University of Oslo
Department of Informatics

A Security Architecture for Monitoring a Nuclear Test-Ban Treaty

Henrik Grindal Bakken

Cand. Scient. Thesis

31st July 2003

Abstract

For a long time, an effort to achieve a comprehensive nuclear test-ban treaty banning all nuclear explosions world-wide has been made. The Comprehensive Nuclear-Test-Ban Treaty (CTBT) is the result of this effort.

A monitoring regime, which is to assist the participating countries in verifying compliance with the Treaty, is being set up by a preparatory commission. To protect the data from a global network of monitoring stations (the IMS network), digital signatures are applied, and a public key infrastructure (PKI) has been set up to accommodate key exchange.

In this thesis, we will describe the IMS network, the proposed security architecture, its protocols and the PKI. We also make a threat analysis for the system.

We have designed and implemented an application, `imsparse`, for one part of the security scheme: receiving and executing commands at the monitoring stations.

Finally, we have commented on the appropriateness of the proposed architecture, and how it meets the threats we defined.

Our conclusion is that most of the threats are dealt with by the implementation of this security scheme, but the physical securing of the sensors may cause problems.

Preface

This thesis is submitted to the Department of Informatics at the University of Oslo as part of my *Candidatus scientiarum* (Cand. scient.) degree.

Thanks

I would like to thank everyone who helped me finish this work. Many thanks to everyone at NORSAR for letting me work there, and in particular to Ulf and Jan for help. Thank you also to Gonzalo Perez and Edward Wokabi from the CTBTO for answering my questions.

Most of all, I wish to thank my tutor on this thesis, Leif Nilsen from UniK, without whom I could never have finished. And finally thanks to Marte for being helpful and very patient.

Abbreviations

A number of abbreviations will be used in this thesis, not all of them well known.

	CTBT specific
CRF	Central Recording Facility
CTBT	Comprehensive Nuclear-Test-Ban Treaty
CTBTO	Comprehensive Nuclear-Test-Ban Treaty Organisation
IDC	International Data Centre
IMS	International Monitoring System
NDC	National Data Centre
PTBT	Partial Test-Ban Treaty
PTS	Provisional Techincal Secretariat
SO	Station Operator
TS	Technical Sectretariat
	General
ASN.1	Abstract Syntax Notation Number One
AC	Access Control
ACL	Access Control List
CA	Certification Authority
C&C	Command and Control
CDP	CRL Distribution Point
CP	Certificate Policy
CRL	Certificate Revocation List
CSP	Critical Security Parameter
DN	Distinguishing Name
DOS	Denial of service
OS	Operating System
PKC	Public Key Cryptography
PKCS	Public Key Cryptography Standards
PKI	Public Key Infrastructure
RA	Registration Authority
URI	Uniform Resource Identifier

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Comprehensive Nuclear-Test-Ban Treaty

1.1.1 History

In order to continue the long-ongoing effort in the field of nuclear disarmament, the international community has set up a comprehensive treaty preventing further testing of all nuclear weapons. The Partial Test-Ban Treaty (PTBT) of 1963 prohibits surface, ocean and space testing and is already in effect. In 1992, several countries including the US, Russia, and the United Kingdom, have agreed to, and abode with, a moratorium to stop all nuclear tests.

In 1996, the Comprehensive Nuclear-Test-Ban Treaty (CTBT) was signed. This treaty bans *all* nuclear tests.

Although the Treaty text[Com94] is finalized, the Treaty has not yet entered into force. For the Treaty to do so, it is required that 44 states listed in the Treaty's Annex 2 ratify it. Currently, 41 of those states have signed the Treaty, but only 31 have ratified it. In addition to the Annex 2 states, there are 126 signatory states, and 71 ratifications.[1]

It is important to keep in mind that the PTBT *is* in effect, and has been for a long time. The difference between PTBT and CTBT is essentially the inclusion of underground tests, and thus the ban on all nuclear tests.

[1]The CTBTO web-site, `http://www.ctbto.org`, provides an up-to-date list of countries which have signed and ratified the Treaty, both Annex 2 countries and other member states.

1.1.2 Organisation

Before the Treaty enters into force, several preparations must be made. The most important, and most demanding, is the implementation of a verification scheme. To establish the necessary infrastructure and technological solutions, the Preparatory Commission for the Comprehensive Nuclear-Test-Ban Treaty (PrepCom) has been formed. Paragraph 1 of the Annex to the Treaty[Com94] specifying the mission for the Preparatory Commission:

Annex

1. There is hereby established the Preparatory Commission for the Comprehensive Nuclear-Test-Ban Treaty Organisation [...] for the purpose of carrying out the necessary preparations for the effective implementation of the Comprehensive Nuclear-Test-Ban Treaty, and for preparing for the first session of the Conference of the States Parties to that Treaty.

PrepCom consists of two organs: the plenary body composed of all signatory states, and the Provisional Technical Secretariat (PTS).

The two main targets for PrepCom, is to promote Treaty signature and ratification, and to establish a global verification regime to monitor compliance with the Treaty. The latter part is the objective of the PTS, and is the focus of this thesis.

When the Treaty enters into force, the Comprehensive Nuclear-Test-Ban Treaty Organisation (CTBTO) will be established (and the Preparatory Commission will cease to exist). The seat will be at the same facility as the Commission is located now, in Vienna, Republic of Austria. It will consist of three organs: the Conference of the States Parties, the Executive Council, and the Technical Secretariat (TS). The latter will replace the PTS. We will use the name PTS throughout the thesis, even though their tasks will be handled by the TS when it is established.

The Role of the Organisation It is important to note that the CTBTO's (future) role, is *not* to monitor compliance with the Treaty. As outlined in Article II of the Treaty[Com94]:

1. The States Parties hereby establish the Comprehensive Nuclear-Test-Ban Treaty Organisation (hereinafter referred to as "the Organisation") to achieve the object and purpose of this Treaty, to ensure the implementation of its provisions, including those for international verification of compliance with it, and to provide a forum for consultation and cooperation among States Parties.

The responsibility to act on evidence (or indications) of non-compliance lies with the participating nations. The mission of the CTBTO is to implement the necessary mechanisms.

1.1.3 Verification Regime

To monitor compliance with the Treaty, a verification regime is required. The goal of this verification scheme is to be able to *detect and locate* all nuclear explosions.

This goal, however, is impossible to truly reach. There will always be a lower threshold (with respect to magnitude of the explosion) under which the verification regime cannot reliably function. The problem of lowering—and agreeing on—this threshold is outside of the scope of this thesis.

The establishment of this regime is the primary task for the Preparatory Commission. According to the CTBTO web pages, the regime consists of four elements[Pre]:

An International Monitoring System A network of stations and laboratories monitoring the earth for evidence of nuclear activities is being deployed. It is called the International Monitoring System (IMS), and is discussed in detail later in this section and in chapter 4.

Consultation and Clarification Process If a participating state suspects non-compliance with the Treaty, it has the right to request a clarification on any relevant matter. Directions for such clarification are a part of the verification regime.

On-Site Inspections If clarifications made by the state subjected to suspicion of non-compliance are not satisfactory to the requesting state, and a suspected nuclear explosion is detected by a station of the monitoring system, any participating state has the right to request an on-site inspection.

Confidence-building Measures The confidence-building measures are two-fold:

1. To contribute to the timely resolution of any compliance concerns arising from possible misinterpretation of verification data relating to chemical explosions, such as, for example, large mining explosions; and
2. To assist in the calibration of stations that are part of the IMS.

Of these elements, this thesis concentrates on the former. The IMS network consists of 321 stations and 16 radionuclide laboratories[Com01b]. There are four kinds of stations: seismic, hydroacoustic, infrasound, and radionuclide. We will briefly describe the functionality of the latter three types in chapter 4, and give a more detailed description of the seismic stations.

The data is transferred from the stations to the *International Data Centre* (IDC), which is located in the CTBTO headquarters in Vienna. This transfer is either directly from the stations to the IDC or via a *National Data Centre* (NDC). All state parties will have access to the data from all stations through the IDC.

1.2 Security Problem

A problem with seismic data and underground tests is that the waves generated by an explosion do not travel as far as waves that travel in the atmosphere or in water. The effect is that to accurately detect and locate an underground explosion, stations in relatively close proximity to the event may be required.

This, in turn, means that proper detection in some cases may depend on data from the testing country's *own* stations. There is consensus that it is impossible to have a monitoring regime that relies on the honesty of the participants, so it is agreed that the sensor data must be authenticated upon reception. It is also accepted that the authentication method used must be adequate to prove to an arbiter that the data is authentic.

A trustworthy monitoring scheme for the CTBT intuitively depends on two critical factors:

- Secure gathering of the IMS data, and
- authenticated transfer of that data to the monitor.

It is not the goal of this thesis to say anything about the politics surrounding the Treaty and the nuclear disarmament struggle, but it is a fact that the lack of satisfactory, secure technology for data transfer is one of the reasons why underground test explosions were not included in the PTBT.

The surveillance techniques for detecting explosions under water, in the atmosphere, and in space are more than adequate, but in 1963, no acceptable solution for underground tests existed. In [Sim91], Gustavus Simmons sums up the difficulties faced in implementing a trustworthy monitoring regime. Although secure emplacement of sensors in sealed-off boreholes was feasible, the secure transfer of the data to other parties, was not. Substantial work was carried out at Sandia National Laboratories in the US to find a way to allow "mutually distrusting (and potentially deceitful) parties [...] to both trust a data acquisition system [...]".

