Multi Level Security in a Public Key Infrastructure

Edward Stempel

february 2000

Verslag afstudeeropdracht
Edward Stempel

Faculteit der Informatica,
Universiteit Twente,
Enschede

Afstudeercommissie:
Dr. J.H. Hoepman (Universiteit Twente)
Dr. A. Helme (Universiteit Twente)
Drs. J.R. Brands, mtd (Philips Crypto BV)
Prof. Dr. Ir. C.J.A. Jansen (Philips Crypto BV)
Prof. Dr. S.J. Mullender (Universiteit Twente)

Multi level security is a concept that already existed before computers were invented. The concept of multi level security is not very difficult. However, implementing it in real systems is.

Section 1 decribes some basic models about computer security and multi level security. These models are very theoretical, but are very useful for understanding the concepts and definitions.

Section 2 describes the basics of cryptography and how it can be used to offer some security services and establish trust. Also the concept of a Trusted Third Party and the Public Key Infrastructure are explained here because they are closely related to cryptography.

Section 3 describes issues about access control in a distributed environment and how mechanisms that deal with access control are affected by, and deal with such an environment.

Section 4 explains some mechanisms that are useful for making multi level security work. In particular the VKaart system is decribed.

Section 5 describes how an architecture (The MLS architecture) should be designed for having Multi Level Security in a distributed environment.

Finaly, section 6 shows some conclusions.

Contents

List of Figures

List of Tables

Preface

Somewhere in September 1998 I started looking for a place where I could graduate on the subject of securing distributed computer systems. At the same time I attended the course Secure Systems Engineering which was lectured by Sape Mullender who is professor at the department of computer science at the University of Twente. I asked Sape if he knew some companies that did some research in the field of securing distributed systems. Sape grabbed a biscuit-tin out of his desk that was bulged with visiting cards. He browsed the cards and gave me a few names and addresses, among which was Cees Jansen's address at Philips Crypto.

I contacted Cees Jansen and made an appointment with him to talk about some projects at Philips Crypto. I was particular interested in the so-called TTP/PKI project that was to be started early in 1999. Because I also need to earn some money for my living, Philips Crypto and I agreed that for a year I would work 2 days per week for the TTP/PKI project and would spend the 3 days per week at graduating and writing my thesis. The work was closely related to my thesis so this would benefit both my work and my thesis.

This thesis discusses multi level security, how the concept of multi level security can be implemented in a system using trusted third parties and in particular using a public key infrastructure.

Introduction

Early 1999 the following description for this thesis was formulated:

> Implementing Multi Level Security using a public key infrastructure.
>
> At Philips Crypto B.V. a pilot project will be started in order to set up a Public Key Infrastructure based on Trusted Third Parties and X.509.v3 in a hierarchical organization that uses Multi Level Security. For his thesis at Philips Crypto B.V, Edward Stempel will investigate the possibilities of a Multi Level Security (MLS) environment, and how a Public Key Infrastructure (PKI) can be brought into action to support MLS. A model will be developed for implementing a PKI that supports MLS. This model should deal with the constraints imposed by the PKI pilot. The existing security model for the Philips Products VKaart and VPN-Guard are the basis for these constraints. In particular, the following topics will be investigated: What different models of MLS are known? And which models are most appropriate to fit the PKI-pilot at Philips Crypto?

The first questions that arise are 'what exactly is security?' and 'what is multi level security?' A lot is written about computer security, and one thing about it is very clear: It is not a product one can buy Commercially Of The Shelf. Security is not solely about technical measures in hardware and software. It is also about the awareness of security issues by the people dealing with the information that should be secured. It is also about how the business is organized. There is no standard way to implement security. A complete defense against attacks depends on a broad range of parameters, hence making every solution unique. The subject is much to complex for standard solutions, and potential attackers are to clever and resourceful to be stopped by such solutions. [CE00] warns for to optimistic opinions that Public Key Infrastructures can solve a lot of security problems. It makes clear that there exist numerous problems that are easily overseen.

Often multi level security is seen as a set of rules how access control should be filled in. There is more to MLS that needs to be explained here. This thesis adds another dimension to the notion of multi level security, based on the conventional (paper) world: Multi level security is also about how access control should be implemented for different security levels.

This thesis describes very theoretical (i.e. not practical) models of security and particular multi level security. It describes what a Trusted Third Party (TTP) is and what a Public Key Infrastructure (PKI) is. Besides, it describes existing models and systems such as Kerberos and the Philips Crypto VKaart system. Based on the knowledge of these descriptions a model is described how one can build a distributed computer security system that is multi level secure.

1. Security models and Multi Level Security

1.1. threats

A threat to a computer system is defined as

> any potential occurrence, malicious or otherwise, that can have an undesirable effect on the assets and resources associated with a computer system.

Having defined a threat, a vulnerability of a computer system is defined as:

> some unfortunate characteristic that makes it possible for a threat to potentially occur

These definitions can be found in [Amo94], but are common in most literature about computer security. Using these definitions, an attack on a computer system is defined as:

> some action taken by a malicious intruder that involves the exploitation of certain vulnerabilities in order to cause an existing threat to occur.

A threat can also occur without the involvement of an attacker, i.e. unintentionally. This is called an incident. An incident therefore is defined as:

> The occurrence of one or more threats which are not caused by an attack

In this context, security is about preventing threats to occur. Although we will mostly talk about security in the context of preventing attacks, it is good to realize that security is not solely about preventing (malicious) attacks, but also about preventing incidents. Furthermore, security is not solely about preventing threats, but also closely related to fault tolerance.

Threats can be categorized into three different types:

- disclosure threats
- integrity threats
- denial of service threats.

The disclosure threat involves the dissemination of information to an individual for whom that information was not intended. The integrity threat involves any unauthorized change of information stored in a computer system, or in transit between computer systems. The denial of service threat arises whenever access to some computer system resource by a legitimate user is blocked.

1.2. Security services and models

Most literature about computer security distinguishes principals into objects and subjects. A subject is a person, or a program (perhaps acting on behalf of a person) that operates on an object. A subject is defined as an active computer system entity that can initiate requests for resources and use these resources. An object is a computer-related resource, such as a piece of information, a file system, or a printer. In some cases a computer process also can be considered to be an object. In general a object thus is a passive computer system repository that is used to store or process information. So an object is a computer resource. However the object may become a subject or vice versa in another request (see [Amo94]).

Different security services can be distinguished. A security service is concerned with the prevention of a threat. Different security services deal with different type of threats. [Hel97, pp 26-27] states that the level of security a system can provide, is defined and measured by estimating how well this system can handle specific threats. A secure system can offer one or more of the following security services:

- Authentication: An authentication service is concerned with establishing the true identity of a principal, eventually by verifying whether a claimed identity is authentic.
- Confidentiality: A confidentiality service is concerned with the protection of information against disclosure threats.
- Integrity: An integrity service is concerned with the protection against the unauthorized modification of information. Therefore this service deals with integrity threats.
- Availability: The availability service is concerned with the accessibility of objects in a computer system and therefore it is concerned with protection against both denial of service attacks and general failures in the system.
- Accountability: Accountability services are concerned with monitoring and auditing of authorizations, and serve in creating logs of access requests and granted services. This service is closely related to non-repudiation.
- Non-repudiation : A non-repudiation service is concerned with preventing the successful but illegitimate denial by someone that some operation was carried out, whilst that operation in reality was carried out.

Security models are used to define policies and relationships between subjects and objects in a computer system. In particular they deal with access control: Which subjects are allowed to perform some predefined actions (e.g. read or write) on which objects. There are different security models, each aiming at modeling some set of security services. Consequently these different models are useful for different purposes. For example The Bell LaPadula [DB73] security model is mostly concerned with confidentiality and therefore its use is appropriate in systems where keeping information secret is vital, such as in military systems. However the BIBA model is more concerned with integrity services and therefore its use is appropriate in systems where integrity of data is important such as in commercial systems (e.g. a banking system).

1.3. Security labels

A security label is defined using security levels and security categories.

A security level is an attribute that can be associated with principals to help denote their degree or sensitivity. For example in a military environment the set of military security levels can consist of unclassified, confidential, secret and top-secret. This attribute is hierarchical in the sense that there exists a total ordering between the levels: top-secret is higher than secret, which in turn is higher than confidential, which is higher than unclassified. In commercial environments another labeling of levels may exist.

A security category is defined as a non-hierarchical grouping of principals that share concerns. There is no ordering between the different categories. Besides they have no relation to the hierarchic security levels. Examples of categories are "medical", "human resource", "commercial", and "management". Categories are used to ascertain if someone really needs to know some information. For example, someone in the commercial department does not need to know medical information of employees. The so-called "need-to-know" principle is used in many organizations where confidentiality is an important issue. This principle simply states that someone only may access information if he or she needs this access for doing his work. Many different "need-to-know" categories can be distinguished.

A security label now is defined as an attribute for principals that denotes their hierarchical sensitivity and a set of need-to-know attributes. A security label thus consists of a security level and a possible set of categories (i.e. an element of the powerset of categories). So an security label is defined as a pair or tuple. For example, suppose in the military two levels are distinguished: confidential and unclassified. There are also two different categories: army and navy. Now, the eight possible security labels are:

- (unclassified, $\emptyset$)
- (unclassified, $\{$army$\}$)
- (unclassified, $\{$navy$\}$)
- (unclassified, $\{$army, navy$\}$)
- (confidential, $\emptyset$)
- (confidential, $\{$army$\}$)
- (confidential, $\{$navy$\}$)
- (confidential, $\{$army, navy$\}$)

Given a security label x=(l,c), the functions *lev* and *cats* are defined as lev(x)=l and cats(x)=c.

Now labels can be associated with subjects and objects by means of a clearance- or classification function. For both these functions the co-domain is the set of labels. For the clearance function the domain is the set of subjects and for the classification function the domain is the set of objects. The signatures of these functions are:

- Clearance: $s \to l$

- Classification: $o \rightarrow l$

In order to compare labels in the ordered set of labels, a boolean function with domain a pair of labels has been defined. This function is called the dominates function and is defined as follows:

dominates $(x_1, x_2) \Leftrightarrow \text{lev}(x_1) \geq \text{lev}(x_2) \wedge \text{cats}(x_1) \supseteq \text{cats}(x_2)$

In the same way an equals function can be defined:

equals $(x_1, x_2) \Leftrightarrow \text{lev}(x_1) = \text{lev}(x_2) \wedge \text{cats}(x_1) = \text{cats}(x_2)$

1.4. Security models and security policies

A security policy defines the conditions under which subject-accesses to objects are allowed in a specific system. These conditions are usually expressed as a set of rules that define the access requirements in the system. In order to enforce the security policy, appropriate mechanisms must be present on the system. Thus policies are the requirements and a mechanism is an implementation that should satisfy such a requirement.

A security model is a representation of a class of systems, in which unneeded details are abstracted in order to highlight a specific property or set of behaviors, i.e. security models are abstract descriptions of system behaviors. For example, a model can be described as a set of policies.

Often a policy is described in an informal manner by using a table. For example table 1.1 describes under which conditions subjects may access objects per type of access. In this example a subject has read access to the object if his clearance dominates the classification of the object. For having write access the subject's clearance must equal the object's classification.

Access Type	Relation (Subject, Object)
Read	Dominates
Write	Equals
Execute	Dominates

Table 1.1.: Policy table

This type of policy expression is easier to understand than formal expressions. However logical errors (and hence vulnerabilities) can be introduced easily using this way of expression, especially when the system is more complex.

[Amo94] describes a way to write down a security policy in a formal way. He assumes a system is represented by states and transitions that cause well-defined changes from an initial state. These changes are initiated by subjects requesting access to objects. A mediation function then determines whether an access is allowed. If the access is not allowed, the system state does not change. So, this type of function will be a Boolean function, relating the elements of the different sets (e.g. the set of security labels or the set of subjects). Now the security policy can be described by means of such functions. Note that in order for these functions to be meaningful, it is essential that the elements of the sets are identified unambiguous. For example the set of subjects has to be complete and must include the identities that request access to objects. The specifier of a security system will have to make his own functions in a similar way, according to his needs.

The function "allows" determines which accesses are allowed, and which are not:

allows: subject $\times$ objects $\times$ accesses $\rightarrow$ Boolean.

This mediation function can be used in combination with other useful aiding functions to create policies. Examples of such aiding functions are:

- the function "own" that determines which subject owns which object
  own: subjects $\times$ objects $\rightarrow$ Boolean.
- The function "admin" will determine if a subject is an administrator of the system.
  admin: subjects $\rightarrow$ Boolean.
- The function "dominates" as defined in the previous section.

With these functions formal policies can be described by using the mediation function "allows". Below some examples follow.

- The unconstrained policy: unconstrained: $\forall$s$\in$S, o$\in$O, a$\in$A: allows(s,o,a) $\Leftrightarrow$ true.
- The ownership policy: ownership: $\forall$s$\in$S, o$\in$O, a$\in$A: allows(s,o,a) $\Leftrightarrow$ own(s,o).
- The own/admin policy own/admin: $\forall$s$\in$S, o$\in$O, a$\in$A: allows(s,o,a) $\Leftrightarrow$ own(s,o) $\vee$ admin(s).
- The dom policy: dom: $\forall$ s$\in$S, o$\in$O, a$\in$A: allows(s,o,a) $\Leftrightarrow$ dominates(label(s), label(o)).

1.5. The Bell-LaPadula model

Elliott Bell and Leonard LaPadula who worked at the MITRE Corporation published the Bell-LaPadula model [DB73] in the early 1970s. The U.S government funded their research on security models and the prevention of disclosure threats. Therefore this BLP-model aims at confidentiality only. Bell and LaPadula recognized different levels of information, which are ordered in a hierarchy. They did not describe the concept of the non-ordered categories.

1.5.1. Functioning of the BLP model

In the BLP model the subjects and the objects are associated with a security label. The BLP-model basically has two rules: The No Read Up (NRU) rule and the No Write Down (NWD) rule. The latter is also commonly used in literature as the $*-property$.

