it is nearly impossible to capture the same finger in exactly the same way a second time. This will result in a different raw image of the fingerprint and therefore a distorted set of minutiae that have to be matched to a stored set of minutiae of the same finger. There are many reasons why two fingerprint captures are not the same. The main reasons are described below (taken from [13]).

Displacement: When placing the finger on the capturing device it is almost impossible to place it in the same position as the first time. When scanning a finger at 500 dpi, even a 2mm displacement will cause a shift over 40 pixels of the fingerprint, and therefore a displacement of 40 pixels of the minutiae.

Rotation: The finger can be rotated at different angles with respect to the capturing device. Some capturing devices guide the finger when it is placed, but even such a physical help cannot prevent this problem fully. In practice rotations up to ± 20 degrees can be observed.

Partial overlap: Displacement and rotation can cause the scanned part of the template fingerprint to be different from the scanned part of the input finger. Larger displacement and rotation cause a smaller overlap.

Non-linear distortion: The 3-dimensional vector is scanned on a 2-dimensional surface. Due to the elasticity of the skin this mapping will never be exactly the same, resulting in non-linear distortion in the scanned fingerprint.

Pressure and skin condition: Due to the shape of the finger, when placed on a flat surface, the middle of the finger will be pressed harder then the outside. This results in different distortion of the ridges in the centre compared to the edges. Also dryness of the skin, skin diseases, sweat, dirt, grease and air humidity all contribute to non-uniform contact between the finger and the reader.

Noise: This can be caused by latent fingerprints that are still present on the reader.

Feature extraction errors: During the feature extraction process, which is described in Section 8.3, errors can be made due to the imperfection of the used algorithms.

8.3 Minutia Extraction

Once the fingerprint has been captured the image has to be processed to find what we are looking for in the image. Before we can determine the positions and directions of the minutiae, we need to process the image first. It is not possible to accurately determine this information from the scanned image. The image is processed in 6 steps and these are depicted in Figure 8.9. Below we describe the 6 steps in more detail.

Original: This is just the original image resulting from the scan. It can be seen that this picture is not very clear and not usable for detection of the positions and directions of the minutiae. The resolution with which the image is scanned generally lies between 250 and 625 dpi with 500 dpi being more or less the standard value. The size of the image generally lies between 0.5 and 1.25 squared inches with 1 squared inch being the standard value. The scanned values are commonly 8-bit values, i.e. between 0 and 255.

Orientation: The first step in enhancing the picture consists of determining the orientation of the ridges. To do this, a so called adaptive match filter is used. The filter is called adaptive because it orients itself to the local ridge flow. It is called match because it should match the ridges instead of the noise in the image. We will not go into the mathematical details of this operation.

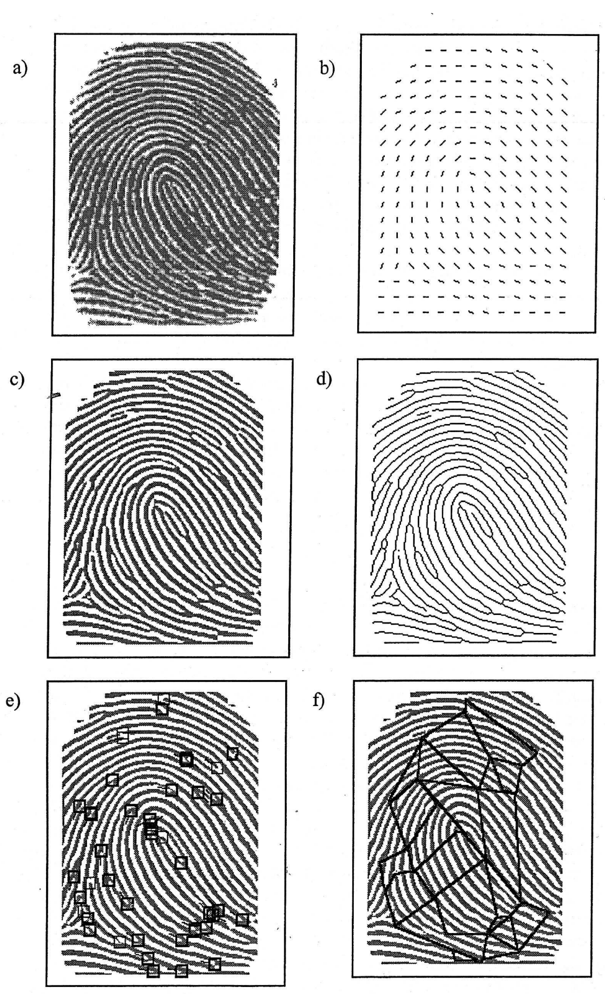


Figure 8.9: Steps in Minutiae Extraction

Binarized: So far each pixel had a grey scale value between 0 and 255. Now, depending on a threshold, these are replaced by binary values 0 and 1. If the value of the pixel is above the threshold the new value becomes 1, otherwise it becomes 0. The problem is that all fingerprint images have different contrast characteristic, so not one single threshold can be used for every image. Even within one image the contrast can differ, e.g. if the finger is pressed more firmly in the centre compared to the edges. To perform the binarization, a so called locally adaptive threshold is used, where the threshold is locally adapted to the local contrast characteristics.

Thinned: In this step the ridges are decreased in width till they are only one pixel wide. This will help in determining the locations of the minutiae. If performed well, this step will not loose connectivity in the ridges. Furthermore attention should be paid that no erroneous bifurcations are introduced. These will look like normal bifurcations with one very short branch. A good implementation will recognise such a bifurcation and will remove it from the image.

Minutiae: This is the step where the ridge endings and bifurcations are detected in the image. We will go into more detail below.

Minutia graph: In this case minutiae are connected to form a graph which contains so called neighbourhoods. Each neighbourhood consists of at least 3 minutiae. Later these neighbourhoods are used in the process of minutiae matching. We will describe the matching process in Section 8.4

The reason we process the image is the fact that at the end we want to be able to detect the positions and directions of the minutiae. Given the thinned image of the fingerprint, this is actually a very easy task. A ridge termination is located at the ending of a line and a ridge bifurcation is at the junction of three lines. Because the ridges are only one pixel wide we can determine the so-called *crossing number*, $cn(\mathbf{p})$, of each pixel $\mathbf{p}$. Let $\mathbf{p_0}, \mathbf{p_1}, \ldots, \mathbf{p_7}$ denote the pixels around pixel $\mathbf{p}$, counted in a circular manner, then the crossing number $cn(\mathbf{p})$ is defined as:

$$cn(\mathbf{p}) = \frac{1}{2} \cdot \sum_{i=1}^{8} |val(\mathbf{p_{i \bmod 8}}) - val(\mathbf{p_{i-1}})|,$$

where $val(\mathbf{p_0})$ denotes the value 0 or 1. It is easy to see that if $val(\mathbf{p}) = 1$, then pixel $\mathbf{p}$ is

- an intermediate ridge point if $cn(\mathbf{p}) = 2$;
- a ridge ending if $cn(\mathbf{p}) = 1$;
- a ridge bifurcation if $cn(\mathbf{p}) = 3$.

In part d of Figure 8.9 many terminations and bifurcations can be detected, some of which are not marked in part e of that image. Due to the noise in the original image extraneous minutiae will be detected. This can (partly) be eliminated by using thresholds. For example a bifurcation where one of the branches has a length below the threshold is removed. The threshold needs to be determined in a empirical way. Also two endings close together, with opposite directions, are likely to be caused by a broken ridge due to for example a scar or noise. Also the endings at the boundaries of the image are almost certain to be caused by the limitation of the size of the image and will not be true ridge endings.

All the minutiae that remain after the elimination step will form the *minutia template* for that finger. Generally from each minutia the type (termination or bifurcation), the (x, y) location, and the direction is stored in the template. Not every matching algorithm needs to take this information into account. Often the type of minutia is disregarded because discrimination of one from the other is difficult.

The number of minutiae detected ranges normally from 10 to 100, depending amongst others on the quality of original image. Each minutia can be stored using 27 bits. One bit is used for the type, 9 bits for x position and 9 for y position (assuming a 500 dpi image of 1 square inch) and the direction is stored in 8 bits. If we assume that it is stored in 4 bytes, the template will require up to 400 bytes of data. It is not uncommon that even larger templates are used, say up to 1024 bytes.

8.4 Minutiae Matching

Various techniques are used to match a template fingerprint with an input fingerprint. Most approaches can be classified into three classes of matching approaches.

Correlation-based matching: The correlation between the images of the fingerprints are used to match fingerprints. The correlation between the pixels of the image is computed within these approaches. Obviously displacement and rotations have to be taken into account when applying correlation-based matching.

Minutiae-based matching: This is the most widely used and most popular technique. Minutiae-based matching tries to maximize the number of minutiae between the template fingerprint and a rotated and displaced version of the input image.

Ridge feature-based matching: Minutiae-based matching does not work very well in low quality fingerprint images, because minutiae extraction is very difficult. Other features of the ridge pattern of the fingerprint, such as local orientation of the ridges, may be detected more reliable but are less distinctive in general. Depending on the quality of the images this may however be the only way of matching fingerprints. Minutiae-based and correlation-based matching algorithms may be seen as a subclass of ridge feature-base matching algorithms.