The gathering of the data is mainly outside of the scope of this thesis, although some aspects of it will be addressed.

In addition to the transfer of the data from the sensors to the IDC, there is a need for communication the other way. Calibration and key management commands may be permitted from both the PTS and the station operator (SO). These commands will be able to greatly affect the operational status of the stations. It is therefore necessary to have a secure framework for proper authentication of the commands at the stations.

1.3 Proposed Security Solution

To guarantee the authenticity and integrity of the data from the stations, PrepCom has proposed to apply digital signatures (see chapter 2 for details on digital signatures) to them.

Several parties will be requesting data from the stations and verify their signatures, as well as performing remote maintenance and configuration. Therefore it has been decided that a Public Key Infrastructure (PKI) (see chapter 3) is to be established to accommodate key retrieval and verification.

The digital signatures are not the only means of security in the IMS system, though. When an earthquake, a volcanic eruption, or a similar physical event takes place, the signals that are generated will show up on sensors all over the world. When these events will happen is impossible to predict. This fact actually works as an important security mechanism. It is called *geophysical authentication*, and we will briefly come back to it in chapter 7.

1.4 Motives

To get an idea about why solid security measures must be deployed, we need to look at what motives potential attackers will have. A good beginning is to look at Article I of CTBT treaty text[Com94].

ARTICLE I BASIC OBLIGATIONS

1. Each State Party undertakes not to carry out any nuclear weapon test explosion or any other nuclear explosion, and to prohibit and prevent any such nuclear explosion at any place under its jurisdiction or control.

2. Each State Party undertakes, furthermore, to refrain from causing, encouraging, or in any way participating in the carrying out of any nuclear weapon test explosion or any other nuclear explosion.

The goal of the Treaty is to bring a stop to all nuclear test explosions. Our perspective is the security scheme set up to allow authenticated transfer of the sensor data from the stations to the IDC.

The main danger to this system is perceived to come from the participating nations. They agree on a common goal to stop the development of nuclear weapons, with a hope of a future without nuclear weapons altogether. They do not, however, trust each other to all abide with the rules laid down to achieve this goal, hence the need for a verification regime.

Although that verification regime is set up to prevent undetected nuclear explosions exclusively, it does bring other dangers. Other potential attack-situations arise with the possibilities tampering with the system provide.

A threat analysis for the verification regime is provided in chapter 5.

1.5 Scope of the Thesis

In this thesis, our contributions have four main focal points:

- A description of the IMS network, the IMS PKI, and the protocols used;
- An identification of the threats to the system;
- The design and implementation of an application for receiving command and control messages at the IMS stations;
- A brief analysis of the security architecture and how it meets the threats we have identified.

1.6 Use of Sources

Much of the system description in this thesis originate from documents that are internal to the PrepCom and not generally accessible. We have tried to quote from these sources in a way to provide the reader with the necessary information.

1.7 Organisation of the Thesis

This thesis is organised into four parts.

The first part will cover basic cryptography in chapter 2 and PKI in chapter 3. In addition to covering the technical background material, we will look at the historical aspects of the combination cryptography and nuclear test-ban treaties.

The second part will, in chapter 4, describe the International Monitoring System, including the security solution proposed by the PTS, before a threat analysis is made in chapter 5.

The third part, chapter 6, describes our work on implementing one part of the security architecture.

Finally, the fourth part is the analysis of the defense mechanisms proposed by the PTS and comment on the quality on the scheme (chapter 7).

We will then present our conclusion of the thesis.

Chapter 2

Cryptography

Although the aim of this thesis is to describe and analyse a more complex system and not the cryptographic algorithms used, an introduction to the building blocks is required.

2.1 Security Services

Usually, when the use of cryptography is required in a system, it is not really the *cryptography* itself that is in demand, but the *information security services* it can provide. What is required is not *encryption*, but *confidentiality*; not a *digital signature*, but a way to ensure *authenticity*. Information security has many uses, and cryptography is a key building block in many of them. We describe some important security services in this section.

2.1.1 Confidentiality

The oldest and most basic application of cryptography is that of *confidentiality* (or secrecy). The need for two or more entities to communicate over a channel which is vulnerable to tapping is often called upon.

In the IMS network, there is little need for confidentiality. As we shall see later in section 2.6, one of the most demanding tasks has been to *avoid* the use of secrecy, but still achieve authentication of the data.

2.1.2 Authentication

Although the insecure nature of computer networks (and in particular the Internet) often commands a great need for confidentiality, the need for trustworthy identification and authentication is possibly even greater. Further, cryptographic authentication channels are very often used in conjunction with secrecy channels.

In [DH76], Diffie and Hellman explain the need for proper authentication: *"Authentication is at the heart of any system involving contracts and billing. Without it, business cannot function. Current electronic authentication systems cannot meet the need for a purely digital, unforgeable, message dependant signature. They provide protection from third party forgeries, but do not protect against disputes between transmitter and receiver."*

We can separate authentication into two cases: Entity authentication and message authentication.

Entity Authentication The objective of *entity authentication* or *identification* is to allow one entity to assure itself of the identity of another entity or to let one entity identify itself to another. An important requirement is that the material used to identify entity A can not later be re-used by another entity to impersonate A. A consequence of this requirement is that the entity requesting to be identified must take an active part in the identification process. An effect of this again, is that entity authentication is real-time.

Based on Handbook of Applied Cryptography[MvOV97], we list the following requirements for an entity authentication scheme:

- Ability to enable an entity A to be properly identified by another entity B, while
- B must be unable to re-use the material from the identification of A to successfully impersonate A to C, a third party; and
- C, having observed the identification process between A and B must, even after observing a large number of identification processes, be unable to successfully impersonate A to any entity, D.
- The process must be feasible and efficient to perform, both for A and B.

It is common that no otherwise meaningful information is conveyed between A and B as part of an entity authentication process.

Message Authentication In [MvOV97], *message authentication* (or *data origin authentication*) is referred to as a type of authentication identifying the source of the specific data. [MvOV97] also defines *data integrity* as the property that data has not been (in an unauthorized manner) altered since the time of creation or modification by an authorized source. The inseparability of the two properties is stressed, as a message with its origin authenticated, but which is since then altered by an unauthorized entity, no longer can be said to originate from the original source. Conversely, if the source can not be determined, there is no foundation for integrity.

In this thesis, we will consider message authentication to be both the identification of the source of the message as well as its integrity.

To provide message authentication, we need a function working on an arbitrary message and some information identifying the author. The function must be efficient to compute, both producing the authentication and verifying its authenticity. We require that the function allow verifiers to assure themselves that the following hold:

- The message originates from the indicated sender;
- The message has not been altered after leaving the sender;
- Both the production and verification of the authentication data are efficient.

With entity authentication, we stressed that it must be impossible for an entity C to observe and copy the material used by A to identify itself to B and later (successfully) re-use that material to identify itself as A. In the case of message authentication, this does not apply. Due to the nature of digital data, it will always be possible for C to obtain a copy of a message from A including the authentication material, and re-use that to later certify that the *same message* originates from A. In this case, C would only act as a temporary storage for A's message. We can say that while entity authentication is valid only for the instance it is directly applied to, message authentication is not limited in time.[1]

What C *must* be unable to do in a good message authentication scheme, is to use a message authenticated by A (or many such messages) to produce a *different* message, which is accepted by B as authenticated by A.

In the IMS network, the primary authentication demands will be on message authentication; both authentication of commands to stations and authentication of data from the stations.

2.1.3 Non-repudiation

A service tightly bound to (message) authentication is *non-repudiation.* With message authentication, a recipient can be convinced of the integrity and

[1]The message authenticator can of course include a time when it ceases to be valid, but in general, this is not the case.

source of a message. There are many situations where the recipient also would like to be able to later *prove* (to an arbiter or another third party) that the sender actually sent this message.

We say that a mechanism providing non-repudiation has the property that the authorized originator of an authenticated message is unable to later deny having authenticated the message.

A standard example on non-repudiation from the literature is a stock broker acting on behalf of a client. If the client sends the broker an authenticated order to purchase stocks in a company and the broker obeys, then in the event of the company's stock going down, chances are that the client will accuse the broker of acting without his consent, thereby not being responsible for the loss. If the authentication mechanism on the order provides non-repudiation, the broker can use that to prove to a neutral arbiter that the client really did place the order.

2.1.4 Access Control

Access control means controlling which entities (users, computer programs, etc.) are allowed to access (read, write, execute) which applications, services, or data. Very often, access control is the ultimate goal for computer security applications.

Implementing access control always starts with entity or data origin authentication. This can involve determining the identity of the requester or verifying some credentials.

For each request for access, the access control system can give an answer of yes or no. Every time the entity requests access to resources, the system verifies and, if applicable, grants access.

Later in this chapter, we will describe two different approaches to implement access control. One is based on issuing credentials to users which prove their right to access. The other is using lists, located at the access-granting host, where the identification of a requesting user is looked up.

Granularity

When discussing access control, an important issue is *granularity.* With granularity, we mean at how detailed a level the access control is applied.

In every system utilizing access control, the "upper level" would likely be simply access to the system as a whole. A second level can be controlling which files, services, or applications the user are allowed to access. A third level will be what the user is allowed to do with these files, services, and applications (e.g. read, write, execute). This is a level of access control normally found in an operating system.