No Read Up. Classified information may not be accessed by persons with a lower security clearance than the information. Therefore the NRU rule states that a subject with security label x will only be granted read-access to objects with security label y if security label x dominates security label y. For example, a person with security label secret will not be allowed to read a document that is classified top-secret, but will be allowed to read documents that are classified up to confidential.

A formal description of the NRU-rule for some subject s and some object o is:

NRU: allows(s,o,read) $\Leftrightarrow$ dominates(clearance(s), classification(o)).

No Write Down Classified information can "leak" to persons with a lower security clearance than the classification of the information, if someone with a high security clearance can write information with a lower security label. The information will then be labeled with a low classification, but may contain information that should have had a higher security label. Therefore the NWD-rule states that a subject with security label x will only be granted write-access to objects with security label y if security label y dominates security label x. For example a person with security clearance secret will not be allowed to write (into) a document classified confidential. However, this person will be allowed to write a document that is classified secret or top secret. A formal description of the NWD-rule for some subject s and some object o is:

NWD: allows(s,o,write) $\Leftrightarrow$ dominates(classification(o), clearance(s)).

1.5.2. Tranquility in the BLP-model

In reality, operations from subjects to objects are no atomic operations, i.e. operations are not performed uninterruptable and hence an intermediate state of the operation can be visible. In fact these operations are often comprised of sequences of read and write accesses. For example, when writing to a file, first the directory has to be read, in order to find (a place for) the file. Then the file will be written, and the directory will be written (updated). This non-atomic property of operations introduces a risk for information leakage when one ore more security labels change during an non-atomic operation. This problem is quite fundamental and has some serious consequences because in operating systems there must exist (non-atomic) processes that can change their security labels during their execution. Examples of these processes are the process scheduler, the memory management function and the (hardware) resource manager. This is in essence why in hardware of computers a mechanism is built that has different privileged states and where so-called unprivileged operations are trapped [AS91]. In other words: In order to perform properly, an operating system must already have some form of multi level security implemented in the kernel!

In order to prevent information leakage the NRU- and NWD-rules require that the security labels of subjects and objects are not changed during an operation. Therefore the strong tranquility property has been introduced. This property simply states that the security labels of subjects and objects never change during an operation.

The strong tranquility property reduces the degree of flexibility during operations. It makes changes of security labels during an operation impossible, even when such a change would not violate the NWD- or NRU-property. For example, when someone with an initial security clearance top-secret is performing some read-accesses on an object with security level confidential, this person will not be allowed to lower his security clearance to secret, although such lowering will not violate the NRU- or NWD-rule. Therefore the weak tranquility property has been introduced. This property states that the security label of subjects and objects never change in such a way as to violate a defined security policy (like the NRU- and NWD-rules).

1.5.3. Criticism on the BLP-model

The BLP-model allows a subject to write to an object with a security-label that dominates its own label. At the same time the subject is not allowed to read the same object. This causes the so-called "blind write". The possibility of blind writes enhances the integrity risk. It is not just the blind write that enhances the integrity risk, it is the fact that an subject can write to an object with a higher security-level that enhances the integrity risk. This fact enables the subject to append to an existing object with a higher security-level, or even to overwrite it! Almost at all times this will be seen as unacceptable. But then again, the BLP-model strives for confidentiality only. If one wants to strive for integrity too, one has to use another model.

If one applies the BLP-model to a distributed system, another problem will arise. Assume there are two systems: Secret subject A on system A and confidential object B on system B. Whenever subject A wants to read an object on system B, it has to issue a read-request to system B. However, issuing such a request will violate the NWD-rule, because subject A is not allowed to write (i.e. send) information to object B. This problem also arises for subjects like device drivers and memory managers (see section 1.5.2 and [AS91]). In order to solve this kind of problems a trusted subject has to be introduced that is allowed to override the NWD-rule (and perhaps also the NRU-rule). Such a trusted subject typically will be an administrator of a system, or the operating system. This solution thus passes the problem with confidentiality on to the trusted subjects. Therefore this criticism seems to be fundamental, because the problem can only be passed on to another subject which then has to deal with it.

1.6. The BIBA model

In the mid 1970s, Ken Biba who worked at the MITRE corporation, published the BIBA integrity model. He developed this model because he realized that the BLP-model introduced integrity problems. The BIBA-model consists of four different instances:

- the Mandatory Integrity Model,
- the Subject Low-Water Mark Model,
- the Object Low-Water Mark Model
- the Ring Model.

1.6.1. Functioning of the BIBA models

The Mandatory Integrity Model The Mandatory Integrity Model is quite the opposite of the BLP-model. It has two rules that should ensure the integrity of objects. These rules are the No Read Down (NRD) rule and the No Write Up (NWU) rule. A formalized description of the NRD and the NWU-rule for some subject s and some object o is:

NRD: allows(s,o,read) $\Leftrightarrow$ dominates(classification(o), clearance(s)).

NWU: allows(s,o,write) $\Leftrightarrow$ dominates(clearance(s), classification(o)).

The NWU rule ensures that a subject with a certain security label (say x) and thus has the security clearance (for integrity) x cannot produce (write) information with a higher security classification than x. The NRD rule ensures that a subject with security clearance x cannot acquire information with a security classification lower than its security clearance (x). So nobody can write (to) an object that has a higher integrity than his own clearance.

The Subject Low-Water Mark Model The Subject Low-Water Mark Model relaxes the No Read Down rule from the former model. A subject may access an object with a lower security-level than his own, but the subject will have to lower his security-level to that of the object, before it can perform a read-access on the object.

The Object Low-Water Mark The Object Low-Water Mark Model relaxes the No Write Up rule. A subject may write into an object with a higher security level, but only after it lowered the security level of the object.

The Ring Model The Ring Model allows a subject to obtain read-access to an object with a higher security-level, but does not allow the subject to alter (i.e. have write-access to) the object.

1.6.2. Comparison of the BIBA model to the BLP model

The BIBA model has a lot in common with the BLP-model. It is often said that the BIBA model is the BLP-model upside-down. It also requires tranquility properties in order to prevent the occurrence of integrity threats. Note also that this model aims at preventing the occurrence of integrity threats, but it certainly does not aim at preventing the occurrence of confidentiality threats. To a certain extent these two requirements are conflicting and therefore combining these requirements will reduce the usefulness of the model. Nevertheless it is possible to combine the BLP-model and the BIBA-model by putting together the rules of both models. This will result in a read-equal rule and a write-equal rule which satisfy the constraints of both models. This problem may be solved by using different labels (per principal) for integrity and confidentiality. Clearly, using the same label will give trouble. Moreover, it is logical to use different labels for different attributes, as the level of integrity should be considered to be another attribute than the level of confidentiality.

2. Cryptography and trust

2.1. Cryptography

> Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication and data origin authentication ([ea97]).

Cryptography thus is about mathematical techniques which can be used to implement security services.

An encryption algorithm is an algorithm that implements a mathematical function $e(p) = c$. It can be used to make a piece of text unreadable to others. Say p is a piece of readable text or plain text. When we apply the function $e(p) = c$ to p, we get the unreadable- or cipher-text c. We say that we encrypt p. In order to regain the original plaintext p from the ciphertext c, we need the inverse function of e: $d(c) = p$ in such a way that $d(e(p) = c) = p$. In this way we can decrypt c.

Anyone who knows the decryption function d and the ciphertext c can regain the plaintext by applying the decryption function to c. Therefore these functions need an additional parameter that is not known to anyone; the so-called key K. Then we have the functions: $e_K(p) = c$ and $d_K(c) = p$. These functions are good cryptographic functions if it is computationally infeasible to regain p, if one knows c and the functions d and e. Therefore it is necessary that there is a very large number of different keys possible to apply to these functions. Otherwise it would be possible to do an exhaustive search on all keys, i.e. trying all possible keys.

2.1.1. Symmetric encryption

If the key used for encryption is the same key used for decryption, or it is computationally feasible to reduce the decryption key from the encryption key (or vice versa) we are talking about symmetric encryption. Symmetric encryption is also known as secret key encryption, because in most cases the key should be a shared secret between a limited group of parties. Well known symmetric encryption algorithms are DES, Triple-DES, IDEA, Blowfish and RC4.

If two parties, say Alice and Bob exclusively share the same key K, then they can set-up a confidential communication-path between them, i.e. without anyone being able to read the plain text that flows from Alice to Bob and vice versa. Another useful property is that if Alice receives a piece of ciphertext, that she can decrypt (with her key K) into some plaintext that makes sense[1] , she knows that that piece of plaintext originates from Bob or herself and no-one else. This property is the basis for some authentication protocols, such as the challenge-response protocol (see section 2.1.7) and Kerberos (see section 3.4).

2.1.2. Asymmetric encryption

In contrast to symmetric encryption, asymmetric encryption uses different keys for encryption and decryption. The functions are $e_{K_1}(p) = c$ and $d_{K_2}(c) = p$. The keys are K_1 for encryption and K_2 for decryption. It must be computationally infeasible to reduce K_2 from K_1. Asymmetric encryption is also called public key encryption,

[1] In order to enable someone to ascertain wether a plaintext makes sense, some redundancy should be added to the plaintext.

because often one key is made public and hence called the public key, while the other key is kept private to one party and hence this key is called the private key. It should be noted that not always the decryption key is the private key. The scheme where the encryption key is private and the decryption key is public, is also possible (and useful). However, is should remain impossible to derive the private key from the public key, i.e. this requirement is linked to which key is made public, instead of whether it is used for encryption or decryption. Well known asymmetric encryption algorithms are RSA, and ElGamal.

Assume two parties, Alice en Bob generate their own key pairs and carefully keep their private key to themselves. We now call Alice's and Bob's public keys K_A en K_B respectively. Their private keys are called K_A^{-1} and K_B^{-1}. Alice and Bob know each others public keys, because they received them in such a way that they are sure the public key belongs to the other. Alice and Bob now can set-up a confidential communication-path between them. In order to do so Alice encrypts her message to Bob using Bobs public (encryption) key K_B. Only Bob can decrypt this message because only he has the corresponding (decryption) private key K_B^{-1}.

2.1.3. Digital signatures

Another useful application cryptography is about making digital signatures. For RSA it works the other way around compared to encryption. However, working the other way around is not always possible. This method is possible using RSA, but not possible using ElGamal! However a algorithm to create digital signatures based on ElGamal does exist, but this algorithm works differently from the example below:

Alice encrypts her message using her private key K_A^{-1}. Now, anyone who has her public key K_A can decrypt her message. So, the message will not be confidential in this way. However the message is authenticated, because only Alice could have encrypted this message with K_A^{-1}, because only she possesses that key. So as long as Bob believes only Alice possesses K_A^{-1}, he knows for sure that the message originated from Alice. This property is used for putting digital signatures on documents and for authenticating parties, e.g. via X509v3 certificates. It should be noted that different keys should be used for making signatures and encrypting data for confidentiality.

It is obvious that with public key encryption, distributing the keys is less problematic than with symmetric key encryption. However asymmetric algorithms are much less easy to compute than equivalent symmetric algorithms.

2.1.4. Hash functions

Another type of cryptographic function is the hash function. A hash function is a function that takes a piece of text of any size as input and produces as output a piece of text with a predefined size. Say M is the message, then $H(M) = h$ is the hash function of M and h is the hash value of M. Hash functions can be used to detect alterations to large documents. Therefore a good hash function should have the following properties:

- Given a certain M, it should be easy to compute h.

- Given a certain h, it should be computational infeasible to find a certain M, so that $H(M) = h$. (one-wayness)
- Given a certain M, it is computational infeasible to find a M', so that $H(M) = H(M')$. (second pre-image resistance)
- The hash function should be collision-resistant, i.e. the chance that two messages have the same hash value should be near zero. Obviously the hash value must have a size that is large enough to accommodate this property.

A hash function may have a key as extra parameter. Such a hash function is called a Message Authentication Code or MAC.

2.1.5. Signatures and certificates using RSA

Consider the scenario in which someone wants to sign a digital document, i.e. add some piece of information to the document, stating he agrees to the document. Of course it must be impossible to falsify this extra piece of information. Adding a signature to the document is done as follows (see left-side fig 2.1). Assume M is the document.

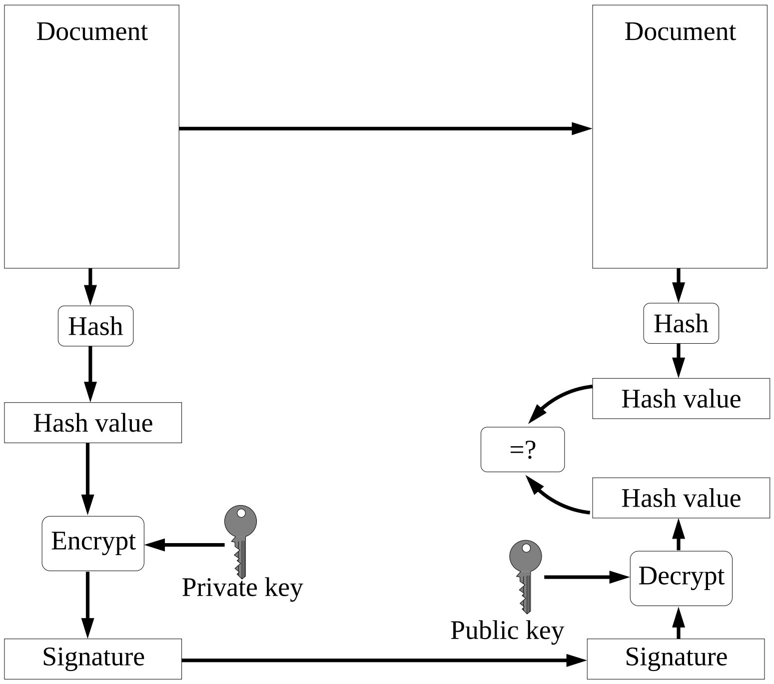

Figure 2.1.: Adding and verifying a signature

1. The signer calculates the hash value of the document $h = H(M)$.
2. The signer encrypts the hash value using his private key: $s = e_{K^{-1}}(h)$
3. The signer adds this value s to the document M

Now anyone who possesses the public key of the signer can verify the signature (see right-side fig 2.1):

1. The verifier calculates the hash value of the document $h = H(M)$.
2. The verifier decrypts the signature s that is added to the document using the public key of the signer $h' = d_K(s)$
3. The verifier compares h' to h. If $h' = h$ then the signature is genuine. The verifier now knows that this document was signed by someone who possessed the private key K^{-1}.