In the remainder of this chapter we will only describe minutiae-based matching in more detail. We denote by $\mathbf{T}$ the set of minutiae of the template and by $\mathbf{I}$ the set of minutiae of the input fingerprint. Each minutia $\mathbf{m}$ is described by four parameters: $\mathbf{m} = \{t, x, y, \theta\}$, where t denotes the type of minutia (ridge termination or ridge bifurcation), (x, y) denotes the position of the minutia and θ denotes the direction of the minutia. The number of minutiae in the template is not necessarily equal to the number of minutiae in the input fingerprint. Let $\mathbf{T} = \{\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_m\}$ and $\mathbf{I} = \{\mathbf{m'}_1, \mathbf{m'}_2, \ldots, \mathbf{m'}_n\}$, where m and n are the number of minutiae extracted from the template resp. input fingerprint image.

We say that the minutia $\mathbf{m}_i$ from the template $\mathbf{T}$ matches the minutia $\mathbf{m'}_j$ from the input $\mathbf{I}$ if both the *spatial distance* (sd) and the *direction difference* (dd) are small, i.e. below some threshold value. The sd and dd are defined as follows:

$$sd(\mathbf{m}_i, \mathbf{m'}_j) = \sqrt{(x_i - x'_j)^2 + (y_i - y'_j)^2} \tag{8.1}$$

and

$$dd(\mathbf{m}_i, \mathbf{m'}_j) = \min(|\theta_i - \theta'_j|, 360^o - |\theta_i - \theta'_j|) \tag{8.2}$$

Problem with matching fingerprints is that the input fingerprint can be displaced and rotated in comparison to the template fingerprint. More errors are possible but are left out from this discussion. Also the type of minutia is not considered at this point.

To compensate for displacement and rotation we can undo this using a *map* function that rotates and displaces the minutiae from the input again, to hopefully return to a similar placement and rotation as in the template fingerprint. Let (Δ_x, Δ_y) be the displacement we will compensate and let θ be the counter clock rotation. Let further $\mathbf{m''}_j$ be the result of applying the displacement and the rotation to $\mathbf{m}'_j$, i.e.

$$map_{\Delta_x,\Delta_y,\theta}\left(\mathbf{m}'_j = (t'_j, x'_j, y'_j, \theta'_j)\right) = \mathbf{m}''_j = (t''_j, x''_j, y''_j, \theta''_j), \tag{8.3}$$

where:

$$\begin{cases} t''_j & = & t'_j \\ x''_j & = & \cos(\theta)\cdot x'_j - \sin(\theta)\cdot y'_j + \Delta_x \\ y''_j & = & \sin(\theta)\cdot x'_j + \cos(\theta)\cdot y'_j + \Delta_y \\ \theta''_j & = & \theta'_j + \theta \end{cases}$$

Now let *mm(.,.)* be the decision function, deciding whether two minutiae are close enough to be possibly matched together. We say that two minutiae are close together when both the spatial and the directional differences are below some threshold values, i.e.:

$$mm(\mathbf{m}_i, \mathbf{m}''_j) = \begin{cases} 1 & \text{if } sd(\mathbf{m}_i, \mathbf{m}''_j) \leq r_0, \text{ and } dd(\mathbf{m}_i, \mathbf{m}''_j) \leq \theta_0, \\ 0 & \text{otherwise} \end{cases} \tag{8.4}$$

In Figure 8.10 we see some examples of minutiae matching. We first see that $\mathbf{m}_1$ is not matched because there is no minutiae from the set **I** close by. Minutia $\mathbf{m}"_3$ cannot be matched for a similar reason. Finally we see that, although minutiae $\mathbf{m}_3$ and $\mathbf{m}"_6$, are close together in spatial distance, they are not matched together because they are far away in directional distance.

The task of minutiae matching is now to find the optimal values of Δ_x, Δ_y, and θ such that the number of matches is maximized. Obviously there is an important restriction! Each minutiae in the template **T** must be matched to either 0 or 1 minutiae in the input **I** and also the other way around. For this purpose we define a function $P(.)$, where the input is an index $1 \leq i \leq m$ from a template minutiae and the output is either an index $1 \leq j \leq n$ from the input minutiae or 0 if the template minutia is not matched. We find the following properties of $P(.)$:

1. $P(i) = j$ implies that $\mathbf{m}_i$ is matched with $\mathbf{m}'_j$;
2. $P(i) = 0$ implies that $\mathbf{m}_i$ is not matched with any $\mathbf{m}'_j$;
3. if $\forall_{i\in\{1,2,\ldots,m\}}\,[P(i) \neq j]$ then $\mathbf{m}'_j$ is not matched with any $\mathbf{m}_i$;
4. Assume that $i \neq i'$, then $P(i) = P(i')$ if and only if $P(i) = P(i') = 0$.

The matching problem can now be defined as choosing those values of Δ_x, Δ_y, θ, and $P(.)$ that maximize the following expression:

$$\sum_{i=1}^{m} mm(\mathbf{m}_i, map_{\Delta_x,\Delta_y,\theta}(\mathbf{m}'_{P(i)})) \tag{8.5}$$

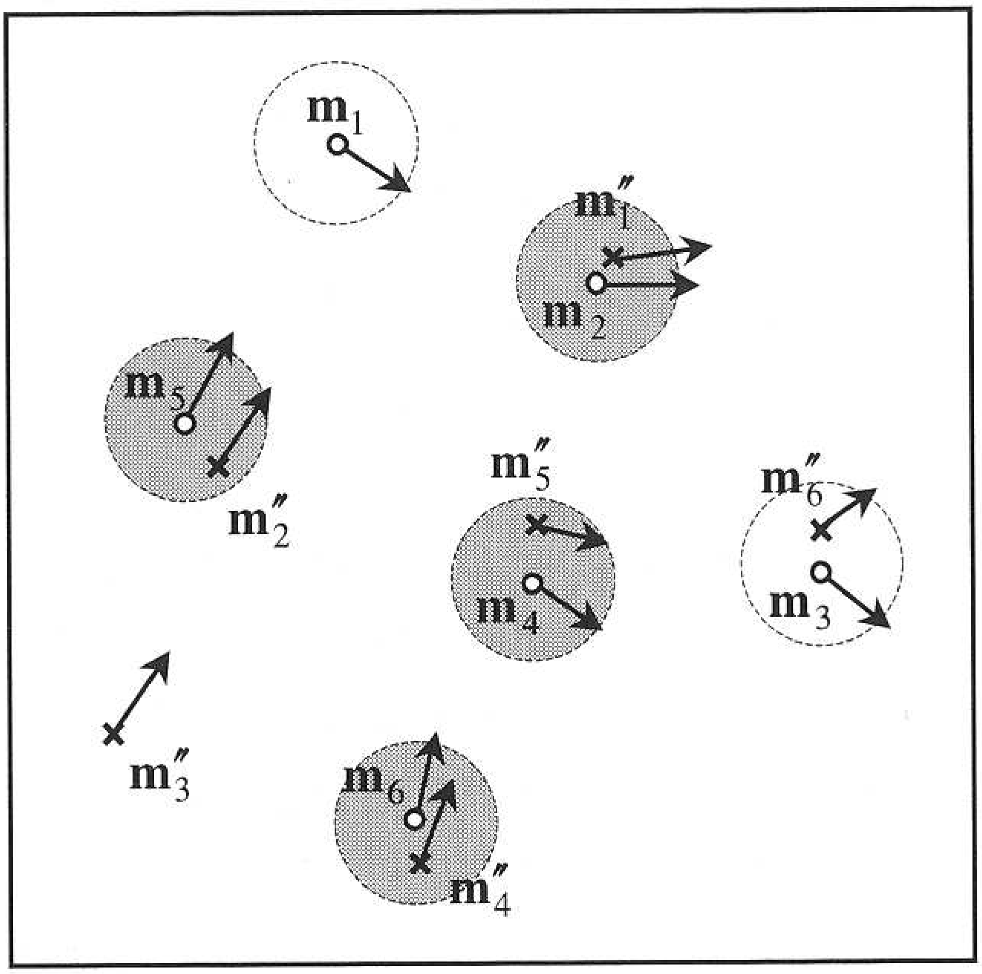


Figure 8.10: Mapping of minutiae based on distance and direction

The distance between the two fingerprints can be defined in various ways. The most simple solution is to use the expression in 8.5 almost directly. If the maximal number of matches is M, the distance function could be defined as $d(\mathbf{T}, \mathbf{I}) = 1/M$ and the threshold could be set to $1/t$, where t is the minimal required number of matches between two sets of minutiae.

Obviously other distance metrics can be used. For example the spatial and directional differences could be incorporated into the formula. Also the ratio of matched minutiae of the total number of minutiae in $\mathbf{T}$ could play a role. Furthermore we could refine the matches by also considering the type of the minutiae in order to not match a ridge ending with a ridge bifurcation.

Although it is easy to formulate the minutiae matching problem, it is much harder to solve it. We already mentioned that ridge types are not included in function $mm(.,.)$, but there are more problems that need to be solved. One of the problems is the scale of the fingerprint. If for example different dpi values are used during scanning of the template and the input fingerprint, we might have to compensate for this by scaling the input image to the size of the template.

Another thing that has to be taken into account is that two minutiae that are closest together not necessarily need to be matched by the $P(.)$ function. In 8.11 we see that it is best to match $\mathbf{m}_1$ with $\mathbf{m'}_1$ and $\mathbf{m}_2$ with $\mathbf{m'}_2$ then to match $\mathbf{m}_1$ with $\mathbf{m'}_2$ and leave the other two minutiae unpaired.