It is not necessarily the lowest level imaginable, though. There can for example be limitations on the operations allowed within a specific application, or restrictions on allowing the user to alter these permissions for himself or others.

2.2 Cryptographic Protocols

What characterizes a cryptographic *protocol*, as opposed to an algorithm, is that it involves two or more parties, is distributed, and is designed to solve a specified problem rather than to perform a single task.

A cryptographic protocol is a series of steps to be taken for each participant. These steps should be precisely defined and should deterministically specify what each participant should do at any given time of the protocol.

2.3 Cryptographic Primitives

There are several families of cryptographic primitives, each with its own area of use. We will limit ourselves to the following:

- Symmetric encryption algorithms
- Asymmetric encryption algorithms
- Cryptographic hash algorithms
- Message authentication codes
- Digital signature algorithms

In each of these families, there exist a multitude of algorithms. For each category, there are many algorithms we consider to be secure, but also a lot of already broken ones. A factor which will always come up when discussing the strength of an algorithm, is key length[2]. Although a long key-length will never be a guarantee for quality, a too short key will render any otherwise good algorithm unsafe due to simple “brute force” attacks.

[2]For hash algorithms, there are no keys, but the length of the hash is significant.

2.3.1 Symmetric Encryption Algorithms

A symmetric cryptoalgorithm, also known as a *one-key* or *standard* cryptoalgorithm, is an algorithm where the sender and receiver(s) share one secret key, K. Actually, it is somewhat misleading to talk about *one* symmetric algorithm, there are usually two; an encryption algorithm e_K and a decryption algorithm d_K. We will refer to such a pair of algorithms as one algorithm (this also goes for digital signature algorithms and public key cryptoalgorithms).

The one property we demand from every symmetric algorithm is that it satisfies $d_K(e_K(x)) = x$. If you encrypt something and decrypt the result with the correspondig decryption algorithm, you should get the original value back. In addition, it is obvious that finding K, given some ciphertext, or pairs of corresponding plaintext/ciphertext, should be difficult. Finding the plaintext from the ciphertext without access to K must of course also be difficult.

Block Ciphers and Stream Ciphers

There are two fundamentally different types of symmetric cryptoalgorithms, *block ciphers* and *stream ciphers*.

With a block cipher, each block (the size of the block varies with the algorithm) of plaintext data is encrypted using the same key, K, to obtain the ciphertext. A stream cipher encrypts each character (which is usually a single binary digit) of the plaintext with a different key. The keys are obtained successively from a generated *keystream*.

A distinction between the two types is that while encryption and decryption of a ciphertext block using a block cipher depend only on the key and the ciphertext[3], encryption and decryption of a character using a stream cipher also depend on the internal *state* of the algorithm.

[MvOV97] outlines the main use of stream ciphers with three advantages over block ciphers: One is that they have a very low error propagation, another that they allow each character (bit) to be individually processed on reception as well as permitting limited or no buffering. The third advantage is their speed; stream ciphers tend to operate faster than block ciphers. The last point is a truth with modifications, though. With stream ciphers, very efficient encryption and decryption is possible, but the setup of the keystream may take some time. For example with rapid key change, this overhead can be problematic.

[3]This only applies when the block cipher is used in Electronic Codebook Mode. It should be noted that block ciphers can be used in other modes to mimic the operation of stream ciphers. Details on this in the next section.

Block ciphers are still more widely used than stream ciphers. The primary reason being their versatility. Another important advantage for block ciphers is that they do not require an initialization when the key is changed.

Besides offering strong confidentiality, block ciphers are used in construction of stream ciphers, pseudorandom number generators, MACs, hash functions, and even digital signature schemes (see "Modes of Operation", below).

There are several well-known and presumed strong symmetric algorithms in use today. The Data Encryption Standard (DES)[Nat77], has been the most widely deployed since its arrival in 1977. DES was an official US standard until its replacement by the Advanced Encryption Standard (AES)[Nat01a]. Despite the fact that DES is still in use, it is not regarded as secure, due to its short key-length (56 bit).

Modes of Operation

Originally developed for DES, the four *modes of operation*[Nat80] can be applied to any block cipher: *electronic codebook mode* (ECB), *cipherblock chaining mode* (CBC), *cipher feedback mode* (CFB), and *output feedback mode* (OFB). The modes define four different ways to apply a block cipher to achieve different capabilities.

ECB is the straight-forward block cipher encryption mode. The plaintext is divided into n-bit blocks, where n depends on the algorithm. Each block is encrypted, using the same key, into its corresponding ciphertext block. ECB is simple, but has at least two major weaknesses. The first, and biggest, is that an active attacker can rearrange the order of entire blocks, retransmit blocks, or remove blocks. The second problem is that patterns in the plaintext are not properly disguised. Two similar blocks of plaintext produce the same ciphertext when the same key is used.

A remedy to both these deficiencies is the CBC mode. Each plaintext block is bitwise added modulo 2 (XOR) to the previous *ciphertext* block prior to encryption. With this simple method, both the reordering attacks and the pattern disclosure are prevented. Additionally, a block cipher used in CBC mode can be used to provide message authentication. See section 2.3.4 on MACs.

The two last modes for block ciphers, CFB and OFB are very different. Their objective is to allow a block cipher to resemble a stream cipher.

2.3.2 Asymmetric Encryption Algorithms

The symmetric cryptoalgorithms are strong and very efficient, but have a major flaw: the need for a secret channel to establish the key. In 1976, Whit

Diffie and Martin Hellman published their famous paper, "New Directions in Cryptography"[DH76], in which they proposed a fundamentally new idea for a cryptographic algorithm: The public key cryptoalgorithm.

In a public key cryptography (PKC), there are *two* keys used: one for encryption and one for decryption. The encryption key, K_p, is *public* and (presumed) known by everyone, the decryption key, K_s, is *secret* and known only by the owner. If Alice wants to send an encrypted message to Bob, she can encrypt the message using Bob's public key. Bob (and only Bob) can then use his secret key to decrypt the message.

In standard one-key cryptography, a concatenation of complex functions are normally used to mix the key and the plaintext to create the ciphertext. In PKC, a special brand of mathematical functions are used: *One-way trapdoor functions.*

A function suitable for an asymmetric encryption algorithm must meet the following requirements:

- $d_{K_s}(e_{K_p}(x)) = x$ for each K_s, K_p key-pair;
- Both $d_{K_s}(y)$ (given knowledge of K_s) and $e_{K_p}(x)$ must be efficient to compute for all allowed values of x and y;
- It must be infeasible to find x from $e_{K_p}(x)$ without knowledge of the corresponding K_s;
- It must be infeasible to find K_s from K_p.

The most well-known public key cryptoalgorithms include RSA[RSA78], the first one which was publicly known, and the El-Gamal family[ElG85].

RSA is based on the well-studied problem of integer factorization. El-Gamal was originally based on discrete logarithm over a finite field, but the same general idea has also been used for discrete logarithms over elliptic curves.

A major problem with this form of cryptography is that it is far slower than regular symmetric cryptography. Even with todays fast computers, encrypting and decrypting is too slow for large amounts of data. The general solution is to use an asymmetric technique to establish a secret *session key*, and subsequently use symmetric encryption for the rest of the traffic.

A way of agreeing on this session key, as suggested by Diffie and Hellman in their article, is a *key exchange protocol*. The objective of a key exchange protocol is not to encrypt anything, but to facilitate for the two communicating parties to both obtain knowledge of a shared key, and to prevent any attackers from getting that knowledge.

The Diffie-Hellman key exchange protocol is one such protocol, while another way to solve the problem is to use a generic asymmetric encryption algorithm to encrypt the session key. This approach is called a *hybrid cryptosystem.*

2.3.3 Cryptographic Hash Functions

A hash function, or message digest function, is a function that maps an arbitrarily long message into one of a fixed size.

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

for some l.

Since a hash function has an infinite domain and a limited (although usually rather large) range, it is obvious (follows immediately from the pigeonhole principle) that a hash function will have *collisions*, ie

$$f(x) = f(y) \quad \text{where} \quad x \neq y.$$

For a regular hash function, it is only required that collisions occur relatively rarely, and that a change in x with high probability will change $f(x)$.

With *cryptographic hash functions*, this is not the case. Among other uses, they are important in most digital signature schemes. Signatures are not made on the data being "signed", but actually on the hash of the data (details on digital signatures in 2.3.5). It is then obvious that if it is feasible to find a $y \neq x$ such that $f(y) = f(x)$, then it is feasible to break the signature scheme. If Bob has signed x, the same signature is also valid on y. This leads to the following requirements on all (strong) cryptographic hash functions:

For hash algorithm h, we require that

- it is computationally infeasible to find x and x', $x' \neq x$ such that $h(x) = h(x')$, and that
- it is computationally infeasible to find, given $y = h(x)$, any x' such that $h(x') = y$ without knowledge of x.

A hash function which meets these requirements is called a *one-way, strongly collision-free hash function*[Sti95].

Cryptographic hash functions have two primary uses: As a checksum, to make sure there are no (accidental[4]) changes to a message or file. Secondly, in digital signature schemes.

Several cryptographic hash functions have been standardised. MD5[Riv92] and SHA-1[Nat95] are among the most used algorithms. MD5 creates a 128-bit hash, while SHA-1, which is a US standard, has 160-bit hashes. There is also a variant of SHA-1, called SHA-256, with longer hashes.

[4]Hash functions are also frequently used to "verify" the integrity of source code for software downloaded over the Internet. The idea is that you should obtain the checksum from another source than the code itself, and thereby make forgery a far more complicated matter.