2.1.6. Authenticating public keys

The verifier needs to be sure that the public key and the private belongs to the person who signed the document, or to whom a encrypted message has to be sent, i.e. he must be assured of the binding between the keys and the signer. A possible way would be that the verifier keeps a database of all bindings between keys and persons. However this would be an inefficient method. It would be sufficient if the verifier knows one public key K_A and that public key belongs to someone (A) who is trusted by the verifier to have authority over the bindings between keys and persons (i.e. the authenticity of the key). This authority can write a document stating that some key K_x belongs to person x, and sign that document with his own private key K_A^{-1}. A document signed by some authority is called a certificate. Basically this is all a certificate is. The authority who signs the certificate is called the Certificate Authority.

2.1.7. Proving the possession of a key

Someone may use the possession of a key for authentication. Suppose Alice wants to authenticate Bob. Assuming Bob has a private key (i.e. he exclusively has this key), Bob can authenticate himself by proving he possesses this key. This can be done by executing a challenge-response protocol. In this scenario Alice generates a random string and sends it to Bob. Bob then encrypts this string with his private key and sends the ciphertext to Alice. Alice now decrypts the ciphertext using Bob's public key and compares it to the random string she sent Bob. If these strings are the same, Alice can safely conclude that the string was encrypted using Bob's private key. Therefore Alice knows she is really communicating with Bob. In this example asymmetric encryption is used. Symmetric encryption may also be used, however in that case Alice knows that she communicates with anyone who possesses the secret key. This can be herself, Bob, or even someone else, such as a TTP. The latter is the case in the Kerberos protocol (see section: 3.4).

2.2. Trusted Third Parties

A trusted third party is, as the name suggests, a third party that is trusted by other parties. A Trusted Third Party may be helpful when two or more parties want to securely communicate or trade with each other. Often these parties don't know each other, let alone that they trust each other. A TTP may also be helpful to sort out a disagreement between parties after some transaction. Therefore it is needed that the TTP plays a role in the transaction. The TTP will then be able to reproduce sufficient evidence about the transaction being in dispute. These kind of services are often called notary services

In real life a notary is a good example of a TTP. Therefore often the TTP is compared to the notary. A notary is impartial and trusted by all parties involved. Therefore a notary must comply to some rules that often are written down in national laws. Not just anyone can be a notary. For some transactions the involvement of a notary is required by law. Trading of real-estate is a transaction in which, under Dutch law, a notary must be involved.

If a dispute arises after a transaction has been done, the notary can produce evidence that is recognized by a judge. The judge trusts the notary. Note that a notary is not a judge. This will also apply to the TTP. The TTP can produce evidence concerning a transaction. However most national laws require a judge to settle some dispute between two parties about a transaction. Therefore also a judge needs to trust the TTP. Under Dutch law two parties may agree in advance that their disputes will be solved by some other party than a judge, and that this other party will accept the evidence provided by an electronic TTP. There are few national laws that give a TTP a legal status of provider of evidence. In the few cases that such laws do exist they exclude or at least limit the liability of the TTP, hence removing some essential part that makes a TTP trusted. Besides in those cases a party is responsible for any signature that matches his private key, no matter who actually used the key ([CE00]). This has some serious consequences. There exist a lot of ways to get a digital signature on some document, even if the owner of the digital signature does not wants to sign such a document. The signer even does not have to know he signed such a document! This can especially happen if someone has to sign a transaction using someone other's hardware and software (see also section 4.2). For example, this is the case when paying with a PIN-card or using some banks ATM to withdraw money. The technology of digital signatures is not yet mature enough. This is because a digital signature is about trust, and the technology to provide trust in a digital signature has not matured enough. A lot of different technologies are involved in the use of digital signatures, These technologies are still being developed and yet certainly not trust-worthy enough. Besides, because a lot of technology is involved in the use of digital signatures, the issue is very complex. Therefore the law-maker has to be very careful in assigning a legal status to a digital signature.

2.2.1. Basic forms of TTPs

There are three basic forms of a TTP; the in-line TTP, the online TTP and the offline TTP (fig: 2.2). The in-line TTP must always be involved in the transaction, because the two parties (A and B) do not communicate directly with each other, but communicate via the TTP with each other. The method to establish trust is quite straight-forward in this scheme. The two parties only have to setup a trusted communication path to the TTP. This form TTP may well be used to offer some non-repudiation services, because the TTP can easily log all relevant communications of

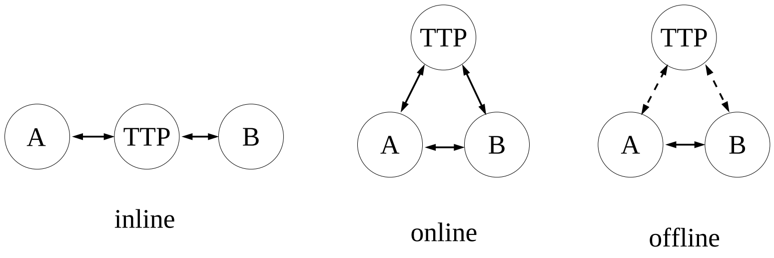

Figure 2.2.: Basic forms of TTPs

the transaction. However, there are three major disadvantages to this scheme: The TTP has to be involved in the complete transaction while the two parties may wish not to share every information concerning the transaction with the TTP. It may be necessary to have a confidential communication path between the two parties without the involvement of the TTP. Besides, the transaction may involve some large piece of information that does not need to be communicated via some trusted (and possibly expensive) communication path. The second major disadvantage is that the TTP always has to be available whenever a transaction has to occur. The third major disadvantage is that, due to its nature a TTP will soon become a bottleneck, if it gets used by a lot of users.

The online TTP also must be available whenever the two parties want to setup a trusted communication path. However the communication path will not go through the TTP. The two parties have a direct communication path with each other. In order to setup this trusted communication path they need some trusted information, e.g. what encryption key can be used to securely communicate with the other party. The TTP typically should provide some authentication service to the parties enabling them to authenticate each other. The TTP also may provide a notary service like a time stamping service, i.e. the TTP signs a document, thereby stating that it existed on a certain time. In this scheme some delegation of trust is needed. This means that the two parties must rely on the trustworthiness of the information the TTP provides about the other parties (e.g. that a certain encryption key belongs to a certain party).

At first sight the off-line TTP looks quite similar to the online TTP. However there are some important differences. There is no need to have the TTP directly involved in each transaction. In fact, it may be possible that the two parties never directly communicate to the TTP. This implies that an offline TTP is not suitable for some typical TTP services like a time-stamping service. But the great advantage is that the TTP does not need to be available every time a transaction occurs or some trusted communication path between the two parties is set up. Still the TTP can provide some trusted information to the parties. The keyword here is “certificate”, because the TTP can publish relevant information that can be trusted in certificates. Such a TTP is often called a Certificate Authority, or in short CA. The concept is quite simple in essence[2]:

1. The parties trust the TTP to have authority over some information and therefore the parties trust that this information provided by the TTP is correct.
2. The parties know the (public) verification key for the TTP’s signatures. Hence they can verify that any statement (a certificate) made by the TTP really originates from the TTP.

[2]However the real implementation is rather complex!

The two rules above make it possible to verify information without direct communication with a TTP. This kind of TTP service is very useful when trusted and static information is needed. However, due to the offline nature, it is not very useful if the trusted information is dynamic, i.e. changes often. In that case the life-time of the trusted information has to be short enough (depending on how dynamic the data is), or there has to be some kind of black-list mechanism such as described in section 3.3.1.

2.3. A Public Key Infrastructure

A CA certifies the binding of public keys to principals. These certificates can be stored in a repository that is publicly available, e.g. in a directory service. These certificates then can be used to authenticate the corresponding principals. The infrastructure of clients (i.e. the programs using the certificates to authenticate a principle), the CAs, and the repositories of certificates is called a Public Key Infrastructure.

A Public Key Infrastructure (PKI) enables users to acquire certified public keys. The keys are certified in the sense that they are bound to some identity. The signer of the certificate (i.e. the TTP) should be trusted to have authority over these bindings. Typically a PKI can be used to acquire a user's public key for verifying his signature, or for authenticating the user, etcetera. The whole idea behind such a PKI is that you can find out what someone's public key is, without previously contacting this person.

Figure 2.3 shows a simplified model of a PKI. A user may get his public key certified by registering this key at a certificate generating TTP. Often this TTP therefore is divided into two sub functions: the Registration Authority (RA), and the Certificate Authority. The users register their public key at the RA. The RA verifies that the user possesses the corresponding private key and checks the identity of the user. The RA gathers the information about the user that should be stored in the certificate. It then passes on this information in a request to the CA to certify the key with the information. The CA creates the actual certificate.

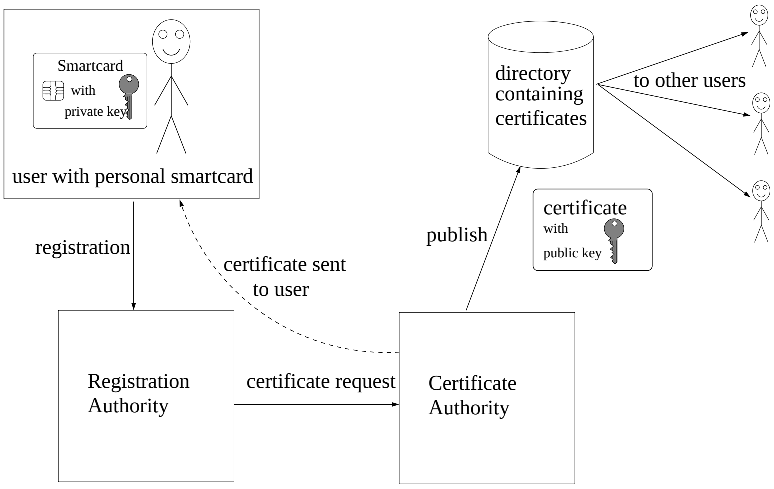

Figure 2.3.: A Public Key Infrastructure

Basically there are two ways to distribute the certificate:

- The CA sends the certificate to the user (eventually via the RA), so the user can pass it on to any principal it wants to authenticate himself to.
- The CA publishes the certificate on a directory, that can be consulted by any principal that may need the certificate. In contrast to the previous method, every principal can acquire the certificate (and hence the public key) without even having any previous contact with the user. This way of distributing a key is very useful for applications where some message has to be sent confidential to the user without having any previous contact with that user (e.g. email!).

The CA may also publish a blacklist of invalidated certificates on a directory (see section 3.3.1). More considerations on the use of offline TTP's and PKIs can be found in section 3.1.3 and section 3.2.

3. Authorization- and authentication mechanisms

3.1. Access control in a distributed environment

Access control is all about what operation subjects are allowed to perform on which objects. Access control can therefore be defined as the boolean function "allows" in section 1.4. This access control function is often represented by a matrix where each row represent a subjects and each column represent an object. Table 3.1 shows an example of such a matrix. In this example the subjects are Alice, Bob, Charles and Deborah. The objects are a printer, a disk, a floppy and a network. The entries corresponding to a subject and an object list the operations the subject may perform on the object (r=read, w=write, -=nothing). In this example Bob may read and write from/to the disk, whereas Charles may only read from the disk.

	printer	disk	floppy	network
Alice	w	rw	r	rw
Bob	w	rw	rw	rw
Charles	-	r	rw	r
Deborah	w	r	r	r

Table 3.1.: Access Control Matrix

The entries in the matrix correspond to the access types that the corresponding subject is allowed to perform on the corresponding object. In other words the subject is authorized to perform the access type on the object. The function "allows" or the matrix makes up a so-called Access Control Matrix (ACM). A whole row now represents the capability list of a subject, i.e. to which objects the subject has access The column now represents an Access Control List (ACL) which lists all subjects that have access to a particular object. [Mul89].

3.1.1. Access Control in real systems

Thus far there is no problem with this abstract view on access control. However when access control has to be implemented on real systems, several problems will arise. There has to be a mechanism that enforces the access control according to the ACM. This mechanism is called the Reference Monitor (RM) (see figure 3.1). Now, the first question one has to ask oneself when implementing the RM is what policy it should enforce and which RM therefore is the most appropriate to enforce the access control.

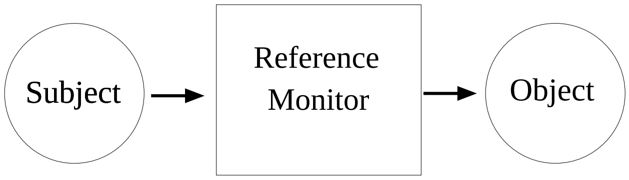

Figure 3.1.: Reference Monitor

It is not wise to leave the decision whether an access should be allowed to a piece of software that is under the control (over the essential tools to access the object) of the user. For example, a piece of software (readable by the user) with the access passwords encoded (not encrypted) in it will be vulnerable to malicious users who may attempt to find the password in the program code. In general it is necessary to shield the RM from the subjects in such a way that the subjects can not tamper with the RM. Besides, the system has to be designed in such a way that it ensures (within the assumptions of the threat model) that all access to objects only can go through the RM. Assuming the threat model requires protection against physical attacks, this implies that the RM (and the object) has to be physically secure from the subjects[1]. There is an important exception to this: Assuming a subject cannot decrypt an encrypted object without possessing the proper keys for doing the decryption, the object may be accessible to the subject, provided the object is encrypted and an unauthorized subject does not posses the proper decryption key. However a (malicious) user may still be able to destroy or alter the object and consequently cause a denial of service or integrity fault to the authorized users. However these attacks can be detected easily. The confidentiality however will not be in danger.

