A critical view on Public Key Infrastructures

Thomas Tjøstheim
Institute of Informatics
University of Bergen

26th February 2004

Acknowledgments

The completion of this thesis has required the help of many people.

Many Thanks to my two thesis supervisors Øyvind Ytrehus and Tor Helleseth. Thank you for giving me confidence in myself and helping me finish this thesis.

Many Thanks to my dad for helpful discussions, and for reading and commenting on my thesis.

Many Thanks to Siren Steen and the Music Library in Bergen for understanding, and giving me time to work on my thesis.

Many Thanks to Trond Rognebakke Bjørstad for helping me with technical challenges in LaTeX.

Many Thanks to Vebjørn Moen for helpful discussions about SkandiaBanken.

Many Thanks to Voss Fjellheiser and King Winter for helping me relax and giving me loads of snowboarding fun.

Contents

Chapter 1

Introduction

The explosive growth and availability of the Internet have been important factors in order to create a global online business market. People can now in the comfort of their own home or office, communicate and trade with other people independent of their geographic location. Issues like buying and selling of merchandise, transfer of documents and production management are just some examples of what can be done completely electronically. Although the Internet has introduced many new possibilities, it has also presented some new security challenges. Traditionally when dealing with any kind of sensitive information, people have met each other face to face, assuring a secure exchange of information. The trustworthiness of the other part could be evaluated by gathering references that described the behavior of that part in similar situations. The problem with the Internet is that it is a large and open network, where it is easy to remain anonymous. When engaging in online operations, several questions become central. How do the participating parties know who they are communicating with? How can one assure that no one is eavesdropping ? Is there any guarantee that the information sent between two parties is not intercepted and altered? What if one of the parties denies the occurrence of a transaction? On which basis can a party gain access to a resource, and how can one control which operations that are performed on a resource?

The answer to these questions lies in the following main objectives in information security [67]:

- *Authentication* is the assurance to one entity that another entity is who he, she, or it claims to be.

- *Confidentiality* is the assurance to an entity that no one can read a particular piece of data except the receiver(s) explicitly intended.

- *Integrity* is the assurance to an entity that data has not been altered (intentionally or unintentionally) during transfer or with time.

- *Non repudiation* is the assurance, to the extent technically possible, that entities remain honest about their actions. The most commonly discussed form of non repudiation is that neither the sender nor the receiver of a message is able to deny the transmission taking place. The basic idea behind non repudiation is that a user is cryptographically bound to a specific action in such a way that a denial of that action, to some extent, constitutes a confession of malice or negligence.

- *Privilege management* is a generic term for what is variously called authorization, access control, rights management, permissions management, capabilities management, and so on. Generally we can say that privilege management is associated with what an entity can see and do.

1.1 Motivation

A Public Key Infrastructure (PKI) is generally considered to provide three primary services: authentication, confidentiality and integrity. In addition a PKI can *enable* the services of: non repudiation and privilege management. Later in this chapter I will give a general PKI definition and describe some important considerations in a PKI.

As more businesses are going online there has been an increase in the market for PKIs. A PKI seems like an attractive way to solve security problems; it comes in one package and can provide all the security services you need. However, it is important to realize that a PKI is not some magic package that you add to your system to make it secure. Before installing any PKI, a careful analysis of the environment should be done in order to point out which security services are needed, and determine how to implement those services. A consequence of the increased focus on PKIs is that there currently is a lot of commercial interests in PKI technology. Some of the larger PKI vendors are: Verisign, Entrust, Baltimore Technologies, RSA security and Microsoft.

The goal of this thesis is to provide a critical and vendor neutral view on PKI solutions. The PKIs covered in this thesis are: PGP, SPKI/SDSI and

X.509. PGP (Pretty Good Privacy) is a free and open PKI with an interesting trust model. PGP has proved popular especially for the use of secure email. Some of the motivation for looking at PGP was to understand some of the shortcomings that prevents PGP from being used in larger environments. SPKI/SDSI (Simple Public Key Infrastructure/Simple Distributed Security Infrastructure) is a new standard that emphasizes authorization. Today more protected resources are being made available on the Internet. Therefore it has also become more important to have good mechanisms for administrating who can access those resources and also control which operations that are to be allowed performed on those resources. X.509 is the traditional PKI, which the majority of PKI vendors use as a basis for their implementations. The motivation for studying X.509 was partly because it is the most widely used PKI standard and partly to get a better understanding of the supposedly complex trust model and naming scheme. In addition I wanted to look at the use of X.509 *attribute* certificates for authorization, in order to compare with the SPKI/SDSI authorization model.

After getting a better understanding of important elements of PKIs, I wanted to focus specifically on storage of keys (Personal Security Environment, (PSE)) and *revocation* of certificates. The security of public key cryptography is dependent on keeping the private keys secret. The weakest link in a PKI is usually between the user and his key. In this thesis I take a critical look at available software and hardware solutions for storing cryptographic keys.

Revoking a certificate implies declaring a certificate invalid before its validity period has ended. Unfortunately revocation is often a neglected part of the PKI. The reason for this is that many revocation mechanisms are complex and expensive to operate. Part of the problem is also understanding the risks involved with not checking for revoked certificates. Many security applications like for example SSL (Secure Socket Layer) and TLS (Transport Layer Security), do not have automatic checks for revocation. In this thesis I take a critical look at some of the main revocation techniques. I also try to illustrate the potential dangers of not checking for revoked certificates, by looking at a known compromised certificate attack on named-server version of SSL/TLS.

After a critical study of theoretical PKI solutions, I wanted to look at a *real life* example of a company that uses PKI technology. SkandiaBanken was selected. This is a popular online bank in Norway. The purpose was to look at different aspects of the PKI, like certificate enrollment, user login, and understand specifically how authentication and authorization are implemented in SkandiaBanken's access control scheme. After discovering some weaknesses in the scheme, a brute force attack was studied. The attack illustrated among other things that digital certificates must be carefully adopted to the system in order to retrieve the intended security services.

1.2 Public key cryptography

Until the mid 1970s only one mechanism was publicly known for the secure exchange of a message over an insecure channel. In order to communicate securely the participants exchanged a secret key via a trusted secure communication channel. The exchange was often done by meeting in person or sending a trusted courier with the secret key. Crypto systems using this mechanism are classified as *symmetric key* or *conventional* crypto systems. The secret key is used both for encryption and decryption of a message.

In 1976 Diffie and Hellman discovered public key cryptography which revolutionized the history of cryptography [1]. The radical difference between public key cryptography and symmetric cryptography is that two related but different keys are used instead of one. The basic idea in a public key system is that each user has a key pair consisting of a public key and a private key. The public key is made publicly available and the private key is kept secret. Anyone can encrypt a message with the recipient's public key, but no one other than the intended recipient who knows the private key can decrypt the message. Security of public key crypto systems is based on the fact that it is believed to be computationally infeasible to find the decryption key, given only knowledge of the cryptographic algorithm and the encryption key [2]. However, it should be *easy* for the intended recipient to decrypt a message. A desired property for public key systems is a *trap-door one-way function*, which is easy to calculate in one direction and infeasible to calculate in the other direction unless certain additional information is known [10]. Figure 1.1 gives an explanation of encryption and decryption in a public key system.

A big advantage when public key systems first were introduced was that distribution of symmetric keys now was simplified. Without any prior communication, the secret key can be encrypted with the recipient's public key and sent over an insecure channel. Previously one would often use special couriers who delivered the keys before any communication could take place. Public key cryptography also reduced the total number of needed keys for a network of communicants. If symmetric encryption is used for a network with

[1] This is not entirely true as it was revealed in 1997 that individuals in the British agency GCHQ knew of some of the concepts of public key cryptography as early as the late 1960s. However, this was not known publicly.

[2] To this date it has not been mathematically proved that this is computationally infeasible, for any known scheme.

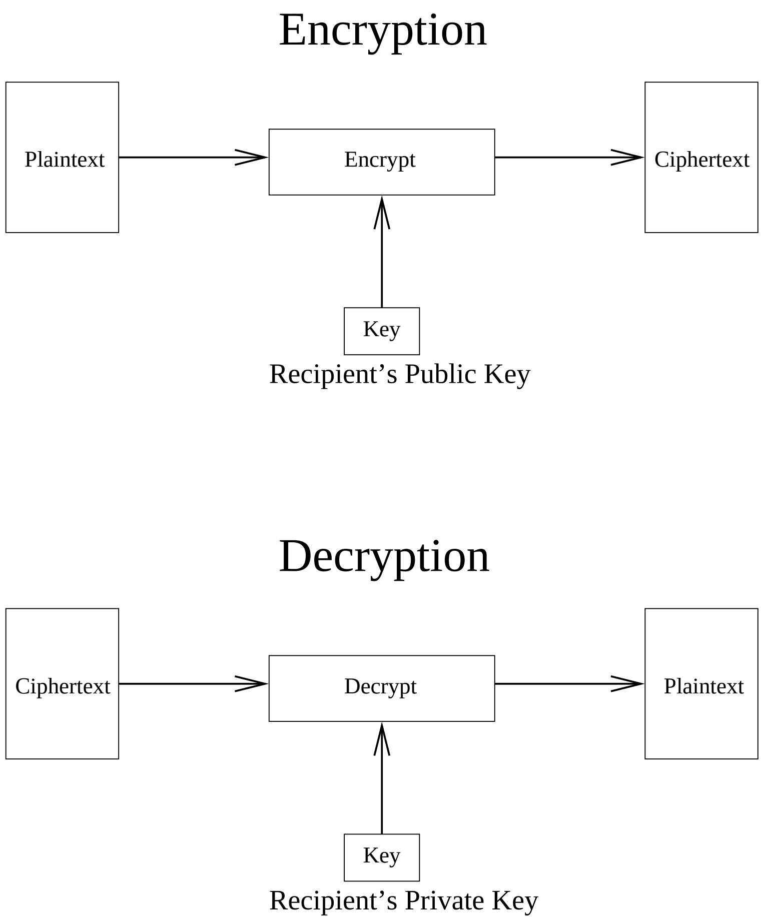


Figure 1.1: Public key cipher model

n people, each user needs one secret key for each of the n-1 other users in the network. This gives a total of $\frac{n \times (n-1)}{2}$ different keys for the whole network. In comparison, if a public key scheme is used, each user only needs a private and public key pair and access to the public keys of the other communicants. This gives a total of 2n different keys.

Another advantage with public key cryptography is the support for *digital signatures*. The concept of a digital signature is that a single entity can sign some data and any number of entities can read and verify its accuracy. A digital signature is more secure than a handwritten signature as it is believed to be computationally infeasible for another entity to forge another entity’s signature. A user Alice signs a message by encrypting the message with her private key. Assuming good key protection, only Alice has knowledge of her private key, so only Alice could have signed the message. Simplified Alice sends the signed message together with the message in plaintext. Bob verifies the signature by decrypting with Alice's public key and verifies that the decrypted message equals the message sent in plaintext.

The disadvantage of public key systems is that they have a much bigger computational overhead than symmetric systems. Symmetric systems have fast encryption and decryption algorithms since they are based on easily implementable substitution and permutation functions. Public key systems are also slower since they depend on bigger key sizes to provide good security.

Diffie and Hellman's proposal solved many of the earlier problems related to key distribution and authentication. However, it still remained to find a secure and efficient way of distributing public keys. When making public keys publicly available it is important to guard against forgery. That is, a user Eve could pretend to be user Bob and send a public key to another participant or broadcast such a public key. Until the forgery is discovered by Bob, Eve can decrypt all messages intended for Bob and can also authenticate herself as Bob by digitally signing documents with her private key. One of the first proposals by Diffie and Hellman was to register each user's public key in a central trusted *Public File*, maintained by a trusted authority. Each entry in the Public File consisted of a (name, public key) pair. The scheme suffered from performance problems in being a central directory. Also there was the vulnerability of someone tampering with the entries.

1.3 Digital certificates

In 1978 Loren Kohnfelder introduced the certificate, that was to simplify and secure the exchange of public keys. In order to secure the binding between names and public keys in the Public File, Kohnfelder made a new digital signed data record, containing a name and a public key. He called this data record a certificate [1]. His intention was to reduce some of the performance problems with the Public File. Since the certificate was digitally signed it could be given to untrusted parties.

Today we can classify certificates into three classes [69], shown in figure 1.2. The classification of a certificate is based on the binding being defined in the certificate. We identify three bindings: name public key (for example X.509 name certificates and PGP certificates), authorization name (for example X.509 attribute certificates) and authorization public key (for example SPKI/SDSI authorization certificates).

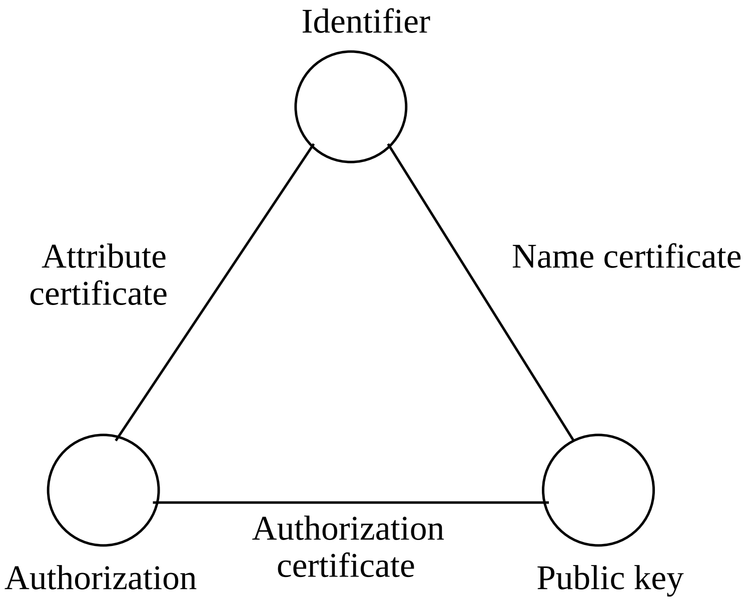


Figure 1.2: Certificate classes

1.4 Core PKI concepts

A Public Key Infrastructure (PKI) consists of the services that are needed to deploy and support technologies based on public key cryptography. In this section I look at how the services of a PKI are provided and discuss some general concepts and considerations in a PKI.

The core services of a PKI are: confidentiality, integrity and authentication. The services of authentication and integrity can both be achieved through the use of digital signatures. Suppose Bob signs a message with his private key and Alice successfully verifies the signature with Bob's public key. Assuming good key security, only Bob has knowledge of his private key, and thus only he could have signed the message. Let us assume that Eve intercepts the message and alters the contents. This would with very high probability lead to a change that would break the validity of the signature and therefore be discovered. The service of confidentiality can be provided through the use of public key encryption. However, typically a PKI also has support for symmetric ciphers for this purpose, where public key encryption is primarily used for the exchange of symmetric keys.

The services of non repudiation and privilege management can be enabled in a PKI since they build on the core PKI services. The primary purpose of non repudiation is to gather evidence attesting to the validity of an event that will be convincing to an impartial third party. Non repudiation is complex, as it is crucial to protect the evidence and difficult to decide how much evidence is needed to convince a third party. A central issue when presenting evidence is secure time stamping. If we assume a private key compromise, it must be possible to prove that this was the case for a specific time period. Also the validity of the compromise must be confirmed with existing revocation mechanisms. However, the whole concept of non repudiation is finally dependent on users protecting their private keys.

Privilege management deals with what an entity is allowed to see and do. Let us assume we have an authority guarding a resource. Further we assume that a number of entities have different privileges they can enforce on that resource, like read, write or execute. However, before admitting any privileges, in many situations it is necessary to establish that an entity has a particular identity or belongs to a specific group or role. Different PKIs offer different solutions in binding authentication and privilege management together. X.509 attribute certificates for example bind a name to an authorization. In order to authenticate the certificate holder, there is a link inside the attribute certificate to a name certificate. In the SPKI/SDSI infrastructure entities are primarily identified by their public keys. A SPKI/SDSI authorization certificate binds an authorization to a public key, thereby linking authorization and authentication directly.

The services of a PKI are based on public key cryptography. A *public key certificate* (authorization and name certificates) secures the public key of the certificate subject, and is therefore central in any PKI. In the following I state some general considerations regarding public key certificates.

- Who issues certificates? Depending on the PKI this could either be a trusted third party, another entity in the PKI or the certificate could be *self issued*. Common for all PKIs is that the certificate issuer signs the certificate and thereby claims the authenticity of the public key. Some general considerations concerning issuing of certificates are: How can multiple certificate issuers cooperate to cover the whole domain of larger PKI environments? On which basis is a certificate issued? Does there exist automatic methods for certificate reissuing? Are old keys stored, so that old material can be decrypted? How are the private keys delivered?

- How are certificates and corresponding private keys stored? What if the private key is lost? In order to verify signatures and encrypt information to other entities it must be easy to obtain the certificates of other entities. Some possible certificate storage devices are: X.500

directories, LDAP servers, Web servers, and corporate databases.

- How can it be determined that a certificate is valid? This includes among other things of verifying the signature on the certificate, checking that the certificate has not expired, that the certificate is being used in a manner that is consistent with current policies and constraints and that the certificate is on the correct format. Does revoked certificates pose any harm to the environment? If so, what is the best revocation mechanism for this environment?

- How do entities in the PKI refer to each other? In larger PKI environments it can be difficult to separate one entity from another. To protect against forgery there can only be a one-to-one relation between an entity and a public key.

1.5 Structure of thesis

This thesis consists of seven chapters. Each section in chapters two, three, four and five is followed by a critical notes section. The purpose of this was to state personal opinions about the schemes, and point out strengths and especially possible weaknesses. In some cases the criticism may seem obvious, but is nevertheless important to discuss. Below follows a brief description of each chapter:

- Chapter 2 describes the three PKIs: PGP, SPKI/SDSI and X.509.

- Chapter 3 describes four classes of revocation mechanisms. CRL (Certificate Revocation List) schemes, Authenticated Dictionary, OSCP (Online Status Certificate Protocol) and short lifetime certificates are covered.