2.3.4 Message Authentication Code

In Handbook of Applied Cryptography[MvOV97], a *message authentication code (MAC)*, also called a *keyed hash function*, is defined as a function h_k depending on two input parameters: a secret key and a message. A good MAC must meet the following three requirements:

1. It must be easy to compute;
2. It must take input of arbitrary length;
3. It must be difficult to forge.

The first property says that for any input message m and any key k, $h_k(m)$ should be easy (and quick) to compute. The second requires that $h_k(m)$ must be possible to compute on every possible m. Finally, we require that without knowledge of k, it should be infeasible to compute $h_k(m_j)$ regardless of the number of $m_i, h_k(m_i)$ pairs observed (where $m_j \neq m_i$ for all observed m_i).

MACs are, as the name implies, used to provide authentication of messages. This is achieved by having two or more entities share a secret key. A message authenticated by a MAC produced by that key, can be applied to convince the involved entities that the message originates from someone or something with access to the secret key, and that it has not subsequently been replaced or altered.

The most common way to implement MACs is to use a regular single key block cipher in CBC mode, and use the final block as a MAC.

An alternative approach for creating a MAC on a message is to use a cryptographic hash function, and to create the digest of both the message and the secret key k. A widely used example of this technique is HMAC[KBC97].

While MACs have many important fields of use, and are subject to much research today, they have limitations:

- A MAC can only be verified by entities in possession of the private key used to generate the MAC.
- Further, a MAC can never provide non-repudiation since all parties able to verify its validity will at the same time be able to produce it. In other words, if Alice composes a message, creates a MAC to provide authenticity, and sends the message and MAC to Bob, she can later (to a third party) deny having sent the message and claim *Bob* created it, and that *he* made the MAC. There is nothing Bob can do to disavow this claim, as he *is* able to create an undetectable forgery.

2.3.5 Digital Signature Algorithms

As we have seen, (symmetric key) MACs provide message authentication, but they have limitations.

Another technique used to implement message authentication, is a *digital signature.*

[MvOV97] describes a digital signature as

> A *digital signature* of a message is a number dependent on some secret known only to the signer, and, additionally, on the content of the message being signed. Signatures must be verifiable; if a dispute arises as to whether a party signed a document (caused by either a lying signer trying to *repudiate* a signature it did create, or a fraudulent claimant), an unbiased third party should be able to resolve the matter equitably, without requiring access to the signer's secret information (private key).

The two major distinctions between a MAC and a digital signature are the opportunity for everyone with access to the public key to verify a signature, and the ability to provide non-repudiation.

A *digital signature scheme* is referred to as the combination of a signature creation algorithm, a verification algorithm, and, if used, a cryptographic hash function.

A digital signature scheme requires a *key-pair*: a *signing key*, which is known to the signing party only, and a *verification key*, which is required to be known by everyone who want to verify the signature.

[MvOV97] mentions two types of digital signature schemes: with message recovery and with appendix. A digital signature scheme with message recovery is a signature from which the entire (signed) message can be recovered. In other words, the message is part of the signature. This is typically used for shorter messages only, and we will not consider them further. Digital signature schemes with appendix, on the other hand, means that the message must be input as a separate argument to the verification function.

Signature algorithms with appendix usually accept fixed-length input only. A signature scheme able to sign fixed-length messages only in unacceptable, though. The solution is to apply a cryptographic hash function to the data before generating the signature. Instead of signing the document, we sign the hash of the document. See figure 2.1 for an illustration of a typical digital signature scheme.

Because of this problem, it is important that only signatures on a small subset of the possible messages are accepted. A high level of redundancy in the acceptable messages will reduce the risk that an existential forgery

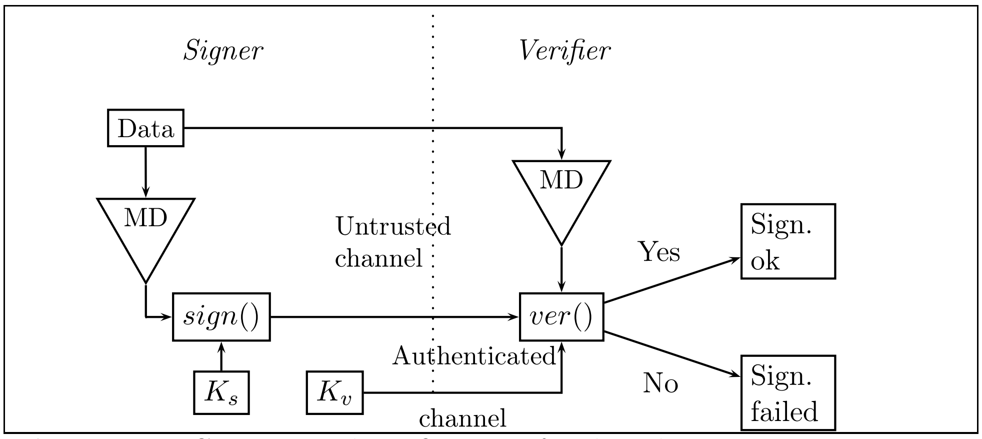

Figure 2.1: Creation and verification of a digital signature. Here, K_s is the signing key, K_v the verification key. The MD boxes are message digest (hash) functions, *sign*() is a function taking a message digest and a signing key and produces a digital signature. *ver*() is a function taking a message digest, a digital signature, and a verification key and returns "Yes" or "No" depending on whether the signature was accepted.

can be successful. When the hash of the message is signed, however, the situation is different. The attacker will have a forged signature on a hash, but he has no way of finding a *message* that corresponds to the signature.

Using a hash is not without problems. There are two important things to consider with the hash function. Firstly, all parties (that is, the signer and all potential verifiers) must know which hash function to use. Since the signature is applied to the hash, verification using another hash function (and therefore another hash), will fail. The consequence is that a digital signature scheme must specify not only the algorithm for creating the signature itself, but also the hash function. The second problem is that if an adversary can find another message that has the same hash as the one a signature is applied to (recall that every hash function will by definition have collisions), the signature will be valid also on the second document. The result is that a digital signature scheme is no stronger than its hash function.[5]

Security of a Digital Signature Scheme

When discussing the security of any security mechanism, it is important to first describe what an adversary would want to obtain by breaking it. Regarding digital signature schemes, the answer is simple; *to create a forgery.*

[5]At least theoretically, a signature scheme is no stronger than its hash function. In practice, exploiting a hash collision is more difficult than exploiting a weakness in the signature creation itself, but there is at least no question that the security of a digital signature scheme depends on the hash function.

A much cited source for the different types of forgeries an attacker can create, is an article[GMR88] by Goldwasser, Micali, and Rivest. In the article, they list four different kinds of forgeries:

A Total Break The attacker is able to calculate the user's private key.

Universal Forgery The attacker is able to find a way to produce undetectable forgeries on arbitrary messages.

Selective Forgery The enemy is able to produce a forged signature on a message chosen prior to the attack by the enemy.

Existential Forgery The enemy is able to forge a signature on at least one message. The contents of the message is uninfluenced by the attacker.

A total break or a universal forgery allows an attacker to sign any message, and a signature scheme prone to such an attack is broken. It is, however, normal to consider the security of a scheme in the context that an attacker is allowed to have messages of his choosing signed (a chosen message attack). In this scenario, it is important that also a selective forgery is difficult.

The technique used in most signature schemes to prevent successful selective or existential forgeries is the hash function. A selective will typically allow an attacker to obtain a forgery on the message he wants by having the private key owner sign other messages, and exploit a mathematical or structural property of the signature function.

When hash functions are used, the attacker will only be able to get a valid signature on a hash, without knowing a message corresponding to the hash. When the hash function is strong, the attack is unsuccessful.

Deterministic vs. Non-deterministic Algorithms

We distinguish between deterministic and non-deterministic digital signature algorithms. We say that an algorithm is deterministic if for each key-pair (k_s, k_v) and each message m there exists only one single signature s such that $ver_{k_v}(m, s)$ is true. Conversely, if an algorithm is non-deterministic, several such signatures $s_1 \ldots$ exist.

Well-known Digital Signature Schemes

The two most common digital signature schemes used today are the Digital Signature Scheme (DSS)[Nat94] and one built on the RSA algorithm[Kal98a]. The DSS uses the Digital Signature Algorithm (DSA), which is based on the El-Gamal signature scheme[ElG85]. We refer to the standards for details on the RSA scheme, and describe DSS in detail below.

An interesting aspect, and one in which the two algorithms differ, is that while RSA (as the only digital signature algorithm to the author's knowledge) encrypts the message digest with the secret key, DSA produces a pair of numbers. This pair, the public key of the signer, and the digest of the signed data, can later be used to verify the signature. Verifying an RSA signature constitutes simply decrypting the encrypted digest with the signer's public key and comparing the decrypt to a self-produced digest of the data.[6]

Another difference between the two schemes is that RSA is a deterministic algorithm, while DSA is non-deterministic.

Digital Signature Standard

The Digital Signature Standard (DSS) was proposed by the U.S. National Institute of Standards and Technology (NIST) in 1991. It uses the Digital Signature Algorithm (DSA) as its signature algorithm. DSA is a variant of the El-Gamal scheme[ElG85]. In 1994, it was adopted as a Federal Information Processing Standard (FIPS 186)[Nat94]. The DSS specifies SHA-1[Nat95] as the hashing algorithm.