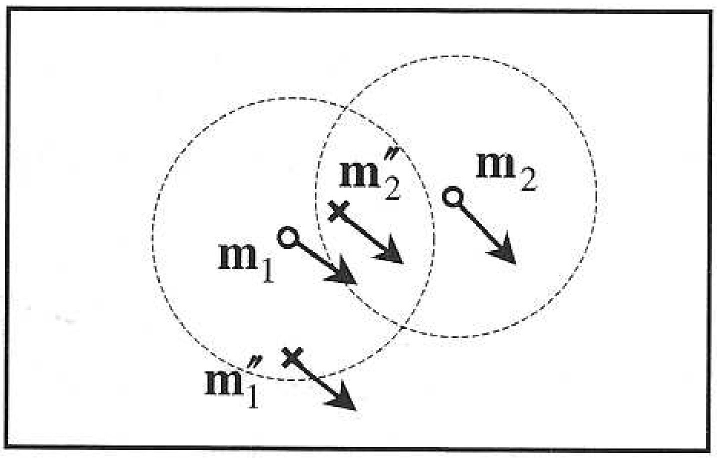


Figure 8.11: Close Minutiae Matching Problem

Chapter 9

Face Recognition

Face recognition is the most common method of biometric recognition. Used by humans that is. Even within a large group of people it is relatively easy to recognize friends or family. Humans have no problems recognizing others by their face, no matter how bad the lighting is or if the face is obscured by hats or sunglasses. Also different facial expressions pose no problems. Automating this process is however very difficult. More on face recognition can be found in [9] and in Chapter 3 of [5].

In the remainder of this chapter we will have a closer look at the various techniques used for face recognition. In Section 9.1 we will give a general introduction. In Section 9.2 we will describe the process of detecting a face within a larger image and we will briefly go into feature based face recognition. Sections 9.3 and 9.4 are used to describe a face recognition method using so-called *eigenfaces*.

9.1 General Introduction into Face Recognition

Face recognition faces a few challenges. In order to be able to use a biometric feature for authentication purposes we need a distance metric that assures a small *intra-class distance* and a large *inter-class distance*. In other words, the distance between faces of two different persons should be large and the distance between two faces of one person should be small. Obviously the challenge is to define a *distance metric between two faces*. Various techniques are used to come to such a distance metric. These will be discussed further on in this chapter.

Let us first focus on the intra-class variations. Various facial images of a single person can vary very much (see figure 9.1). This can have many reasons, some of which are listed below.

- The pose of the head can vary. He can either look directly in the camera, look away or even up or down.
- Illumination conditions can differ. Not only can the environment be brighter or darker, but the source of light can also come from different places, thereby illuminating some parts of the face more and other parts less.
- Many different facial expressions are possible. The person can smile, frown, look happy, surprised or sad. Eyes can be open or closed at the image.
- Different accessories can be used, like glasses, hat, beard, moustache, veil, shawl or otherwise.
- Part of the face can be obscured, for example by a hat, other persons in the image, hands or other objects between the person and the camera.
- The image can be in color, greyscale or even in black and white only. It may even be cartoon like.
- Time can change the appearance of face of a person. Not only the normal ageing of a face, but over time also accidents can happen that affect the face (e.g. scars).

It is obvious that there are many reasons why different images of the same face may give rise to a large distance.

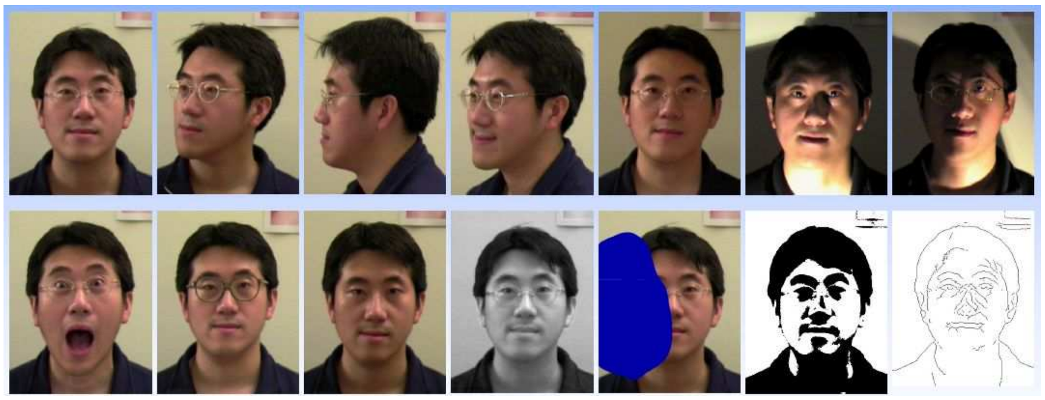


Figure 9.1: Intra-class Facial Variations

Beside the large intra-class variation we also have the problem of small inter-class variation. The most obvious example of that is identical twins. Even humans have problems distinguishing between identical twins. Mentioned as an intra-class variation is ageing. But ageing also poses problems with inter-class variation, e.g. father and son make look alike.

Face recognition can be either *pose-dependent* or *pose-invariant.* We say that it is pose-dependent if a user should present his face in a more or less fixed setting, looking in a particular way to the camera, probably with a neutral expression. In general all the conditions are fixed and the image during the authentication phase will very much resemble the image during enrollment. In case of pose-invariant face recognition, no conditions are imposed on the way the user interacts with the camera. It could even be so that the user is not even aware of the camera or that he is not aware that the process of face recognition is taking place. It is obvious that face recognition is more difficult in the case of pose-invariant recognition.

There exist many techniques for face recognition. In the next section we will go into more detail on face detection and feature detection within a face. Face recognition can be based upon these detected features. This will also be briefly mentioned in the next section. Face recognition can also be based upon a face as a whole. For this also multiple techniques are used, but we will have a closer look at so called *eigenfaces* exploited by a technique called *Principle Component Analysis (PCA).* The PCA technique will, in general terms, be described in Section 9.3 and its application to face recognition can be found in Section 9.4.

9.2 Face Detection

An automated system must perform three steps in the process of face recognition. First it has to locate the face within a larger image. Potential problems here can be that there are more faces in the image and that the person(s) in the image do not necessarily face the camera. From the detected faces, the system has to find the characteristic features, e.g. detect position of eyes, nose, mouth, or ear. Problem here is that these may or may not be on the image or may be obscured, e.g. by sunglasses or beard and moustache. The final step in the process is matching the extracted features with the features in a template to see if the faces match. In case of authentication the features are matched with all of the templates of the specified user. It is possible to store more templates per person to cover for various facial expressions or other possible variations.

Face detection is the first step in the face recognition process. The problems discussed in the previous section are actually for the larger part problematic in the face detection phase. If a face is not detected, or detected incorrectly, the remainder of the recognition process is doomed to fail.

Several tasks are performed during the face detection phase. Obviously the location of the face is detected[1]. Furthermore the location of the *facial landmarks* can be detected. Facial landmarks are specific locations on the face like centre of the eyes, tip of the nose, left and right end of the mouth etc. After detection of the landmarks, the next steps performed in face detection are *normalization* and *cropping*. These two steps actually consist of three tasks, being:

Spatial Normalization: Alignment of the centre of the eyes to predefined locations with a fixed number of pixels between the eyes through the use of rotation and scaling transformations.

Facial Region Extraction: This extracts that specific region from the image that only contains the facial region. In this way it is assured that background and other non-facial related items (e.g. hair style) are removed from the image and do not cause problems during the rest of the recognition process.

Intensity Normalization: The remaining image is transformed into a vector by concatenating all the pixels (row by row or column by column) and the pixels in the vector are normalized to mean zero and unit variance.

9.3 Introduction into PCA

Let $V = \mathbb{R}^n$ be the space of n-dimensional real valued vectors. A basis that is normally used for V is the set of vectors $\mathbf{e}_i$ for $i \in \{1, 2, ..., n\}$ where $\mathbf{e}_i$ is the vector with a single 1 in the i^{th} position and zeroes in all other positions. Any element $\mathbf{x}$ can be expressed in this basis in a very natural way. Normally $\mathbf{x}$ is denoted by $\mathbf{x} = (x_1, x_2, ..., x_n)$, implying that:

$$\mathbf{x} = \sum_{i=1}^{n} x_i \cdot \mathbf{e}_i,$$

or in other words, $\mathbf{x}$ can be written as a linear combination of the basis vectors. Any set of n vectors can serve as a basis for V as long as they are

[1]Various, mathematically complicated, methods can be used for this task. One of them is using the eigenfaces described in Section 9.4. This method is briefly discussed in Section 9.5

linearly independent. This term can best be explained by its opposite. A set of m vectors $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m$ is called *linearly dependant* if there exist values $v_1, v_2, ..., v_m$, not all being equal to zero, such that:

$$\sum_{i=1}^{m} v_i \cdot \mathbf{x}_i = \mathbf{0}^n,$$

where $\mathbf{0}^n$ denotes the all zero vector in V. In other words, a set of m vectors from V is linearly dependant if a non trivial linear combination of them add up to the all zero vector of V. Now we call a set of vectors *linearly independent* if it is <u>not</u> linearly dependant. Furthermore, a linearly independent set of vectors is a basis of V if its contains n elements, where n is the dimension of V. It is easy to check that the set $\{\mathbf{e}'_1, \mathbf{e}'_2, ..., \mathbf{e}'_n\}$, as defined below, is also linearly independent and therefore forms a basis of V.

$$\mathbf{e}'_i = \begin{cases} \mathbf{e}_1 & \text{if } i = 1 \\ \mathbf{e}_{i-1} + \mathbf{e}_i & \text{if } i = 2, ..., n \end{cases}$$

The idea of PCA is to transform the standard basis into another, in a smart way, depending on a set of data. To start explaining PCA look at Figure 9.2 where all the crosses denote samples taken from some kind of experiment. We see that all the crosses are in a cloud around the line $x = y$, which is also drawn in the figure. Every circle in the picture is the perpendicular projection of one of the crosses onto the line $x = y$. For example assume that the cross on the x-axis has coordinates $(3.2, 2.6)$, then the corresponding circle on the line $x = y$ will have coordinates $(2.9, 2.9)$. In general a cross with coordinates $\mathbf{x}_1 = (u, v)$ will be projected onto the point $\mathbf{y}_1 = (\frac{u+v}{2}, \frac{u+v}{2})$. In terms of distance (see Section 3.3), the Euclidean distance between $\mathbf{x}_1$ and $\mathbf{y}_1$ equals:

$$d_2(\mathbf{x}_1, \mathbf{y}_1) = \sqrt{(u - \frac{u+v}{2})^2 + (v - \frac{u+v}{2})^2} = \frac{1}{2} \cdot |u - v| \cdot \sqrt{2}.$$

The total distance between the crosses and the circles is equal to the sum of all the Euclidean distances between a cross and its corresponding circle. The trick of PCA is that, given the set of original points, it finds the best line to project them on in terms of the minimal total distance between the crosses and the circles. This line will give the first basis vector, also called *Principle Component (PC).* Other basis vectors will be perpendicular to this vector.