- Chapter 4 describes a known compromised certificate attack on named server version of TLS/SSL.

- Chapter 5 looks at different software and hardware approaches to storing public and private keys. Conventional password protected file, personal entropy, software smart cards and hardware smart cards are covered.

- Chapter 6 looks at different aspect of the PKI in SkandiaBanken. Issues like certificate enrollment and how authentication and authorization are implemented is studied. A brute force attack is described by generating Norwegian Social Security Numbers (SSNs). A discussion of different attack methods and possible proprietary SkandiaBanken control mechanisms is made.

- Chapter 7 gives a short summary of this thesis and lists some possible further work.

Chapter 2

Three different PKIs

In this chapter I will look at the three PKIs: PGP, SPKI/SDSI and X.509. The main focus in each PKI will be on how users in the PKI are identified, structure of trust model and certificate revocation. For the SPKI/SDSI and X.509 infrastructures I will also cover authorization [1].

2.1 PGP

Pretty Good Privacy (PGP) [10] is a program originally created by Phil Zimmermann in 1991. PGP provides services for confidentiality and authentication and is widely used for electronic mail and file storage applications.

2.1.1 Private and public key rings

Each user in the PGP infrastructure has a private and public key ring. The private key ring holds the user's public keys and corresponding encrypted private keys. Several key pairs are used, to allow different keys when interacting with different groups of correspondents. Another reason is to increase security somewhat, by allowing users to periodically change key pairs.

Associated with each key pair entry is a *KeyID*, a *userID* and a time stamp.

- The KeyID is used for indexing public keys and consists of the 64 least significant bits of the public key. This will with very high probability be a unique identifier for a public key in the key ring. Suppose Bob

[1] X.509 authorization is strictly not part of the X.509 PKI as X.509 attribute certificates do not include a public key. However, the X.509 PKI is closely related, as authorization is dependent on authentication. The motive was also to compare with the SPKI/SDSI authorization model, where a specific authorization is linked directly to a public key

wants to exchange a symmetric key with Alice. Bob first encrypts the symmetric key with one of Alice's public keys, and sends with it the KeyID. Alice uses the KeyID to find the public key that was used for encryption and decrypts with the corresponding private key.

- The userID is typically the user's email, but other types of identification are also possible. Given that the userID is different for each key pair it can also be used to identify public keys in the user's private key ring.

- The time stamp indicates the date/time the key pair was generated.

A user's public key ring contains the public keys of all the other users this user communicates with. Typically this will be the public keys of people with whom the user exchanges email.

2.1.2 PGP trust model

PGP's trust model is commonly referred to as a *web of trust* model. PGP users sign each other's certificates, and each user progressively forms a web of users which he/she trusts communicating with. For example, Alice signs Bob's certificate which she believes is authentic. Suppose Carl wants to communicate privately with Bob. Bob forwards his certificate, and Carl verifies the signatures on Bob's certificate. Carl who trusts Alice as a signer, recognizes Alice's signature and will therefore also trust that Bob's public key is authentic. This is actually a bit simplified, as we will see later that Carl can assign various degrees of trust, and can also decide how many trusted signatures are needed to make a certificate completely valid. If Carl had not known or trusted any of Bob's signers, then Bob would have had to find other signers which Carl trusts to sign Bob's certificate.

PGP operates with two different versions of trust:

1. Trust in certificate, that describes how strong the binding between userID and public key is believed to be. Each user who signs a certificate, attests to the authenticity of the binding between userID and public key.

2. Signer trust, which describes how much a signer is trusted in certifying other certificates. As we shall see later the two different trust types are dependent on each other.

An entry in the public key ring consists of a PGP certificate (see Table 2.1) together with a *trust flag byte.* A trust flag byte consists of three different fields: a *signature trust* field, an *owner trust* field and a *key legitimacy* field.

PGP certificate	
Fields	**Explanations**
`PGP Public Key`	The public key for this entry
`UserID`	The owner of this key, typically given with the email address of the owner.
`KeyID`	The least significant 64 bits of this key.
`Time stamp`	The time of registering of the public key.
`Digital Signature`	The digital signature of one or more users that attests the authenticity of the binding between public key and user that appears in the certificate. Each certificate owner signs his/her own certificate, and other users may also sign the certificate. In general PGP certificates are always *self-signed* and may have more additional signatures.

Table 2.1: Overview of fields in PGP certificate

- The key legitimacy field indicates to which extent PGP will trust that this is a valid public key for the user. Three different values of trust are used: *undefined, marginal* and *complete.* The higher the value of trust, the stronger the binding between userID and public key is believed to be.

- The signature trust field indicates to which degree a PGP user trusts a signer in certifying public keys. Trust levels are: *full, marginal, untrust-*

worthy and *unknown*. Each users evaluation of the other signers is kept secret and therefore only exists within each individual user's public key ring [61]. The trust evaluations are kept secret in order to protect each PGP user's personal opinion about other people's trustworthiness, and also because different people will have potentially different personal opinions about who is a trustworthy signer and not.

- The owner trust field is assigned by the *key ring owner* and indicates how much the certificate holder is trusted to sign other keys.

Suppose Bob is a PGP user. For each time Bob inserts a new certificate in the public key ring, the following routine is performed:

1. PGP assigns a trust value associated with the owner of the public key. If Bob is the owner, the public key also appears in the private key ring and a value of *ultimate trust* is set. Otherwise Bob assigns a trust value that indicates how much he trusts the certificate holder in signing other certificates.

2. When the new certificate is entered, one or more signatures may be attached to it. For each signature, PGP searches Bob's public key ring, to see if the signer is among the known certificate holders. Given so, the *owner trust* field is assigned to the *signature trust* field. If not, an *unknown user* value is assigned.

3. The calculation of the key legitimacy field is done automatically by PGP and is based on entries in the signature trust field. Each user can set configurable parameters that control how many *complete* or *marginally* trusted signatures are needed to make a certificate completely valid. A certificate becomes valid if either one of these parameters are met. Otherwise if neither is met, but at least one type (marginal, complete) a value of *marginal* trust is assigned to that certificate.

Periodically PGP scans the public key ring for updates. This is done in a top-down process. For each owner trust field , PGP searches for all signatures made by that owner and updates the signature trust field. The key legitimacy fields are then also updated on the basis of the new signature trust fields.

2.1.3 Revocation of PGP certificates

PGP certificates can be revoked for several reasons. A certificate holder can for example suspect a private key compromise or he/she may have lost the password to unlock the private key. Another revocation reason is simply that a user may want to avoid the use of the same key over a longer period of time. In the PGP infrastructure, only the certificate holder can revoke his/her certificate. One exception to this is that the certificate holder can choose a designated revoker, that revokes the certificate. This obviously requires that the designated revoker is fully trusted, since the revoker will need access to the private key. When a certificate is revoked the revoker issues a *key revocation certificate* signed with the private key of the certificate holder. The key revocation certificate has the same syntax as a regular PGP certificate, besides it includes an indicator that explains that the purpose of this certificate is to revoke the public key. The revocation certificate is distributed on PGP certificate servers, warning all PGP users, and requesting an update of affected public key rings.

2.1.4 Critical notes

The PGP trust model works well for groups with a small amount of people in a closed environment, for example a small company or department. Under these circumstances it is easy to make decisions based on trusting other participants. Scaling the PGP trust model for larger communities is not easy, because Alice from company A and Bob from company B has no common point of trust.

A potential security hole in the PGP model is the use of multiple signatures to authenticate one particular key. The theoretical reason for having multiple signers is to add fault tolerance. In case one signer makes a bad judgment (signs a certificate that should not have been signed), then the lack of other signers will prevent the certificate from being verified. However, there is no guarantee that the key signers are completely independent. There is a risk that one user could use different private keys to sign the same certificate, or other signers could cooperate to force a certificate valid for another user.

PGP allows users to define parameters, that define the number of *complete* and *marginal* trusted signatures needed to make a certificate completely valid. One drawback with this scheme is that, these parameters are forced to be global. For instance two users who are trusted *marginal* as signers are forced to have the same trustworthiness. In reality one marginally trusted user might be twice as trusted as another marginally trusted users. One way of eliminating the ambiguity in PGP trust levels, could be to assign *trust points*. Each user can then define globally a *trust bound*, that defines a certificate as completely valid.

PGP does not allow any trust chains [61]. A user accepts a certificate if the number of complete or marginal trusted signers meets the requirements set by that user. There is no transitivity, in the sense that if user A trusts user B, who trusts user C, then A also trusts C. This obviously sets certain limitations but can also be an advantage, as each user is only dependent on his/her own trust decisions.

PGP works well for applications like secure email, and has a good trust model for assisting in forming trust opinions about newly encountered parties. However, its trust model becomes too weak for more complex applications like secure e-commerce. Also it is debatable how many people who really understand and use the PGP web of trust model.

2.2 SPKI/SDSI

The SPKI/SDSI infrastructure is the result of the merging of two research projects, namely SPKI and SDSI. The SPKI [4] (Simple Public Key Infrastructure) project focused on a simple and flexible authorization model. While the SDSI [5] (Simple Distributed Security Infrastructure) part of SPKI/SDSI focused more on a local name space architecture to create secure and scalable computer systems.

2.2.1 Naming

A user in the SPKI/SDSI infrastructure is primarily identified by an asymmetric public/private key pair. A public key (or its hash, if hashes are collision free) is therefore a unique identifier for a user. However, it can be difficult for users to refer to other users only by their public keys. The SPKI/SDSI infrastructure solves this problem by allowing each user to build up a *local name space*, consisting of *local names*. The main idea is that a local name replaces a public key as an identifier. This is accomplished by issuing a *name certificate* that defines the local name and binds it to a public key (the *subject* field in the certificate). A local name consists of the certificate issuers public key and an identifier (<public key><identifier>). The identifier acts as a nickname for the subject the local name has been defined for. The advantage of this method is that the certificate issuer can choose identifiers that are meaningful to him/her, that are easy to recognize and remember. At the same time local names are made global, since other users can identify a users local name space from the public key in the local name.

The SPKI/SDSI name certificate is a digitally signed document of the form $(Key, Name, Subject, Validity)_{IssuerSignature}$ consisting of a body of four components and a signature [62]. Table 2.2 explains the components of the name certificate.

SPKI/SDSI name certificate	
Fields	**Explanations**
`Issuer`	The public key of the certificate issuer
`Identifier`	Determines the local name that is being defined.
`Subject`	Determines the new meaning of the local name. The subject can be a public key, a local name in the issuers local name space or a local name in another users local name space.
`Validity Specification`	Time period during which the certificate is valid. The specification has the form (t1,t2), specifying that the certificate is valid from t1 to t2.

Table 2.2: Components in SPKI/SDSI name certificate

Local name spaces are linked when a user defines a certificate binding between a name in its local name space to a name in another users name space [6]. Figure 2.1 shows an example of name space linking.

1. $K_B\ Friend \rightarrow K_{Carol\_Smith}$
2. $K_A\ Carol\_Smith \rightarrow K_B\ Friend$

Figure 2.1: An example of linking local name spaces

In this figure Bob issues a certificate that defines the local name: $K_B\ Friend$ that replaces the public key of his friend: $Carol\_Smith.$ as an identifier. Alice links her name space with Bob's, by issuing a certificate that defines the local name $K_A\ Carol\_Smith$ that refers to the local name in Bob's name space. The advantage with this method is that if for example Carol_Smith changes her key pair, then the user who Alice refers to as Carol_Smith would also automatically change. This is because of the indirect binding between K_A $Carol\_Smith$ and K_{Carol_Smith}. Note that if $Carol\_Smith$ changed her key pair, then Bob would have to change his definition of $Friend$ and issue a new certificate, stating the new public key of $Carol\_Smith$.

A fundamental notion in the SPKI/SDSI infrastructure is the definition of *group names*. A group is characterized by a local name, defined by the *owner* of the group. The local name consists of the owner's public key followed by the group name, for example: K_B $agents$, stating the public key of the owner Bob (B), followed by the group name: agents. A group owner defines a group by issuing to each member, a name certificate that defines a link between the local name of the group and this member's key or name. It is also possible to bind other groups to a group by issuing a certificate that binds the local name of the group to the name of the group being added. One of the main purposes of defining groups is to achieve easier management of Access Control Lists (ACLs). Instead of having an entry for each group member, it suffices with one single entry for the entire group, thereby reducing the size and complexity of the ACL.

2.2.2 The SPKI/SDSI authorization model

A SPKI/SDSI *authorization certificate* binds a specific authorization to the certificate's subject. The specific authorization specified in the certificate could be the right to access a resource like a particular web site, or read a particular set of files, or login to a particular account. An authorization certificate is a digitally signed five tuple of the form $(Key, Subject, Tag, Delegation, Validity)_{Issuer\_Signature}$. Table 2.3 explains the different fields of the certificate.

A SPKI/SDSI ACL is a list of entries, where each entry has the following *required* fields: subject, tag and delegation bit. These fields are the same as those of a SPKI/SDSI authorization certificate. Optionally validity specifications and issuer fields can also be included. If the ACL is to be processed and stored in a secure environment (for example behind a firewall), it is not necessary to sign the ACL or include issuer fields.

How are the ACL and authorization certificates put together to form the SPKI/SDSI authorization model? To clarify it might be best to illustrate with an example. Let us assume that we have a user Alice, who wants to protect some object, for example a web site. Alice starts by issuing authorization certificates to users (public keys) that are to access the web site. She also specifies in each certificate's authorization tag exactly what kind of operation that are to be allowed on the particular web site, for example read, write and which links that can be furthered accessed. For some certificate

SPKI/SDSI authorization certificate	
Fields	**Explanations**
`Issuer`	The public key of the certificate issuer.
`Subject`	Refers to the user (or users), who are receiving the granted authorization. The subject could be either a public key, a local name or a group name.
`Tag`	Explicitly states the authorization being delegated in this certificate.
`Delegation`	A Boolean bit that is set to be true or false. If true, the subject of this certificate can delegate the specified authorization or subsets of it to other users.
`Validation`	Specifies the validity period of the certificate.

Table 2.3: Overview of fields in SPKI/SDSI authorization certificate

subjects she might choose to set the delegation bit in their certificates to be true, so that those users can delegate the given authorization or subsets of it to other users. After issuing the authorization certificates, Alice issues an ACL with the entries corresponding to the certificate subjects she has issued certificates to. In the following let us assume Bob requests access to Alice's web site.

1. Bob sends an initial request to access Alice's web site, and also specifies what kind of operations he wants to perform on Alice's web site.

2. Alice denies the request, since the request is neither authorized or authenticated. Instead Alice responds with a challenge : using this ACL protecting the web site and this authorization tag formed from the initial request you must prove to me that you have the right to access this resource. Alice will only approve the request given that Bob can provide *proofs* of authentication and authorization. Bob is authorized if he can present a valid certificate chain via the web site's ACL to himself. If Bob's public key is already listed directly on the ACL, it is

not necessary to supply a certificate chain. An important observation regarding authentication, is that it is not Bob that is authenticated, but Bob's public key that is authenticated.

3. Bob generates a chain of certificates, for example by using the *certificate chain discovery algorithm*, for details see [7]. The output of the algorithm is as follows: a certificate chain consisting of 0 or more signed certificates, that proves in this case that Bob is authorized to the operation at the time interval specified in the time stamp. The certificate chain is a chain of the form $(\mathrm{SELF}, S_1, T_1, D_1, V_1)$, $(\mathrm{SELF}, S_2, T_2, D_2, V_2)$, ..., $(I_{i-1}, S_i, T_i, D_i, V_i)$, ..., $(I, \text{Bob's public key}, T, D, V)$. Where each certificate $(I_{i-1}, S_i, T_i, D_i, V_i)$ indicates the issuer, subject, authorization tag, delegation bit and validity specifications. The leftmost or start of the certificate chain is the certificate of the ACL owner (where *SELF* indicates that it is a self-issued certificate). The chain ends in the certificate of the user requesting the authorization. Bob signs the tag given from Alice and sends this with the chain of certificates.

4. Alice approves Bob's second request if the public key that verifies the signature on the authorization tag is the same that the chain of certificates authorizes. Alice then checks that the certificate chain goes from herself to Bob. The combination of the signed tag and the certificate chain gives the intended proofs of authentication and authorization, and, if fulfilled, will grant Bob access.

In addition to the authorization scheme presented, SPKI/SDSI provides an additional fault tolerance mechanism in the ability to define *threshold subjects*. A threshold subject can be defined in an authorization certificate and states a requirement that *k out of n* keys must sign a request (similar to PGP's web of trust) in order to get that request approved. The intention is to increase protection. As an example, if we have a *2 out of 4* threshold scheme then any two of the four key holders specified by the guardian, must sign a request to get it approved. The fault tolerance is added since an attacker now must compromise two out of the specified keys in order to fool the guardian. An important assumption for the scheme to be efficient, is that the signers have to be independent. A guardian should verify that the different signatures belong to different users. The use of threshold subjects adds a bit to the complexity of the SPKI/SDSI authorization model, but might be useful for some applications.

2.2.3 SPKI/SDSI trust model

Each user in the SPKI/SDSI infrastructure is its own CA that can issue name certificates and authorization certificates. Name certificates are issued to build up a user's local name space. There is no need for any rules or statements concerning issuing of name certificates, since the issuer is the sole authority on the local name, certificate subject binding. No other users can question the quality of that binding. Users protecting a resource can issue authorization certificates and can set their own policies and regulations on the usage areas of the issued certificates.

The use of trust in the SPKI/SDSI infrastructure is transitive. This becomes apparent from the certificate chains formed by delegation. Let us assume a guardian Bob issues an authorization certificate to Alice, with the delegation bit set to be true. Bob now trusts Alice in delegating the given authorization or subsets of it to other users. Alice is also trusted in deciding the further *delegation depth* by setting the delegation bit to true or false. Let us assume Alice issues an authorization certificate (with the delegation bit set to true) to Carol, who again issues a certificate to Dave. Dave will be trusted to access Bob's resource, given that he can present a valid certificate chain that goes from Bob to Alice, to Carol and finally to himself.