Key Generation A DSS key-pair consists of a private key a, and a public key (p, q, α, y). They are created as follows:

1. Select a random 160-bit prime q.
2. Select a random prime p, $2^{511} < p < 2^{1024}$ where $q|(p-1)$.
3. Select a generator α of the unique cyclic group of order q in $\mathbb{Z}_p^*$.
4. Select at random a, $1 \leq a \leq (q-1)$.
5. Compute $y = \alpha^a \mod p$.

Here, p, q, and α are what is referred to as *system parameters*. In a network (for example a PKI, see chapter 3), it is possible to let all involved entities use the same values for p, q, and α.

DSS allows a key length (the length of p) of 512 to 1024 bits, inclusive. The key length must be a multiplicative of 64.

[6]It should be mentioned that in the literature, the creation of the signature is often referred to as "encrypting" the digest with the private key, and the key-pair is referred to as encryption and decryption keys. An example of this practice is RFC 2315[Kal98b], in which PKCS #7 is defined. In the case of DSA (and, as mentioned, all other widely used algorithms except RSA), this is logically wrong. No *ciphertext* is ever made, and there is no encryption key, only a signing key and a verification key.

Signature Creation Assume that entity A wants to sign the message m, using his private key a and public key (p, q, α, y). Let h be the SHA-1 hashing algorithm.

1. Select a random number k, $0 < k < q$.
2. Compute $r = (\alpha^k \bmod p) \mod q$.
3. Compute $s = k^{-1}(h(m) + xr) \mod q$.

Now, (r, s) is A's signature on m.

Signature Verification Assume that entity B has obtained A's public key (p, q, α, y). Again, let h be the SHA-1 hashing algorithm. Further, let r', s', m' be the versions of r, s, m received by B.

B's steps to verify the validity of the signature is as follows:

1. Verify that $0 < r' < q$ and that $0 < s' < q$. Reject the signature if either condition fails.
2. Compute $w = s^{-1} \mod q$.
3. Compute $u1 = (h(m') * w) \mod q$.
4. Compute $u2 = (wr') \mod q$.
5. Compute $v = ((\alpha^{u1} y^{u2}) \mod p) \mod q$. If $v = r'$, accept A's signature on m, otherwise reject it.

The appendix to the DSS standard[Nat94] gives a proof showing that $v = r'$ when $m = m'$, $s = s'$, and $r = r'$ (i.e. when the message and signature from A are unchanged).

Security of DSS The El-Gamal scheme, and hence also DSS, is based on the problem of computing logarithms in $\mathbb{Z}_p^*$. A powerful algorithm called the *index calculus method*[COS86] apply to this particular problem.

Performance of DSS The performance of DSS is a function of the performance of the two components: SHA-1 and DSA.

The speed of a SHA-1 computation depends on the size of the input.

The parts of the DSA operation that are most significant with regard to performance are the modular exponentiations. For signature creation, there is one ($\alpha^k \bmod p$), and for verification two ($g^{u1} \bmod p$ and $y^{u2} \bmod p$). These require on average each 240 modular multiplications[MvOV97], or 240 for creation and 480 for verification.

2.4 Cryptanalysis

When discussing the quality of a cryptographic algorithm, the resources available to the attacker, we call him Oscar, is an important factor. With resources, we mean both the time and calculating power available, how much he knows about the cryptographic material he wants to break, and his ability to monitor or alter the data involved.

When one analyses the strength of a cryptoalgorithm, its intended use is obviously important, particularly the time frame for which security of the encryption is required. For example, in an interactive login session over the Internet, both confidentiality and authentication of the data is important. Since passwords or other sensitive information may be passed over the insecure channel, it is required that the encryption is strong enough to resist attacks for a long period of time. The authentication mechanisms, on the other hand, is of no concern immediately after the session is over.

Generally, the algorithms in wide-spread use are thoroughly tested and are (believed to be) able to withstand all attacks from the most powerful of attackers for a long time.

Kerckhoffs' Principle In 1883, Auguste Kerckhoffs published a paper titled "La cryptographie militaire"[Ker83] in which he set forth several principles he claimed should apply to every cryptosystem. The most famous of these principles state that the system itself must not require secrecy, and that it can be stolen by the enemy without causing trouble. Notice that he does not say that the system should be public, only that the security of the system should not rely on it being kept secret.

Active and Passive Attacks We always assume that the attacker has access to read the (encrypted, signed, etc) data he is trying to break. Whether he is also able to alter that data on its way from Alice to Bob, is another question. We say that if Oscar can perform an attack after only observation of traffic, it is a *passive attack*. If he edits, replaces, removes, or inserts data between Alice and Bob, he performs an *active attack*. Active attacks require more effort, and often more resources, but an algorithm will not be considered secure if it cannot withstand an active attack.

Available Data Attacks on cryptographic algorithms are often categorised by the amount of data the attacker has access to. It is common to use the following categories:

Known ciphertext The attacker's knowledge is limited to observed ciphertext. A cryptoalgorithm which does not withstand a known ciphertext attack is considered broken.

Known plaintext The attacker has access to plaintext-ciphertext pairs.

Chosen plaintext The attacker is able to have plaintext of his choosing encrypted with the key used by Alice and Bob, and to obtain the corresponding ciphertext. This class can be extended to allow the attacker to choose plaintext depending on the result of the previous encryptions. It is then called an *adaptive chosen plaintext* attack.

Chosen ciphertext In the context of public key cryptoalgorithms (discussed in section 2.3.2), arbitrary pairs of corresponding plaintext-ciphertext pairs of the attacker's choosing must be considered to be available since the encryption key is public. A scenario resembling chosen plaintext for public key algorithms is chosen ciphertext. The attacker is allowed to have ciphertext of his choice decrypted by the secret decryption key, and to obtain the corresponding plaintext.

In all cases, we assume the attacker has access to a substantial amount of the specified data.

We will now look at a few specific types of attack. It is important to separate between attacks where the deficiency is in the algorithm itself and attacks where the implementation is at fault. Typically, a faulty implementation is relatively easy to fix (and often affects only one of many implementations), while a problem with the algorithm itself means the algorithm must be improved or discarded. We start with a special case.

2.4.1 Brute-force Attack

A brute-force attack is the simplest form of attack possible; the attacker tries all possible keys. For symmetric key cryptoalgorithms, it is a goal that brute-force should be the attacker's best choice (or at least of the same order of magnitude as the best). Currently, 128 bits key-length is seen as adequate to resist brute-force attacks for all foreseeable future. As we have seen, DES uses only 56 bit keys, leaving DES vulnerable to even attackers of a relatively modest resource level.

2.4.2 Protocol Failures

As we saw in 2.2, it is often required to combine cryptographic primitives through a well-defined series of steps to achieve the desired goal. However, while cryptographic protocols can provide good solutions to hard problems, they can also introduce security problems not present in the primitives upon which they are based.

Examples of protocol failures include replay attacks and man-in-the-middle attacks.

Replay Attack In a replay attack, Oscar intercepts and saves a valid, signed message from an Alice to Bob. Oscar then, at a later time, resends the message to Bob, who accepts it as valid, and believes it comes from Alice. This is an attack which is impossible to guard against by digital signatures alone; after all, Alice's signature is still valid.

There are many situations where such an attack can be dangerous. We will describe one in chapter 7. To prevent replay attacks, a protocol usually includes either a sequence number, random nonce, time stamp, or some other means for the receiver to detect that the received message is out of order.

Man-in-the-middle Attack A man-in-the-middle attack is an attack where Oscar observes and possibly modifies communication between Alice and Bob without either of them learning of his presence. A good example of such an attack is in public key cryptography.

Assume Alice is going to send an encrypted message to Bob. She does not currently have Bob's public key K_{BE}, so she tells him to send it to her. Bob does as he is asked, but Oscar manages to intercept the key and change it in transit, so Alice ends up with *Oscar's* public key, K_{OE}.

Alice then encrypts the data (or a session key; this does not change the argument) with K_{OE}, which she believes is Bob's key. Oscar again intercepts the traffic, decrypts the contents (he, obviously, has access to his own private key, K_{OD}). He then re-encrypts the message (possibly after modifying it) with K_{BE}. Bob receives this message, decrypts it with K_{BD}, and has no knowledge of Oscar's interception.

2.4.3 Implementation Deficiencies

Many cryptographic protocols with sound design and using safe algorithms have been compromised because of bugs in the implementation (or one of many implementations). We separate between two different types of implementation deficiencies: *side-channel attacks* and *bugs*.

A side-channel attack is a problem where information is leaked from the execution of the cryptographic process.

A typical side-channel attack is a *timing attack*. The idea in a timing attack is that through timing cryptographic processes, it is possible to extract secrets from the system performing them. Dan Boneh and David Brumley recently presented a functional timing attack against web servers running OpenSSL[BB03].

An example of a bug in an implementation is a bad random number generator. If the effective key-space is significantly smaller than the "real" key-space, a brute-force attack may become feasible.

2.5 Non-cryptographic Security Mechanisms

So far, our focus has been entirely on the cryptography and its application to information security. However, security is not limited to cryptography. In this section, we will address some security mechanisms that are *not* cryptographic in nature.

2.5.1 Logging and Auditing

Good logging and auditing procedures are important parts of most security systems.

Logs record events or statistics in a system. These events can include user logins and logouts, access requests to services, material or applications, change of permissions, changes in system operational status, etc.