Now we will turn to finding the basis that PCA will find. In Figure 9.2 we will take as the first basis vector the vector $\mathbf{e}'_1 = (1, 1) = \mathbf{e}_1 + \mathbf{e}_2$, which corresponds to the line $x = y$. The second basis vector will be perpendicular

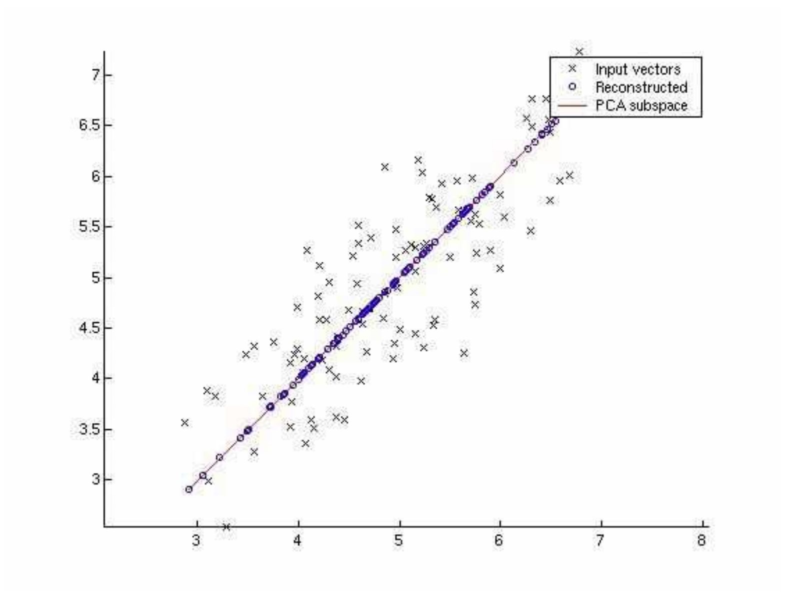


Figure 9.2: Principle Component Analysis in 2 dimensions

to the first one (thereby making sure that they are linearly independent) and in this example it will be $\mathbf{e}'_2 = (1, -1) = \mathbf{e}_1 - \mathbf{e}_2$. Instead of using $\mathbf{e}'_1 = (1, 1)$ and $\mathbf{e}'_2 = (1, -1)$ we can also choose scaled versions of them, as long as $\mathbf{e}'_1$ still represents the line $x = y$ and $\mathbf{e}'_2$ is perpendicular to $\mathbf{e}'_1$. Any point $\mathbf{x} = (u, v)$ can now be written as:

$$\mathbf{x} = (u, v) = u \cdot \mathbf{e}_1 + v \cdot \mathbf{e}_2 = \frac{u+v}{2} \cdot \mathbf{e}'_1 + \frac{u-v}{2} \cdot \mathbf{e}'_2.$$

Knowing that $\mathbf{x}$ lies in the neighbourhood of the line $x = y$, we see that u and v will be almost equal, meaning that the factor in front of $\mathbf{e}'_2$ is much smaller than the one in $\mathbf{e}'_1$. In other words, basis vector $\mathbf{e}'_1$ is "more important" then basis vector $\mathbf{e}'_2$. So by ignoring the factor in front of $\mathbf{e}'_2$, we only lose a small amount of information about the original point $\mathbf{x}$.

PCA is the technique that turns, based on a set of data, the original basis into a new, perpendicular, basis in such a way that every next basis vector contains "less information", i.e. is "less important" then the ones before. Moreover PCA will also give information on how important the basis vectors are relative to each other, or in other words, how much information is contained in each basis vector compared to the other basis vectors. This information can then be used to determine how much information will be lost when not all basis vectors are used, or equivalently, how much information is used when only the first t basis vectors are used.

Let us now see how we can calculate the basis vectors in PCA in $\mathbb{R}^n$.

Let us assume we have m samples $\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_m$, all elements of $V = I\!R^n$. We define $\mathbf{y}$ as the mean of the samples:

$$\mathbf{y} = \frac{1}{m} \cdot \sum_{i=1}^{m} \mathbf{x}_i,$$

and centre all the samples by subtracting the mean value from them, i.e. $\mathbf{x}'_i = \mathbf{x}_i - \mathbf{y}$. Now let X be the $n \times m$ matrix formed by taking as columns the vectors $\mathbf{x}'$. From X we form the covariance matrix $C = X \cdot X^T$. In this matrix the $(i,j)^{th}$ entry is the covariance between the i^{th} and j^{th} coordinate of the vectors $\mathbf{x}'$. It is obvious that $C_{i,j} = C_{j,i}$ and the size of C is $n \times n$. From C calculate the eigenvectors $\mathbf{ev}_1, \mathbf{ev}_2, ..., \mathbf{ev}_n$ and corresponding eigenvalues[2] $\lambda_1, \lambda_2, ..., \lambda_n$ and order them according to decreasing eigenvalues. We will not go into the mathematical details of how the eigenvalues and eigenvectors are calculated in a smart way from the large $n \times n$ matrix C. We will only mention that we can also start with the (much smaller) $m \times m$ matrix $C' = X^T \cdot X$, find the eigenvectors of C' and turn them into eigenvectors of C

Now that PCA has created the new basis $\mathbf{ev}_1, \mathbf{ev}_2, ..., \mathbf{ev}_n$, an original of vector $\mathbf{x} = (x_1, x_2, ..., x_n)$ can be transformed to $\mathbf{x}" = (x"_1, x"_2, ..., x"_n)$ in the new basis. Now we can write

$$\mathbf{x} = \sum_{i=1}^{n} x_i \cdot \mathbf{e}_i = \sum_{i=1}^{n} x"_i \cdot \mathbf{ev}_i,$$

where

$$x"_i = \mathbf{ev}_i^T \cdot (\mathbf{x} - \mathbf{y})$$

9.4 PCA for Face Recognition

Now let us see how PCA can help us recognising faces. Let us assume that the face images are represented as $N \times N$ arrays of greyscale intensities. This can also be considered as a N^2-dimensional vector with values in $I\!R$. Every image can therefore be represented as a point in $I\!R^{N^2}$, which is a huge space, even for a moderate value of N. For example if 256×256 images are used, we are considering points in $V = I\!R^{65536}$. Not every point in V represents a face. There are many points representing images of trains, cars, cows, buildings and many more. The vast majority of points doesn't even represent an image we would recognize as sensible, i.e. most images look like noise. So the subspace of V that represents faces is very small compared to

[2] An eigenvector $\mathbf{ev}$ of a symmetric $n \times n$ matrix M is a vector satisfying $M \times \mathbf{ev} = \lambda \cdot \mathbf{ev}$ and λ is the corresponding eigenvector.

V itself. The central idea is now to find a basis of this subspace of faces such that all images of faces can be represented using this basis. The basis vectors again being points in V, can be represented as images and turn out to be vague images of faces, called *eigenfaces*. So each image of a face can be represented as a linear combination of these eigenfaces.

PCA will perform a transformation of a basis, based on a number of samples. Let us assume we have M sample images, denoted by $\Gamma_1, \Gamma_2, ..., \Gamma_M$, where Γ_i can either be viewed as an image or as an N^2-dimensional real valued vector, depending on the context. Let Ψ be the "average" image, i.e.:

$$\Psi = \frac{1}{M} \cdot \sum_{i=1}^{M} \Gamma_i.$$

Let Φ_i be the difference between the original image Γ_i and the average Ψ, so $\Phi_i = \Gamma_i - \Psi$. The vectors Φ_i will be used to find the eigenvectors (here called *eigenfaces*) and the corresponding eigenvalues. To do so, first the matrix $X = [\Phi_1, \Phi_2, ..., \Phi_M]$ is formed and from that we can find the covariance matrix $C = X \cdot X^T$. From the covariance matrix C the eigenfaces $\mathbf{ev}_i$ and eigenvalues λ_i for $i = 1, 2, ..., M$ are calculated. The matrix C however is enormous, so it might be better to find the eigenvectors $\mathbf{ev}'_i$ and eigenvalues of the matrix $C' = X^T \cdot X$ (which is only $M \times M$). These eigenvalues $\mathbf{ev}'_i$ can be used to find the M first eigenvectors $\mathbf{ev}_i$ and eigenvalues λ_i of C.