2.2.4 Revocation of SPKI/SDSI certificates

The SPKI/SDSI infrastructure does not advocate for the need of any particular revocation mechanisms. The only validation check performed is to check that the certificate has not expired. The SPKI/SDSI certificate guarantee is: "This certificate is good until the expiration date. Period." [9]. The argument behind this guarantee is simply that a certificate issuer in the SPKI/SDSI infrastructure takes the consequences of the certificates he/she issues. A user either issues name certificates to define his/her name space or issues authorization certificates to protect access to some resource they are controlling. The exception here is for users that have been delegated some authority from a guardian, and have a responsibility on the behalf of the guardian to make trustworthy decisions. However, what if a user's private key is compromised? SPKI/SDSI arguments for using certificates with short lifetimes, thereby lessening the risk of compromised keys. In some situations this might not be a good enough protection against compromised keys. It is also resource and time consuming to have to reissue certificates at short time intervals. SPKI/SDSI advocates using a Suicide Bureau (SB) or Key Compromise Agent (KCA) [9] where users register their public keys. If a key is compromised a "suicide note" is published to the SB, who will broadcast it on the SB network. A user's SB can also issue a special "*certificate of health*" [9] indicating that this public key was registered with this bureau on this date, and since then no evidence has been received that the key has been lost or compromised. The certificate of health can then be sent with the original certificates of the user.

2.2.5 Critical notes

The SPKI/SDSI infrastructure is an infrastructure that focuses mainly on authorization compared to authentication. It uses a local name space architecture that is enforced through users defining their local name spaces by issuing name certificates. At the same time local names are made global, since other users can identify a user's local name space from the public key in the local name. The advantage with a local name space architecture is that each user can choose names to identify the other users that are easy to recognize and easy to remember for him/her. It is also easier to scale the SPKI/SDSI infrastructure since a local name only needs to be unique within a users local name space.

The SPKI/SDSI authorization model provides several advantages. Maybe one of the biggest advantages is that trust originates directly from the guardian of the resource instead of from a trusted third party. Intuitively this seems like a better solution compared to having the guardian trust a third party that the guardian might not have any control over. The guardian can state in detail what kind of operations an authorized user can perform on the protected resource, by specifying this information in the authorization certificate's tag. An advantage with this approach is that only one ACL can be used to protect the given resource, where each authorized user might have different specified operations they are allowed to perform on that resource.

The SPKI/SDSI infrastructure does not provide any mechanism for establishing the identity of users. Given a public key, who is the holder of that public key? In the example with a guardian protecting a resource, it is left to the guardian to determine which users shall be granted access. On which basis can a guardian determine which users (public keys) are to be granted any authority? Part of the problem can be solved, since a guardian can delegate to a more qualified user, the responsibility of determining which users should be given access . To prevent any illegitimate users in gaining access, it must either be difficult for illegitimate users to obtain legitimate key pairs, or the guardian must have a good way of determining which public keys (users) to trust. The latter might seem especially difficult for *larger* communities. It may seem as if the SPKI/SDSI infrastructure is best suited for closed environments like for example a company, where it is easier to control who is a legitimate user or not. But even in a closed environment there is a risk for an *inside man* to loan out his key pair to non users. One way of preventing this could be to try to lock the private key in hardware.

A possible disadvantage with the SPKI/SDSI infrastructure is that there currently is no acknowledged revocation scheme. Only the expiration date of a certificate is checked for. However, revoked certificates pose a bigger threat than obsolete certificates, because of the risk of a key compromise. The SPKI/SDSI infrastructure provides some additional protection against key compromises through the definition of threshold subjects. This unfortunately adds a lot of complexity to the scheme, and it is debatable if guardians will bother with this extra precaution.

Another possible weak point in the SPKI/SDSI infrastructure is that there is little control on delegation depth. In the current scheme, if a user receives some authority to delegate a specific authorization, it is up to that user to determine if the receiving user also should be able to delegate the given authorization to other users. The longer the chain of delegation becomes the more probable it is that a mischievous user is able to obtain authorization. A possible solution could be to use integer control instead of Boolean control, to increase control of delegation depth.

The SPKI/SDSI infrastructure has many advantages and is especially well suited for the use of access control. However, it is a quite new infrastructure (1998) and has not yet been implemented in many commercial applications, that are largely dominated by the X.509 infrastructure.

2.3 X.509

X.509 [11] is a recommendation for a framework for public key certificates and attribute certificates published by the ITU-T, a standardization sector within the ITU (International Telecommunication Union). X.509 is today the most widely used certificate standard in Internet applications, examples include: secure MIME encoding (S/MIME), IP security, Secure Socket Layer (SSL)/Transport Layer Security (TLS) and Secure Electronic Transfer (SET).

X.509 certificates have developed through three versions, where the first version was defined in 1988. The third version used today was defined in 1995. What primarily separates the third version from its ancestors is the denotation of certificate extensions. The main purpose of adding certificate extensions was to solve some of the problems of deploying versions 1 and 2 on a bigger scale. Each certificate extension can have different specified purposes like: describing policies, assuring the identity of the certificate subject or include authorization information to access some protected resource. In general we can describe an extension as a three tuple of the form (type, criticality, value). Where type indicates what kind of extension, the criticality field indicates if the extension is marked critical or non-critical (marked by the certificate issuer) and the value describes the explicit contents of the particular extension. Non critical extensions may be ignored by the certificate user, for example an extension for alternate name and name attributes. A critical extension should not be ignored by the certificate user, for example an extension describing that the certificate should only be used for authentication of the certificate owner. In addition to standard extensions that are predefined, it is also possible for a certificate issuer to define their own certificate extensions.

2.3.1 Naming scheme

The X.509 naming scheme is originally based on X.500, which was a global online distributed directory service for certificates. In order to uniquely identify a user, each user was assigned a *Distinguished Name* (DN). In the X.509 infrastructure a certificate issuer binds a DN to a public key by issuing a X.509 name certificate [2] (see table 2.4). Figure 2.2 shows an example of a X.509 certificate, generated with OpenSSL, an open source PKI toolkit [63]. In this example the DN of the certificate subject is the concatenation of the following Relative Distinguished Names (RDNs): "C=NO, ST=Hordaland, L=Bergen, O=Demo United, CN=Ola Nordmann, Email: Olan@ii.uib.no".

An alternative or supplement to the DN scheme is the possibility of identifying a certificate owner by adding an extension called *Subject Alternative Name* to the certificate. The extension indicates alternative name forms like for example email address and URIs [3]. According to [64] the alternative name form should be considered just as binding as the subject DN. Also if the DN is null, one or more alternative name forms must be present, and this extension must be marked critical.

2.3.2 Authorization in X.509

The X.509 infrastructure supports two methods for authorization. Authorization information can either be placed in an extension in a name certificate or in a separate *attribute* certificate.

[2] Also referred to in literature as X.509 identity certificate or X.509 public key certificate.

[3] Uniform Resource Identifier, a generic term for all types of names and addresses that refer to objects on the World Wide Web.

The placement of authorization information in name certificates is usually undesirable for two reasons [65]. Firstly, there is the possibility of difference in lifetimes for the validity of authorization information and the validity of public key and name binding. Authorization information typically has a short lifetime of only weeks or months, whereas the validity of public key name binding is usually valid for much longer. Therefore the total useful lifetime of the certificate is shortened. Secondly, the authority on name public key binding is often a different authority than the authority on authorization. The name certificate issuer therefore has to do some additional steps to obtain the authorization information from the other authority. In some cases this may not even be possible.

Realizing some of the shortcomings of name certificates for carrying authorization information, the U.S American National Standards Institute (ANSI) X9 committee developed an alternative approach, known as attribute certificates. An attribute certificate (see table 2.5) binds a name to a set of attributes. Attributes will typically be information that describes authorization information associated with the certificate holder, like: access privileges, group memberships, policy specifications and roles. A certificate issuer can either include standard attributes (see table 2.6) or define it's own specific attributes.

What is important to notice is that an attribute certificate binds attributes to a name and not a public key. The attribute certificate is believed to be tamperproof since it is signed by a trusted certificate issuer. However, there is no way for a guardian (the certificate verifier) to control that the user requesting access is the same user that is claimed in the attribute certificate. The solution presented in [65] is to include a link to a name certificate within the attribute certificate. An attribute certificate holder can be authenticated by signing the request with the private key corresponding to the public key in the name certificate.

There are two primary models for the exchange of attribute certificates: The *push* model and the *pull* model. In the push model, attribute certificates are pushed from the client to the server. In the pull model the server pulls the attribute certificates from a repository or an on line attribute certificate issuer. The push model has the advantage that performance is increased as no search burden is placed on the server. The pull model has the advantage that it can be implemented without changes to the client or to the client-server protocol. The choice of the push or pull model depends on the surrounding system requirements. The push model is especially suitable for inter-domain cases where the client's rights should be assigned within the client's *home* domain, whereas the pull model is suitable for inter-domain cases where the client's rights should be assigned within the server's domain.

How does a guardian validate a request to access some object? A typical verification sequence could be:

1. The guardian verifies the signed request from the user by applying the public key from the user's name certificate (public key certificate). The guardian also verifies that the name certificate is signed by a trusted CA and validates the signature from the CA.

2. The guardian verifies that the received attribute certificate is in accordance with the requesting user's name certificate. This could for example be accomplished by having a hash of the public key listed in the attribute certificate. The guardian checks that the attribute certificate is issued by a trusted attribute certificate issuer. An important note here is that Farrell and Housley [65] demand that this issuer is directly trusted either by configuration or otherwise. This means that currently there is no support for attribute certificate chains, and hence no support for further delegation of authority. The guardian also checks that the attribute certificate has not expired.

3. If the attribute certificate is verified, the guardian extracts the attributes within the attribute certificate, and the user is allowed to perform what has been specified in the attributes.

```
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 9 (0x9)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=NO, ST=Hordaland, L=Bergen, O=Demo CAs, CN=https://skjor.ii.uib.no:8443
        Validity
            Not Before: Dec 18 12:30:38 2002 GMT
            Not After : Dec 18 12:30:38 2003 GMT
        Subject: C=NO, ST=Hordaland, L=Bergen, O=Demo united,
        CN=Ola Nordmann, Email=olan@ii.uib.no
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:fa:3f:b6:3a:56:1c:ad:56:19:55:c5:a7:06:19:
                    d2:88:d1:58:ce:5a:f6:a4:bb:c1:13:aa:3b:f2:0d:
                    a1:a1:9c:37:32:0f:25:54:4d:a3:39:fd:23:21:bc:
                    e1:0f:bd:81:ac:90:9f:1a:47:aa:97:cb:72:d7:14:
                    70:84:2c:20:12:18:c7:c8:a7:31:95:35:80:58:7b:
                    ca:72:ba:3f:a6:2c:c8:60:b1:40:24:de:76:11:16:
                    de:1f:9d:03:1a:08:69:99:00:ee:44:df:f1:bd:00:
                    83:44:02:9c:04:0d:cf:85:09:8f:66:eb:2d:b6:4d:
                    31:9d:a0:a2:d4:9e:ac:f1:9b
                Exponent: 65537 (0x10001)
    Signature Algorithm: md5WithRSAEncryption
        5f:19:48:27:27:c3:a9:d9:f5:11:90:15:44:9b:55:d6:54:47:
        76:f1:9f:38:38:d0:72:ad:e1:84:ec:60:ad:2a:38:4f:f8:27:
        f3:0a:44:12:e6:0b:39:a7:00:8e:fe:da:d8:91:3f:5a:33:b8:
        2f:cf:9d:72:3a:b7:a9:35:c8:22:71:55:77:8b:3a:fa:1a:f2:
        4c:32:f7:f0:72:c6:d0:54:12:6c:02:94:ef:4f:21:54:8a:72:
        f6:56:a6:8a:b4:97:4b:cd:a5:08:3d:20:d4:c4:25:98:c8:97:
        55:a7:cc:e3:ce:3b:54:1f:5e:20:6e:4c:a7:10:9f:98:6f:6a:
        30:37}
```

Figure 2.2: Example of a X.509 certificate

X.509 name certificate	
Fields	**Explanations**
`Version`	Identifies the version of the certificate (either 1,2 or 3)
`Serial Number`	Unique identifier for the certificate relative to the certificate issuer.
`Issuer`	Distinguished Name (DN) of the CA that issued the certificate and must always be present
`Validity`	Indicates time period for which the certificate is valid. Field is composed of not valid before and not valid after times.
`Subject`	Indicates the DN of the certificate owner and must be non null unless an alternate name form is used.
`Subject Public Key Info`	This field contains the public key associated with the subject and must always be present.
`Issuer Unique ID`	Optional unique identifier of the certificate issuer, rarely used in implementation practice.
`Subject Unique ID`	Optional unique identifier of the certificate owner, also rarely used in implementation practice.
`Extensions`	Optional standards, includes for example: Authority Key Identifier, Subject Key Identifier, Key Usage, and many more.
`SignatureAlgorithm`	The algorithm used by the CA to sign the certificate
`SignatureValue`	Identifier for algorithm used in computing the digital signature on the certificate.

Table 2.4: Overview of fields in X.509 name certificate

X.509 attribute certificate	
Fields	**Explanations**
`Version`	Identifies the version of the certificate (either 1 or 2)
`holder`	Holder of the attribute certificate
`Issuer`	Distinguished Name (DN) of the CA that issued the certificate and must always be present
`SerialNumber`	Unique identifier for the certificate relative to the certificate issuer.
`attrCertValidityPeriod`	The validity period of this certificate
`attributes`	The payload, the attributes linked to the certificate holder
`Issuer Unique ID`	Optional unique identifier of the certificate issuer, rarely used in implementation practice.
`Extensions`	Optional standards, includes for example: Authority Key, Identifier, Subject Key Identifier, Key Usage, and many more.
`SignatureAlgorithm`	The algorithm used by the CA to sign the certificate
`SignatureValue`	Identifier for algorithm used in computing the digital signature on the certificate.

Table 2.5: Overview of fields in X.509 attribute certificate

Standard attributes types in X.509	
Attribute Type	**Explanations**
`Service authentication`	Provides information that can be presented by the verifier to be interpreted and authenticated by a separate application within the target system.
`Access identity`	Identifies the attribute certificate holder to the server/service
`Charging identity`	Identifies the certificate holder for charging purposes. Holders company can for example be the charging identity.
`Group`	Presents information about holder's group memberships
`Role`	Holds information about roles designated for the certificate holder
`Clearance`	Clearance information about the certificate holder (associated with security labeling.)

Table 2.6: Explanations of standard attribute types in X.509

2.3.3 X.509 trust model

In this subsection I will only focus on the trust model for the X.509 name certificate as there currently is no established trust model for the use of attribute certificates.

A certificate issuer in the X.509 infrastructure is commonly known as a Certificate Authority (CA). The CA issues certificates according to a self defined Certificate Practice Statement (CPS) [66]. A CPS is a detailed description of practices followed by the CA in issuing and managing of certificates. Within the CPS is the Certificate Policy (CP) that essentially describes the intended use of the certificate.

What characterizes the X.509 trust model is that CAs are organized into a hierarchy. How these CA hierarchies are built up will depend on the surrounding environment of the PKI. Common for all hierarchic architectures is that there are one or several *root CAs*. A root CA is the starting point of trust, which all entities (intermediate CAs and users) holds as a *trust anchor*.

I will look at three ways of organizing CAs [67]: *strict hierarchy of CAs*, *distributed model* and the web model that is currently used on the World Wide Web today.

1. In the strict hierarchy of CAs there is only one root CA, followed by 0 or more levels of intermediate CAs. Each CA issues certificates to its children entities, whether it be intermediate CAs or end entities (users). All entities in the domain have a common point of trust in the root CA. In order for communication and certificate processing to work, each entity in the hierarchy must be supplied with the root CA's public key. How is secure communication between two users enabled in this trust model? Let us suppose Bob's certificate is signed by CA_2, whose certificate is signed by CA_1, whose certificate is signed by the root CA. The certificate processing consists of first constructing the certificate path from Bob's certificate to the root CA and then validating the path. Alice will only trust Bob if the following sequence of verifications holds: Alice verifies the CA_1 certificate with the root key. Then Alice verifies the CA_2 certificate with the public key of CA_1. Finally she verifies Bob's certificate with the public key of CA_2.

2. In the distributed trust model, trust is distributed between two or more root CAs. In practice this configuration consists of several linked strict hierarchies, as each root CA may have 0 or more subordinate CAs. Figure 2.3 shows an example of a CA hierarchy with multiple root CAs and intermediate CAs and the users shown as the last line of boxes in

the figure. The different root CAs can be interconnected by means of *cross certification*, shown as dotted lines in figure 2.3.

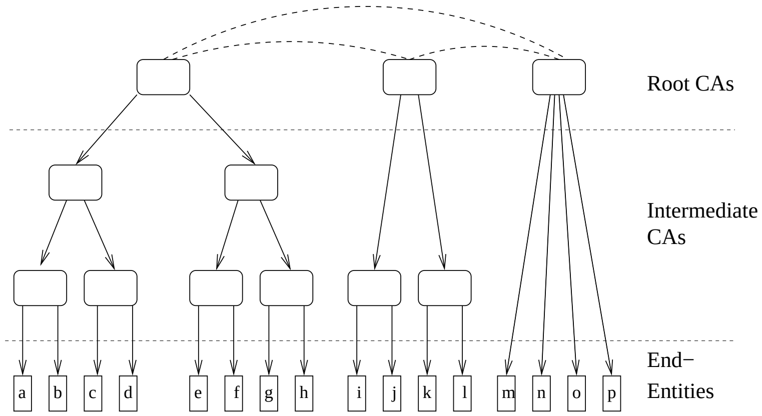


Figure 2.3: Example of a CA hierarchy with multiple CA roots

Let us assume Alice is part of the domain belonging to root CA_1 and Bob is part of the domain belonging to root CA_2. Initially Alice might only trust entities which have been signed by CA_1, and Bob only trusts entities which have been signed by CA_2. In order for Bob and Alice to establish communication CA_1 and CA_2 can cross certify each other by letting CA_1 sign CA_2's certificate and vice versa. Now Alice can verify CA_2's certificate with her trusted copy of CA_1's public key, and then verify Bob's certificate, using her now-trusted copy of CA_2's public key.