3.1.2. Distributing the reference monitor, subject and object

Replicating the object can relieve the denial of service threat, but this will introduce the problem of keeping the several copies consistent with each other. Another problem that may arise is that the RM needs (a part of) the ACM, but (that part of) the ACM is maintained at (a) different place(s): How does the RM know that

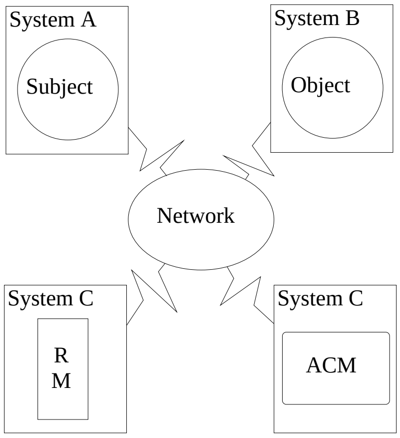

Figure 3.2.: ACM, RM, Object and Subject distributed

a subject (at a certain moment!) is authorized to access an object? Obviously, it is quite realistic to assume that the different parts (subject, object, ACM, RM) are distributed over several systems (see fig. 3.2). Therefore we have to deal with access control in a distributed environment. Note that in a distributed system the lines between the different entities generally are not secure, i.e. some malicious user can eavesdrop on the line (confidentiality), cut the line (denial of service) or even create false messages (integrity) and thereby masquerade itself as someone else (authenticity). These threats cause several problems. Note that this scheme has a major problem, as the subject may access the object without going through the reference monitor. For this reason alone there must be some kind of RM at the object's system.

[1] For example, a unix workstation offers some access control on the filesystem. However, often it is possible to open the workstation and access the harddisk with the filesystem, using another computer.

Authentication of the principal to the Reference Monitor

How does the RM know the authenticity of the subject and the object? The authenticity of both the subject and the object should be ascertained so that the correct access will be granted to the subject. By placing the RM near the object, the problem of authenticating the object may be solved . This means practically

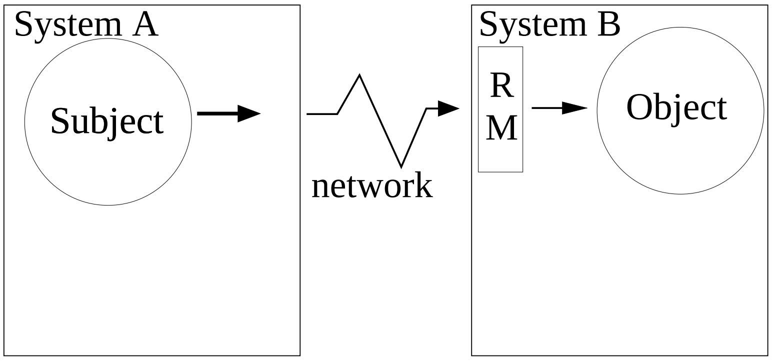

Figure 3.3.: RM placed near the object

that the RM should reside on the same system as the object does (see fig 3.3). In other words: The RM and object should not be separated from each other. The subject however should be separated from the RM to prevent the subject from tampering with the RM.[2] Besides this, it is often desirable for other reasons to be able to separate the subjects from the objects (and thus the RM) . The problem of authenticating the subject is more difficult, because the RM and the subject are not on the same system (trusted by the RM).

Encryption As stated before, the usage of encryption techniques may mitigate the conditions. If the object is encrypted, there is no drawback concerning the confidentiality when placing the object within reach of the subject[3]. This is because the subject still cannot read the object as it is encrypted. However, this solution will then transfer the original problem to the problem of keeping the key confidential. In other words, the key for decrypting the object should still be securely separated from the subjects. The VKaart system that is described in section 4.3 works in this way. Giving the decryption key itself to the subject has the major drawback that the object must be re-encrypted with another key whenever the subject's access-right to the object gets revoked. Of course one should be aware that placing the encrypted object near the subject still leaves the object vulnerable to integrity threats and denial of service treats.

Cryptography may also be used to authenticate a principal. A subject can prove its possession of a key by using encryption. Assuming only that subject has access to this key, one can ascertain the possession of the key by the subject by executing a challenge-response protocol (see section 2.1.7) with the subject. If the RM somehow knows to what user(s) this key has been bound, the RM can authenticate the user(s). There are several techniques to authenticate the subject. These techniques will be discussed in section 3.2 and 3.4. Note that having ascertained the authenticity of a subject does not imply that the RM knows whether the subject may access some object. For that the RM also needs an up-to-date version of the ACL.

[2] Remember that the subject is an active entity, whereas the object is a passive entity
[3] i.e. without intermediation by the reference monitor

3.1.3. The Reference Monitor and the Access Control Matrix

How does the RM know the integrity (correctness) of the ACM? The RM should be assured of the correctness of the ACM. If the ACM would be modified (un)intentionally, some access could be granted that should not be granted, or the other way around. This is of course highly undesirable. The ACM could be sent to the RM via a secured channel and thereby assuring the RM that the ACM originates from the correct manufacturer (i.e. the maintainer). This can be done real-time, which ensures a high level of integrity. But it also introduces a high risk of denial of service whenever the channel between the maintainer of the ACM and the RM gets interrupted.

Using Access Control Lists The ACM can also be stored with the objects, provided that subjects can only change the ACM through the RM (see fig: 3.4). Binding the ACL to each object can accomplish this. In other words: The ACM will

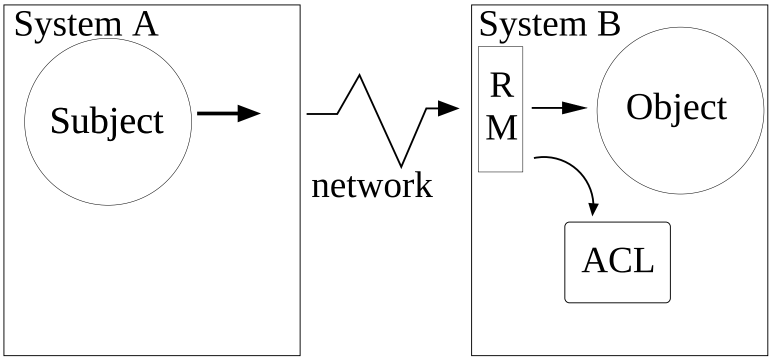

Figure 3.4.: The ACL stored near the object and Reference Monitor

then be protected by the same mechanism it serves. The big advantage of this idea lies in the availability it creates: The ACL is always available whenever the object is available. The drawback however is that the maintainer probably has to change the ACL from a different point than where it resides. This can cause problems with keeping the ACLs up-to-date enough whenever the maintainer cannot reach the ACLs due to a denial of service. In a way, this means that either the integrity of the ACM gets compromised or its availability gets compromised. In fact there will always remain a tradeoff between availability and integrity in respect to whether the ACM represents the most recent version. This is an important conclusion to recognize.

Using Capabilities The ACM can also be stored with the subjects, provided that the subjects cannot change the ACM themselves. Binding the capabilities to the users can do this (see fig: 3.5). Each user can present his capabilities to the RM. Cryptography can ensure the integrity of the capabilities. This can be done by having the maintainer of the ACM digitally sign the capability. This scheme also has the disadvantage that the maintainer has to change the ACM from a different point than where the the capability resides. Besides that, revoking the access rights means that all possible copies of the capability should be destroyed, which is almost impossible to guarantee, or some kind of black list with capabilities should be maintained. This in turn will cause problems due to the necessity of its availability. Reducing the validity of the capability to a limited period can mitigate

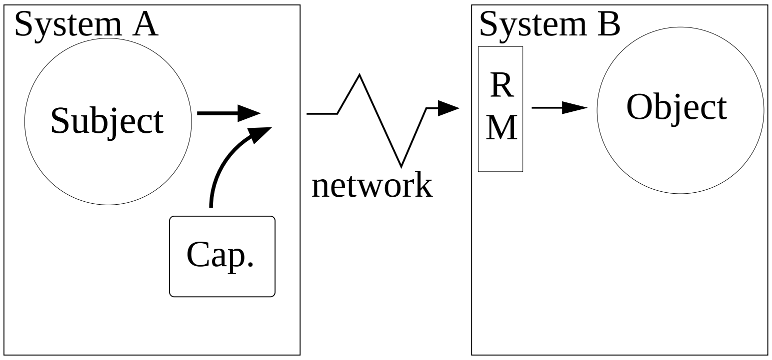

Figure 3.5.: The Capability stored near the subject

these problems. However, revoking a capability before the end of its validity period will still raise many problems. An attribute certificate (see section 3.2.2) is an implementation of capabilities.

Offline checking versus online checking of the ACM Offline checking of the ACM (for example by means of an ACL or a capability) has a great advantage with respect to the availability of the (needed part of the) ACM. However, it introduces a period in which some access right has been granted in the past to one ore more subjects regardless of the actual access right that the maintainer at that actual moment wants to give to the users. This is the validity period of the capability (see section 4.1). This means that the maintainer of the ACM cannot revoke this access right unconditionally and immediately. (consider the scenario where the maintainer is prevented from updating an ACL or a capability). The only cure to this problem is by letting the RM make an online check of the capabilities or ACL's validity at the maintainer. The major drawback of online checking is that it introduces a weakness in the availability of the verification process.

For example, consider the usage of the telephone directory (i.e. the book that the phone-companies distribute). It is known that already at the moment that this directory is distributed, it is (partial) outdated because every day people move and hence the new addresses and phone numbers of these people cannot be in this directory. To overcome this problem, the phone company has a service where one can check the most recent version of the directory. This service provides an online check[4], e.g. a service phone number where one can ask an operator to query the most recent database of the company.

It is obvious why the phone company still distributes the books with the phone directories: Exclusively relying on the online service would overload this online service and hence make it impossible to find some address or phone number at all. One can safely say that there is a tradeoff between the availability of data and the actual correctness of the data. These problems are (partially) dealt with in section 3.3.1.

[4]For technical reasons these directories are not really online, but are updated on a high frequency basis, e.g. 4 times a day.

3.2. X.509 certificates

In a distributed environment there is a need for trusted information about principals. This information may involve a public key for authentication, the Access Control List of an object, or the capabilities of an subject. In most cases the lifetime of these kinds of information will have different orders of magnitude: In general, the public key for authentication of a principal will have a longer lifetime than the lifetime of information about authorization. This is because authorization information (e.g. the Access Control Matrix) can be very dynamic, and hence this information may change often. Authenticity information normally will not change. Hence the binding of authentication keys to principles will change seldom, for example when the key lifetime expires or the corresponding private key is (suspected to be) compromised.

Obviously a certificate should have a lifetime of the same order of magnitude as the lifetime of the information incorporated in the certificate has: If the lifetime of the certificate is too long, the certificate will probably be revoked relatively early in its lifetime. Then a new certificate has to be issued for that piece of information. This will result in an unnecessary administrative overload and large (perhaps making it not maintainable) black-lists containing old certificates. If the lifetime of the certificate is too short, lots of certificates about the same unchanged piece of information need to be generated, which also generates an unnecessary administrative overload. If certificates must be generated in a higher frequency, the CA must be available to generate these certificates more often as well. In the extreme case the CA will become a sort of online TTP. It is for this reason that there are different type of certificates.

3.2.1. X.509 public key certificates

In 1988 ITU (formerly CCITT) published the first recommendation X.509. It defines a standard Public Key Infrastructure certificate format and is part of the X.500 directory recommendations. The X.509 public key certificate is typically designed for information with a relatively long lifetime. Besides a certificate format, X509 contains a protocol intended for setting up authenticated communication between two principals, assuming each knows the public key of the other (which could be found in a X.509 certificate). Unfortunately the protocol in this first version of the X.509 recommendation contains two weaknesses which can be exploited by an intruder [BL92]. These flaws were fixed in later versions of X.509. The second version X.509 was published in 1993. It also contains a sort of black-list mechanism called the Certificate Revocation List, or in short CRL (see section 3.3.1). In 1997 [x5097] was published. It comprises the third version of the X.509 recommendation and is therefore also known as X509v3. This is the current version at the moment of this writing.

3.2.2. X.509 attribute certificates

The X.509v3 recommendation includes a standard format for a different kind of certificate: the attribute certificates. These certificates are designed for information with a relative short lifetime, such as information about authorizations. The attribute certificate is coupled to a public key certificate. In this way the binding of a user to his public key can have a long lifetime, while the information in the attribute certificate can have a shorter lifetime. This makes attribute certificates a good medium for dynamic information such as authorizations.

3.3. Revoking certificates

3.3.1. Certificate Revocation Lists

The major disadvantage of certificates (and all capabilities in general) is that once they are issued, the information in it is assumed to be trustworthy as long as the signature on it can be trusted. However, this information may become invalid for several reasons, but it is still signed by a trusted signature. Therefore the signature has a certain lifetime. This means that within the certificate, besides the information itself, two points in time are signed. The signer of the certificate thereby states that the certified information is only to be considered trusted between these two points in time. This limits the lifetime of the certificate to some finite length of time. Evidently, the verifier of a certificate has to know the right time!

However, within the interval of the certificates lifetime, the information must be considered trustworthy because the certificate is valid at that time, no matter what. To reduce the risk that the information remains certified when it should not, two measures can be taken. The most logical measure would be to reduce the lifetime of the certificate. This however would force the signer to create certificates more often. From the viewpoints of availability, efficiency and security there is an upper bound and a lower bound to the lifetime.