Now, given the image Γ of a face, we can express it in terms of the eigenfaces. We find the coordinates ω_i from

$$\omega_i = \mathbf{ev}_i^T \cdot (\Gamma - \Psi),$$

where i runs from 1 to $M' \leq M$. Note that not all eigenfaces need to be used, but only the M' most relevant ones. The vector $\Omega^T = (\omega_1, \omega_2, ..., \omega_{M'})$ describes the contribution of the eigenfaces in the new face Γ. Define $\Omega_i^T = (\omega_1^i, \omega_2^i, ..., \omega_{M'}^i)$ as the contribution of eigenfaces of the image Γ_i of the i^{th} enrolled user.

We can now use the vector Ω to see if the image is indeed a face. If it is, the coordinates ω_i should be small and the total weight of the vector Ω should be small too. Furthermore, the distance between Ω and Ω_i can be used to see if the image resembles the face of user i. Again the distance should be small and a suitable distance metric should be used. We can now distinguish 4 different cases as we see in Table 9.1.

We see that if both the weight of Ω and the distance between Ω and Ω_i small are, the user is authenticated as being user i. It is also possible that Γ is recognized as a face, but it does not resemble the face of user i. Opposite

$w(\Omega) \rightarrow$ $d(\Omega, \Omega_i) \downarrow$	small	large
small	Face, recognised as user i	Non-face, resembling user i
large	Face, not recognised as user i	Another image

Table 9.1: Face Recognition Possibilities

to this we have the possibility that we have a "non-face image" that still resembles user i. Obviously this will be very rare.

To perform the above mentioned measurements, we need to select a suitable distance metric $d(.,.)$ and a suitable weight function $w(.)$. Both functions can depend on the coordinates only, or also incorporate the eigenvalues λ_j to have a weighted distance function. When matching two vectors Ω and Ω_i it is more important that the first few coordinates are close together then the last few, because the corresponding eigenfaces are more important. A possible distance functions are described below, but many other distance functions can be found also.

$$d_4(\Omega, \Omega_i) = \sqrt{\sum_{j=1}^{M} \lambda_j \cdot (\omega_j - \omega_j^i)^2}$$

The eigenvectors that are used are derived from a sample set of M images. We made no assumption on the source of these face images. It could be that images of many different people, in many different poses have been used. In that case the eigenfaces describe a general way a person looks. It could also be that per person a set of sample images is used, where the person in the pictures displays various facial expressions and looks in multiple directions. From this set of sample images we can again find a set of eigenfaces describing this specific person more precise. In this case a test image Γ will be expressed within the subspace of faces of user i, which should give better results for recognising the specific person. Obviously this will lead to more work and a larger template for each user. Instead of using an overall set of eigenfaces and eigenvalues each user has his own personal set of eigenfaces and eigenvalues. For authentication purposes this might not be a big problem, but identification becomes more cumbersome because an image of an unknown persons must be expressed in many different (personal) eigenfaces to see if it matches.

9.5 Face Detection - revisited

Eigenfaces can also be used to detect a face within a larger image. As mentioned in Section 9.2, we will briefly describe how this works. As explained before span the eigenfaces the subspace containing face-like images. So within a larger image we can look for a smaller part which lies within this "face subspace". For this purpose we take an $N \times N$ sub-image Γ and transform it using the eigenfaces to see if the weight of $\Omega^T = (\omega_1, \omega_2, ..., \omega_M)$ is small. Obviously the face (or faces) in the larger image do not need to have the same size as the sample faces that were used to build the eigenfaces. So also other size sub-images should be considered. These $K \times K$ images should be resized to $N \times N$ to fit the eigenfaces.

Chapter 10

Signature Verification

Signatures are used in many places to authenticate oneself. For example when signing contracts or legal documents, getting money from the bank or applying for a credit card. Also digital signatures are already widely used in the electronic world. Documents are not only protected against unauthorized disclosure by encrypting them, but in many cases they are also signed (using for example RSA or DSA) to provide prove of the identity of the signer. But there is one more form of signature that combines the “real life” with the “electronic world”. In that case a user uses special equipment to write his signature. The signature is converted into electronic data. This data can be processed to see if it matches a pre-recorded template of this persons signature. The data can also be used to add a “real life” looking signature to the bottom of an electronic document.

There is a great variety in special equipment that can be used to capture the signature. In the *offline* case, the user writes his signature with a normal pen on normal paper and this is digitized using a scanner. In this way only the shape of the signature will be captured. In the *online* case the signature is written with special equipment and directly stored in an electronic form. This can for example be done by using a special tablet that records the position of the pen on the tablet and records that information. Another option is using a special pen that can record position, motion or maybe even pressure or tilt of the pen. In the online case not only the shape of the signature is recorded, but also extra information like the dynamics of the writing of the signature. Which information is used for authentication purposes depends on the specific algorithms used.

In what follows we will mostly follow [4], but we will also use information from other sources. In Section 10.1 we will first go into more detail of testing signature verification systems. The system should have a small FAR, i.e. it must be resistant against forgeries.

In [4] a verification system is described that we will use here to explain the steps taken in a signature verification system. Signature verification systems can use different sets of information for authenticating people. In Section 10.2 we will describe the general signature verification system from [4]. We will not go into all details of this system but we will describe in Section 10.3 the concept of *critical points* and we will describe in Section 10.4 what features are used in the system described in [4].

10.1 Signature Forgeries

Signatures seem easy to forge. When forging a signature on paper the forger needs to copy the shape of the signature as best as possible. A person accepting the signature will visually inspect the signature to see whether or not it resembles a template (for example a signature on a special card, on the back of a credit card or in a passport). This visual inspection should convince the person of the correctness of the (forged) signature.

When trying to forge a signature in an online signature verification system, a forger must also be able to mimic the dynamics, making it harder to forge. It depends heavily on the specific dynamic information used by the signature verification system how well the forger must mimic the moves of the genuine owner of the signature. For online systems we distinguish two types of forgeries, the *skilled* and the *random* or *zero-effort* forgery.

A forgery is called skilled if the forger had access to one or more genuine signatures. From this he could study the shape and practice on getting the shape right. He might even have information on the dynamics of the signing process. A forgery is called zero-effort if there was no access to a genuine signature. The forger has in this case no knowledge of the shape of the signature. He might have the name of the person whose signature he tries to forge. From this he has to come up with a signature that he hopes resembles the true signature in shape and possibly in dynamics.

When performing tests with only random forgeries, the FAR is obviously much lower then when skilled forgeries are used. Random forgeries can be used to see how well the system can work in an ideal world where nobody tries to cheat, but where genuine signatures of different people may look alike. Obviously skilled forgeries are a much more realistic threat for a signature verification system. Most reports on these systems therefore use skilled forgeries to see how well the system performs.

10.2 General System Description

In [4] the author describes a system for signature verification. This system consists of a number of subsystems, being *preprocessing*, *feature extraction*, *threshold selection*, and *matching*. During the enrollment a user provides a number of signatures, usually 3 to 10, and this set of signatures is stored directly or as a template. According to [4] all signatures provided during enrollment will make up for 1 template, which is updated every time a new reference signature is provided. The advantage of using a template is that it is much easier to compare this to a given signature during the verification process. Keeping the signatures as is has the advantage that different writing styles can be modelled. It is not necessary to convert all the signatures into a single template. Every signature can be converted into a template by itself. The matching stage will be a bit different. In that case the input signature can be matched with all signature templates to see if at least one matches very well. Maybe the decision is based upon the average match between the input signature and all templates, or maybe even a predetermined number of templates must match with a given threshold. This can be very dependent on the system that is used.

Below we describe in a bit more detail the various subsystems in the verification system.

Preprocessing: Noise is removed from the recorded data and the signature is smoothed. It is normalized in size and possibly time dependencies are removed in case only the shape of the signature is used.

Feature extraction: Local and global features are extracted. A global feature described the signature in one or a few values, describing certain characteristics of the signature. Local features can be either *spatial* (related to the shape) or *temporal* (related to the dynamics). Combinations of the two are also possible.

Matching: To perform matching a presented signature has to be compared to the (set of) template(s) of the same signature. Depending on the specific algorithm the global and/or local features are compared. The distance between the templates and the presented signature will be used to decide whether or not they are from the same person.

Threshold selection: The threshold can be either global, i.e. the same for all users, or it can be user dependent. Depending on the signatures presented during the enrollment the threshold for a specific user can be different from the threshold for another user.

We will not go into more detail of all of these subsystems. In the following sections only a few interesting parts are described in more detail.

10.3 Critical Points

When preprocessing the online captured signature, various things will take place. These are briefly described below.

Resampling: The signature contains spatial and temporal information. In order to be able to compare the shapes of two signatures, we need to get rid of the temporal information. This is done by using resampling. Before resampling, the temporal information must be saved because resampling loses all this information.

Smoothing: To get rid of noise in a signature it is smoothed with a Gaussian filter. This will remove the small changes, but will leave the overall structure of the signature unchanged. Smoothing is performed for x-direction and y-direction separately.

Critical Points: These are points within the signature that are supposed to carry more important information. We will go into more detail of critical points below.

Size Normalization: The best way to compare two signatures is to resize them to a fixed size. In this way we can eliminate false non-matches due to variations in space where the signature can be written. Normalization can keep the ratio of the signature, but it can also be normalize both in height and width, leading to loss of writer dependant information.

The critical points are used later on during matching of signatures. The author of [4] uses 2 types of critical points. The first type is the endpoint of a stroke[1]. The second type is where the trajectory of the signature curve changes. In Figure 10.1 we see 12 different classifications of critical points. The top 4 result from changes in x- or y-direction. In the bottom 8 we see that the signature comes from or goes to a horizontal or vertical line. In all cases i denotes the critical point and the arrow displays the writing direction.