3. In the web model a number of root CA certificates are pre installed into the browser. Examples of CA vendors that are default in Netscape and Explorer browsers are: Verisign, Entrust Technologies, Thawte Certification Division and Baltimore Technologies to name a few. A user can also modify the initial set of certificates, by adding or deleting certificates. The certificates hard coded into the browser determines which public keys the browser user will initially trust as CA roots for verification.

   The structure of the web model is somewhere in between the strict hierarchy model and the distributed trust model. Like the distributed trust model, the web model consists of two or more strict hierarchies. Typically the web model will have many CA roots as browsers from Explorer and Netscape have up to a hundred pre installed root CAs. What separates the web model from the distributed trust model is that root CAs are not connected by means of cross certification. In order for a browser user Bob to communicate with users from other domains, he is made a relying party of all domains represented in the browser. In this way Bob can communicate with all other users that are part of the same domains that he is. Figure 2.4 shows an example of the web model.

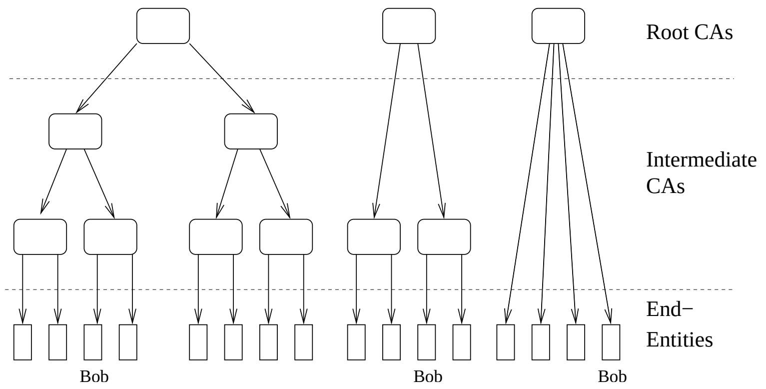


Figure 2.4: Example of a web model hierarchy

2.3.4 Revocation of X.509 certificates

X.509 supports a number of different revocation techniques that I will cover in chapter 3.

2.3.5 Critical notes

The X.509 certificate standard has been the dominating certificate standard in Internet applications till now. The advantage with the X.509 standard is that it has a general and flexible format. Part of the reason that X.509 has become so widely distributed is that it was available as an international standard at the time when a number of vendors were ready to begin implementing products. In being the most used certificate standard, X.509 has also received a lot of criticism. What has especially been criticized is the uniqueness of DN names, trust issues, and general complexity in certificate processing.

How unique are DNs? A CA verifies that the DN of the certificate subject is unique within the CA's domain. This means that a user can have different DNs in different domains or the same DN in different domains. So we do not always have a one-to-one relation between DN and certificate holder. Also there is the possibility of two users in different domains having the same DN.

This can particularly be a problem when different CA domains are connected through cross certification.

Another problem with the current DN scheme is that of making unique DNs within larger CA domains. The problem has been attempted solved by adding more personal attributes, like employee number, Social Security Number and email. This practice is unfortunate as it can lead to privacy concerns, as the DN of the certificate subject is often stated in plain text inside the certificate. Another drawback with more detailed DNs is that it becomes harder for a user to recognize other users in the same domain.

We see that there is a problem with using DNs as globally unique identifiers. In practice the certificate extension: subject alternative name is often used either with a DN or as a separate identifier.

The use of attribute certificates for authorization is not a mature field, and there are few implementations that support X.509 attribute certificates. One reason that attribute certificates have not yet been put into wide use is that there currently is no established infrastructure around attribute certificates. For example there is no support for chaining of attribute certificates, since an attribute certificate issuer must be directly trusted. In the X.509 authorization model trust *flows* from the certificate issuer instead of from the guardian of the resource. The advantage of this approach is that it may be easier for the trusted third party to issue and manage certificates. However, the result is lack of control, for example the guardian might have little control on which basis a user has issued an attribute certificate.

What is actually meant with the term trust in the X.509 infrastructure? The ITU-T Recommendation X.509 specification, defines trust as: "generally, an entity can be said to *trust* a second entity when it (the first entity) makes the assumption that the second entity will behave exactly as the first entity expects" [11]. The problem with this statement is that it is hard for users to quantitatively measure expected behavior in another entity. So how can users decide if a CA is to be trusted or not? CAs are often *self issued* (root CAs) or depend on self issued CAs (intermediate CAs). A possible help for users is the Global Trust Register [68], that distributes a list over the most commonly used root CA public keys. The list is distributed in a secure non electronic way. However, for the web model, the browser vendor defines for the browser user which CAs to trust. Even though it is possible for users to modify the pre installed list, it is debatable if most users are educated enough to do this.

The CPS describes some of the legal responsibilities of the CA, in the case of for example a certificate compromise. This can be assuring for users, the problem however is that a CA defines its own CPS. Unfortunately CPSs have a tendency to be long documents explaining why it is not the fault of the CA if something goes wrong.

2.4 A comparison of the three PKIs

Table 2.7 summarizes and describes some of the differences and similarities in the SPKI/SDSI, PGP and X.509 infrastructures.

Infrastructure	**Characteristics**	**Specification**
SPKI/SDSI	Trust model	Trust originates from the guardian of the resource. In order to access a resource a user has to provide a chain of certificates from the guardian to this specific user's public key. A user in the chain delegates its authorization to the next user in the chain.
	Name Space	Local
	Revocation	None. Advocates using a Suicide Bureau and issuing special *certificates of health*.
	Certificate Issuer	Each user can issue certificates. No CA structure.
	Certificate Types	Name certificate, binds a name to a public key. Authorization certificate, binds an authorization tag to a public key.
	Signatures	Each certificate contains one signature and belongs to the issuer of the certificate.
PGP	Trust model	*Web of Trust* = "multiple path of certification, to achieve fault tolerance in compensation for the fact that amateur certifiers are signing certificates "[17].
	Name Space	Global
	Revocation	Certificate owner issues a *key revocation certificate*
	Certificate Issuer	Each user can issue certificates.
	Certificate Types	Name certificate, binds a public key to a *userID* and *keyID*.
	Signatures	Each certificate can have multiple signatures. The first signature belongs to the issuer of the certificate.
X.509	Trust model	Hierarchical
	Name Space	Global
	Revocation	Supports different revocation mechanisms like for example CRLs and OSCP.
	Certificate Issuer	Only a CA can issue a certificate. The CA's are organized in a hierarchy of trust.
	Certificate Types	Name Certificate, binds a public key to a name. Attribute certificate, binds a name to a set of attributes.
	Signatures	Can have multiple signatures in the case of cross certification.

Table 2.7: Comparison of SPKI/SDSI, PGP and X.509

Chapter 3

Revocation mechanisms

Revoking a certificate implies declaring a certificate invalid before its validity period has ended. There are many reasons for revoking a certificate, for example suspected key compromise, detected key compromise, change of subject name and change of relationship between a subject and an authority [18]. In the following subsections I will not concentrate on why a certificate is revoked, but look at some of the main mechanisms for certificate revocation.

3.1 CRL

Certificate Revocation List (CRL) [18] is the traditional method for revocation of certificates. A CRL is a time stamped, digitally signed list of certificate serial numbers belonging to revoked certificates inside a CA domain. Figure 3.1 shows the ASN.1 syntax of a CRL [24].

```
CertificateRevocationList ::= SIGNED SEQUENCE{
    signature        AlgorithmIdentifier,
    issuer           Name,
    lastUpdate       UTCTime,
    nextUpdate       UTCTime,
    revokedCertificates
              SEQUENCE OF CRLEntry OPTIONAL}

CRLEntry ::=SEQUENCE{
    userCertificate SerialNumber,
    revocationDate UTCTime}
```

Figure 3.1: ASN.1 syntax for CRL

A CRL can be thought of as an equivalent to a *blacklist* of credit cards not to be accepted. A CRL is issued periodically by the CA covering the certificate domain. A CRL has a validity period from the time it was issued to the time for the next expected update, specified in the *nextUpdate* field. A new CRL is either issued at the expected update time or before if a specific event occurs, like the revocation of a CA certificate. It is important to notice that at any time only one CRL for a domain is valid [25]. The CRL scheme is part of the X.509 standard which also supports some of the CRL extensions. One of the main critiques against CRLs have been that they do not provide the intended service and are too costly to implement [9, 21]. Even so, CRLs are among the most widely used revocation mechanisms, primarily due to the different extensions and modifications of the basic CRL scheme that can provide better performance. There is a number of modifications and extensions available either as standards or only proposals. I will therefore only cover a few of the *well-known* standards.

3.1.1 Delta CRL

Delta CRLs are X.509 CRL extensions supported by X.509 version 2 and up. Delta CRLs provide a way for limiting the number of reissues of a full size CRL. A delta CRL is a digitally signed list of only the last changed entries since the full size CRL was issued. This is a big advantage since Delta CRLs are generally much smaller that the full size posting of a CRL, and therefore also can be updated and reissued at a higher rate with less cost.

3.1.2 CRL distribution points

CRL distribution points is another X.509 CRL extension, and is supported by X.509 version 3. The CRL distribution point scheme is another scheme that tries to improve performance by addressing the size of a CRL. This is done by dividing the total population of a CRL for one CA into a number of *segments*, where each segment is associated with a CRL distribution point. Each certificate has a pointer to its CRL distribution point so that the verifier can effectively check revocation information for this certificate. The main advantage with this scheme is that it is more scalable, since the segments can be distributed on different hosts/directories and each verifier will with high probability request less than all segments.

3.1.3 Dynamic CRL distribution points

One problem with the CRL distribution points scheme is that a subject is assigned to one segment for the whole lifetime of that subject's certificate. The issuing CA has to define a static partitioning of the subjects in the domain before issuing a certificate.

Two CRL extensions are defined in [22], suggesting a method of dynamic partitioning of segments. A CRL *Scope Field* contains a *scope statement* that indicates a range of certificates that are covered by the associated CRL partition. The CRL *Status Referal Field* associates scope statements with CRL distribution points.

A certificate verifier can from the CRL distribution point referred in the certificate, obtain a CRL with a Status Referal extension. This extension might include the current certificate and a pointer to a new location for the CRL.

3.1.4 Critical notes

The CRL scheme has some obvious weak points. In practice CRLs do not fulfill the requirement of recency, that is to provide freshly updated information for applications that depend on this. There will always be a delay between a revocation event and the reissuing of a CRL, whether it be seconds, minutes, hours or days. The more often a CRL is issued the better security against use of revoked certificates, but at the same time performance in terms of bandwidth use and processing delay will increase. One way to decrease traffic is to allow clients to *cache* a CRL. However, this will not work well in practice. Either the CRL has too long validity time and serves little or no use, or clients will have to download a new CRL at rapid time intervals in order to get good security.

Another point of interest is when a new CRL is issued. As mentioned before there can only be one valid CRL for a domain at any time. This means that all the clients will have to fetch a new CRL at the same time, creating a *peak load* that can easily lead to a break down. There is also a vulnerability for a Denial of Service (DoS) attack at this stage.

Other considerations are the length of a CRL. If a CA is covering a large domain of clients, the size of the CRL can become very large and consume even more resources. This however I think is not one of the main problems with the scheme, and can be solved by applying the many modifications and extensions to the *basic* CRL scheme.

The CA is responsible for maintaining the CRL, but what kind of guarantee is there for the CA's own certificate? In [18] the use of Authority Revocation Lists (ARLs) are mentioned. It is a CRL that is only used for managing revoked CA certificates. The ARLs are issued by *CA-certifying* CAs. This is one way for clients to check the status of the CA certificate in their domain. Unfortunately it adds to the complexity of an already complex revocation mechanism, and there is given no guarantee for the CA-certifying CA. In practice a compromised CA private key or certificate is very rare, and if it happens the CA vendor will probably make sure every client finds out through newsgroups and mailing lists. Sun did this when their CA key was compromised [27].

3.2 Authenticated Dictionary

An Authenticated Dictionary (AD) is a data structure that supports authenticated membership queries and update operations [31]. A typical implementation of an AD is based on the use of a hash tree, where the leaf nodes represent certificate serial numbers and the root is signed by the CA. Currently only Certificate Revocation Tree (CRT) implementations of ADs are being used on a larger scale, and I will therefore only focus on this implementation of ADs.

The Certificate Revocation Tree scheme was introduced by Kocher [29] in 1998 and is based on the use of Merkle hash trees [28]. A CRT system is a three party scheme consisting of a Certificate Authority (CA), directories and end users (certificate verifiers). The CA collects the revocation information from for example a CRL and builds the CRT. All the certificates for a domain are divided into ranges by sorting the certificate serial numbers. Let C_i be a certificate with serial number i. A range is defined by $<$a,b$>$ where a and b are the serial numbers of two revoked certificates and where all certificates with serial number m in the range a$<$m$<$b are not revoked. The ranges are compiled into data structures L_0, L_1, ..., L_{N-2}, L_{N-1}, where N is the number of revoked certificates. In addition the reason for revocation, date of revocation and other related information can be packed in the data structure. The leaf nodes of the hash tree are constructed by using the hash function H:$N_{0,i}$ = H(L_i), where $N_{0,i}$ indicates the hash value of node i at level 0 in the tree. A node at the next level of the Merkle hash tree is computed by concatenating the hash value of its two children nodes, for example $N_{1,0}$ = H($N_{0,0}||N_{0,1}$). The CA computes all the values up to the tree root: $N_{r,0}$, where r is the height of the tree (r=$log_2(N)$, if the tree is complete). The root of the tree and other information like expiration date of the tree is then signed.

The digitally signed root and the tree is then distributed to directories, which basically are sets of servers that makes the data available to all end users. The directories do not have to be trusted since the root is digitally signed by the tree issuer, and an end user who obtains the data, only has to check that the data from the directory is not defect or expired.

To understand how the CRT scheme really functions it is best to look at an example. Let us assume that we have 7 revoked certificates with the following serial numbers: 5, 8, 9, 13, 17, 23 and 42. The CA first divides all certificates from one domain into ranges defined by the given revoked certificates, and creates the data structures L_0, ..., L_7, shown in figure 3.2. The leaf nodes of the tree are calculated by applying the hash function H to the data structures L_0, ..., L_7, the rest of the tree is determined from these leaf nodes. Let us assume that an end user Bob wants to verify a chain of certificates. For each certificate he sends the serial number and the name of the certificate issuer to the nearest directory, since a CRT can include certificates from several CAs. The directory responds with the Merkle tree leaf for each certificate, the smallest number of intermediate nodes needed for Bob to calculate the tree root and the the digitally signed root. Let us assume that one of the certificates Bob requested revocation status for, has serial number 59. For this serial number the directory will then respond with the values L_7, $N_{0,6}$, $N_{1,2}$ and $N_{2,0}$ (shaded in gray in figure 3.2), which are the values needed for Bob to calculate the root node: $N_{3,0}$. Bob verifies that his calculated root equals the submitted root from the directory. In addition Bob should also verify the signature of the root, check that the leaf node describes the right certificate, and check that the CRT has not expired.

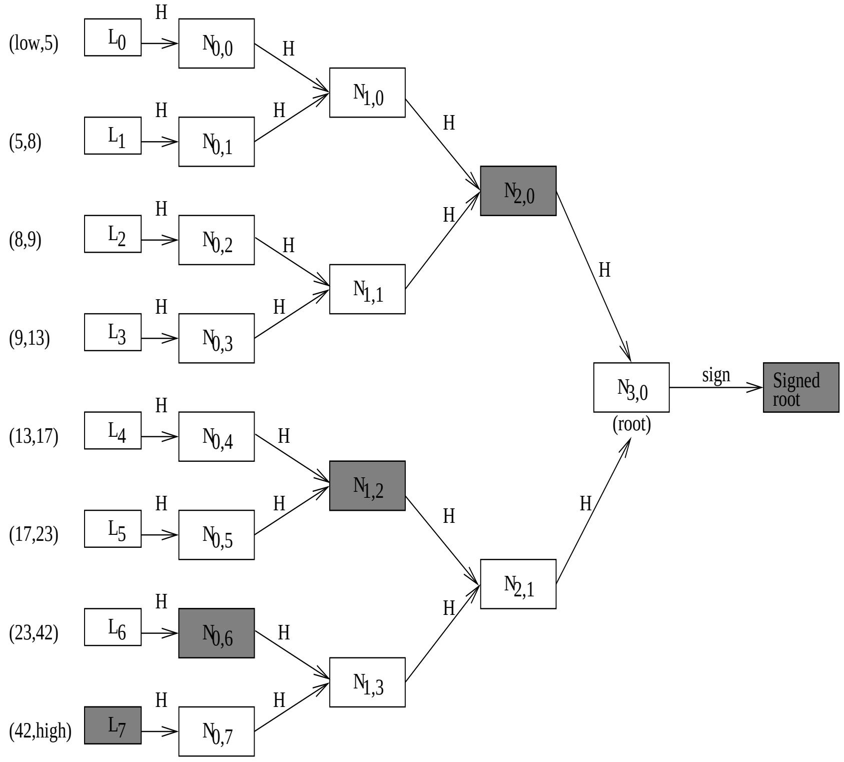


Figure 3.2: Sample CRT

3.2.1 Critical notes

The efficiency of the CRT scheme is really good. The amount of bandwidth needed is much less than for other schemes since the amount of data only increases with the logarithm (base 2) of the number of leaf nodes [28]. The computational load for both tree issuer and end user in constructing the root is also low, since *low cost* hash functions are used in the calculation. Another performance saving element is that an end user requesting the status for a chain of certificates only has to perform one signature verification operation. Another bandwidth advantage is that the directory only sends a subtree of the CRT needed for the end user to compute the root, contrary to the CRL scheme where the whole CRL is transmitted to the verifier. The performance advantages of the CRT scheme has lead to the adaption of CRTs in several commercial applications [32].