Auditing is analysing the logs to detect abnormalities. Auditing is both a tool (log analyser) and a policy. The goal of auditing is to extract the important parts out of the logs. Logs are often sizable and manual processing is impossible. With an auditing tool, one can define the events that are considered normal and events that must be reported (auditable events). Defining these events require a knowledge about the system; what the security goals are and how they may be violated. This in turn dictates what information it is necessary to log.

It is important that critical logs are securely stored. Logs stored on a compromised host can no longer be trusted to be correct.

2.5.2 Access Control Lists

A common way to implement access control to applications or data is to maintain access control lists (ACLs). ACLs are tables with a row per user and a column for each privilege the user may or may not have access to. Like all access control, ACL lookup depend on proper entity authentication (of the user). ACL usage, however, also rely on message authentication (of the ACL).

We will discuss several alternatives for using cryptographic techniques to protect the integrity of the ACL in chapter 6.

When we use ACLs for access control, what happens is that the entity requesting access identifies itself to the host, the host looks the entity up in its lists, and accepts the entity's request if it is valid. This is analogous to a guard having a list of all the people who are allowed entrance. The guard asks people to identify themselves, if happy with the identification, he checks his list and accepts entrance if the person is on the list.

2.5.3 Certificate-based Access Control

Certificate-based access control differs fundamentally from ACLs. Here, each entity has a proof of identification (the certificate; we will address this in detail in chapter 3). This "ID card" not only contains the entity's identification, but also its credentials. The analogy to this approach would be that the guard does *not* have a list, but that every person allowed in is issued a membership card (or, rather, have stamps in their ID cards). The guard would now accept for admission only those who could produce a valid membership card.

One significant difference between these two methods, is that with an ACL, the host always controls exactly who have access. With certificate-based AC, there are two modifications to that scenario:

- The host does not necessarily decide who has access, and
- the host does not necessarily at all times know who has access.

An advantage with using certificates is that the need for maintaining the ACL is relieved from the host.

One potentially both useful and difficult aspect with a certificate-based system is the one of interoperability. It is inconvenient if a different certificate must be issued to every user for each resource he needs access to.

In chapter 6, we will describe how certificate-based access control can be implemented using the X.509 certificate standard. We will also discuss interoperability.

2.6 History

The history of the ancient science of cryptography, is well documented[Kah67, Sti95]. There is, however, one interesting aspect of its more recent history which directly applies to this thesis and the CTBT.

Before the (public) discovery of asymmetric cryptography in 1976 by Diffie and Hellman[DH76], Sandia National Laboratories had already worked for decades trying to come up with an acceptable solution to the challenging demands of a verification system for an underground test-ban treaty.

The problem is that the monitoring body ("the monitor") might require data from the nation being monitored ("the host") to successfully detect a breach of the treaty.

The first question is: How can the monitor trust the host-controlled sensors to provide the authentic information? One solution was to install a conventional encryption key at the station (installed and kept secret by the monitor) and encrypt all data with it. With enough redundant information (time stamps or serial numbers, as well as some well-known text at the beginning or end of each data block, for example), the monitor would be satisfied that the data was authentic.

This approach, however, is unacceptable for at least one immediate reason: the host has no way of verifying that the data transmitted from the station is limited to the data it is *supposed* to transmit.

This problem prompted research into a field were little was publicly known at the time: message authentication without secrecy. According to [MvOV97], the first mention of using DES to provide a MAC-like mechanism was in 1977 by Campbell. The term MAC did not surface until the end of the 1970s, when it was tied to authentication of financial messages[Ame86]. In 1980, FIPS 81 (DES Modes of Operation)[Nat80] standardised MACs based on DES, using CBC mode or CFB mode.

This was in the early 1970s, and public key cryptography was not publicly known.[7] According to [Sim91], Simmons, Stewart, and Stokes found their solution in 1974. They proposed to form an authenticator, much shorter than the message, using a hash function, and encrypt it with a secret key. The encrypted authenticator is then appended to the message. A technique resemblant to MACs.

The encrypted authenticator was in many ways satisfactory. It was proposed that the host could be handed the key used to create the authenticator once the monitor had successfully verified the authenticator. The host could now be satisfied that the station transmitted only the data it was supposed to transmit, and the monitor could be satisfied with the integrity of the data.

[7]Although Diffie and Hellman for a long time were credited with first discovering public key crypto, it was revealed in 1997 that the researchers Malcolm Williamson, Clifford Cocks and James Ellis from Government Communications Headquarters (GCHQ) in England had come up with the idea in the early 70s. The note written by James Ellis, "The Possibility of Non-Secret Encryption" is now published at CESG's web site: `http://www.cesg.gov.uk/site/publications/media/possnse.pdf`. Digital signatures were not anticipated by the English researchers. There is no question that Diffie and Hellman's results were independent of those from GCHQ.

But what if a station picked up a nuclear event, and promptly sent the incriminating evidence to the monitor? The monitor, verifying the authenticator, is certain that the data is authentic, and that a nuclear explosion has taken place. But instead of a confession, the host would claim that the data is a forgery; that the authenticator is not the one created by the sensor, but by the monitor. There is nothing the monitor can do to prove that the authenticator was created by the station, since it itself (the monitor) has the exact same opportunity as the station to create the authenticator.

The demand for non-repudiation is evident. The monitor must be able to prove to a third party that the message truly *is* authentic and originating from the host-controlled station. An acceptable answer to this new question did not surface until public key cryptography and digital signatures were proposed by Diffie and Hellman in 1976. It quickly became obvious that digital signatures were a perfect fit in treaty verification.

Already from the early days of PKC, its application to the CTBT was perceived as one of its important applications[Dif88]. The requirements of the treaty verification system was evidence that PKC provided a useful tool not found elsewhere.

Chapter 3

Public Key Infrastructure

In this chapter we will give an introduction to Public Key Infrastructure (PKI) in general. What is the purpose of a PKI, how does it work, what problems can it solve, what are the pitfalls?

3.1 What is a PKI?

In chapter 2 we saw that the existence of efficient and strong algorithms allow the generation of—for all practical purposes—secure digital signatures as well as impenetrable encryption. We also saw that asymmetric encryption algorithms allow exchange of session keys without the need of a secret channel.

There are still two issues left before a confidential and/or authenticated communication network can be established:

- Distributing public keys.
- Verifying public key validity.

In a wide-spread environment, both these issues raise major problems. When Alice makes her public key available to Bob and sends him a message signed[1] with the corresponding private key, how does Bob know this really *is* Alice's key? It could well be Oscar, pretending to be Alice, who sends *his* key and *his* signed material. Although the need for a *confidential* channel for key exchange is no longer present, we still very much require an *authenticated* channel. We must be able to tell that the source of the key really is Alice and that it has not been tampered with in transit. Bob might additionally

[1]The same argument applies to encrypted messages. The key-transfer will just be the other way around, Alice needs to get Bob's public key.

want to be certain that Alice really is in possession of the corresponding secret key. A third problem for Bob, is finding out if this “Alice” really is the “Alice” he knows.

In other words, we need a method for allowing the entities of our community to obtain the other entities’ keys in a trustworthy manner.

To meet this requirement, a system called Public Key Infrastructure (PKI) has been conceived.[2] The goal of a PKI is to establish a method for all involved users to retrieve the public key of any other participant and to be certain of its authenticity.

Exchanging public (or, for that matter, symmetric) keys between a few people is easy, at least if the people in question live close to each other or have some other means of communicating directly. This direct approach, however, does not scale to bigger systems at all. In a PKI, the solution is to establish a central authority which issues, signs, distributes, and, when necessary, revokes proofs of identification. This central authority becomes the basis for all trusted communication in the system. It is therefore essential that all the members of the system has access to an authenticated copy of its verification key.

3.2 Entity Naming

When Diffie and Hellman introduced public key cryptography to the public in 1976[DH76], they claimed that the new technique would put an end to the key distribution problem once and for all. They proposed that a “phone book” of public keys where to be made available to all. Its entries would include name, address, and, instead of a phone number, the entity’s public key.

While this is certainly a compelling idea, it has severe weaknesses. Identifying people by their names work well on a small scale, but quickly runs into problems. In [Ell00], Carl Ellison describes some of the problems with the Diffie-Hellman approach. He concludes that, as human names on average only offers about 20 bits of entropy, so names alone are far from adequate.

Ross Anderson also comments the Diffie-Hellman solution in [And01] (pp. 124-136). One of his remarks is that addresses tend to change, and that they make a poor basis for a naming scheme. In a project attempting to develop a register of entities and their public keys, Anderson and his research group found that the main cause of changes to the directory was change of addresses. Compromise or loss of keys did not command a single change in the directory[ACL$^+$99].

[2]One can say that *any* mechanism for managing and distributing public keys is a PKI. In this thesis, we define a PKI to be an infrastructure as described in this chapter.

Finding a naming scheme that both allows for a unified global naming system and is intuitive, has proven to be difficult. When Alice wants to send an encrypted message to Bob, she must be able to look Bob up and find his private key. In a small company's private PKI, this is simple; there is just one Bob. What, then, with a major national or international PKI? The world is full of Bobs. How can Alice find the right one?

3.2.1 Distinguishing Name

One suggestion which aims to solve the naming problem is *distinguishing names* (DNs). DNs originates from the X.500 directory service standards.

A DN is a globally unique name for an entity, consisting of a root and a *relative DN* (RDN). The root again, is defined by *its* root, and *its* RDN. An RDN is a pair: attribute and value.