In Figure 10.2 we see a part of a signature that is enlarged and the big dotes in this enlargement denote the critical point distinguished in this

[1]A stroke is the collection of all consecutive samples between the moment the pen touches the paper (or the tablet) and the moment it is lifted again

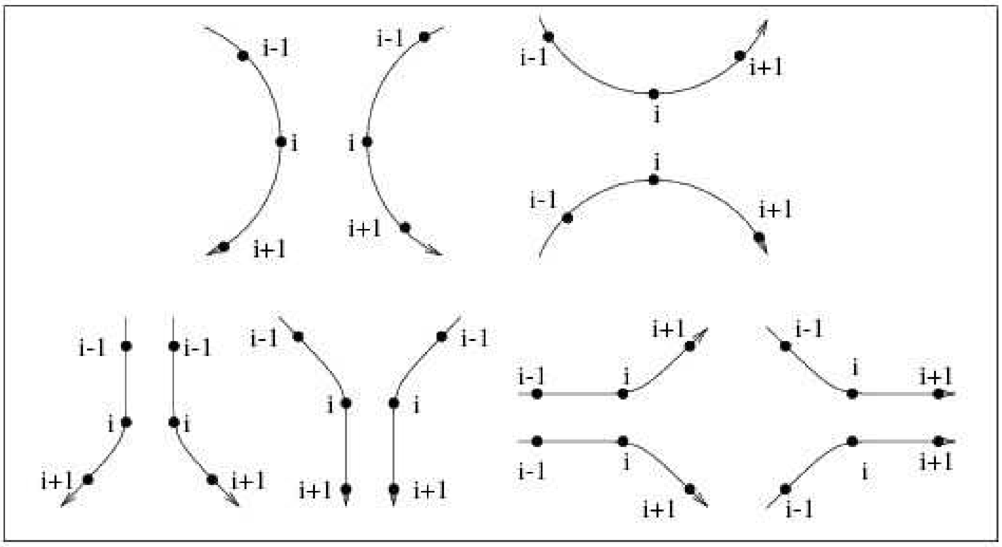


Figure 10.1: Critical Points due to Trajectory Change

part. We clearly see 11 marked critical points. Assume the writing direction was from left to right and numbering the critical points along the signature trajectory, we see that points 1 and 11 are the ending points of a stroke. Points 2, 4, 9 and 10 are clearly points where the signer drastically changed the writing direction. Points 3, 5, 6, and 7 (and possibly also point 8) are points where this change in writing direction is not so drastic, but these are merely changes due to curvation in the writing.

Looking at the figure, we see that in the neighbourhood of point 4, due to the small loop, we would expect two extra critical points. Following the writing direction, just in front of point 4 we see a right to left change in the writing direction and just after point 4 we see a left to right change. These are however not been marked as a critical points. One of the rules in [4] when determining critical points is that the cannot be too close together. In this case this seems reasonable because a next time the signature is written this small loop might not be there.

Looking further at the figure, we also see that the writing direction changed (from down to up) at the point where the signature curve between points 4 and 5 and between points 6 and 7 cross. This particular point might be too close to point 5 according to the rules in [4], but in a next writing of the signature they might be a bit further apart and the specific point might come up as a critical point. In general the decision which points to include or exclude must be taken with much care.

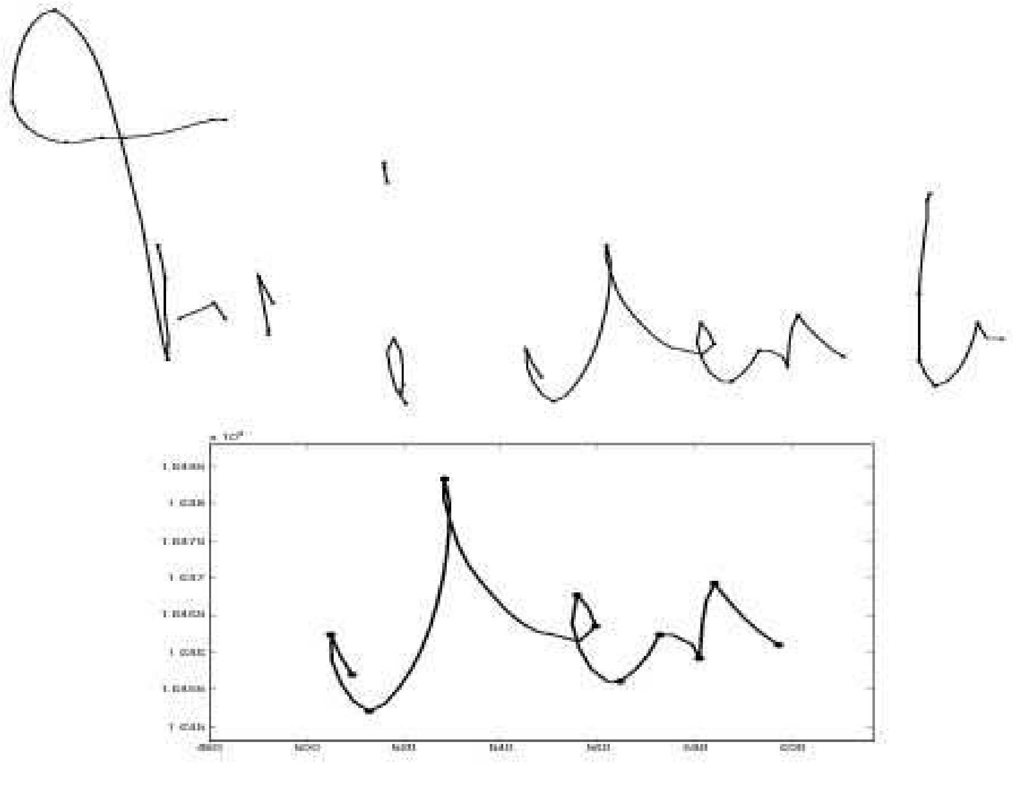


Figure 10.2: Example of Critical Points

10.4 Feature Extraction

Information needed for the matching algorithm can simply be the x- and y-coordinates which are recorded during the signing of the algorithm. But more information, both global and local, can be retrieved from the dataset.

10.4.1 Global Features

A global feature expresses the entire signature with a single value. It depends on the complete signature. More then one global feature of a signature can be used when performing authentication. In general it will be hard, if not impossible, to use a single global feature to discriminate between various signatures.

In [4] the number of strokes is used as a global feature of a signature. For example in Figure 10.2 we see a part of a signature that already contains 7 strokes. In general though signatures will not contain many strokes. Authentication where people write their names in block-writing would give rise to more strokes. In that case the number of strokes will be approximately the number of letters in a name, not counting dots, dashes or other things.

Other global features are easy to come up with. Below we give a short list of examples that can easily be extended.

Size: The size of the signature. This can be measured in many different ways, for example the difference between maximum and minimum values in x- and y-direction or the difference between the starting point and the ending point of the signature. Another option for size is taking the sum of the Euclidean distances between every two samples. This can easily be measured per stroke and these values can be added to get the total size. Also the distance between the strokes (i.e. where the pen went up and where it came down again) can be taken into account. Many other options are available for the size of a signature.

Time: The time needed to write the signature taken from the time the pen goes down the first time till the last time the pen goes up. The time needed per stroke is obviously related to the number of samples, assuming that the sample rate is fixed. Possibly some devices cannot measure time between pen-up and pen-down. In such case the time has to be taken as the pen-down time.

Speed: Speed is the distance the pen travelled per time unit. For speed similar options are available as for size. We can easily calculate the average of the maximum speed for a single stroke or for the complete signature.

Acceleration: More or less the same options are available as for speed.

10.4.2 Local Features

On a local level we can also derive some features. These local features can be classified either as *spatial* or as *temporal*. A spatial feature reflects on the shape of the signature while a temporal reflects on the writing of the signature, e.g. speed and direction. In [4] 5 local spatial features are used. These are displayed in Figure 10.3.

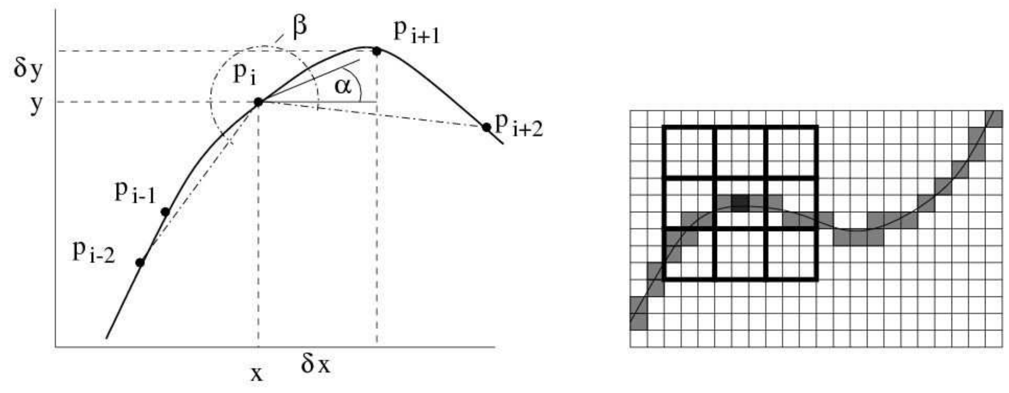


Figure 10.3: Local Spatial Features

The spatial features of point $p_i = (x_i, y_i)$, used in [4], are the following:

Position: For the position the absolute y-value y_i of p_i is used.

Change: The change is represented by $\delta x = x_{i+1} - x_i$ and $\delta y = y_{i+1} - y_i$

Angle: The angle α between the tangent of the signature curve in the point p_i and the positive x-axis is not used directly, but instead $\sin(\alpha)$ and $\cos(\alpha)$ are used[2].