The CRT scheme is still a new concept and not yet an official standard so there can still be some unknown disadvantages with the scheme. One obvious weak point is updating of the revocation tree. Since all intermediate nodes in the tree are dependent on each other, the whole tree or large parts of the tree will have to be reconstructed for each update. The question then is how often should a CRT be updated. The answer is as often as possible to provide good timeliness but at the same time offer good performance. Naor and Nissim present a better solution for updating by using a 2-3 trees described in [30].

The communication in the scheme goes either between CA and directory or between end user and directory. As for all schemes depending on a database there is a possibility for a DoS attack by *flooding* the directory with a large number of queries.

3.3 OCSP

Online Certificate Status Protocol (OCSP) [23] is a standardized protocol for certificate revocation checking. In theory OCSP is especially designed to be an alternative for operations that require recently updated revocation information. In [23] high-value fund transfers or large stock trades are mentioned as examples.

An OCSP protocol session consists of an exchange of messages between a *requester* and a *responder*. The requester first sends an OCSP request for one or more certificates that the requester wants to check revocation status for. The responder receives the OCSP request and first checks that it has the right semantics and that the request contains the information needed by the responder. If any of these conditions are not fulfilled an error message is returned, otherwise a signed OCSP response message is sent. The signature requirements stated in [23] require that the signature either be by the CA that issued the certificate, from a trusted responder or by a responder designated by the CA. The signed response message includes the revocation status for the certificates listed in the initial request. Each certificate is given one of the possible status values *good*, *revoked* or *unknown*. The state good indicates that the certificate has not been revoked. The revoked state indicates that the certificate has either been revoked for a limited time or permanently. The unknown state indicates that the responder does not know anything about the certificate being requested. When receiving the response the requester should go through a checklist, that among other things include validating the signature, comparing the identity of the signer with the intended responder, for more details see [23].

One of the performance problems with the OCSP scheme is that the responder has to sign each response message. This causes a big processing delay, one solution OCSP provides is to pre process or pre construct OCSP responses prior to the actual request. In this way the requesters do not have to wait for their responses to be processed, and the responder will be able to handle more queries. Every preprocessed response has a time period ranging from when it was made to when the next update will at the latest happen. Unfortunately it also leads to a larger delay between a revocation event and updating of the certificate status. Another problem with this approach is that since the requester can ask for the status of more than one certificate, the number of possible preprocessed responses can get exceedingly large.

In [23] several security considerations of the OCSP scheme are mentioned. One obvious weak point is a DoS vulnerability. Since the process of signing is so expensive an attacker can easily flood the responder with a huge number of requests and eventually crash the system. The attacker can also exploit the fact that error messages are unsigned. The attacker can send a huge number of false error messages to the requesters and block them from service. Another security consideration is that the attacker can replay a preprocessed response for a certificate that had status good but that currently is revoked. This can be a possible scenario because the preprocessed response can still have valid time period, even though the certificate has been revoked.

3.3.1 Critical notes

As mentioned in the introduction of this subsection, OSCP is in theory especially designed for time critical applications that depend on recently updated revocation information. Whether this works in practice or not depends on several factors. First of all the OSCP scheme only describes the actual OSCP protocol, that is the message exchange between requester and responder. There is no description of how the OSCP server or responder receives its revocation information. Implementations may choose different sources for obtaining revocation information. Whatever revocation source is chosen, the OSCP scheme will still suffer from the weak points of that revocation scheme. However, the certificate verifier may be able to achieve more recently updated information, since the verifier does not have to download a CRL for example and use unnecessary resources for storing. The OCSP responder will probably have better resources for storing and can provide more frequent updates.

The OSCP protocol defines three different status values a certificate can have. These definitions are too vague and should be replaced with some easier and more concise terms. For instance the value good means that a certificate has not been revoked, but gives no guarantee that the certificate has not been expired. The unknown state is also unclear, and can basically mean anything from the certificate never having been issued to the responder not finding the right CRL for it. A certificate could for example have the status either valid or non valid. A certificate could be non valid if it is either revoked, expired or if it is impossible to find information for this certificate. Optionally, the non valid field could have an add-on field, describing why the certificate is not valid.

3.4 Short lifetime certificates

The certificates used today typically have relative long lifetimes lasting a year or several months. Certificates with long validity times have increased probability of a compromised key event or another event that should cause revocation of the certificate. An ongoing discussion is if it is possible to replace all or parts of revocation systems with certificates that have short lifetimes. Proponents argue that short lifetime certificates reduce the risk of an event that should imply revocation of the certificate. In [9] Rivest introduces a new certificate guarantee, where a certificate is definitely valid between time periods *date1-date2* and expected to be valid from *date3-expiration date*. For operations where high risk is involved (for example high value fund transfers) the verifier definitely will want a certificate with lifetime in the range *date1-date2*. Less risky operations can accept a certificate which is expected to be valid. This approach can limit the number of certificate reissues, and thereby also reduce the load on the CA.

The SPKI/SDSI infrastructure suggests using no revocation scheme, except for key compromise events, where a Suicide Bureau is used together with specific *certificates of health*, described in the SPKI/SDSI subsection.

3.4.1 Critical notes

The main problem with short lifetime certificates is the increased load on the CA. The rate of certificate reissues must stand in relation to the security vulnerabilities for the given environment. What is the best proof, a newly reissued certificate or newly updated revocation information? If a certificate holder must have the ability to get a certificate reissued at any time, then this is probably more resource demanding than updating revocation information that typically have predefined periodic updates. Frequent reissues may lead to peak loads for the CA and periods without service. Naor and Nissim presents a solution in [30]. Instead of reissuing certificates, certificates are updated, this is carried out by the use of a hash tree, and each user has to update his own path to the root node.

3.5 Discussion of revocation mechanisms

In this chapter we have looked at four different classes (identified by Myers in [33]) of mechanisms for revoking certificates. It is hard to say that one revocation strategy is superior to all others, since the strategies all have their pros and cons. Instead of developing a *global* revocation standard, I think it is important to identify the most important requirements for each environment and then apply the best suited revocation mechanism.

One of the most crucial requirements of revocation solutions is that the revocation information is recent, to prevent the use of revoked certificates. None of the current revocation mechanisms can provide total protection against revoked certificates, and I do not think it is realistic that PKI users can not tolerate any exposure to revoked certificates. However, for environments where the use of revoked certificates can have a serious impact, it is important that the revocation information be updated at a high enough rate to keep exposure to revoked certificates at a minimum. Other important parameters for evaluating revocation mechanisms besides recency are : security, scalability, standard compliance and performance (for example bandwidth use and processing delay).

An alternative solution can be to mix several revocation mechanisms together. Both the OCSP and the CRT scheme are for example dependent on revocation sources, like CRLs. One advantage with this scheme is that it can increase performance, scalability and timeliness. Unfortunately this will also add to the complexity of the scheme. There can also be a problem with non-repudiation. If several revocation sources are used, it can be difficult or impossible to prove what source an end entity has been using at a given moment [18].

Unfortunately revocation mechanisms are one of the most expensive parts of a PKI. A survey done by the MITRE Corporation in 1994 estimated that yearly running expenses of an authentication infrastructure derive almost entirely from administration of revocation [20]. Much of the reason for these high costs is that the revocation information has to be updated as often as possible to prevent the use of revoked certificates.

Chapter 4

Known compromised certificate attack on named-server version of TLS/SSL

A compromised public-key certificate is a valid certificate where the corresponding private key is known to someone other than the owner of the certificate. A compromise is classified as either *known* or *unknown*, depending on whether the compromise is discovered or not. As soon as a compromise is detected, the certificate should be revoked and the owner should request a new certificate.

An unknown compromised certificate represents a big threat as there is no way for a certificate user (certificate verifier) to discover the compromise. An attacker with access to the compromised private key can fake signatures in the certificate owner's name, and also decrypt all encrypted information sent from the certificate owner.

However, a known compromised attack might also be a threat. The article [44] describes a known compromised attack on *named-server* version of TLS/SSL. A named-server version of TLS/SSL means that only the server side is authenticated, while the client remains anonymous. This is the scenario for instance for many online shopping sites. The known compromised certificate incident is defined to be an incident where:

1. The private key of the certificate owner is discovered compromised.
2. The owner revokes the certificate.
3. The owner obtains a new certificate from the CA.

The TLS protocol [45] is a successor of the SSL protocol and is also back wards compatible with SSL. So it suffices to only describe the attack on the

TLS protocol, as the same attack can be mounted on the SSL protocol. Table 4.1 gives an overview of the notation used in the named-server version of TLS while table 4.2 shows the six message stages in the protocol.

KU_i	Public key for subject i.
KR_i	Private key for subject i.
`C`	Client.
`S`	Server.
T_i	Time stamp generated by subject i.
N_i	Random nonce generated by subject i.
$\{...\}_{KR_i}$	Signed by subject i, with key KR_i.
$\{...\}_{KU_i}$	Encrypted by subject i with key KU_i.
$\{i, KU_i\}_{KR_{CA}}$	Subject i's public key certificate.

Table 4.1: TLS Notation

$$
\begin{array}{ll}
C \rightarrow S : (N_C, T_C) & (M_1) \\
S \rightarrow C : (N_S, T_S) & (M_2) \\
S \rightarrow C : \{S, KU_S\}_{KR_{CA}} & (M_3) \\
C \rightarrow S : \{N'_C\}_{KU_S} & (M_4) \\
S \rightarrow C : \{H(K_{AB}, AB_5, (M_1, M_2, M_3, M_4))\}_{K_{AB}} & (M_5) \\
C \rightarrow S : \{H(K_{AB}, AB_6, (M_1, M_2, M_3, M_4))\}_{K_{AB}} & (M_6)
\end{array}
$$

Table 4.2: TLS Protocol

where

$$K_{AB} = \mathrm{F}((N_C, T_C, N_S, T_S), N'_C)$$

and

$$AB_5 = \text{"server finished"},\ AB_6 = \text{"client finished"}$$

The master key K_{AB} is calculated by a function F, that takes as input the messages M_1 and M_2 in addition to the master secret N'_C. The first two messages are sent in plaintext, so we see that an attacker needs access to the master secret N'_C in order to calculate the master key K_{AB}. One way of stealing the master secret N'_C is by a Man-in-the-Middle (MITM) attack combined with the use of a compromised server certificate. The attack will only succeed under the assumption that the client/user fails to check the status of the server certificate. This is unfortunately a reasonable assumption as most clients/users fail to test if the server certificate has been revoked. Very few applications today support revocation mechanisms like CRL or OCSP. For SSL/TLS, CRL or OSCP must be run separately from SSL/TLS [47]. Typically the user will be using a web browser client like Netscape or Explorer, where SSL and TLS are integrated. The TLS protocol will be transparent for the user and a X.509 certificate will be sent to authenticate the server. The web browser will only warn the user if one of the following is *not* true :

1. The certificate has been signed by a recognized CA
2. The certificate is currently valid and has not expired
3. The common name on the certificate matches the Domain Name Server (DNS) name of the server

As described in the subsection on the X.509 trust model most browsers today have a list of trusted authorities already hard coded into the browser cache, defining for the user which CAs to trust or not. The first check consists of checking if the CA is listed in the initial set of trusted CAs and verifying the signed certificate with the public key of the given CA. If one of the points 1-3 are not true, the browser will present a warning dialogue window. Given that the all points are verified, the client responds to the server with a challenge encrypted with the public key presented in the server certificate, as part of the TLS protocol. The server authenticates itself by successfully solving the challenge, proving that it possesses the private key to the certificate.

What the browser does not do is to check if the certificate has been revoked. This is up to the user. The *normal* user is typically not very security oriented so he/she does not bother to check with a CRL or other revocation mechanisms. Also a potential risk as we saw in the section on revocation, is that the revoked information has not been updated, as none of the currently implemented revocation mechanisms can absolutely guarantee completely updated information at all times. So we see that the MITM attack on named server version on TLS/SSL is a real threat. Also, making the problem worse are pre made hacker tools that can be downloaded so that people without highly detailed knowledge can carry out the attack. The MITM attack on named server TLS/SSL for example can be deployed with a hacker tool called dsniff developed by Dug Song [46].

Table 4.3 shows the attack protocol on named server TLS, while figure 4.1 gives a short explanation of each step in the attack protocol.

$C \to S : (N_C, T_C)$	(M_1)
$S \to C : (N_S, T_S)$	(M_2)
$S \to (C)I : \{S, KU_S\}_{KR_{CA}}$	(M_3')
$(S)I \to C : \{S, KU'_S\}_{KR_{CA}}$	(M_3'')
$C \to (S)I : \{N_C'\}_{KU_S'}$	(M_4')
$(C)I \to S : \{N_C'\}_{KU_S}$	(M_4'')
$S \to (C)I : \{H(K_{AB}, AB_5, (M_1, M_2, M_3', M_4''))\}_{K_{AB}}$	(M_5')
$(S)I \to C : \{H(K_{AB}, AB_5, (M_1, M_2, M_3'', M_4'))\}_{K_{AB}}$	(M_5'')
$C \to (S)I : \{H(K_{AB}, AB_6, (M_1, M_2, M_3'', M_4'))\}_{K_{AB}}$	(M_6')
$(C)I \to S : \{H(K_{AB}, AB_6, (M_1, M_2, M_3', M_4''))\}_{K_{AB}}$	(M_6'')

Table 4.3: Man-in-the-Middle attack on named server TLS

The attack does not work on versions of TLS/SSL where both the client and server is authenticated (named server, named client version). The reason for this is message M_4 in the protocol:

$$C \to S : \{C, KU_C\}_{KR_{CA}}, \{N_C'\}_{KU_S}, \{...H(M_1, M_2, M_3''), ...\}_{KR_C} \quad (M_4)$$

The parts shown as "..." are tags and other parameters that are not important for this discussion. We see that the hash of the first three messages is signed with the private key KR_C of the client. The attacker is unable to fake the clients signature as he does not have access to the client's private key KR_C. The attack is discovered in message steps M_5 and M_6 where different hash values are computed as a result of the attacker failing to fake message M_4.

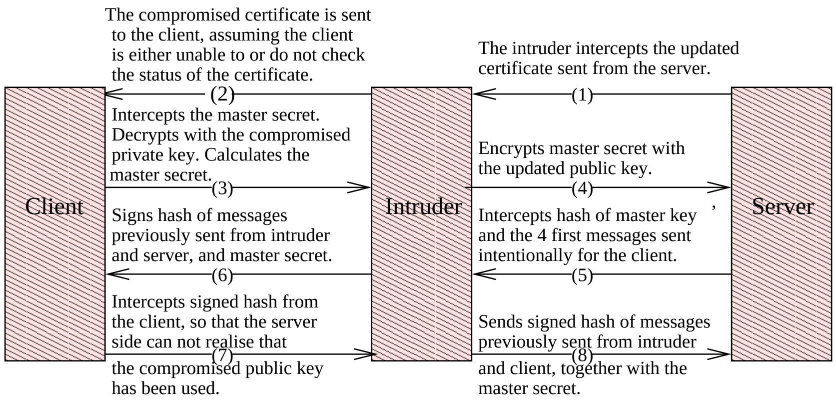


Figure 4.1: Description of MITM attack on named server TLS

Chapter 5

Personal Security Environment

A Personal Security Environment (PSE) can be thought of as a *container* that stores the private and public keys (certificates) of a user. The main purpose of the container is to protect the keys and make the keys available for the owner in a simple, but safe manner. PSE implementations can roughly be divided in two: a wide variety of software solutions and different smart card approaches, which I will consider in the following sections.

5.1 Software implementations of PSEs

Conventional software implementations of PSEs usually consists of a DES or AES encrypted file. A key owner is prompted for a password from which a symmetric key is derived [1] in order to decrypt the file. The main disadvantage with this approach is that the *normal* user has a tendency to choose bad passwords that are easy to guess. The problem is that users choose passwords that are easy to remember, consisting of *real* words (words that are listed in a dictionary) and that are short in length. This introduces a vulnerability for a dictionary attack. A dictionary attack uses an application that tries to crack the password by generating different words and combinations from available dictionaries. With todays fast computers and easily accessible hacking tools this becomes a simple task for the attacker. A metric for evaluating the strength of a password is obtained by looking at its *entropy*, which is a mathematical measure of possible patterns within random data [36]. For each random variable X with outcomes $X = \{x_1, ..., x_n\}$ having probabilities $p_1, ..., p_n$, the entropy of the variable X is defined to be [60]

[1] Some general methods for deriving a key from a password are described in PKCS #5 [34], for example by hashing the password together with a salt.

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x). \tag{5.1}$$

According to [35] English has about 1.3 bits of entropy per character, this means that the average encoding of a 30 character English password can be represented by about a 40 bit string, so breaking the password is comparable to doing a brute force attack on a 40 bit key. With non English characters, symbols like #, &, etc. , uppercase and lowercase there is generously 4 bits of entropy per character. So to protect against a brute force attack, the user should choose *random-looking* passwords consisting of at least 14 characters. This is comparable to doing a brute force attack on a 56 bit key, which is still possible, but requires some resources. Obviously choosing passwords of 14 characters and more is not very user friendly, and either the user will choose a shorter and easier password to remember or write down the password. The worst case scenario is where the security of the network is only as good as the weakest password. Such a scenario could consist of a set of users having access to some sensitive database, where the attacker can gain access to the database by cracking the weakest password.

5.1.1 The personal entropy scheme

The second method I will look into uses personal entropy to protect the secret key. The full scheme is described in the article [37]. Instead of depending on the protection of one single password, the scheme encrypts the PSE with the answers to several personal questions, hence the term personal entropy. The scheme is designed in such a way that a user can forget a subset of the answers and still recover the secret key, whereas the attacker will have to guess a large subset of the answers to break the scheme. The scheme uses a secret sharing technique (modified Shamir [38] scheme is used) with parameters n and t, where t<n and the secret data is divided into n parts. The encryption of the n parts is carried out separately and the passwords should be independent of each other. The scheme is designed so that if a user forgets up to (n-t) of the parts, it is still possible to recover the secret. An attacker on the other hand will have to guess at least t of the passwords to be able to retrieve the secret. The best brute force attack, given that the passwords are truly independent of each other is to pick t of the n passwords with as little total entropy as possible.