The other way is to create a black list of certificates. All certificates on this list should not be trusted. So all certificates on such a list are revoked. Hence such a list is called a Certificate Revocation List. A certificate may incorporate places where these CRLs are distributed, the so-called CRL distribution point(s). The CRL's are signed as well. Besides, the CRL can be upgraded at predefined intervals. The verifier of the certificate now has to consult the CRL to make sure a certificate still can be trusted. Because the CRL will be updated at predefined intervals (the next update will be stored in the last CRL), a verifier also can ascertain the CRL is recent enough. This may prevent a successful attack (i.e. by using a revoked certificate) by applying a denial of service attack on the CRL distribution point.

3.3.2. Online Certificate Status Protocol

In order to offload the certificate handling from a reference monitor (client) to a server, the IETF is working on the so-called Online Certificate Status Protocol. This is a protocol where a client can offload verification of a certificate to a server. The server then gives a variety of valuable information about a certificate, such as whether or not the certificate is valid. The reason for making such a protocol is that for deciding whether or not a certificate should be trusted (and specific for attribute certificates), the client (i.e. reference monitor) must have implemented a lot of specific rules. Delegating certificate-based access control decision functions, including the processing of attribute certificates to an OCSP server would mitigate such problems. It should however be noted that using OCSP implies having to trust the OCSP server, i.e. adding another TTP to the already existing TTPs.

3.4. Kerberos

3.4.1. Kerberos basics

Kerberos is an authentication and authorization mechanism or service. It originated in 1986 at the Massachusetts Institute of Technology, where it is still being developed. Its use is widespread and today there are more developers of Kerberos than the MIT only. The most recent version of Kerberos today is version 5. Kerberos is a interesting protocol because there are quite some applications that are Kerberos-enabled (also known as Kerberized). Therefore Kerberos can be used as an intermediate authentication mechanism between lots of existing applications and other authentication mechanisms such as a public key infrastructure. In august 1999 Microsoft announced to incorporate Kerberos as the core security protocol into Windows 2000 [Cha99].

Kerberos is based on the concept of a Trusted Third Party. The TTP in the Kerberos service is trusted in the sense that each principal that can authenticate itself using the Kerberos service (and hence the TTP), is who it claims to be. Kerberos is based on the Needham and Schroeder key distribution protocol and uses symmetric, or secret key, encryption. Kerberos is a client-server application, where the clients are the principals that want to authenticate themselves to other parties. The servers, in this context, are the Trusted Third Parties of the Kerberos system[5]. These TTPs keep a database in which they store client-keys that are exclusively known to the client and the server. Therefore the Kerberos server can use these keys to authenticate a client. Kerberos generates session-keys which are given to two clients and no one else. These session keys are used by the Kerberos-clients to communicate with each other. The Kerberos implementation at MIT uses the DES encryption algorithm. However all cryptographic functions are put in a library that may be replaced by another cryptographic library. In this way Kerberos may be implemented using other cryptographic algorithms [JS88].

Kerberos is used to perform mutual authentication and authorization between a client and a server. Therefore there are two Kerberos servers: The Key Distribution Center (KDC) and the Ticket Granting Server (TGS). When a user logs in at the

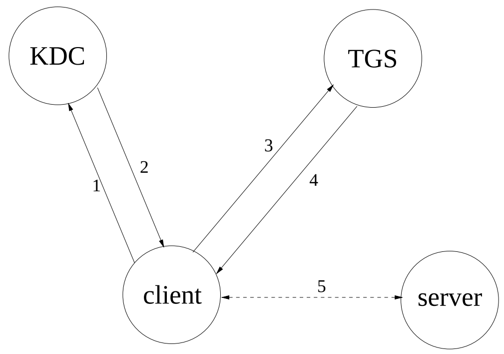

Figure 3.6.: Kerberos

client machine, the client sends a request (fig: 3.6, 1) for a so-called ticket granting ticket to the KDC. The KDC then checks this request and if the check evaluates

[5] Do not confuse the Kerberos client-server model with the client-server model between the principals.

okay, it sends the ticket granting ticket to the client (fig: 3.6, 2). Now the client may act on behalf of the user. When the clients wants to use a service from a server, acting on behalf of the user, it has to authenticate itself to the server (and optionally the server itself to the client). The client therefore has to get a so-called server-ticket. To get the server-ticket, the client sends a request for a server ticket to the TGS (fig: 3.6, 3). In this request the clients incorporates the ticket granting ticket (which proves that the client is acting on behalf of the user) it received earlier from the KDC. The TGS now checks this request, including the ticket granting ticket, and if this check evaluates positive, the TGS returns a server-ticket to the client (fig: 3.6, 4). The client can use this server-ticket to setup an authenticated channel between the client and the server (fig: 3.6, 5). Furthermore it can use the server-ticket to setup a confidential channel to the server. Kerberos then has reduced the initial key distribution problem, i.e. the problem of distributing $O(m^2)$ keys for m principals to an initial problem of distributing $O(m)$ keys.

The messages between the client, server, KDC and TGS are constructed in such a way, that the system cannot be compromised by a man-in-the-middle attack. For more information on these messages one can consult [JS88] or any other of the many articles about Kerberos.

3.4.2. Kerberos and public key cryptography

The basic Kerberos protocol is based on symmetric or secret key cryptography. This implies that the KDC and the user have to share a secret key. Therefore there is no way to distinguish a compromised KDC from a client because they both possess the same key. This naturally has some consequences for the clients ability to repudiate a transaction. The KDC has to maintain a database of secret keys which it has to protect from the outside world. If asymmetric, or public key cryptography were to be used the KDC would not have the user's secret key and could therefore not pose himself as the user. Moreover, the KDC then does not need to maintain a database with secret keys. The KDC only has to protect its own private key, just like the users should do.

The Internet Engineering Task Force is working on a proposal to enhance the Kerberos standard with the ability to do the initial authentication based on the usage of public key cryptography. This proposal will update [JK93] which specifies version 5 of the Kerberos protocol. At the moment of this writing the proposal is a draft that expires on April 30, 2000. In this draft the user still sends a request for a ticket granting ticket to the KDC. However, in this request the user sends his certificate along with the request and he signs the request using his private key corresponding to the certificate. The KDC can verify the signature on the request using the certificate. The KDC then knows the authenticity of the initial request and can proceed in the same way as the conventional Kerberos version 5 protocol describes. The user certificate may be signed by the KDC or by some Certificate Authority that the KDC trusts. The rest of the Kerberos protocol does not need to be changed. Hence only the KDC part and the initial Kerberos-login at the user-part of existing Kerberos authentication mechanisms need to be changed. It should be noted that such a scheme does not provide a non-repudiation service to the servers, as the client authenticates himself to the TGS using a Kerberos-ticket made by the KDC, instead of using his own private key. Hence either the client or the KDC may have used the ticket. Off course, if the KDC is malicious it may have distributed copies of the user's ticket to other clients. In general a non-repudiation can not be made by some intermediate TTP only. It must always involve the verification of something that is signed using the client's private signature key where only the client possesses that key, e.g. by using a challenge-response algorithm.

4. Assumptions and considerations

4.1. Multi Level Security and validity periods

The notion of multi level security access control can be viewed from different perspectives. On the one hand, access control can be multi level secure because of the implementation of a certain policy in the Access Control Matrix such as the No Read Up policy. On the other hand, it can be important how the access control mechanism is implemented. I.e. an access to a high level secure object will require another mechanism than access to a less high secure object. E.g. a low secure object can be accessed using an offline checking mechanism like a certificate, whereas a high secure object may only be accessed by using an online checking mechanism. In between, there exists a mechanism that uses a cached (part of the) Access Control Matrix where the cached ACM has to be updated after some predefined time has been elapsed. For example, this can be a system using certificates and a Certificate Revocation Lists where the validity period of the CRL will be a parameter for the level of security: The validity period is the elapsed time (since the latest online check) before a new online check must be done. The smaller the validity period of the CRL, the more up-to-date the CRL (and thus the relevant part of the ACM involved) will be. Using a validity period of zero seconds will practically be the same as using an online verification mechanism. Setting this period to an infinite time will be practically the same as using a certificate mechanism without using a CRL (see fig: 4.1). Obviously it is important for the Reference Monitor to have the

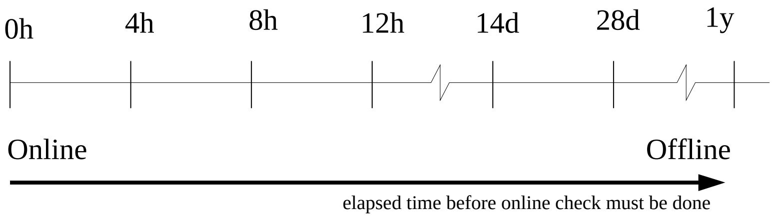

Figure 4.1.: Online vs offline checking

most up-to-date version of the Access Control List: If the RM has no up-to-date ACL, it will grant an access request from a subject whose access rights have been revoked after the RM got its copy of the ACL and cached it. So there is a certain risk in using cached copies of the ACL. However, the major benefit of using cached ACLs is that an ACL will always be available. There is a tradeoff between the integrity and the availability.

When building a distributed Multi Level Secure Access Control Mechanism it is assumed that for different levels, different validatity periods are the most appropriate, because the shorter the validity period is, the more up-to-date the certified information is. This in turn will reduce the risk that access will be granted when it should be denied, or vice versa. This notion adds another dimension to the notion of Multi Level Security: It is not only about how the ACM should be filled in, but also what mechanisms should be used in enforcing the access control. Choosing the right validity period should be based on how well the objects (and subjects) should be protected, but also on other criteria such as the costs of keeping the RM online and how practical or desirable it is to keep the RM online.

4.2. The Trusted Computing Base

4.2.1. The trust architecture

[Hel97] describes a trust architecture of a computer system. In this system a computer has an interface to other computer systems (the outside world), e.g. a computer network. Furthermore, the computer has a human-visible interface to the human user, e.g. a monitor and a keyboard (see fig: 4.2). The idea is that the

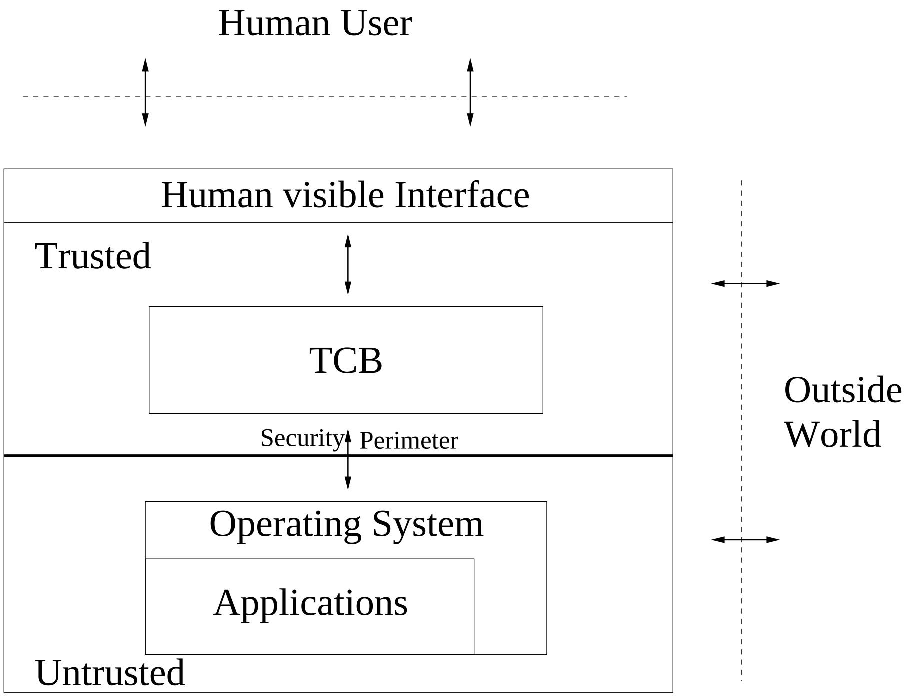

Figure 4.2.: The trust architecture (adapted from [Hel97])

computer is acting on behalf of the human user. However the user can only see what is presented to him through the human-visible interface. In some way the user must be guaranteed that what is presented to him is genuine and that any information the user gives to the system will be processed correctly. This counts for any piece of information that needs a security service[1]. For example, say the user is presented a transaction contract to buy a product from company x for the price of \$200. The user is now requested to authorize the payment of \$200. The user now needs several security services: He must be assured that he is dealing with company x (authentication). He also wants to know that the deal that the computer presents to him really is what the company is offering him (integrity). We say that there must exist a trusted path or a secure channel between the company and the user. The user also must be assured that when he enters some confidential information (e.g. a password), this information can only be seen by the party that needs this information. Besides it is often desirable that none of the parties can successfully deny a transaction that actually did occur (non-repudiation). Some kind of logging of certain critical events therefore may be necessary.

It is obvious that the human user cannot build a trusted path between himself

[1] regardless what kind of security service: integrity, confidentiality, authenticity etcetera

(through the user interface) and the other parties based on cryptographic algorithms. This is where the Trusted Computing Base or TCB comes in the picture. The human user is presented some information through the human-visible interface, e.g. a contract. This interface now has to be under control of the TCB, and the human user somehow must be assured that the human interface is controlled by the TCB (for example by flashing a led, that only the TCB can control). Furthermore, all information that is entered by the human user (e.g. through the keyboard) will only be seen by the TCB. Now the TCB can perform the needed security services for (or on behalf of) the human user. However the human user must be guaranteed that the TCB is functioning properly. Therefore the TCB must be a certified and unmodifiable subcomponent of the computer system.