Curvation: The curvation β is the angle between the line through p_i and p_{i+2} and the line through p_i and p_{i-2}.

[2]The reason to use the sine and cosine is that there are no problems around 0. The difference between an angle $\alpha = 1$ and $\alpha = 359$ is large but the sine and cosine function correct that due to their cyclic nature.

Grey value: The grey value in point p_i is determined using a 9×9 pixel neighbourhood as is displayed in Figure 10.3. This square is divided into nine 3×3 squares and for each the grey value is determined as the sum of the pixel values within the small square. Specifically, in that figure the grey values will be $(0, 0, 0, 2, 3, 3, 3, 0, 0)$, when counting from left to right and from top to bottom.

The total number of local spatial features used in [4] is therefore equal to $(1 + 2 + 2 + 1 + 9) = 15$. Beside the spatial local features also the use of temporal local features is explored in the document. The speed at both resampled points (see Section 10.3) and at critical points is used. The speed at a critical point is directly known for a critical point is also a sampled point. The speed at a resampling point is calculated under the assumption that the speed varies smoothly between sample points. The speed at resampled points is calculated per stroke, so there are no boundary problems between strokes.

10.5 Feature Matching for Signatures

After the features have been extracted the matching problem has to be solved. In the previous section we already saw that a lot of features are extracted from the signature. Of course, not all of these features need to be used in a matching algorithm. In the remainder of our description we will assume that only the spatial local features are used. Using also temporal features or global features can be done in a similar way.

All of the critical point of the signature can now be described with a 15-dimensional vector $\mathbf{f} = (f_1, f_2, ...f_{15})$ that we will call a *feature vector*. Assume that in a particular signature M critical points are identified, then the complete signature **sig** is described using M 15-dimensional vectors, i.e. $\mathbf{sig} = (\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_M)$. Assume $\mathbf{sig_1}$ and $\mathbf{sig_2}$ are two signatures we want to compare. In general the do not contain an equal number of critical points. Let $\mathbf{sig_1}$ contain M critical points and assume that $\mathbf{sig_2}$ has N critical points. We are now faced with 2 different tasks. First we have to find a way to compare 2 different feature vectors, i.e. define a distance metric between $\mathbf{f}_i$ and $\mathbf{f}_j$. The other task is to compare two different signatures, i.e. two sequences of feature vectors.

Let us look at the last problem first and let us assume that we already have some distance function $d_{feature}(.,.)$ between two feature vectors. Obviously we want to match the critical points from one signature to the critical points of the other. Due to various reasons extra critical points can be identified within a signature or critical points can be omitted from the signature. So we need to find a technique to match feature points in $\mathbf{sig_1}$ with those in

$\mathbf{sig_2}$. Obviously the first and last critical point of $\mathbf{sig_1}$ correspond to the first and last critical point of $\mathbf{sig_2}$. The remainder in between is uncertain. Let us assume that we want to match $\mathbf{sig_2}$ (acting as the input signature) to $\mathbf{sig_1}$ (acting as the template signature). There will be critical points in $\mathbf{sig_2}$ that are not in $\mathbf{sig_1}$. We want to *delete* these critical points from $\mathbf{sig_2}$. On the other hand we can also have critical points in $\mathbf{sig_1}$ that have not been recognised in $\mathbf{sig_2}$ and such points have to be *inserted* in $\mathbf{sig_2}$. All the other critical points of $\mathbf{sig_2}$ can be matched to a point in $\mathbf{sig_1}$. Let us give a small example to make things more clear. Let $\mathbf{sig_1} = \{\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_7\}$ and $\mathbf{sig_2} = \{\mathbf{f}'_1, \mathbf{f}'_2, ..., \mathbf{f}'_8\}$. Obviously $\mathbf{f}_1$ is matched to $\mathbf{f}'_1$ and $\mathbf{f}_7$ is matched to $\mathbf{f}'_8$, being the starting and the ending point of the signature. Moreover, let the matches be as described in Table 10.1, i.e. the feature vector in the top row is matched to the one in the bottom row, and a "-" meaning it is not matched.

$\mathbf{sig_1}$	$\mathbf{f}_1$	$\mathbf{f}_2$	-	$\mathbf{f}_3$	$\mathbf{f}_4$	$\mathbf{f}_5$	$\mathbf{f}_6$	-	$\mathbf{f}_7$
$\mathbf{sig_2}$	$\mathbf{f}'_1$	$\mathbf{f}'_2$	$\mathbf{f}'_3$	$\mathbf{f}'_4$	$\mathbf{f}'_5$	-	$\mathbf{f}'_6$	$\mathbf{f}'_7$	$\mathbf{f}'_8$

Table 10.1: Matches between $\mathbf{sig_1}$ and $\mathbf{sig_2}$

From the table we see that we need to delete points $\mathbf{f}'_3$ and $\mathbf{f}'_7$ and insert point $\mathbf{f}_5$ between points $\mathbf{f}'_5$ and $\mathbf{f}'_6$. There are certain costs related to these insertion and deletion operations. These costs can be described using two distance functions $d_{ins}(.)$ and $d_{del}(.)$. Also there are costs related to matching feature vectors $\mathbf{f}_i$ to $\mathbf{f}'_j$ and these costs are described using the distance function $d_{feat}(.,.)$. The two distance metrics $d_{del}(.)$ and $d_{ins}(.)$ can be related to the distance function $d_{feat}(.,.)$, e.g. by defining that the second component of the feature distance function is equal to 0.

We need a technique called *Dynamic Time Warping (DTW)* or *string-editing* to match strings of unequal length in the way we described above. We will not go into much detail here but refer to [14], where the authors also use the technique for matching strings of features resulting from signatures. Also in [4] we find a description of the DTW technique.

So we see that, once we have decided how to define the distance metric d_{feat}, we can calculate the distance between two strings of features, i.e. between two signatures. This distance metric depends on which of the 15 features of the feature vectors are used. If positioning information is used then for that part of the feature vector the Euclidean distance $d_2(.,.)$ from Section 3.3 can be used. In case information about the angle is used the absolute difference of the difference in angles can be used. This should be done carefully because the difference between an angle of 359 degrees and

1 degree is not 358 degrees but only 2 degrees. For the other parts of the feature vector also distance metrics can be derived. After coming up with the distance metrics for the various parts of the feature vector they need to be combined in some smart way, probably using a weighted sum or something similar.

The overall quality of the signature verification process depends on many things. There are many variables as we can read in the above discussion. We can vary the distance metric of the feature vectors, define the costs of insertions and deletions differently, use another method to match two strings of unequal length, use different subsets of the extracted features or even define new ones. Also the global features can be used in the definition of the total distance between two signatures.

Especially the definition of the distance between two feature vectors will need a lot of attention when designing the system. Finding the best, or at least a very good, distance metric $d_{feat}(.,.)$ will be a task that will take a lot of time and patience. Many different metrics need to be tested and compared to each other before one is found that is good enough.

Chapter 11

Keystroke dynamics

User authentication through keystroke dynamics is not a very new technique. Already during World War II telegraph operators could recognize the sending operator by his typical keying rhythm (of Morse-code). Keystroke dynamics is the process of analysing the way a person types at a keyboard and identifying him based on this.

Keystroke dynamics analysis can well be used during typing of a username and password to enhance the security of the password. In such case the data that is typed is fixed and also the time this information is typed in is fixed (during login at a computer). In such a case we talk about *static verification*. The opposite of this is *continuous verification*, in which case the typing behaviour is analysed during a complete session. In case of static verification we only check if login is done by the correct user. Using continuous verification we can also check if, during a session, the user has changed. Of course typing rhythm can also change during a session due to other causes like tiredness or drowsiness. Over a longer period of time someone's typing rhythm can change due to for example change of keyboard (from a standard US keyboard to a Norwegian one with extra symbols) or familiarity with typing a specific text (such as username and password).

11.1 Extracting Typing Information

To perform keystroke dynamic analysis we need to extract information during typing. The information that can be extracted during typing consist of at least the following features:

Latency: The time T_l between the release of one key and the depression of the next key. So this is the time we are "not" using the keyboard.

Key-down: The time T_d a key is held down. This is also called *duration.*

Force: The pressure applied to a key when pressing it down.

Finger placement: The position where the finger is placed on the specific key of the keyboard.

Finger choice: Which finger is used for which key of the keyboard.

The first two items of the above list are easy to measure using keyboard interrupts. Small programs can be written to collect these measurements. The last two can easily be obtained through video surveillance of the keyboard. When using a special keyboard also force can be measured. For the moment we will assume that only latency (T_l) and key-down time (T_d) will be used for analysis because this can be easily measure with only a small program and a standard keyboard.

As with signature verification, we can again look at global and local features. Global features are things like average typing speed, use of backspace, and many more. Local features are related to specific keystrokes, e.g. looking at how long the letter 'e' is pressed down, maybe in relation to the next or the previous letter.