Encryption of the secret s :

1. Ask the user n questions : q_1,..., q_n to generate the answers a_1,..., a_n

2. Generate a random number r_s, that is used as a salt.
3. Compute the hash of question, answer and salt : h_1 = H(q_1 + a_1 + r_s),..., h_n = H(q_n + a_n + r_s).
4. Divide the secret s into n parts : s_1,..., s_n according to the (n,t) threshold scheme.
5. Encrypt the n parts by applying corresponding hash value : $E_{h_1}(s_1) = c_1$,..., $E_{h_n}(s_n) = c_n$
6. Keep q_1, ..., q_n, r_s and c_1, ..., c_n and discard the rest.

Recovering the secret s :

1. Ask the user the n questions and retrieve the answers : a'_1, ..., a'_n
2. Calculate the hash values consisting of salt: r_s, answers and questions : h'_1 = H(q_1 + a'_1 + r_s), ..., h'_n = H(q_n + a'_n + r_s)
3. Decrypt the cipher texts by adding the corresponding hashes : $D_{h'_1}(c_1)$, ..., $D_{h'_n}(c_n)$
4. By using the (n,t) threshold scheme recover the secret s by choosing a subset of t shares.

5.1.2 Software smart cards with cryptographic camouflage

The main idea in this scheme [39] is that only one password will decrypt the true key, while many passwords will produce decryptions that apparently look like the real key. An attacker doing a brute force (dictionary) attack will recover many plausible keys and not be able to separate the right key from the others. The only way the attacker allegedly can test if he has the right key is to try to authenticate himself by signing a challenge from an authentication server/entity. If the scheme is properly implemented and followed, the number of possible keys is too high and the attacker will be suspended access due to multiple authentication failures. So under the right circumstances the PSE will be protected against a dictionary attack, much the same way a smart card does, but completely in software.

In most software implementations of PSEs there are several ways for an attacker to verify if he has the true private key or not. Therefore it is important to eliminate these possibilities in order to make the scheme work.

One problem with encrypting the whole representation of the private key is that it contains some structure defined by the format. Hence the attacker can encrypt all possible keys with the format and all false encryptions will with high probability not give the same structure. A good example mentioned in [39] considers encrypting the modulus of an RSA key, where almost all false decrypts will have moduluses with small factors, unlike the true modulus.

An important restriction in the camouflage scheme is that the user's public key must be kept secret, and only trusted authorized users can have access to it. If the attacker knows the public key he can easily check if he has the corresponding private key, by for example encrypting some known text with the public key and verifying with the private key and see if he gets the same text. It is also important to protect signatures made by the user. If the signing process has been done in some standard form , for example with PKCS#1 [40] where the private key is first hashed and then padded in a deterministic way. Then the attacker can get hold of the data signed and can test all keys by signing the data and compare against the original signature. This attack can be prevented by padding the data in a random way with the aid of a secure pseudo-random generator.

5.1.3 Critical notes

In this section we have looked at three different software implementations of PSEs. The traditional software implementation consist of an encrypted file, where the encryption key is derived from the users password. This scheme is vulnerable to brute force/dictionary attacks, where much of the problem lies in the fact that users choose passwords that are easy to remember, hence easy to break. The fact that todays computers are only getting faster and faster only increases the problem, requiring longer and more difficult passwords. A safe key is roughly around 80 bits, and this is comparable to a 20 character password consisting of apparently random symbols. One way to guard against brute force attacks is to slow down the amount of passwords that can be checked for. A common way of hindering dictionary attacks is to use a *salt*. The salt is consists of 12 random bits that are padded with the user's password. The salt is stored in plain text together with the password file. When a salt is used the words in the dictionary need to be padded with each of the $2^{12} = 4096$ possible values of the salt. What is important to notice is that the use of a salt only stops attacks that aim at finding a random person's password. Since the salt is stored in plain text it is possible to do regular dictionary attack on a single user with this salt value. Another method for slowing down the attack is for example to hash the user-password a large number of times before storing it. What might be hard to accomplish in practice, is to find suitable solutions that at the same time can slow down a dictionary attack and still be fast enough not to be of any annoyance for regular users.

The personal entropy scheme solves some of the user problems with the first scheme. Instead of having to remember long complicated passwords the user can instead answer the questions of many personal questions to unlock the PSE. However, experience [37] has shown that finding good general questions is hard. The best solution which gives the highest entropy per password/answer is to let users construct their own questions that trigger some personal memory. However, many users are either reluctant to make questions or can not come up with good questions. Another possible problem with the scheme is that an attacker can gather a database on its subject, by for example tracking information on the Internet or from others.

Another problem is the size of questions that have to be answered. Let us assume that the questions can all be answered with English first names; from [37] we have that this will give an entropy of about 8 bits per password. If we would like a key of about 96 bits we have to choose t=12, from [37] we have that the corresponding n must be 20 or larger [2]. So in order to provide good security against a brute force attack we see that a large number of questions have to be answered by the user. The size of t becomes a trade off between security and user-friendliness.

The advantages of the personal entropy scheme is that it is easy to implement and does not require any extra resources or suffer from any restrictions. One solutions mentioned in [37] is to combine the personal entropy scheme with the *standard* scheme to avoid user annoyance. The main idea is to use the high entropy password to unlock a password with low entropy that is easy to remember, for example a 4 digit PIN. The low-entropy password could be used for all authentication operations until system shutdown.

The software smart card scheme is not vulnerable to a brute force attack given that is implemented and used in the correct settings. The scheme is not dependent on long passwords, but makes it infeasible to do a brute force attack by making it *impossible* for the attacker to distinguish the correct key from the many false keys. Compared to a hardware smart card there is not any need for buying smart cards or card readers. A disadvantage compared to the hardware implementation of a smart card is that all cryptographic calculations are carried out on the host processor, and is thus more vulnerable to unauthorized copying from viruses.

The software smart card key scheme has several restrictions and its usage

[2] Given that the user answers each question correctly with probability 0,95 and that the probability is close to 1 (0,99998) that the user will answer t of the n questions correct.

areas are therefore also limited. The scheme is most suited for use of digital signatures in a closed PKI where only trusted entities can verify signatures.

5.2 Smart card implementation of PSE

A smart card setup of a PSE consists of a smart card together with a smart card reader connected to the host computer. Typically an authorized user will have to use a PIN code together with the card in order to gain access to the system. Generally a cryptographic smart card contains the private and public keys (certificates) of a user and also has the ability to do some cryptographic calculations on the smart card, like digitally signing a document.

A smart card is a portable storage device that is embedded with either a microprocessor and a memory chip or only a memory chip. Memory-only-cards have a limited number of possible functions, usage areas are for example storage of telephone credits, transportation tickets or electronic cash. Having the microprocessor embedded makes it possible to add and delete information and also do some calculations on the actual card. Cards with microprocessors are dependent on some kind of energy support to power the microprocessor. Smart card with microprocessors are classified into contact cards and contactless cards, depending on how the card receives its energy [42]. Contact cards have gold plates or contact pads that communicate via direct electrical contact with the card reader. Often keyboards, PC's and PDA's have built in card readers. Contactless cards communicate via radio frequency (RF). A small wire loop is embedded inside the card and this acts as an inductor to supply energy to communicate with the reader. When the card is put into a RF field an induced current is created in the wire loop, and this is used as the energy source. The smart card readers for contactless cards usually consists of a serial interface for the computer and an antenna that communicates with the card.

Smart cards where invented in 1974 by inventor Roland Moreno and are used in a wide variety of applications today. "A typical smart card today has an 8-bit microprocessor operating at 5 MHz, 256 to 1024 bytes of RAM, 6 to 24 KB of ROM, 1 to 16 KB of EEPROM (Electrically Erasable Programmable Read-Only Memory), and sometimes an on-board encryption module" [41]. Figure 5.1 shows a general *crypto* smart card setup.

Physical attacks on smart cards can be categorized into two main categories : *invasive* and *non-invasive attacks* [43]. A non-invasive attack (also known as a eavesdropping or side-channel attack) concerns all attacks on the smart card where the chip package on the smart card is left untouched. Examples of frequently used non-invasive attacks are tampering with supply

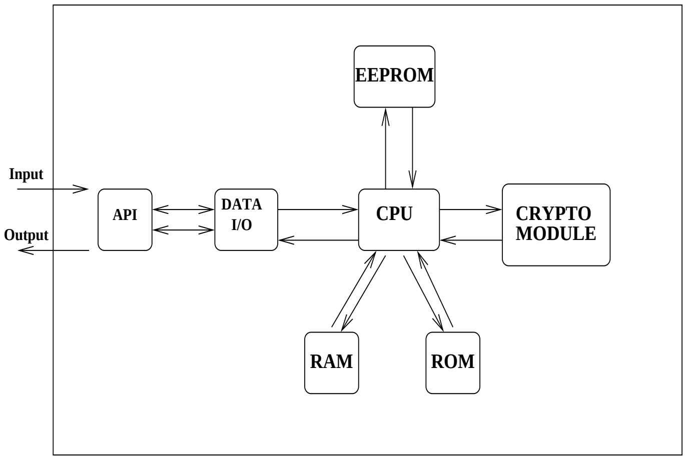


Figure 5.1: Basic crypto smart card configuration

voltage and clock signal to the CPU. By applying under-voltage and over-voltage it is possible to disable protection circuits and force the CPU into making wrong decisions. This can also be achieved by reducing the clock frequency, which will have the effect that the transistors will scale differently to the reduced frequency and can cause a possible error. By doing comprehensive testing it is possible for the attacker to find out which instructions are forced by applying different voltages and clock frequencies.

Non-invasive attacks are dangerous since they are hard to detect and do not require any expensive equipment. However, a successful attack requires a great deal of testing to determine the effect of various attack methods and the attacker must also have a very detailed knowledge of the different components in the chip package.

Invasive attacks are more complicated and require expensive equipment. The first step in the attack is to remove the chip package from the smart card. Once the chip has been opened it is possible to glue the chip to a test package and perform several attacks. A laser cutter is used to move the passivation layer, usually consisting of silicon oxide or nitride. With the help of a microscope and several micropositioners it is possible to access the stored data via the memory bus where all the data is available at a single location. Invasive attacks are more serious as it is possible to read information stored on the card, however they are far more expensive than non-invasive attacks.

5.2.1 Critical notes

Storing cryptographic keys on a smart card has several advantages. As mentioned in 5.1 most software implementations of PSEs suffer from a brute force attack vulnerability. The fact that computer speeds are increasing more and more only makes the attack more feasible. However, when the keys are stored on a smart card, there is no way for the attacker to try a brute force attack on the key space, as long as he does not have any access to the actual smart card. The attacker can of course try to steal a card and then try a brute force attack on the PIN code. Typically a PIN code is 3 to 8 digits long and there is a *three time rule* so that the attacker will be denied access after typing the incorrect PIN code three times. But as we saw earlier once the attacker has the actual smart card it is possible to do invasive and non-invasive attacks. These attack forms are a real threat, but the actual risk is lower since only few attackers have the detailed knowledge required to carry out the attack, and the case of a stolen card is not that frequent.

Another advantage with crypto smart cards is that cryptographic operations can be done on the card. This way it is harder for viruses or Trojan horses to copy or tamper with information. However, smart cards are vulnerable to certain types of virus attacks. Let us assume that we want to sign a document, the smart card receives the hash of the proposed document and signs the hash on the card by using the embedded encryption module. Then there could be a possibility for a virus switching the hash of the document with the hash of a completely different document, saying something like : "I give you my house and car". The problem is that the scam often is not detected before it is used against the user.

The use of smart cards for storing cryptographic keys/certificates has not yet been put into wide use. Some of the reason for this is the relative high cost of buying smart card readers and smart cards. Another problem is that there are lot of competing companies developing different standards that often are incompatible. Other problems are limited storage space and potential trust splits. Trust splits can occur when a party responsible for example for the manufacturing of the card can take over the functionality of for example the card owner.

Chapter 6

Access control in SkandiaBanken

In this chapter I will present an outline of SkandiaBanken's access control scheme. The outline has been based on information found on SkandiaBanken's web pages [48] and from email contact with SkandiaBanken's technical support [49] [1]. It has been difficult to find precise information as the scheme is developed by the company EDB Fellesdata [52] and the fact that details of the scheme are proprietary. I will discuss possible security holes, and how these could have been used to compromise a SkandiaBanken account. In particular I am interested in studying a *real-life* example where digital certificates are being used for authentication. It is also important to illustrate that the analysis of a security protocol is very difficult, due to the many ways in which an attacker can take advantage of the protocol environment.

6.1 Customer registration/Certificate enrollment

SkandiaBanken is a bank that specializes in online banking services. In Norway it opened the 27th of April 2000, and it is a branch of the Swedish SkandiaBanken. SkandiaBanken in Norway has about 140 employees and more than 200,000 customers.

When registering as a new customer, one has to fill out a form shown in figure 6.1. SkandiaBanken first controls that the entered SSN (Social Security Number) is valid [2]. Given a valid SSN, SkandiaBanken verifies that the entered name matches the SSN and sends a registered mail to the address listed in the national register. The customer can then collect a four digit PIN code

[1] The information gathered is based on the existing SkandiaBanken scheme July 2003.

[2] This is done by verifying that the two last control digits calculate correctly based on the first nine and ten digits of the SSN.

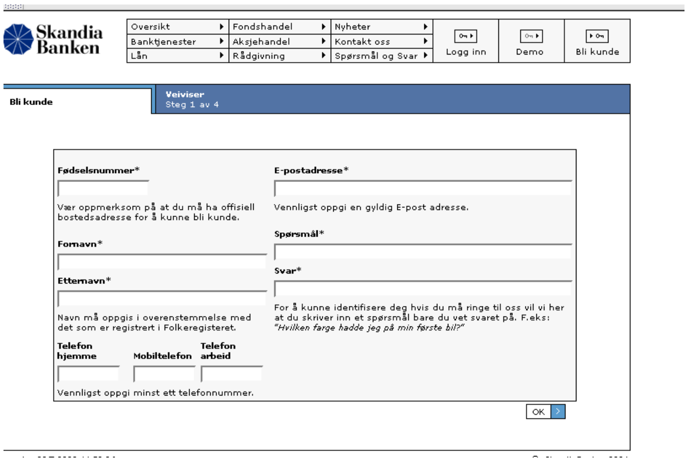


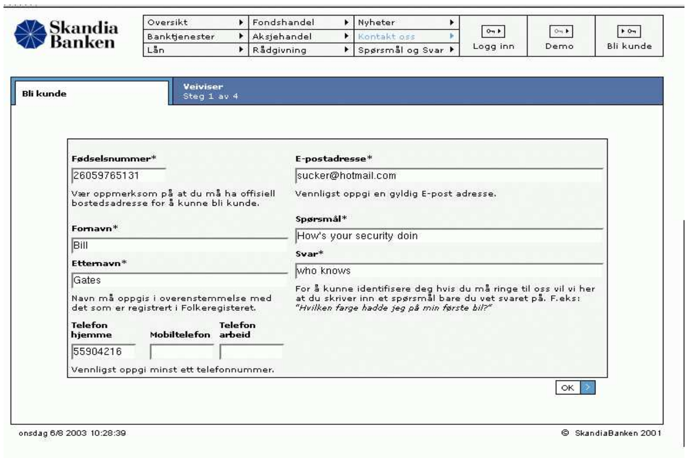
Figure 6.1: Customer registration

at the nearest post office. During the process of looking at SkandiaBanken's customer registration, some interesting observations were done. Several different SSNs were tried with different *made up* names and emails. Table 6.1 shows some of the SSNs tried and the response from SkandiaBanken on each one. We see from table 6.1 that SkandiaBanken does not give appropriate responses for some SSNs. Apparently SkandiaBanken first checks if a SSN is valid by calculating the two last control digits. Then it checks that the new customer is 18 or older. When testing for age SkandiaBanken does not only look at the birth date, but also uses the individual number as an indicator of age. If age and SSNs verify correctly the customer's name and SSN is tried matched against the name and SSN listed in the national register. We see that there is a *bug* for the last SSN. As shown in figure 6.1 and table 6.1 we see that it is a valid SSN and the name Bill Gates is apparently not checked for in the national register. The reason for this could be that the information is not available for people born before 1900 or SkandiaBanken simply do not expect customers of this age [3]. Figure 6.1 shows the response to the fake registration.

Before login into the account, the customer has to download a *personal* certificate and possibly a server/SkandiaBanken certificate [4]. The customer must download a new personal certificate for each new host that is being used to connect to SkandiaBanken. A user downloads a certificate by entering the

[3]After meeting with SkandiaBanken's head of security in Bergen it was discovered that the reason this SSN was not checked for was because the person with this SSN was dead.

[4]This will depend on which browser the customer is using. Newer versions of Explorer, for example, already have SkandiaBanken listed as a trusted authority and the certificate is hard coded into the browser.



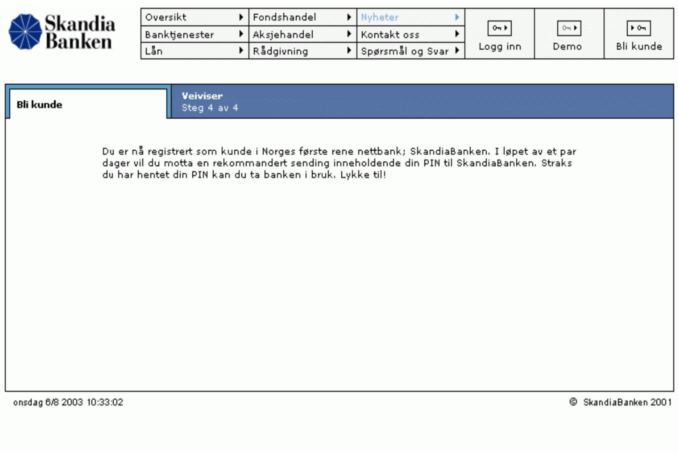


Figure 6.2: Fake customer registration

SSN	SkandiaBanken response	Explanation
30100090045	SSN on the wrong format.	Two last control digits calculate wrongly.
30100090046	Sorry we have an age limit of 18.	Person with this SSN born in the year 2000.
10018590056	Sorry we have an age limit of 18.	SSN does not belong to real person, individual number (900) in SSN belongs to person born in period 2000-2054 and not 1985.
10018439714	The given information does not match against the information registered in the national register.	SSN is valid but name and SSN do not match
26059765131	Registered as a customer	SSN is valid and belongs to a male born in 1897.