4.2.2. The adapted trust architecture

Suppose someone wants to use a conventional computer configuration where in it is difficult to place a TCB between the non-trusted OS en applications on the one hand and the Human visible Interface on the other hand. Is there a way in which the human user can still get some guarantee that the Human visible Interface is not compromised by a faulty OS or application? Figure 4.3 shows an adapted trust architecture where the TCB has to use the OS to communicate with the Human visible Interface (and hence the user). Now the OS (and possibly the applications) also have to be certified in some way. A possible solution is to get the TCB sign the OS once it is approved by the user[2]. Such a method is used in the VKaart system that is described in section 4.3.

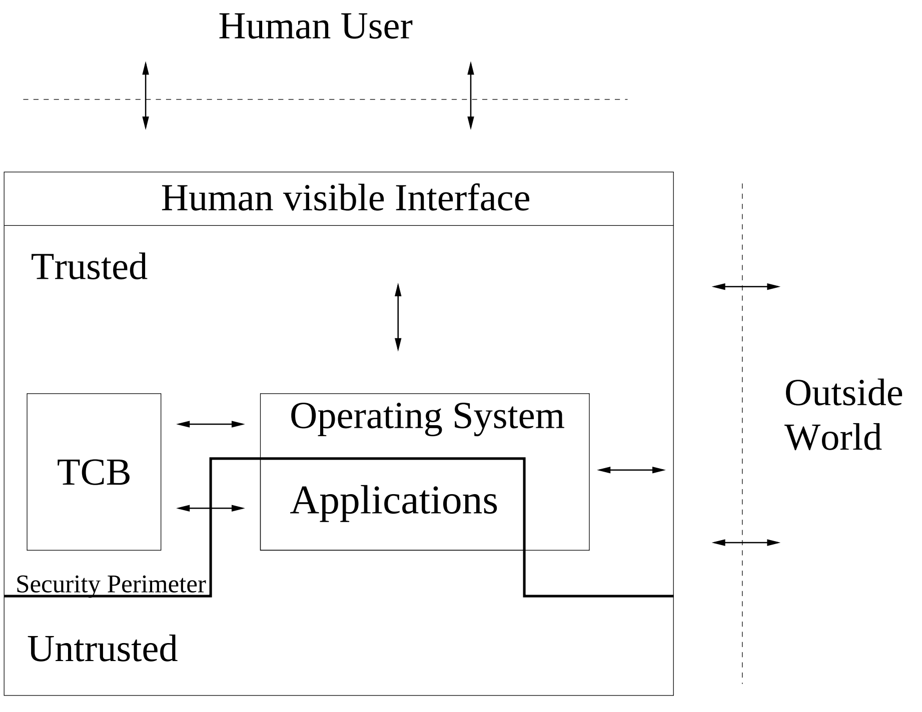

Figure 4.3.: The adapted trust architecture

A great disadvantage of this solution is that it is practically infeasible to assess

[2] This might require a special user such as a security officer

whether an OS or even an application can be trusted to function properly. However, if sufficient guarantees can be given about the proper functioning of the OS, then this trust architecture is almost as safe as the previous trust architecture. However one should keep in mind that this in fact is a rather big "if". If one is concerned about the security, sufficient guarantees should be as close as possible to 100% guaranteed, because if one does not strive to the 100%, an attacker almost certainly will benefit from the remaining percentage that cannot be guaranteed. Trying to keep some weaknesses secret, i.e. having a policy of security by obscurity, has proven not to work. Moreover not all operating systems (e.g. Microsoft Windows) do have a clear, well-defined distinction between what is part of the OS and what is part of an user application. Therefore it is not clear where the security perimeter is situated. Add to this that the security of OS functions is often not guaranteed as well. Hence guaranteeing that such a system is secure, is factually impossible.

4.3. The Philips Crypto VKaart system

4.3.1. simplified overview of a VKaart station

The Philips Crypto VKaart station (fig: 4.4) consists of a Personal Computer with the Windows NT4.0 operating system on it. Some of the Operating System functions are adapted for the VKaart station. The PC has a PCMCIA adapter in which a dedicated PCMCIA card (the VKaart card) can be inserted. The VKaart card will be the Trusted Computing Base (TCB) of the VKaart station. Optionally the PC has a smart card reader which can be used to authenticate the user.

The VKaart station encrypts all user files on the hard disk. The actual encryption is done by the TCB. This TCB also stores the necessary keys. These keys therefore never have to leave the TCB.

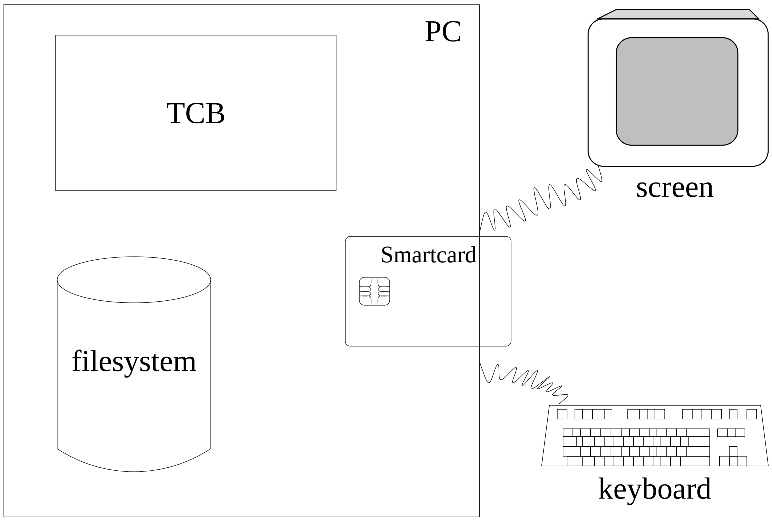

Figure 4.4.: VKaart system

All system files (needed by the Operating System) are digitally signed by the TCB. During the booting of the OS, the TCB checks the signatures of these files in order to detect the eventual tampering with these files. If the signatures are okay, the TCB will perform normally. If not, it will block the TCB, thereby preventing the user from running a corrupted VKaart station. The user still may boot the system as a non-VKaart system, but all encrypted files will then be unreadable.

The user has to authenticate himself to the VKaart station and in particular to the TCB before the system permits access to the encrypted files. The user can do this either by entering a password on the keyboard (something the user knows), or by inserting a special personalized smart card in the PC (something the user has). Optionally the user may choose, what security level he will use after logging in. After the authentication the security level of the user (i.e. his clearance) will be ascertained by the TCB.

The VKaart system works in sessions, i.e. it has one clearance for all functions at a time. This clearance is ascertained at the login of the user. The user may choose to alter this clearance during a session. In order to do this the VKaart system first must close all (encrypted) user files, because of the tranquility property (see section 1.5.2).

4.3.2. multi level secure file access

When the OS receives a call to access a user file, it has to decide whether this access should be allowed or not. The ACL for the file is stored in the file itself, i.e. it is attached to the encrypted file. Furthermore a header is attached to the file. In this header additional information about the file is stored. In particular its security level is stored in there, but also a reference to the owner of the file is stored in the header. The access control of the file is regulated by a complicated key management scheme which is shown in figure 4.5. The file is encrypted using a fast

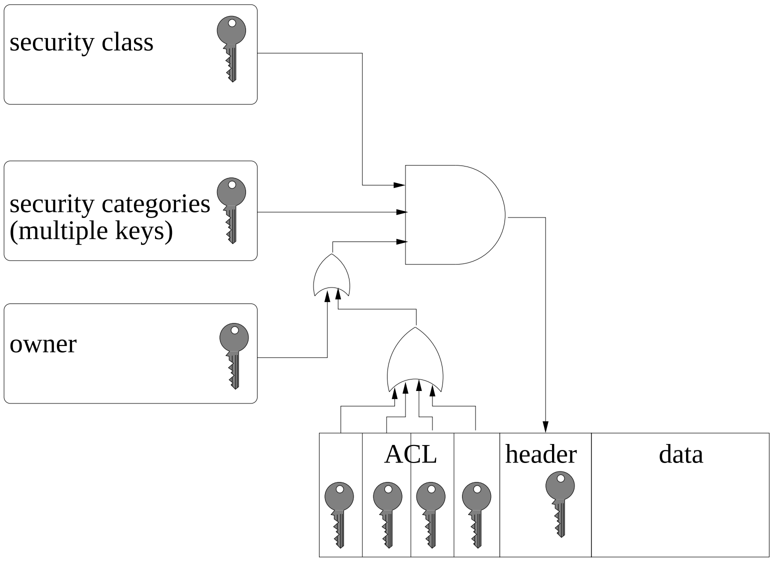

Figure 4.5.: Key management scheme for file encryption in the VKaart system

symmetric encryption algorithm. The symmetric key for this encryption on its turn is also encrypted. This key is encrypted using an asymmetric encryption algorithm (and therefore) using an asymmetric key (the so-called Key Encryption Key). The Key Encryption Key can only be derived (using a mathematical algorithm) if one possesses the following keys:

- The necessary security classification key (exactly one key)
- The necessary security category keys (the number of these keys varies from 0 to 24)
- The owners key or the ACL-key (exactly one key)

The symmetric file encryption key that is encrypted with the KEK, is stored in the header of the file. The classification level and the security category form the mandatory access control attributes. The ACL keys or the owners key form the discretionary access control attributes. Only the owner of a file and the users in the ACL having write access may change the ACL.

The ACL lists all users that are allowed to access the file. For each such user the ACL-key is encrypted with the user's public key and stored in the ACL. In this way only the users who are listed in the ACL can retrieve the ACL-key from the ACL.

Besides, the file is digitally signed by the owner. Any tampering or altering of the file will be detected. Other users can verify the signature on the file because the owners signature verification key is stored in a certificate.

Note that because the file is encrypted, the file can be safely stored anywhere without any risk of disclosure. Therefore it is possible to store the file on a normal (i.e. not secured) network file server.

4.3.3. Key management

Decryption of the ACL-key and the derivation of the Key Encryption Key is done in the TCB. Infact all encryption and decryption is performed by the TCB. This means that all keys do not leave the TCB for the decryption or encryption of a file. The users' public keys (one for signature verification and one for file encryption) are stored in so-called X.509-like certificates. These are Philips Crypto proprietary certificates. The certificates are signed by a trusted third party called the Security Management Station (SMS). The SMS is a special VKaart station that is stored in a secure place. The SMS servers a group of VKaart stations, called the VKaart domain. All keys for the security levels and security categories of a VKaart domain are managed by the SMS.

When a VKaart station is set up, its VKaart card (the TCB) is initialized by the SMS. This is done by physically inserting the TCB in the SMS (hence securing the channel between this TCB and the SMS). The SMS then loads all necessary keys into the TCB. After installing the VKaart card in the VKaart station, the TCB still can receive secured information from the SMS, because the SMS can distribute (encrypted) data to the VKaart station via a networked file server or even by floppy disks.

4.3.4. E-mail security.

The VKaart system can also be used to encrypt and digitally sign email. This is implemented by a utility that encrypts and signs the message which in turn will be attached to a conventional (i.e. non-encrypted) e-mail message using a normal e-mail client. A problem in this system is that the VKaart system only has access to the public keys that belong to its own VKaart domain. Secure e-mailing outside its own domain is therefore not possible. This is where the usage of a Public Key Infrastructure based on standards would benefit the VKaart system.

4.3.5. Secure communication with other VKaart stations

Every VKaart station can set up a secure communication channel to another VKaart station in the same VKaart domain. This cannot be done easily across different VKaart domains because the stations need to share each others public keys which are stored on the SMS'es of the VKaart domains. However cross-certifying other SMS'es is possible.

5. A MLS model using TTPs

5.1. Separate TTPs for each category or level.

5.1.1. Basic assumption.

The function of a TTP is to provide information that one can trust to a certain extent. For example a TTP gives you a key, stating that the only ones who have this key are you, another person we call Bob, and off course the TTP himself. This example is typical for systems that use secret keys for authentication, such as Kerberos. Another type of TTP is the PKI that is mentioned in section 2.3. The key word here is "trust": Who decides which keys belong to which persons and how well should that decision be trusted. Traditionally (i.e. in the non electronic life) these kind of decisions are often made by the end users. For example, a physician will not pass on his client's medical record to just anyone who asks for it, even not when such a person claims to be another physician. The physician will check whether the person requesting the record, indeed is a physician.

How does Alice know whether she should trust Bob? Traditionally this question would be solved by a personal meeting between Alice and Bob, or by requesting a reference from a (ore more) trustworthy person or organization that both Alice and Bob know. In the example of the physician such a person could be another physician, or perhaps the client, whose record we are talking about. A possible organization that could say something about the question whether Bob is a physician or not would be something like the association of physicians.

In general, automated systems should fit an already existing system or organization. Otherwise the automated system will not function in the organization and will be rejected. Especially in large organizations it will be difficult, if not impossible to impose a technical system on a organization, if that technical system does not fit the organization. As a consequence the technical solution to establish trust should be set up in a flexible way, i.e. in a way that it can fit in with the already existing way trust is established. Within small organizations such flexibility will not be as essential as in large organizations. This is because it is much easier to convince a small group of persons that a new solution will work as well, or even better than it is to convince a large group of persons, all the more because in larger organizations other matters may be of more concern than the possible benefits of the new system. Especially establishing trust between large organizations will be difficult, and cannot be subject to a single solution like establishing a global trusted third party. It is not without a reason that there does not exist such a global TTP.

5.1.2. Different categories or levels implies different TTPs.

The establishment of trust between physicians (possibly) involves other channels of trust than the establishment of trust between two persons from another group (e.g. between human resource managers). In order to make the technical solution with TTPs as much fitting to the existing situation, it will thus be necessary to have different TTP's for different categories. This may also count for the security levels, providing that for each level another way of establishing trust is needed.

When these principles will be applied to a architecture that provides trust, the architecture will contain different TTPs, each providing trusted services for a specific subset of categories or levels. This also means that the clients (i.e. the reference monitors) should know which TTP should be consulted for which level.