The "tree" of DNs does not have one (global) single root. When you set up a tree of DNs, you choose both the names of your attributes and what is the top level. It is not uncommon to use "dc" (domain component) as the attribute for all but the leaves of the tree (e.g. "dc=com, dc=company, dc=marketing"). Another approach is to use "c" or "o" (country or organisation), "ou" (organisational unit), etc. The RDN of end-users often has an attribute name of "cn" for "common name".

Attribute names are normally case-insensitive. For values, this depends on the attribute. Due to the nature of the DN, the order of the pairs is significant.

An example of a naming scheme using DNs is the one used in the IMS PKI. For IMS stations, the root will be

```
O=CTBTO, ou=International Monitoring Regime, ou=IMS Stations,
  L=<Site specific>, cn=<Subject specific>
```

3.3 Components of a PKI

3.3.1 Certificates

A certificate is a data structure designed to provide some means of binding an entity's *identity* to its public key. It will typically be a document including (but not limited to) a unique (within the PKI) name (e.g. a DN) of the owner, a name uniquely identifying the issuer, the public key of the owner (with details about algorithm used etc), period of validity. The document is then signed by a trusted authority.

There are several certificate formats. The standard used in the IMS PKI is the X.509 format[Int97b], described in RFC 3280[HPFS02]. See section 6.4.1 for details on the X.509 certificate standard.

Version 3 of X.509 certificates open for various pre-defined or custom extensions to be added to a certificate (this is discussed in detail in 6.4.1), including "Key Usage" and "Extended Key Usage". These extensions are designed to allow the issuer to state the intended use of this specific certificate. Primarily, the usage extensions are present to provide information on what the certificate should *not* be used for. One main constraint is telling other end-users that this certificate is an end-user certificate, and should not be accepted as the certificate of an issuer. Other useful constraints include telling end-users that a certificate is issued for signing purposes only (i.e. not for encryption) and a pointer to the relevant certificate policy (see section 3.4.1).

An extension may be marked critical or non-critical. A client using a critical extension is required to understand and process that extension, or to discard the certificate.

3.3.2 Certification Authority

The CA[3] is *the* central piece of any PKI. Its primary task is to verify identities and to issue, distribute, and revoke certificates.

Key Generation In some PKI solutions, the CA is responsible for generating end-users' key-pairs. The common solution is that the users generate a key-pair themselves and then issue a certificate request to the CA, but if for example the users are very low on computational power or simply lack the ability to generate keys, or the PKI policy states that private keys should be backed up at the CA, the alternative can be useful.

In the case of keys being generated by the CA, there are two major problems: The secret key must be safely conveyed to the owner, and we would have a problem non-repudiation-wise. The secret key is now known by the CA. This causes a problem, because although the CA *is* trusted by the users to identify entities, it is not necessarily trusted to properly guard knowledge of entities' secret keys.

With regard to key generation and key backup, a distinction is often made between keys intended for signing and keys intended for encryption. For a

[3]In [AL99], Adams and Lloyd point out that even though the term "Certificate Authority" is wide-spread in the literature, it is not correct, at least not in connection with X.509. In the RFC[HPFS02], the term "Certification Authority" is used exclusively, and, as Adams and Lloyd argue, the CA is the authority on *certification* (issuing certificates), not on the certificates themselves, which they refer to as a *policy authority*.

signing-only key, there is normally no need for a backup of the private key. Should it be lost, we can still verify the old signatures, and a new one can be issued for the future. Encryption (or, actually, decryption) keys are not as simple. If the private key of an encryption key-pair is lost, every document encrypted with the corresponding public key is lost (barring a break of the algorithm). In many situations, this is an unacceptable risk, and care will be taken to have a backup of all decryption keys for emergency use.

Identification and Certificate Issuing Before issuing a certificate to an end-user, the CA, often via a Registration Authority (RA, see below for details), has to somehow verify the identity of the requester. It is normally also required that the CA gets a proof that the user is in possession of the private key corresponding to the public key the users wants a certificate on.

Certificate Distribution When the CA has made and signed a certificate, it is (again, often via the RA) conveyed back to the requester. Since the certificate is signed and therefore (barring a break of the cryptographic algorithms or protocols used) unforgeable, there is no need for a secret or authenticated channel from the CA back to the end-user. Now, the user can redistribute his certificate to anyone—it is for example common to include the certificate with a digital signature—but it is also desirable that other users can obtain the certificate without approaching the owner.

Distribution of issued certificates is the CA's job. A common procedure is publish all valid certificates the CA issues in a directory, and provide end-user access. See section 3.3.5 below.

Certificate Revocation In the case of an end-user's private key being compromised or the user's privileges stripped or changed, the CA must communicate to its users that the certificate of that user no longer is trustworthy.

The first step in that process is obviously to remove the certificate from any directory it is listed in, but that is not enough. The certificate is (presumed) known by everyone, and is still valid since it has the CA's signature on it, and is not expired. The CA's next step will be to *revoke* the certificate by posting it on a Certificate Revocation List (CRL). See 3.3.3 below for details on CRLs.

CA Organisation It is possible to have not only one CA, but a hierarchy of CAs. In a wide-spread PKI, having all the certificates issued by one CA can become impractical as well as less secure. It can be impractical because access to the CA's private key (which is required to issue a certificate) should be limited to as few people as possible. The work-load on those people can become heavy.

The reason it can be less secure is mainly that compromise of the CA private key will affect *every* certificate of the PKI. If only one sub-CA is compromised, most of the PKI could live on.

There are many models for PKI hierarchies. We will now briefly describe two widely used models.

Single root pyramid In a single root pyramid, there is a root CA issuing sub-CA certificates. When an end-user uses a certificate, that certificate will typically be issued by a sub-CA. The user will then (in an unauthenticated manner) obtain the sub-CA's certificate (note that the user will also need to verify revocation status of all certificates throughout the chain, see certificate revocation below). This CA certificate is issued by another sub-CA, or possibly the root CA. The user will need to follow this chain all the way to the root. The root CA's authenticated certificate will need to be known by every end-user.

Cross-certification While a single root pyramid may be suitable for some environments (for example a company's own PKI or the PKI used in the IMS network), it is unsuitable for others (like the Internet). An extension is a cross-certification solution. With cross-certifications, what was the root CA with a single root pyramid now become *peer CAs* in a larger hierarchy. Let Alice be part of pyramid A with the root, or peer, CA CA_A (so she has CA_A's certificate installed), and Bob is certified by CA_B in pyramid B. Alice gets hold of Bob's certificate, and wants to verify it. She follows the chain of sub-CAs, and reaches CA_B. She does not, however, have this certificate installed. Still, since CA_A and CA_B are cross-certified, Alice can consult CA_B's certificate, signed by Alice's own CA, stating that CA_A trusts CA_B as a peer CA. She will therefore trust CA_B.

3.3.3 Certificate Revocation List

All certificates should be fitted with an expiry date. The primary reason for this is that what was considered a strong enough key length or good enough algorithm one year might not be adequate in, say, ten years. There is of course also the off chance that the private key is compromised without the owner's knowledge.

The is no guarantee that the key will stay safe for its entire projected lifetime, though. Someone other than the owner may obtain knowledge of the private key, the owner might be stripped of his or her privileges, the algorithm may be broken, etc.

In such case, as mentioned briefly above, the CA needs to be able to revoke the certificate. In a PKI, that means posting the certificate's identification (in X.509, its serial number) on a revocation list.

An absolutely fundamental requirement for the CRL approach to work is that *all* entities check the most recent CRL *every* time they consult a certificate.[4]

There are several ways of implementing the CRL itself. The minimal details the CA must be able to communicate are (for each certificate on the list):

- Unique (within the CA's domain) identification of the certificate,
- date of revocation.

Additionally, every CRL implementation should allow the CA to optionally specify reason for the revocation. An option for stating the *end date* of the revocation is useful as well. With this possibility, the CA can temporarily revoke certificate. There are several scenarios where this can apply. Examples are users temporarily quitting their job or changing their position or if a suspected compromise of the private key is being investigated.

As is the case with certificates, RFC 3280[HPFS02] specifies a standard for CRLs. There are defined two versions of the X.509 CRL, the change from version 1 to version 2 being the inclusion of extensions. As with X.509 certificates, an extension can be marked critical or non-critical. A critical extension must be understood and processed by a client for the client to accept the CRL. Extensions on X.509 CRLs can either be on a per CRL basis (Delta CRL indicator is an example) or on a per entry basis (reason for revocation, for example).

When a certificate is put on the CRL, it normally stays there until its expiry date (there is no use keeping expired certificates on the CRL, as they are invalid anyway). In a large PKI system, the CRL could potentially become rather sizeable, and keeping in mind that it is to be checked by every entity in the system (almost) every time a certificate is consulted, the bandwidth required can become a problem, both centrally and at the nodes, particularly nodes with limited bandwidth or memory.

A possible remedy to this situation is to abandon the flat CRLs with all revoked certificates in a long list—also called *Complete CRLs*—and use a different approach. We will describe some alternatives here.

Delta CRLs The idea of the *Delta CRL* approach is to issue a complete CRL on regular, relatively long intervals, and issue smaller (possibly empty)

[4]Every time is perhaps not required, but at least sufficiently often. How often is considered sufficient varies from PKI to PKI, but a range from one hour to one day should cover most.