Table 6.1: Testing of SSNs

4 digit PIN code along with the corresponding SSN. There is a *three time rule*, meaning that a user who types the incorrect PIN code three times will temporarily lose access to the account. The customer then has to contact SkandiaBanken who will issue a new registered mail with a new PIN code. An interesting observation is that given any valid SSN of a SkandiaBanken customer, it is possible to deny this customer access by entering the wrong PIN code three times.

If PIN code and SSN match, SkandiaBanken issues a X.509 certificate for this user and a corresponding private key is generated. Where and how the personal user certificate and private key is stored depends on the browser used. Mozilla 1.01 for example stores all certificates in the file cert7.db and corresponding private keys are stored in key3.db. The keys are stored in a *software security device* [50]. The user must enter a *master password* to unlock any private key from the PSE. A similar system applies to Netscape browsers.

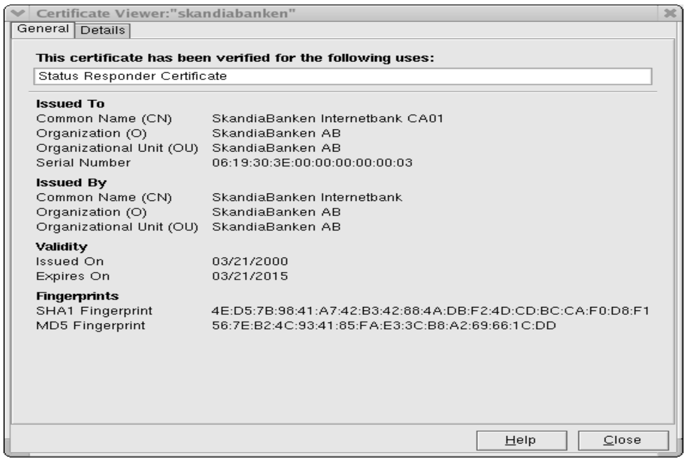


Figure 6.3: SkandiaBanken certificate

SkandiaBanken uses two self-issued X.509 Verisign certificates, a CA01 and a CA02 certificate. The reason for this is probably to divide the customer population between different CA's to achieve easier issuing and managing of certificates. Figure 6.3 shows an example of a SkandiaBanken CA01 certificate and figure 6.4 shows the details of the certificate. Some important differences between the server certificate and customer certificate are :

1. The common name (CN) field is set to the name of the customer and the organization (O) field is set to the SSN of the customer.
2. The validity of a client certificate is only for a year.
3. The certificate is issued by SkandiaBanken CA02 or CA01

6.2 SkandiaBanken login

SkandiaBanken secures the transaction between the client and server by using SSL 128 bits with two way authentication embedded. Simplified this means that the transaction between client and server is encrypted with a 128 bit session key and that both the client and server are authenticated. The following is an outline of the SkandiaBanken login as the details are kept secret:

1. The customer enters the SkandiaBanken login menu and the browser initiates an SSL session with the SkandiaBanken server. Most of the

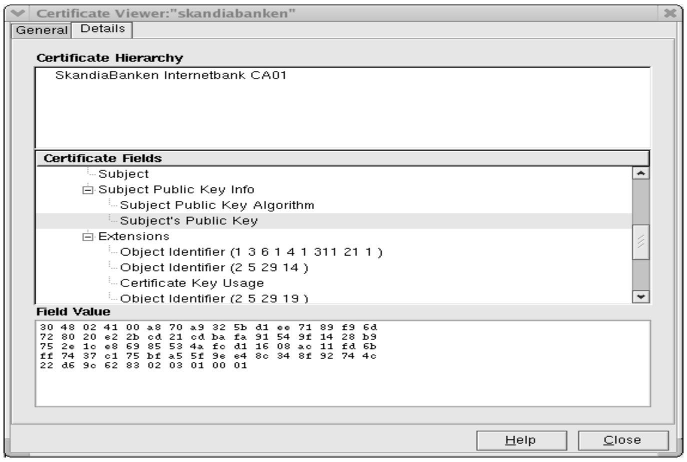


Figure 6.4: SkandiaBanken certificate details

steps of the SSL session are transparent to the user, and are done automatically between browser and server.

2. The browser sends a challenge to the SkandiaBanken server. The server authenticates itself by signing the challenge with its private key and returns it to the browser along with its certificate path. The browser automatically verifies the signature by applying the public key listed in the SkandiaBanken certificate. The browser also checks that the certificate chain ends in a trusted point. The browser will inform the user if the certificate is issued by a non trusted CA, the certificate's validity period has ended or if the certificate's common name (CN) does not match the domain name (DN) of the server. There is no automatic check for revocation status of the server certificate.

3. The user chooses a certificate and enters his/her PIN code and corresponding SSN. The browser sends a message that includes a signed challenge and certificate path. SkandiaBanken verifies the signature by applying the public key listed in the user certificate. In addition SkandiaBanken should also verify that this certificate belongs to the correct customer by comparing the SSN in the certificate with the SSN entered by the user. After the initial *handshake* a session key is exchanged to encrypt the transaction. The customer finally gains access to the account by entering a valid PIN code.

6.3 Brute force attack on PIN code

SkandiaBanken advocates having a user-friendly security solution [51]. All you need to log into SkandiaBanken is your PIN code in combination with your SSN and personal certificate. The only thing the user is requested to remember is the 4 digit PIN code. One of SkandiaBanken's slogans are : "Our code is so simple that you simply remember it. There is a certain element of security also in this" [5]. I agree that having a PIN code with only 4 digits might prevent a user from writing the code down or storing it on their cellular phone. However, a 4 digit PIN code is not very secure against a brute force attack, there are only 10000 possible PIN codes and more than 200000 SkandiaBanken users. SkandiaBanken's argument is that the PIN code is only valid together with the user's SSN and personal certificate, so it is sufficient to use a 4 digit PIN code. In the following I will try to prove that argument is dubious.

An attacker doing a brute force attack on the PIN code supposedly needs the corresponding personal certificate and SSN. However, the attacker only needs the SSN as the attacker can verify a PIN code by attempting to download a new certificate. The attacker hence has three attempts at guessing the correct PIN code for each SSN belonging to a SkandiaBanken customer. Thus there is a probability of $\frac{3}{10000}$ that the attacker will guess the correct PIN code for a given SSN. We see that if the attacker is able to acquire the SSNs of *many* SkandiaBanken users there is a pretty good probability that the attacker will be able to gain access to several accounts.

How easy is it for an attacker to acquire SSNs? The Norwegian SSN is actually not confidential [53] information. Given a reasonable documentation for its use, the SSNs can be requested from the national register. Many public institutions like hospitals, banks and tax authorities have legal access to people's SSNs, but it can be difficult for private persons to argue for the need of many SSNs. Therefore it might be better for an attacker to generate the SSNs. This is not hard given the infrastructure of the Norwegian SSN [54]:

The Norwegian SSN consists of 11 digits: $x_1x_2x_3x_4x_5x_6i_1i_2i_3c_1c_2$.

$x_1x_2x_3x_4x_5x_6$: indicates the birth date of this individual in the order *ddmmyy*.

[5] Freely translated from [51].

$i_1 i_2 i_3$: Is called the individual number that is used to separate people born on the same date. The national personal register gives SSNs in the order of birth messages that are received. They start with the highest available valid individual number for that day and proceeds down wards for each new message. The individual number is based on the century the person is born in, shown in table 6.2. It is also possible to distinguish boys from girls by looking at i_3, which is odd for boys and even for girls.

$c_1 c_2$: are control digits that are calculated as weighted sums of the respectively first 9 and 10 digits.

$c_1 = 11 - (3x_1 + 7x_2 + 6x_3 + x_4 + 8x_5 + 9x_6 + 4i_1 + 5i_2 + 2i_3)(mod 11))$.

$c_2 = 11 - (5x_1 + 4x_2 + 3x_3 + 2x_4 + 7x_5 + 6x_6 + 5i_1 + 4i_2 + 3i_3 + 2c_1)(mod 11))$.

When generating a SSN the highest available individual number for the given birth date is chosen. However, if either c_1 or c_2 is calculated to be 10 ($mod 11$) the SSN is discarded. We can assume a uniform distribution of when c_1 and c_2 are calculated to be 10 ($mod 11$) and give an estimate over how often this will happen:
$p(c_1 \cup c_2) = p(c_1) + p(c_2) - p(c_1 \cap c_2) = (\frac{1}{11}) + (\frac{1}{11}) - (\frac{1}{11})^2 = (\frac{21}{121})$.

Individual number	**Year in birth date**	**Born**
500-749	>54	1855-1899
000-499		1900-1999
500-999	<55	2000-2054

Table 6.2: Correspondence between individual number and birth date

We see that an attacker easily can generate SSNs. However, we have to take into consideration whether a generated SSN belongs to a *real* person and if the SSN belongs to a SkandiaBanken user. I will look at three different ways of generating SSNs. The purpose of the two first methods are mostly to illustrate some of the possible problems with this attack strategy. Afterwards I will discuss different assumptions and the likelihood of an attack.

I first make some general assumptions:

- Some SSNs are *dummy* numbers due to errors with birth date registration, immigrants for instance can have incomplete information about exact birth dates. Also some generated SSNs will probably belong to dead people. The number of these SSNs for this attack are so few that I will not consider them in the probability calculations. Also I will exclude any numbers for leap days since this will complicate our SSN generation.

- It can be discussed if an attacker is better off with guessing two or three times for each SSN. For now we assume the attacker only guesses two different PIN codes for each attack. The reason for this is that an attacker then probably can verify more PIN code SSN combinations without being detected as quickly. If three wrong attempts are made the customer with this SSN will temporarily be denied access. A large number of denials in a *short* time period would seem suspicious. I will discuss the possibility of guessing three times later.

- We assume that it is possible to develop a program or script that can login to SkandiaBanken's pages and automatically enter different SSNs and PIN code pairs.

- We have to consider using many hosts in the attack, as it can seem hard to spoof the IP address of one host and get the response from SkandiaBanken if the PIN code SSN combination was correct or not. I will discuss this in more detail later.

I start by looking at a *best case scenario* for SkandiaBanken where we generate all possible SSNs for people that are 18-100 years old. This way we guarantee covering the SSNs of all 200000 customers in SkandiaBanken. All of SkandiaBanken's customers are born in the 20th century and are therefore given individual numbers in the range 000-499. Hence for each day in the year we get 500 possible SSN, but an estimate of $\frac{21}{121}$ will be invalid numbers.

We get the following amount of SSNs : $(500) \times (365) \times (83) \times (\frac{100}{121}) \approx 12,3 million.$

P(Guessing correct PIN code for one SSN) $= \frac{2}{10000} = \frac{1}{5000}$.

P(At least cracking one PIN code) = 1 - P(cracking no PIN codes).

P(At least cracking one PIN code) $= (1 - (\frac{4999}{5000})^{(200000)}) \approx 1$.

Anticipated number of cracks when checking all SSNs : $\mu = \frac{200000}{5000} = 40$.

The drawback, from the attacker's point of view, is of course the huge number of SSNs that he/she has to check for. A *large* portion of the SSNs will neither belong to *real* people nor to SkandiaBanken users. We see that there is not only a problem with the efficiency of the attack, but also there is a high probability for SkandiaBanken detecting the traffic amount as being suspicious. One way of increasing the ratio of SkandiaBanken SSNs is to concentrate the attack on a specific age group that we know has a high frequency of SkandiaBanken users. From [55] we have that 34 % of customers in pure online banks, like SkandiaBanken are males in the age group 26-35 years old. We therefore generate all possible SSNs for all males that are between 26-35 years old. We then guarantee getting the SSN of all males in this age group that are customers in SkandiaBanken. The expected number of customers in SkandiaBanken in this age group hence are: $(200000) \times (0.34) = 68000$.

Number of SSNs : $(250) \times (365) \times (10) \times (\frac{100}{121}) \approx 754132$.

$P(\text{At least cracking one PIN code}) = (1 - (\frac{4999}{5000})^{(68000)}) \approx 1$.

Anticipated number of cracks when checking all SSN one time: $\mu = \frac{68000}{5000} \approx 13.6$.

It is seen that the number of SSNs has been decreased, and there is a higher concentration of SkandiaBanken customers in this age group. However, one still has to generate a lot of SSNs belonging to non-existing people. In the last generation advantage is taken of the assigning of SSNs to new born and immigrants. As mentioned before a person is given the highest available SSN for that day. Instead of generating all SSNs for one day we can use birth statistics to shorten the amount of SSNs. For the period corresponding to males in the age group 26-35 there is an average of about 30824 births [56] per year. This gives an average of: $30824/365 \approx 84$ births pr day. I will disregard the fact that immigrants also are assigned SSNs and discuss later a more precise estimate of SSNs assigned to people born in the period 1969-1977.

Number of SSNs : $(84) \times (365) \times (10) = 306600$.

The probability of a successful attack will be approximately the same as for our previous generation, given that our estimation of SSNs is accurate.

6.4 Discussion of brute force attack

I have looked at three different ways of generating SSNs belonging to SkandiaBanken customers. The last method involved the least number of SSNs and contained the highest percentage of SkandiaBanken SSNs. I will therefore focus the discussion on the last attack method. What I would like to discuss further are assumptions about:

1. Running time.
2. Possible control routines in SkandiaBanken.
3. Insecurity in data.
4. Variations in attack strategy.

6.4.1 Running time and possible control routines

The running time will depend on work load, CPU power and possible control routines in SkandiaBanken. I will assume a standard CPU is used as processor speed is not the main problem. The work load consists of generating a set of SSNs and testing each SSN against SkandiaBanken. We can assume that the generation of SSNs is straightforward given the infrastructure of the Norwegian SSN. The bottleneck is the testing of each SSN. One has to make a program or script that can automatically log into SkandiaBanken's certificate download menu and enter a SSN together with two different PIN code combinations. Giving a precise estimate over the number of SSNs that can be checked for in a certain time interval is difficult when one has not actually implemented and tested a program that does it. However, the biggest uncertainty lies in the fact that SkandiaBanken's control routines are proprietary. A strategy for an attacker would therefore be to first try to establish a rough *sketch* over SkandiaBanken's control scheme. One way of doing this is to use an anonymous IP address, for example by logging onto a wireless network without access control (e.g many home networks). It is then possible to run a series of tests and study how SkandiaBanken reacts to different events. The attacker could for example study for a given time period the amount of *allowed*:

1. Certificate download attempts.
2. Traffic from one IP address.
3. Failed certificate download attempts.

Given an outline of SkandiaBanken's authentication and control scheme the attacker can then optimize a brute force attack.

How could a global control on the number of certificate download attempts influence the attack? This could certainly limit the speed of the attack, as the attacker would not be able to test as many SSNs. However, SkandiaBanken has to allow users to download new certificates every time they use a new host. So a reasonable amount of certificate downloads must be accepted. The advantage for the attacker is that he or she can spread the attack over time and gather PIN code SSN pairs before carrying out the actual attack. Suppose that 306600 SSNs are generated, trying to cover all boys born in the time period 1969-1977. One can look at the time perspective when checking different sets of SSNs each day. Table 6.3 shows some different scenarios, the second column indicates how many percent of SkandiaBanken's customers this corresponds to. The third column shows the anticipated time, given that one can check that many SSNs each day.

SSN pr day	%	**Time**
1000	0.5%	$\simeq$ 307 days
5000	2.5%	$\simeq$ 62 days
10000	5%	$\simeq$ 31 days
20000	10%	$\simeq$ 16 days

Table 6.3: Running time overview

What could be bad news for an attacker is if SkandiaBanken has control mechanisms for detecting a suspicious amount of failed certificate download attempts. There is already some control, as each user is only allowed to enter the wrong PIN code two times. The problem is if there is some sort of global control surveying the total amount of failed certificate download attempts in for example one day. Any such limit would probably be lower than the number of valid certificate download attempts. This is only speculations, and the attacker can still try to find a number of SSNs lying near SkandiaBanken's *allowed* limit.

Given that SkandiaBanken has implemented control on the amount of traffic from one IP address this could complicate the attack further. However, there are several strategies the attacker could apply to try avoid being detected. One alternative is to only try *a few* SSN PIN code combinations each day. This of course takes long time, but will still represent a security risk, as the attacker over time probably will find at least one PIN code SSN pair. Another possibility could be to spoof IP addresses from one or more hosts. The problem is not that of finding valid IP addresses, but the attacker also needs to know if a SSN PIN code pair was valid so the attacker needs a way of acquiring the response from SkandiaBanken. According to [57] this is difficult, but it is possible to *sniff* the response if an IP address is spoofed on the same subnet as the *real* machine is on. A more complex strategy is to make other hosts do the work for the attacker. One way of doing this could be to first make a program that automatically logs into SkandiaBanken and tests SSNs for valid PIN code combinations. The attacker can then try to spread this program to *many* hosts, either via a virus or Trojan horse. The attacker could use a good *random picker* that distributes the set of SSNs to the hosts. What remains for the attacker is to get the results back from the infiltrated hosts. This could be achieved by having a trigger in the program that logs onto for example an IRC server when a PIN code SSN match has been found. The attacker can then withdraw the information from the IRC server[6].