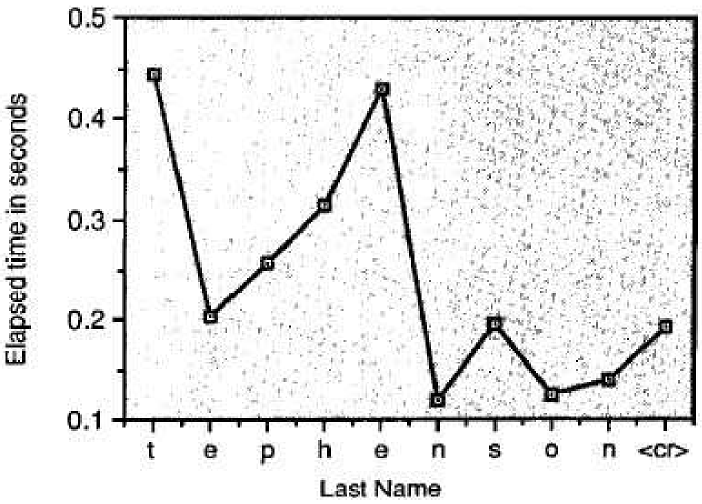


Figure 11.1: Latencies when typing "Stepenson"

Many local features are be used and are in fact described in literature. Some researcher only use the time between various keystrokes (i.e. the sum of T_l and T_d), others use both timings separately. In Figure 11.1 we see an example where only latencies are used to find the way a person is typing his last name. More elaborate is even to look at combinations of 2 or even 3 or more letters. Specific combinations occur a lot in written texts and are therefore typed more frequently, like “ing”, “and”, “the”, “in”, “we”, “are” and “or”. Analysis can be based on either a limited number of such letter combinations or on the whole set of 2 or more letter combinations. In Figure 11.2 we can see the duration T_d and latency T_l of some bigrams found during typing. Only the most common bigrams are using in this figure.

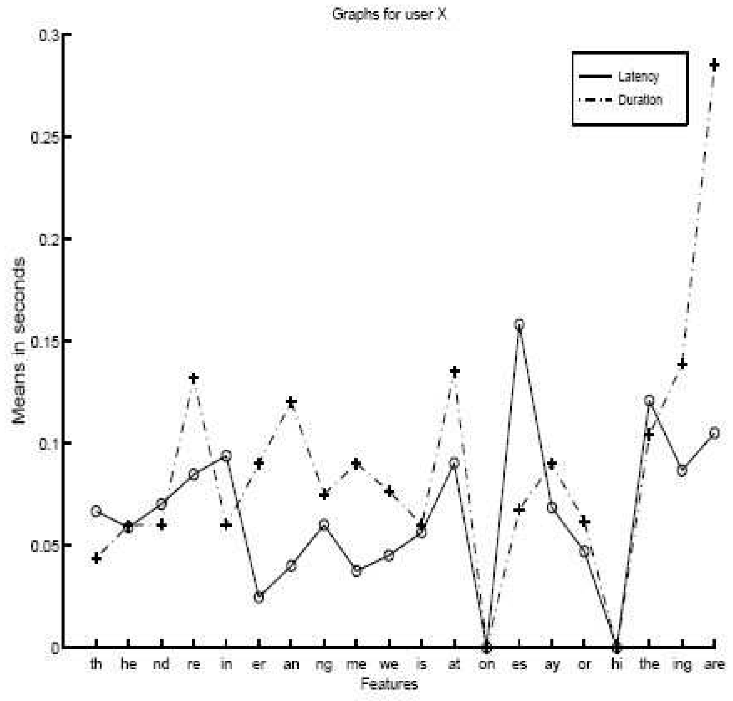


Figure 11.2: Keystroke Latencies and Duration for Various Bigrams

When using rare special characters it is clear that a user must “search” for them on the keyboard and timing can differ a lot from one time to another. This would imply that using these timings doesn’t make the analysis very reliable, but on the other hand, this might just be the information that can be used to distinguish people.

11.2 Matching Keystroke Patterns

After deciding what to use in the analysis, also a distance metric must be chosen. In [7] the authors use only latency to authenticate people and they simply use distance metric $d_1(.,.)$ as described in Section 3.3. In general any of the metrics from Section 3.3 or a more elaborate one can be used. We will describe one example of a more elaborate distance metric here. Assume we are comparing keystroke dynamics when users type their totally randomly chosen, N character long, passwords. Each password $\mathbf{pw}$ is defined by $\mathbf{pw} = (pw_1, pw_2, ..., pw_N)$, where the pw_i stand for the characters in the password. The time the first character (pw_1) was typed is taken to be equal to 0. The timing measurements for the latencies in the template are denoted by $\mathbf{T} = (T_1, T_2, ..., T_{N-1})$. Note that the number of latencies is one less then the number of characters in the password, unless for example after typing the password, the user presses the "Tab" or "Enter" key. In such case also the latency between the last password character and the next key pressed can also be used. In Figure 11.1 we see that indeed the latency between the last 'n' and the "enter" key is used. Let the input timings be denoted by $\mathbf{I} = (I_1, I_2, ..., I_{N-1})$, then we can define the distance between I and T for example as:

$$d_{keystroke}(\mathbf{I}, \mathbf{T}) = \sum_{i=1}^{N-1} \frac{|I_i - T_i|}{Prob(pw_i)},$$

where $Prob(pw_i)$ denotes the probability that character pw_i is used in every day texts. In this case a letter, which is not used so often used during normal typing, gets a higher weight when it appears in a password. The reason behind this would be that it would be more difficult to find for a non-trained user, i.e. for someone with another password, but easier for the person who uses this character often in his password. Actually the value T_i (and thus also I_i) is the time between the typing of pw_i and pw_{i+1}, so maybe a combination of the probabilities of these two characters must be used. Or maybe the probability of the bigram occurring in normal day usage.

When we do not only use latencies but also key-down timings, the distance metric has to be adjusted for that. We could just add an extra summation to the above mentioned distance metric, counting the difference in key-down timings, again using the probabilities of the specific characters that are used. We could for example use single probabilities for the key down times and products of two probabilities for the latency timings. Anything is possible here.

When using 2-letter combinations (called *bigrams*), we could also use a distance metric that also incorporated the probability of occurrence of such a bigram. For example the bigrams "th" and "he" (both occurring in the trigram "the") occur much more then the bigram "fr", i.e. a deviation in the typical way of typing one of those bigrams should be considered more severe than a deviation in the way of typing “fr”.

11.3 Possible Applications

It sounds obvious that it is easier to analyse fixed texts then free texts. When using a fixed text it is easier to record the specific typing behaviour of a user. This text should not be too short for otherwise no good analysis is possible. Also, for practical reasons, the text should not be too long because that would take too much time from a user. When using keystroke characteristics to authenticate a user, the password and username might be too short for a good authentication. But in addition to this, the user might also be presented with a sentence he has to type. This can be either a fixed sentence or one from a limited number of sentences. It may on the one hand be a sentence the user has freely chosen or it may come from a fixed number of sets. Furthermore, the user may or may not have seen this specific sentence before or even has had some tries to type it before. The corresponding user typing characteristics might come from a general template of this user or might be very specific for the presented sentence.

Many computers lock themselves after a period of inactivity. There even exist very high security applications which lock themselves after a fixed period of time, regardless of user activity. In both cases the user must enter his password to get access to the computer again. In such a case the user could be asked to type not his password but to retype a given sentence, so the typing characteristics can be used to determine whether or not this is the legitimate user.

11.4 Learning curve

One problem that is very specific for keystroke dynamics is the learning curve that is involved. When a person is using a new password for the first time, he needs not only to remember the password, but it seems as if he also needs to find the keys on the keyboard. Obviously he is well aware of the position of the keys (at least we assume he is a well trained computer user), but the various combinations of letters in the password is new to him. The longer he uses the same password, the better he remembers it and the easier it is to “automatically find” the keys on the keyboard without thinking. A normal biometric system with a one time enrollment and multiple uses might not be a correct procedure here. Also use of multiple templates, as is often done with signature verification, might not be a good approach. Beforehand we can already predict that the extracted features will change in the beginning to stabilize after a while. The changes between the i^{th} and the $(i+1)^{th}$ time will be small but after a while the difference between the first and the n^{th} time a password is entered might be substantial.

In such a case it might be a good idea to update the template on every entry of the input biometric. Meaning that every time a user enters his password and the features match the features in the template, the template is adjusted according to the way the password is entered this last time. Because we assume a learning process, the way the password is entered the last time is more important then the previous times.

Bibliography

[1] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.

[2] Colin Boyd and Anish Mathuria. *Protocols for Authentication and Key Establishment.* Springer, 2003.

[3] Brostoff and Sasse.

[4] Friederike D. Griess. On-line signature verification (master's project report). Technical Report MSU-CSE-00-15, Department of Computer Science, Michigan State University, East Lansing, Michigan, July 2000.

[5] Anil Jain, Ruud Bolle, and Sharath Pankanti, editors. *Biometrics: Personal Identification in Networked Society.* Kluwer Academic Publishers, 2002.

[6] Anil K. Jain, Arun Ross, and Salil Prabhakar. An introduction to biometric recognition. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(1):4–20, January 2004.

[7] Rick Joyce and Gopal Gupta. Identity authentication based on keystroke latencies. *Communications of the ACM*, 33(2):168–176, February 1990.

[8] Daniel V. Klein. "foiling the cracker" – A survey of, and improvements to, password security. In *Proceedings of the second USENIX Workshop on Security*, pages 5–14, Summer 1990.

[9] Stan Z. Li and Anil K. Jain, editors. *Handbook of Face Recognition.* Springer, 2005.

[10] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[11] Nalini Ratha and Ruud Bolle, editors. *Automatic Fingerprint Recognition Systems*. Springer, 2004.

[12] Henk C.A. van Tilborg. *An Introduction to Cryptology.* Kluwer Academic Publishers, 1988.

[13] James Wayman, Anil Jain, Davide Maltoni, and Dario Maio, editors. *Biometric Systems: Technology, Design and Performance Evaluation*. Springer, 2005.

[14] M. Wirotius, J.Y. Ramel, and N. Vincent. Distance and matching for authentication by on-line signature. In *Fourth IEEE Workshop on Automatic Identification: Advanced Technologies*, pages 230–235, 2005.