5.2. Constraints and assumptions

The Multi Level Secure architecture is distributed over several (computer-) systems. The systems are interconnected by an open network. However, the network does not provide any security service. Hence any message received from the network is not trusted without additional (proper) security services. For example, in order to trust that some message originated from system x, the message must be authenticated using some authentication service. The same goes for the confidentiality and the integrity of the message. The network does not provide any availability-service. Hence a denial of service attack will not be prevented. However the system must be designed in such a way that a denial-of-service attack may not result in a successful attack on the authentication-, integrity- or confidentiality services that are provided in this architecture. On the other hand it is also desirable that a denial of service will not always result immediately in the denial of service of some access. This issue is closely related to the validity periods.

The threat model for the MLS-architecture does not include a physical attack to insert a Trojan Horse on a user's system. However the threat model should address the theft of a system by means of a physical attack. The background of this is that a user's system may be stolen and that measures should be taken to prevent disclosure threats or integrity threats due to theft. However the chance that an attacker physically breaks in to the system without being noticed by the user will be considered so small that no extra measures need to be taken to prevent this.

The MLS-architecture has to provide the following services:

- authentication
- confidentiality
- integrity
- if possible, non-repudiation

The protection against the disclosure threats is considered to be more important than the protection against integrity threats. However, vulnerabilities for integrity threats should be minimized as much as possible without making the architecture too complex. The Access Control Matrix should be filled in according to the Bell Lapadula scheme, i.e. access may be restricted more than the BLP model does, but the access may not be relaxed more than in the BLP model.

5.3. Usage of personalized user TCBs

Human users are considered to be subjects that cannot authenticate themselves using cryptography. They may be able to authenticate themselves using other mechanisms such as password authentication, retina scans, fingerprints etc. Ultimately, the human user may authenticate himself using a personalized Trusted Computing Base that can perform some cryptographic calculations to authenticate itself (and under the assumption that no-one else but the human user has access to the TCB, also the human user will be authenticated). Therefore some (private) keys will be stored exclusively on this TCB. We call this TCB the user-TCB. In fact the human user authenticates himself to the user TCB and after that, this TCB will act on

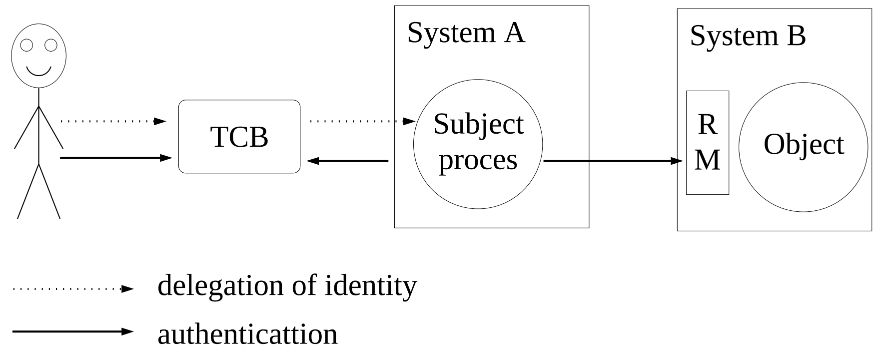

Figure 5.1.: Authentication and delegation of identity

behalf of the human user. We say that the human user delegates his identity to the user TCB. The user's system that should run the (subject) process on behalf of the user, must be authenticated to the user TCB as well. The human user, nor his TCB knows if the user's system can be trusted. Therefore it may be necessary to have the user TCB check the authenticity of the system. The user TCB delegates his identity to the subject-process after the check evaluated positive. Now the subject process can act on behalf of the human user and must authenticate himself to the Reference Monitor on the object's system in order to access the object. (see fig: 5.1).

This MLS-architecture will focus on the authentication of the user TCB to the system and specificly on the authentication of the system TCB to the Reference Monitor. The authentication of the human user to the TCB will be left aside. This does not imply that the authentication of the user to the TCB is trivial. In fact it is not, because it is part of the whole authentication chain, and a chain is as strong as its weakest link.

5.4. Authentication of the user's system to the user TCB

The subject process on the user's system will act on behalf of the user. Therefore his TCB must delegate its identity to the user's system. Before the user TCB will do that, it should verify the user's system's authenticity and integrity. The user's system needs its own certificate to authenticate itself to a user TCB. We call such a certificate a system-certificate. This type of certificate will be issued by a special TTP: The system TTP.

However, the system must store its private key in a secure way. Therefore the system has its own TCB, called the system TCB. The system TCB will also check the integrity of the system by signing the relevant files[1] on the file system. The system and its TCB must be designed carefully: It should be impossible to fool the system TCB with a corrupted system, otherwise it would be possible to masquerade a Trojan horse in the corrupted system, letting the TCB believe that the system is okay. Therefore the system TCB has to be involved in the booting process of the system. Off course such a system will be vulnerable to physical attacks, but our

[1] especially the Operating System's files and perhaps the system's boot files.

goal[2] here is to prevent attacks from outside (via the network) on the integrity and confidentiality of the system.

The task of the system TCB can be compared to the task of the VKaart-card in a VKaart system. The User TCB can be compared to the smart-card in a VKaart system. The user TCB will require the system TCB to authenticate itself before the user TCB will delegate the human user's identity to the system.

5.5. Separation of authentication and authorization.

The reference monitor must decide if some access-request to an object must be honored. Therefore the RM first must authenticate the subject requesting access. Once de identity of the subject is known, the RM must decide whether that subject may access the object. Hence in order to authorize a subject (i.e. give it access to some object), the object must be authenticated first. Authentication and authorization thus are complementary mechanisms. Recognizing this concept, it appears to be advisable to deal with authentication and authorization separately, as they involve different aspects and requirements, e.g. concerning the lifetime of some identity versus the lifetime of some authorization. This implies that authentication and authorization may possibly be implemented in different ways, with different techniques.

The X509 public key certificates are designed for authentication. They are designed for establishing trust in the binding of a key (pair) to some identity. This kind of information typically should be considered static. The keys belong to some principle and should remain to be so forever. However, a principle may lose his key, or his key might be stolen. For this reasons there should be a mechanism to end the trusted binding between the key and the identity of the principal. Another reason to limit the lifetime of the trusted binding is that the lifetime of a key depends on its usage, i.e. the more a particular key is used, the more vulnerable it will be to some kinds of attacks (for example, the known plaintext attack). Summarized, the lifetime of a public key certificate will be limited by the following criteria:

- the lifetime of the binded information (e.g. the identity),
- the lifetime of a key (depending on the strength of the key and how often it is used),
- the chance that a key gets compromised (lost or stolen).

The X509 attribute certificates are designed to link dynamic information to a (more static) X509 public key certificate. An attribute certificate typically contains information that has a short lifetime, this in contrast to the public key certificate. An attribute certificate therefore is very useful to contain authorization information. The nature of authorization information desires that it can be revoked on short-term basis. A certificate attribute therefore will have a short lifetime. (Otherwise the blacklist, or CRL would become unmanageable large).

Kerberos is a good example of seperating authentication from authorization: The KDC authenticates a principle and issues a authentication certificate (the ticket

[2] based on the threat model, see paragraph 5.2.

granting ticket). The principle uses this TGT for getting an authorization to a server, by requesting a server ticket from a Ticket Granting Service. The TGT can be compared to a X.509 public key certificate, making the KDC the CA. The server ticket can be compared to an attribute certificate, making the TGS an attribute authority.

It should be noted that the technique of using certificates implies that there exists a validity period (see section 4.1). For authorizations where correctness is more important than availability, offline mechanisms such as certificate mechanisms should not be used.

5.6. Protection of the object

There are two basic scenarios for where the object may reside: In a secure place or in an insecure place. A secure place will be a place where no unauthorized subject (e.g. a person) can (physically) access the object. We assume a secure place has an optimal[3] online secure connection with the maintainer of the Access Control Matrix. Complying with this assumption should be relatively easy, and therefore considered practical, because the number of secure places should be limited and maintainable, this in contrast to the number of insecure places. In a secure place the object may be stored un-encrypted on a disk. If the object is stored in a secure place it does not need to be encrypted before storage. There is a major advantage to such a scenario: There is no need for long-term encryption keys for storage. However, in order to access the object, additional measures should be taken to do that securely. In that case cryptography may still be a good solution, i.e. the object has to be encrypted only during its transport to the subject.

The insecure place can be accessed by a subject, e.g. because it is stored on the local hard-disk of the subject's PC. In this case encrypting the object before its storage should be used. If the subject needs to access the object, it must be decrypted first. A tamper resistant trusted computing base will be needed to do this in a secure way.

Table 5.1 lists three different places where the object may reside: In a secure place, at the objects place and somewhere else in a insecure place. These three scenarios have different consequences with respect to the security. Some major consequences are summed up in the table. Because the vulnerability to attacks is dependent on

place of the object	secure	on subjects system	elsewhere
long-term encryption-key needed	no	yes	yes
short-term encryption-key needed	yes	no	yes
denial of service threat possible	yes	no	yes
secure audit of access possible	yes	no	no
revocation can be done immediately	yes	if online	if online
tamper resistant TCB at object needed	no	yes	yes

Table 5.1.: Placing the object in a secure place vs. in an insecure place

the place where an object resides, the possible places where an object may reside

[3] Assuming a secure place stores objects that should be protected optimally, the RM should be optimally protected against the DOS-attack which makes the RM permit access when that access has recently be revoked.

may be different for each level. The best way to implement such a requirement would be to set up a policy that users should adhere to. This policy simply states which objects may be stored on which places, according to their security level. Table 5.2 shows an example of such a policy. It should be noted that such a policy

level/place	secured	user's system	file server in office	laptop
top-secret	yes	no	no	no
secret	yes	no	yes	yes
confidential	yes	yes	yes	yes
sensitive	no	yes	yes	yes
unclassified	no	yes	yes	yes

Table 5.2.: Allowed places for objects according to their level (example)

might be implemented through technical measures, forcing users to adhere to such a scheme. However, this will make such a system inflexible to exceptions, possibly moving users to get round such a system. Making the users aware of this policy and explaining why it exists will probably be much more effective than forcing the users to adhere to such a policy without explaining why the policy exists.

Another important aspect that is not related to security in this context, but is substantially influenced by the choice where an object will reside, is the maintainability of keeping the object consistent. This is because in some scenarios the object has to be replicated at different places. This problem can be mitigated by keeping the object at one or a limited number of places.

5.7. Supported authorization mechanisms

The Multi Level Secure architecture should support different authorization mechanisms because different mechanisms will have different impacts on how well some type of attack can be handled. Hence the level of security is influenced by the reference monitor's choice for some authorization mechanism [Hel97, page 26] (see also section 4.1), and the place where the object (and the reference monitor) will reside.

The choice for some mechanism implies also the choice of some validity period. In this context choosing another validity period should be seen as choosing another mechanism. For example, table 5.3 shows some validity periods per level.

level	validity period
top-secret	0 seconds
secret	1 hours
confidential	24 hours
sensitive	48 hours
unclassified	infinite

Table 5.3.: Validity periods for different levels

In principle the reference monitor will consult different TTP's for different security levels or categories (see section 5.1), i.e. it should be possible to use a unique

TTP for each security level or category, but it should also be possible to combine different security levels or categories into one TTP. Figure 5.2 shows a simplified infrastructure. In this infrastructure there are are different TTP's for different levels. They are connected to some (untrusted) network. The subject here will access objects on behalf of some user. It got the necessary credentials to authenticate itself as the user from the user TCB. In this scenario the objects can be on the same system, or elsewhere (on a server).

In case the object resides on the same system (as the left object in the figure does), it is considered to reside on a insecure system (we assume an attacker has physical access to this system). Therefore the object here will be encrypted, and the reference monitor on the insecure system will be built in a tamper resistant trusted computing base.

The other object in figure 5.2 resides on a different system, the server system. That system may be secure or insecure. Either way it will be guarded by the reference monitor on the server system. This reference monitor decides whether or not an access request from the subject should be honored. In case it honors the request, this reference monitor will setup a trusted path (i.e. authenticated confidential and integer) to the subject. This path will be used by the subject for accessing the object. In case the server system is not a secure system, the object also has to be encrypted. In that case the reference monitor of the server will also be built in a (tamper resistant) trusted computing base. This reference monitor will encrypt and decrypt the stored object on the server. So in such a case the reference monitor has to perform two cryptographic functions: Encrypting and decrypting the object for storage on the one hand, and setting up a trusted path to the subject (by using cryptography) to the subject. It is also possible to let the client's RM do the encryption and decryption. The server's RM can still be in control over the accesses by giving the session (encryption/decryption) key of the object to the clients RM if access should be allowed.

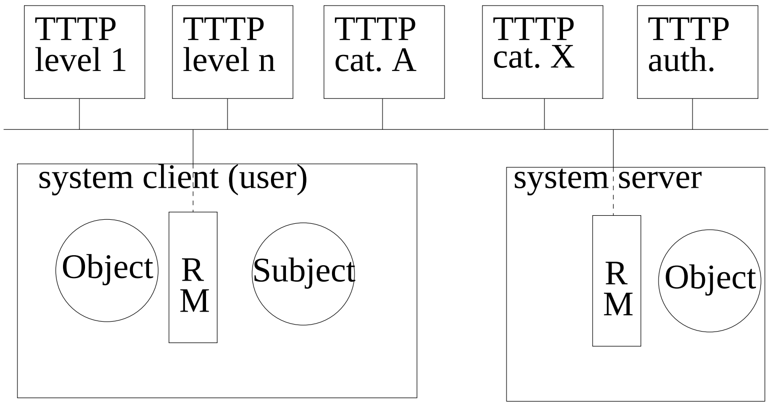

Figure 5.2.: Simplified Multi Level Secure architecture

These requirements will make the implementation of the reference monitor quite complex and extensive. With the current technology, this will make it difficult to implement such a reference monitor in a TCB, which is necessary in all scenarios where the object is not stored in a secure place. However, a solution where the objects only may be stored in secure places will have to many disadvantages, hence making this solution not realistic.