6.4.2 Insecurity in data

In our last generation of SSNs we used the mean of boys born in the age period 1969-1977. This could give an inaccurate result as birth rates vary with month and also from year to year. Also I have not considered immigrants at all. One way to include immigrants is to use statistics over the national population sorted by age and sex [59]. Further one would like to decrease the generation of SSNs of *non existent* people as much as possible in order to do the attack both quicker and reduce the probability of detection. This can be accomplished by for example generating half of the expected SSNs for that day. This way we have a very high probability of generating SSNs of only *real* people, because of the chronological assigning of individual numbers. The total population of males in the age group 26-35 is 340414 [59], this gives an average of ≈ 94 SSNs for each day. If we generate for example 47 valid SSNs for each day, we generate: $(47) \times (365) \times (10) \approx 171550$ SSNs. We can assume every person in this age group has the same probability of being a SkandiaBanken customer, independent of the SSN. So we can further assume that we generate half of all the customers in the age group 26-35.

Number of SkandiaBanken customers in this age group: 68000.

[6] This could be done by using iroffer [58], a software program that acts as a file server for IRC

We cover 34000 of these customers.

Anticipated number of cracks for one cycle of the SSNs is then:

$\mu = \frac{34000}{5000} = 6.8.$

6.4.3 Variations in attack strategy

The attack strategy we have presented so far focuses on SkandiaBanken not detecting any part of the attack. Another strategy could be to use an anonymous IP address and not worry about any part of the attack being detected. Instead of spreading the attack over time to avoid detection, we could focus on verifying as many SSNs as possible before SkandiaBanken reacts. The advantage of doing the attack quickly is that this might surprise SkandiaBanken and several PIN code SSN pairs can be found before SkandiaBanken gets the chance to respond. How is it possible to increase the speed of the attack? The attacker could start by guessing three times for every SSN. This will increase the probability of guessing correctly and at the same time suspend all customers whose PIN codes are incorrectly guessed three times. This will create even more chaos for SkandiaBanken, and might be an advantage for the attacker. In order to do the attack as quickly as possible the attacker can use many different hosts and parallel processing. If we look at the probability, when guessing three times, and using the same set of SSNs as earlier:

Anticipated number of cracks: $\mu = (34000) \times (\frac{3}{10000}) = 10.2.$

One can also increase the attack efficiency slightly by using PIN code filtering. One can assume that PIN code numbers with all 4 digits equal or 3 equal digits in the PIN code are rare, and most likely will either be changed by the customer or are not issued by SkandiaBanken. The number of possible PIN codes then becomes: $10000 - (10) - (4 \times 9 \times 10) = 9630$.

The expected number of cracks then becomes: $\mu = (34000) \times (\frac{3}{9630}) \approx 10.6.$

Another point worth mentioning is that after several attacks have been made, an attacker can make a list of SSNs belonging to SkandiaBanken customers for later attacks. It could be interesting to test if SkandiaBanken reacts differently when guessing incorrectly three times for a SSN not belonging to a SkandiaBanken customer. The attacker can then filter out all SSNs not belonging to SkandiaBanken customers and increase the efficiency of the brute force attack even more.

6.5 Conclusion for this chapter

I have looked at some aspects of SkandiaBanken's access control system. I have looked at the registration of a customer (certificate enrollment) and the main structure of SkandiaBanken's login scheme. Further, I have discussed the possibility of a brute force attack on the PIN code. Several weak points have been discovered in SkandiaBanken's present solution. Maybe most important is that SkandiaBanken's use of certificates seems to have little security function. SkandiaBanken's access control scheme consists of three main objectives: identification, authentication and authorization. Simplified we can say that *proof of identification* in the scheme is the customer's SSN, *proof of authentication* is the users certificate and that the 4 digit PIN code authorizes the user to access his/her account. As has been seen in the brute force attack, an attacker only has to check SSNs and PIN code pairs in order to try to download a new customer certificate. This is a convenient solution for customers as they only have to remember a 4 digit code. The trade off is bad security, the SSNs are too easy to generate and the PIN code is too short to withstand a brute force attack. One way of guarding better against brute force attacks could be to use a user name instead of a SSN. The reason this is not done is probably because it is easier to automate customer registration by using an SSN with the corresponding name against the national register.

One of the weakest points discovered in the scheme is that it is possible to exclude any customer by trying the incorrect PIN code three time for this customer's SSN. By using the SSN generation described earlier it is possible to quickly do a huge Denial of Service (DoS) attack and shut down service for a significant amount of SkandiaBanken's customers. This is hard to defend against, and the attack can be repeated as long as the present solution exist.

Another problem is that SkandiaBanken has little control over its own security solutions. The company EDB Fellesdata has made the current solution, which again uses elements of Verisign products in it. This can not only cause trust splits, but can also be a problem when it is important to respond quickly to attacks.

I have only studied a brute force attack on SkandiaBanken, which I think gives a good overview of the security of the system. There are also many other possible attack types that I have not had time to study. What could be especially interesting is looking at the danger of logging attacks. That is attacks that logs input from the keyboard. This could for example be done by either attaching a hardware device to the back of the machine or using software. Certain situations are more exposed to this kind of attack, for example when using an Internet cafe. It is also possible to spread logging programs via viruses or Trojan horses.

Chapter 7

Summary and further work

In this thesis, I have taken a critical look at PKI solutions. I started by identifying some important security considerations when engaging in electronic transactions. By looking at the three PKIs: PGP, SPKI/SDSI and X.509, I showed that there are many ways of organizing a PKI. The number of people in an environment, and the service and security requirements of an environment will all influence the structure and complexity of a PKI. Further, I have studied in a more general perspective, mechanisms for certificate revocation and different solutions for storing cryptographic keys. Finally, different PKI aspects in SkandiaBanken's access control scheme were studied. A copy of the chapter on SkandiaBanken was sent to SkandiaBanken's division in Bergen. A meeting was agreed upon 1st of December 2003. SkandiaBanken is now currently changing some of its solutions, and have started logging the email and mobile phone number of customers. The weaknesses described in this thesis will most likely be fixed.

I ended Chapter 1 of this thesis with some general considerations about PKIs. Many of these considerations were studied in the thesis but I did not have time to look into the following:

- On which basis is a certificate issued? In order to guard against forgery it is important to have good procedures that secures the identity of the entity requesting a certificate. However, care must be taken so that personal information is kept secret.

- How is the private key delivered? Is the private key generated in such a way that the certificate issuer can not obtain any information about the private key? How does the browser communicate with the PSE when a user signs a document online? Which risks are associated with this?

- A lot of the current implementations of PSEs only focus on protecting the private key. Little research has been done on the necessity of also protecting public key certificates. This is for example relevant to SkandiaBanken, as an attacker could learn SSNs by studying the certificates of customers.

- How can one communicate between different PKIs? Cross certification is one solution. However, this adds complexity, and there can be a problem with multiple certification paths and CAs operating with different certificate validity times. Another possibility is the use of *bridge CAs*.

- Discussion of certificate lifetimes. Why is it often one year? How does key size relate to the lifetime of a certificate?

- How are old keys stored? This is important in order to decrypt old information and for non repudiation purposes.

Bibliography

[1] Loren Kohnfelder's bachelor thesis in electrical engineering. Towards a Practical Public Key Cryptosystem (1978).

[2] CCITT Rec. X.501 (1994) | ISO/IEC 9594-2:1994, Information Technology - Open Systems Interconnection - The Directory: Models.

[3] Advances in public key certificate standards, Security, Audit and Control, 13 (1995, ACM Press/SIGS AC,9-15).

[4] Carl Ellison, B. Frantz, B. Lampson and T. Ylan. RFC 2693: SPKI Requirements. The Internet Society. September 1999. See `ftp://ftp.isi.edu/in-notes/rfc2693.txt`. Last visited 20th February 2003.

[5] Ronald L. Rivest and Butler Lampson. SDSI - A Simple Distributed Security Infrastructure. See `http://theory.lcs.mit.edu/~rivest/sdsi10.ps`. Last visited 20th February 2003.

[6] Toni Nykanen. Attribute Certificates in X.509. November 2000. See `http://www.hut.fi/~tpnykane/netsec/final/`. Last visited 24th February 2004.

[7] Dwaine Clark, Jean-Emile Elien, Carl Ellison, Matt Fredette, Alexander Marcos, Ronald L. Rivest. Certificate Chain Discovery in SPKI/SDSI. See `http://theory.lcs.mit.edu/~rivest/ClarkeElElFrMoRi-CertficateChainDiscoveryInSPKISDSI.ps`. Last visited 24th February 2004.

[8] Dwaine Clarke's Master's thesis. SPKI/SDSI Http Server/Certificate Chain Discovery in SPKI/SDSI.September 2001. See `http://theory.lcs.mit.edu/~cis/theses/clarke-masters.pdf`. Last visited 24th February 2004.

[9] R. Rivest. Can We Eliminate Certificate Revocation Lists? In *Proc.* of *Financial Cryptography 98; Springer Lecture* Notes *in Computer Science* No. *1464 (Rafael Hirscfeld, ed.)*, February 1998. See `http://theory.lcs.mit.edu/rivest/revocation.pdf`. Last visited 24th February 2004.

[10] William Stallings. Cryptography and Network Security: Principles and Practice. Second edition. Prentice Hall 1998.

[11] ITU-T X.509 (93)| ISO/IEC 9594-8:1995 Information Technology - Open Systems Interconnection - The Directory Authentication Framework.

[12] ITU-T Recommendation X.500 to X.525(1993)|ISO/IEC 9594:1994, Information technology - Open Systems Interconnection - The Directory.

[13] See `http://www.ejeisa.com/nectar/dedica/3.2/doc0001.html`. Last visited 18th March 2003.

[14] S.Farell [Baltimore Technologies], R.Housley [RSA Laboratories]. An Internet Attribute Certificate Profile for Authorization. April 2002. See `ftp://ftp.isi.edu/in-notes/rfc3281.txt`. Last visited 20th March 2003.

[15] Warwick Ford, and Michael S. Baum. Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption. Prentice Hall PTR, 1997, page 251.

[16] Ravi Sandhu. SSL, PKI and Trust. 2002. Lecture notes.

[17] Carl Ellison. SPKI/SDSI and the Web of Trust. 4 April 2001. See `http://world.std.com/~cme/html/web.html`. Last visited 24th February 2004.

[18] Andre Årnes, Mike Just, Svein J. Knapskog, Steve Lloyd, Henk Meijer. Selecting Revocation Solutions for PKI, in Proceedings of the 1999 Symposium on Network and Distributed System Security.

[19] A. Arsenault and S. Turner. PKIX Roadmap. Internet Draft, "Work in progress, IETF PKIX working group", October 1999.

[20] Stefan A.Brands. Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy. Page 11. MIT Press, 1st edition August 28th, 2000.

[21] Silvio Micali. Efficient Certificate Revocation. Technical report, Massachusetts Institute of Technology, March 1996.

[22] ITU and ISO/IEC. Final Proposed Draft Amendment on Certificate Extensions. April 1999.

[23] Michael Myers, Rich Ankey, Ambarish Malpani, Slava Galperin, and Carlisle Adams. X.509 Internet Public Key Infrastructure: Online Certificate Status Protocol. IETF RFC2560, June 1999. See http://www.ietf.org/rfc/rfc2560.txt. Last visited 31th March 2003.

[24] RFC 1422 - A.2 Certificate Revocation List Syntax. February 1993

[25] Andre Årnes. Public Key Certificate Revocation Schemes. Queens University, Kingston, Ontario, Canada. February 2000.

[26] Petra Wohlmacher. Digital Certificates: A survey of Revocation Methods. University of Klagenfurt, Austria. See http://www.acm.org/sigs/sigmm/MM2000/ep/wohlmacher/. Last visited 22nd April 2003.

[27] Peter Gutmann. Everything you Never Wanted to Know about PKI but were Forced to Find Out. University of Auckland. See http://www.govis.org.nz/insecurity/pgutmann.pdf. Last visited 24th February 2004.

[28] R. Merkle. Secrecy, Authentication, and Public Key Systems. Ph.D. Dissertation, Department of Electrical Engineering, Stanford University, 1979.

[29] Paul C. Kocher. On Certificate Revocation and Validation. Chief Scientist, Valicert.

[30] Moni Naor and Kobi Nissim. Certificate Revocation and Certificate Update. Weizmann Institute of Science.

[31] Hiroaki Kikuchi, Kensuke Abe and Shohachiro Nakanishi. Performance Evaluation of Public-key Certificate Revocation System with Balanced Hash Tree. Tokai university.

[32] Patrick McDaniel and Aviel Rubin. A Response to "Can We Eliminate Certificate Revocation Lists?". See http://www.patrickmcdaniel.org/pubs/finc00_pres.pdf. Last visited 24th February 2004.

[33] M.Myers. Revocation: Options and Challenges. In Rafael Hirschfield, editor, *Financial Cryptography*, volume 1465, pages 165-171, Anguilla, British West Indies, February 1999. Springer.

[34] PKCS #5 - Password-Based Cryptography Standard, RSA Laboratories Technical Note, Version 2.0, March, 1999. See `http://www.rsasecurity.com/rsalabs/pkcs/pkcs-5/index.html`. Last visited 12th May 2003.

[35] Bruce Schneier, Security in the real world: How to Evaluate Security Technology, remarks from CSI's Netsec Conference in St. Louis, MO, on June 15th, 1999.

[36] Chris Thorn, Randomness and Entropy - An Introduction. 2003. See `http://www.giac.org/practical/GSEC/Chris_Thorn_GSEC.pdf`. Last visited 24th February 2004.

[37] Carl Ellison, Chris Hall, Randy Milbert and Bruce Schneier. Protecting Secret Keys with Personal Entropy, 28 October 1999. See `http://www.schneier.com/paper-personal-entropy.pdf`. Last visited 24th February 2004.

[38] A. Shamir, How to Share a Secret, *Comm. of the* ACM *22*, 11, Nov. 1979, pp. 612-613.

[39] D. N. Hoover and B. N. Kausik. Software Smart Cards via Cryptographic Camouflage. See `http://www.arcot.com/products/arcot_ieee.pdf`. Last visited 24th February 2004.

[40] PKCS#1: RSA Encryption Standard, RSA Laboratories. Technical Note, version 2.1, June 2002. See `http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1/index.html`. Last visited 16th May 2003.

[41] Ronald Ward, Secure Telecommunications, ECE 636. Survey of Cryptographic Smart Card Capabilities and Vulnerabilities.

[42] Tolga Kilicli. Smart Card HOWTO, revision 1.04 2001. See `http://en.tldp.org/HOWTO/Smart-Card-HOWTO`. Last visited 28th May 2003.

[43] Sergei P. Skorobogatov. Copy Protection in Modern Microcontrollers. See `http://www.cl.cam.ac.uk/~sps32/mcu_lock.htm`. Last visited 29th May 2003.

[44] W. Wen, T. Saito and F. Mizoguchi. Science University of Tokyo, Japan. The "Ex-employee" Attack on Certificate-based Authentication Protocols.

[45] T. Dierks, C. Allen. RFC 2246. The TLS Protocol Version 1.0. See http://rfc.sunsite.dk/rfc/rfc2246.html. Last visited 9th July 2003.

[46] Dug Song. University of Michigan. See http://www.naughty.monkey.org/dugsong. Last visited 11th July 2003.

[47] Atsuko Miyaji, Wu Wen and Seiichiro Hangai. Cryptography and Computer security. Review of Radio Science 2000-2002.

[48] See http://www.skandiabanken.no/SKBWeb/NO/Oversikt/pressemeldinger /pressemelding_030402.pdf. Last visited 21st July 2003.

[49] Ricky Sookermany, customer support in SkandiaBanken 21st July.

[50] See http://www.mozilla.org/projects/security/pki/psm/help_21/glossary.html. Last visited 24th July 2003.

[51] See http://www.skandiabanken.no/skbweb/No/ , questions and answers menu with the submenues technical and security. Last visited 28th July 2003.

[52] See http://www.edb.fellesdata.no/ Last visited 28th July 2003.

[53] Forvaltningsloven §13. See http://www.cfje.dk/cfje/Lovbasen.nsf/(ID)/LB00165243?Opendocument. Last visited 24th February 2004.

[54] See http://www.skatteetaten.no/personer/folkeregistrering/article.jhtml?articleID=14068. Last visited 31st July 2003.

[55] Monica Hjorth. "En kartlegging av nettb@nkkunders holdninger. Hovedfagsoppgave i Informasjonsvitenskap. Universitetet i Bergen. November 2002.

[56] See http://www.ssb.no/emner/02/02/10/fodte/tab-2003-04-30-01.html. Last visited 12th October 2003.

[57] See http://www.lasr.cs.ucla.edu/classes/239_1.spring03/slides/lecture2.pdf. Last visited 20th October 2003.

[58] See http://www.iroffer.org. Last visited October 22nd 2003.

[59] See http://www.ssb.no/emner/02/01/10/folkemengde/. Last visited October 22nd 2003. Prentice Hall 2002.

[60] Wade Trappe and Lawrence C. Washington. "Introduction to cryptography with coding theory".

[61] See http://www.cs.ucl.ac.uk/staff/F.AbdulRahman/docs/pgptrust.html. Last visited November 11th 2003.

[62] Joachim Biskup and Sandra Wortman, University of Dortmund, Information Systems and Security. Towards a Credential-Based Implementation of Compound Access Control Policies. November 11th, 2003.

[63] See http://www.openssl.org/. Last visited 19th December 2003.

[64] See http://www.faqs.org/rfcs/rfc3280.html. Last visited 5th January 2004.

[65] S. Farrell and R. Housley. "An Internet Attribute Certificate Profile for Authorization". April 2002. See http://www.ietf.org/rfc/rfc3281.txt. Last visited February 24th 2004.

[66] See http://www.ietf.org/rfc/rfc2527.html. Last visited 13th January 2004.

[67] Carlisle Adams and Steve Lloyd. Understanding PKI. 2003 by Pearson Education, Inc. Second Edition.

[68] See http://www.cl.cam.ac.uk/Research/Security/Trust-Register/. Last visited January 20th 2004.

[69] See http://www.simc-inc.org/archive9900/Feb00/ellison/sld001.htm. Last visited January 22th 2004.