Figure 5.3 shows a MLS architecture. The figure shows at the left-side 4 TTPs which

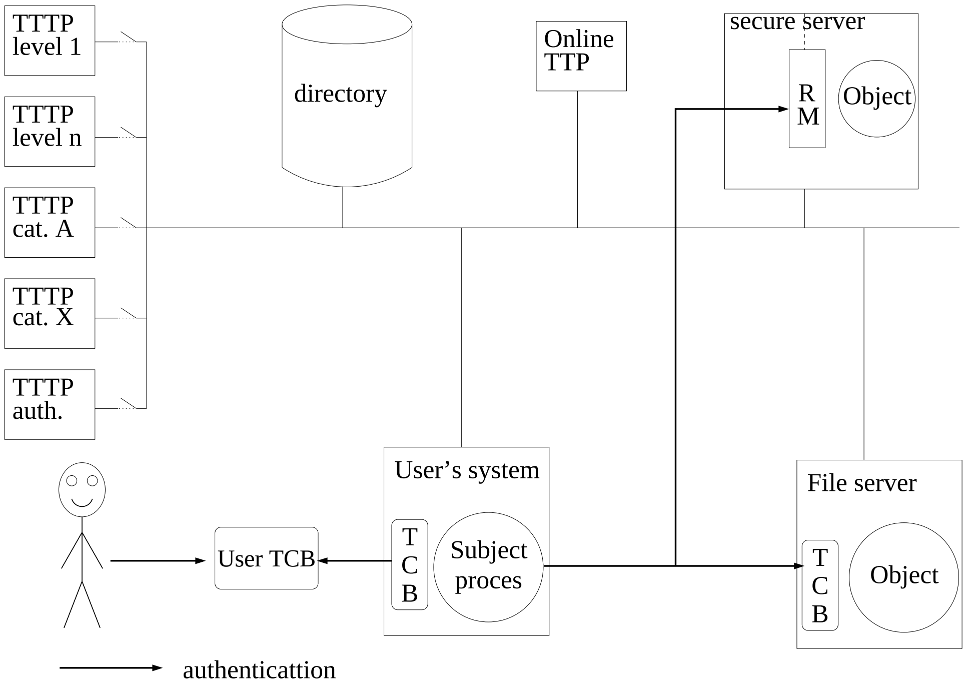

Figure 5.3.: Multi Level Secure architecture

are offline TTPs. They use the directory to store their certificates in. The directory of course is online. There is also an online TTP, that can be used for authorization mechanisms with a validity period of 0 seconds (for example for authorizations of top-secret accesses). There is also a secure server, i.e. a server that is in a secure place and stores objects, such as a file server in a secured room. There is also a normal file server that is not stored in a secure place. Hence it has a reference monitor incorporated in a tamper resistant TCB.

The user's system also has its own tamper resistant TCB. This TCB can be used to authenticate the system to other parties (such as the user TCB, the file server and the secure server). Besides, this TCB can protect the object that is stored on the local system and eventually, an object that is stored on a remote insecure file server.

The user has his own personal TCB that he can use to authenticate himself to the user's system and other parties (e.g. for non-repudiation purposes).

6. Conclusions

6.1. Implementation of the reference monitor

The place where objects should reside has to be chosen carefully. A great deal of this problem should be implemented by organizing the infrastructure. This will probably have more consequences on the organizational level than on a technical level.

The implementation of all the requirements on a TCB will be difficult and more research should be done into this topic. The current development of middle-ware for authorization may resolve some of these problems. For example IBM's "Policy Director" [DAS99a] may offer some solutions, also based on public key infrastructures [DAS99b]. Note that this kind of middle-ware performs functions, that OCSP is being developed for. However the documentation of these products did not mention OCSP as being one of the used protocols. In October 1999, a group of big computer corporations announced they started the Trusted Computing Platform Alliance [IBM99]. This is a joint effort to standardize the implementation of trusted computing bases. A lot of other companies have joint their effort.

Not implementing the requirements in this thesis can reduce the concept of multi level security to one aspect; the filling-in of the access control matrix conform some security policy like Bell-Lapadula. As long as it will indeed be impossible to implement all the requirements on a TCB, it might be advisable to reduce the levels of security a reference monitor in a TCB can handle. At the same time the reference monitor in a TCB should delegate a part of the access control decisions to some remote reference monitor that is placed in a secure place (e.g. using OCSP). There are serious implications to such a scenario, but it would require to much effort to work it out in this thesis.

6.2. Maintainability

An important factor to make a product successful, is how well it can be maintained. Two types of maintainability can be distinguished. On the one hand the product should be maintainable in that it should not take to much effort to extend the architecture with new functionality. On the other hand, operating the product should not require to much effort.

6.2.1. The architecture

When building the architecture, existing standards should be used as much as possible. It is also important to search for standards that are under development now. RSA and the IETF have developed some useful standards and are still developing them. It is also useful to keep up with the developments of The Trusted Computing Platform Alliance. A lot of applications should be enabled to work with this architecture. This cannot be done by one company, because the number of applications and machines involved is too large. Therefore following (and possibly participating in developing) standards will be a crucial factor for the success of the architecture. Obviously the designing of the architecture should be done very careful in order to make it sufficiently maintainable.

The user's system in the MLS architecture has a lot in common with the VKaart station. However, as stated before, the use of industry-wide standards in the development of such an architecture is essential. The VKaart architecture currently does not conform to the necessary standards. Still VKaart might be adjusted, so it will conform more to industry-wide standards.

Throughout this thesis authentication and authorization have been separated. In order to make a practical (i.e. working) system first and enhancing it later, it is possible to first make a system where authorization and authentication are not separated. I.e, in such a system a successful authentication automatically means the authenticated subject is authorized as well. This can be done using X.509v3 public key certificates only. However, prudent engineering will be necessary, in order to make it possible to enhance such a system to use more sophisticated authorization mechanisms.

6.2.2. Operating the architecture

The nice thing about this architecture is that it creates some separate functional roles in maintaining the architecture and providing trust. The TTP(s) for authentication are responsible for enabling ordinary users to authenticate themselves. Therefore the users should be registered at this TTP. The registration will involve the authentication of the user by conventional means, and then the issuing of a certificate with the users public keys. The user should store his private keys on his personal trusted computing base. Therefore it is necessary to use a TCB that complies with important standards (e.g. RSA's PKCS#11). Applications that are involved in authorization (e.g. the user's system or the system that stores an object) must be provided with the proper hardware (a TCB) and software. This should be part of normal system maintenance, and therefore it must be possible to perform some function (e.g. auditing, reconfiguring the reference monitor) from a distant point. The TTP's for authorization are responsible for enabling the right users to the right resources (i.e. maintaining their part of the access control matrix, or their access control lists).

When set-up in the proper way, these tasks will be executes independent of each other. However, it will require a lot of organizational effort to set up the right procedures and responsibilities. Besides, clear agreements on responsibilities, liabilities and tasks should be made in advance.

6.3. Multi Level Security

Multi level security has more than one dimension. It is not solely about filling in the access control matrix according to some model like the Bell-Lapadula model. It is also about the implementation of the access control. The MLS-architecture that is sketched in section 5 is mainly concerned with multi level security for confidentiality. If one strives to have multi level security for integrity as well, the architecture will become more complex. Besides that, the principals should have another label for their integrity next to their label for confidentiality. In short multi level security is about:

- the places where objects resides
- the TTPs that should be consulted

- the authorization mechanism to be used, and in particular
- the validity periods for these mechanisms
- filling in the access control list according to some policy (BLP, BIBA, etcetera)

6.4. Offline versus online

A distributed access control mechanism requires some form of communication between the different systems. Sometimes a user's system will be isolated from other systems for such a long period that problems arise concerning the validity periods (except for the infinite validity period of course).

[HTSK99] describes a nice solution that creates small certificates that can be transferred orally (e.g. by phone). The system can be useful in situations where there is no online connection to a TTP or CRL, but where it is necessary for the reference monitor to ascertain if a access request should be granted, based on a special certificate, specially made for such an occasion. It may be useful to integrate such a system in the MLS architecture to solve the problems that arise when a user's system is isolated for a long period.

The more confidentiality and integrity are considered to be vital, the shorter the validity period should be. This will ultimately result in a validity period that is near zero. In that case, checking an ACM will be done practically online. So in cases where confidentiality and integrity are very important, a PKI is not useful for implementing access control. An online mechanisms should then be used.

7. Appendix

Bibliography

[Amo94] E Amoroso. *Fundamentals of Computer Security Technology.* Prentice Hall, 1994.

[AS91] P. Galvin A. Silberschatz, J. Peterson. *Operating System Concepts.* Addison-Wesley, 1991.

[BL92] M. Burrows E. Wobber B. Lampson, M. Abadi. Authentication in Distributed Sytems: Theory and Practice. Research Report 83, Systems Research Center of Digital Equipment Corporation, Palo Alto, California, USA, february 1992.

[BP97] V. Rijmen B. Preneel, editor. *State of the Art in Applied Cryptography,* volume 1528 of *Lecture Notes in Computer Science.* Springer, june 1997.

[CE00] B. Schneier C. Ellison. Ten Risks of PKI: What You're not Being Told about Public Key Infrastructure. *Computer Security Journal,* 16(1), 2000.

[Cha99] D. Chappell. Exploring Kerberos, the Protocol for Distributed Security in Windows 2000. http://www.microsoft.com/MSJ/0899/kerberos/kerberos.htm, aug 1999.

[Cla87] Wilson Clark. A Comparison of Commercial and Military Computer Security Policies. In *Proceedings of the IEEE Symposium on Security and Privacy.* IEEE, IEEE Computer Society Press, Oakland, California, USA, 1987.

[CvP97] H. van Son C.L.M. van Pul, M.J. van Weert. Vkaart System, Operational Concept Description. Stg. Confidentieel, july 1997.

[DAS99a] IBM / DASCOM. Policy Director - Concepts and Architecture. requested by email to info@securit.be, october 1999.

[DAS99b] IBM / DASCOM. Policy Director and Public Key Infrastructures. requested by email to info@securit.be, december 1999.

[DB73] L. LaPadula D. Bell. Secure Computer Systems: Mathematical foundations, Vol I. Technical report, Mitre Corporation, 1973.

[ea97] A.J. Menezes et al. *Handbook of applied cryptography.* CRC Press, 1997.

[Hel97] A. Helme. *A System for Secure User-controlled Electronic Transactions.* PhD thesis, University of Twente, augus 1997.

[HTSK99] Arne Helme and Tage-Stabell-Kulø. Offline Delegation. In *Proceedings of the 8th USENIX Security Symposium.* The USENIX Association, Washington, D.C., August 1999.

[IBM99] Intel Microsoft Compaq IBM, HP. TCPA$_2 0 Pres.R1 - 5.pdf. www.trustedpc.org, october 1999.$

[JK93] C. Neuman J. Kohl. RFC1510 - The Kerberos Network Authentication Service (V5). Technical report, Internet Engineering Task Force, 1993.

[JS88] J.I. Schiller J.G. Steiner, C. Neumann. Kerberos: An authentication service for Open Network Sytems. http://www.mit.edu, march 1988.

[Ker83] A. Kerckhoffs. La Cryptographie Militaire. *Journal des Sciences militaires*, january 1983.

[MA94] R Needham M. Abadi. Prudent Engineering Practice for Cryptographic Protocols. Research Report 125, Systems Research Center of Digital Equipment Corporation, Palo Alto, California, USA, june 1994.

[MB96] J. Lacy M. Blaze, J. Feigenbaum. Decentralized Trust Management. In *Proc. IEEE Conference on Security and Privacy*. IEEE, IEEE, Oakland, May 1996.

[MN93] S Osborn M. Nyanchama. Role-Based Security: Pros, Cons & Some Research Directions. *Security Audit & Control Review*, 11(2):11–17, spring 1993.

[Mul89] S.J. Mullender, editor. *Distributed Systems*. ACM Press, 1989.

[Neu95] P.G. Neumann. *Computer related risks*. Addison-Wesley, 1995.

[RA] S.J. Bezuidenhoudt R.J. Anderson. On the Reliability of Electronic Payment Systems.

[RA96] M. Kuhn R.J. Anderson. Tamper Resistance - a Cautionary Note. *The second USENIX Workshop on Electronic Commerce Proceedings*, pages 1–11, November 1996.

[Roe] M. Roe. Cryptography and Evidence. Master's thesis, University of Cambridge. email from Arne Helme.

[sch96] B. schneier. *Applied Cryptography*. Wiley, 1996.

[(SE97] ETSI Technical Committee Security (SEC). Telecommunications Security; Trusted Third Parties (TTP); Requirements for TTP services. ETSI Guide v.1.1.2, European Telecommunications Standards Institute, 650 Route des Lucicoles - Sophia Antipolis, Valbonne - France, july 1997.

[Sud91] T.A. Sudkamp. *Languages and Machines*. Addison-Wesley, 1991.

[SvS99] J.H.S. Geldenhuys S.H. von Solms. Managing Multi Level Security in a Military Intranet Environment. *Computers & Security*, 18(3):257–270, 1999.

[vS98a] H. van Son. VKaart Security Management System, System/Subsystem Design Description. Stg. Confidentieel, january 1998.

[vS98b] H. van Son. VKaart Security Management System, System/Subsytem Specification. Stg. Confidentieel, january 1998.

[Wel97] I. Welch. *A Reflective Architecture for Securing Commercial-Off-the-Shelf Applications*. PhD thesis, University of Newcastle upon Tyne, Dept. of Computing Science, august 1997. Thesis Proposal, http://www.ncl.ac.uk/ n5830358/work/PhD/thesis-proposal.html.

[x5097] Recommendation X.509: Information Technology - Open Systems Interconnection - The Directory: Authentication Framework. Technical report, ITU-T, 1997.

[YZ93] J. Seberry Y. Zheng, T. Hardjono. New Solutions to the Problem of Access Control in a Hierarchy, 1993?

Index