CRLs on shorter intervals naming only their *base CRL* and any new certificates revoked after the release of the base. Every delta CRL contains every revoked certificate since the release of the base, so a client does not need to check more than the most recent delta.

A delta also contains the serial number of the base CRL it refers to, so a new end-user or a user returning from an off-line period, can easily first fetch the most recent delta, then find out which base is required and get that one too.

Delta CRLs are implemented in X.509 as a *CRL extension*: Delta CRL indicator. The extension must be marked critical (if it was not marked critical, and ignored by client software, the software would read what is only a (small) part of the entire CRL, but interpret it as the full CRL).

CRL Distribution Points CRL Distribution Points (CDP), also referred to as Partition CRLs, allow the revocation lists of one CA domain to be distributed as multiple CRLs. It also allows a client verifying a certificate to more easily locate the relevant CRL.

This is implemented in X.509 as a *certificate extension*. The value of the extension is a list of items. Each item must have either a DN of the CRL issuer (only if the certificate issuer and the CRL issuer are different entities) or a description of where to obtain the CRL.

If the value field is not an (alternative) issuer, it should be a *Uniform Resource Identifier* (URI)—e.g. an LDAP address or an HTTP address—specifying where to retrieve the CRL from or a DN relative to the issuer (we will not address the technical details here, please refer to section 4.2.1.14 of RFC 3280 [HPFS02] for the formal definitions). It is possible to provide several location specifications (URIs or other methods); in that case, they should all refer to the same CRL.

As mentioned above, CDP is an extension on the *certificate*. It is recommended not to be marked critical, and can serve simply as an indicator for the entity verifying the certificate on where to find the CRL. It can be combined with any supported CRL format, for example Delta CRL.

3.3.4 Registration Authority

The RA is the “extended arm” of the CA. In a small PKI, the RA can be incorporated in the CA, but on a larger scale, it is common to try to limit the extent of the CA wherever possible. The common tasks for an RA is to perform the identity verification of users and distribute the key material to and from end-users.

Typically, the end-user requesting a certificate will arrange to interact with the RA in a way for the latter to satisfactorily verify the former’s identity.

The RA will then make certain that the user has access to the corresponding private key before forwarding the request to the CA.

3.3.5 Directories

When a certificate is issued to an end-user, Alice, she normally receives a copy of the certificate. If she wants to send a signed message to Bob, she can easily transmit the certificate along with the message (this is the standard behavior in for example S/MIME[DHR$^+$98]). Even if she does not include the certificate, it is normally rather easy for Bob to find, because he would know Alice's distinguishing name (as well as the issuer's DN). Every CA must have some means of distributing its issued certificates to end-users, and every distribution method must at the very least support lookup based on a complete DN.

When the roles are reversed, however, it is not equivalently simple. Consider the scenario where Alice wants to send an encrypted message to Bob. She knows that Bob has a certificate issued by a CA known to her, but she is not certain of Bob's DN with that CA. In an effective PKI solution, the CA must provide a repository for certificates and CRLs facilitating simple lookups as well as more complex searches. The X.509 standard is not (anymore) tied up to a specific directory access standard, but LDAP[YHK95] is a widely used protocol. LDAP does not specify the structure of the directory itself, and it is of little interest to end-users. It is possible to allow the underlying directory be accessed both through (e.g.) LDAP and other protocols.

3.3.6 Time stamping

Not every PKI will include a time stamping authority (TSA), but can be very useful. The idea of a (secure) time stamping service, is to provide a service that all entities in the PKI trusts to associate the correct time with a document. To have a document time stamped, a user can send a cryptographic digest of a signed message (i.e. of the message *including* the signature) to the TSA, which in turn creates a document containing the digest and the current time. The document is then signed by the TSA and returned to the user.

The advantage of supplying the TSA with a digest only, and not the entire document, is of course that potentially confidential material does not have to be exposed, only a "fingerprint" of it, but it is adequate for reliable time stamping.

Of course, such a time stamp can not prove when the message was signed by the user, and certainly not the time the user, if applicable, transmitted the (signed) message to the recipient(s), but what it *can* prove, is that the message was signed by the user no later than the time stamp from the TSA. The user can use that property to for example prove that the signature was made before the certificate expired or was revoked.

3.3.7 A PKI Overview

We will now describe a typical PKI certificate creation and usage process. Figure 3.1 shows a scenario where Alice first obtains a certificate from the CA, and Bob later acquires that certificate. We here assume that both Alice and Bob are in possession of an authenticated copy of the CA's certificate.

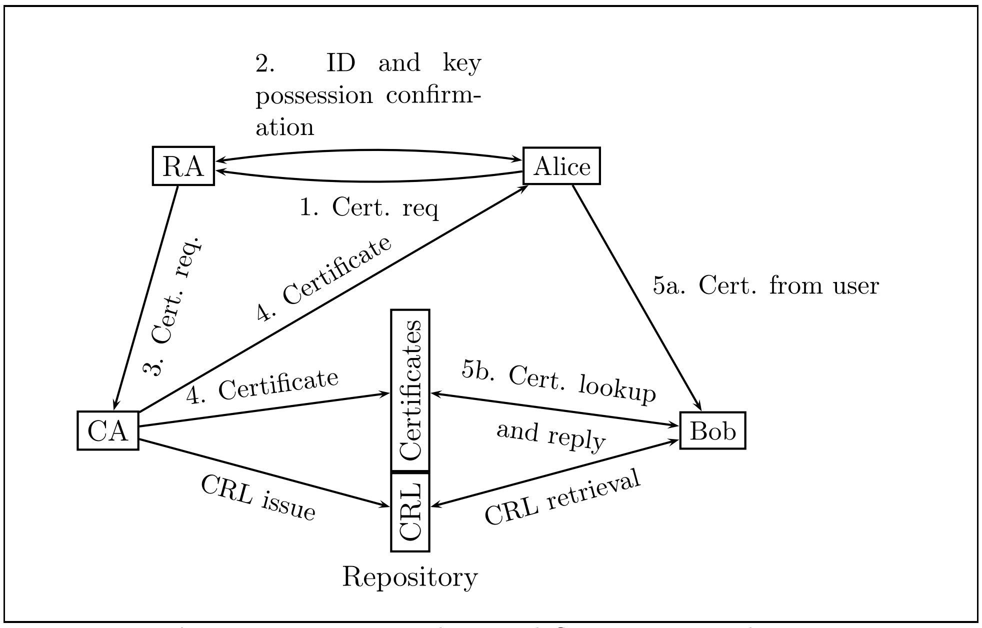

Figure 3.1: Data and control flow in a typical PKI.

In step 1, Alice has generated a key-pair (for digital signatures or encryption) and contacts the RA with a certificate request. At this point, Alice would of course verify the identity of the RA by checking its certificate, issued by the CA.

The RA proceeds in step 2 by confirming Alice's identification and that she possesses the private part of the key-pair. When satisfied, the RA (step 3) forwards the certificate request to the CA.

If the CA accepts the request, a certificate for Alice is created and signed, and then (step 4) published. It would be common for the CA to both send a copy of the certificate back to Alice (possibly via the RA), and to publish it in its certificate directory.

Finally, in step 5, Bob obtains Alice's certificate. In the case where Bob wants to send Alice an encrypted message, he is likely to look her up in the directory, if he has received a digitally signed message, the certificate was probably included. It is also important that every time Bob uses the certificate (he only needs to retrieve it once if he saves it), he fetches the last CRL from the directory, and verifies that Alice's certificate has not been revoked.

3.4 Trust

In a system with a need for trustworthy identification of its members (people or devices), it is implicit that anyone who wants to verify another entity's identity, needs to *trust* some part of the system. One of the primary tasks for such a system is to allow the users of the system to transfer the trust around to where they need it.

Trust, however, is not an on-off issue. Recall from section 3.3.6, that a TSA would be trusted by all entities to provide reliable time stamps. The users may not, however, trust the TSA to not disclose the information it stamps, so they only provide it with digests. This is a good example of an entity trusted to perform one task, but not another.

In a PKI system, it is implicit that users must trust all their CAs. If a user, Bob, does not trust his CA to do this job, it has no value for him.

An important point to make here, is that while Bob trusts his CA(s), he does *not* trust all the users for which the CA has issued a certificate. If Alice has been provided with a certificate by a CA trusted by Bob (and he is in possession of the CA's public key and is certain of its integrity), Bob accepts that the *identity* of Alice is correct, not that Alice will behave the way Bob expects (or hopes). The CA does not vouch for its users' "character", it just ties their identity to their public key.

3.4.1 Policies

In a PKI, we need to be able to formalize trust. One important tool in that regard, is a Certificate Policy (CP). CPs are developed by the CA or by a dedicated policy authority.

CPs are often used as the legal framework around certificates. An end-user which is issued a certificate must agree to abide by the accompanying CP. A CP will describe the responsibilities of both the CA (and other managing entities in the PKI) and the subject of the certificate.

Examples of what a CP can specify are

- Rules for storage of private keys;

- Areas of use of the certificate;
- Financial and legal liabilities in connection with abuse of certificates.

RFC 2527[CF99] defines CPs in the X.509 framework.

3.5 Problems Encountered in a PKI

3.5.1 End-user Identification

One of the primary needs a PKI usually fills, is the problem of securely distributing keys when a face-to-face meeting is out of the question. Every user trusts the CA never to issue certificates incorrectly identifying an entity as something it is not. This trust is absolutely necessary for the PKI to function properly.

There have been incidents where important actors in the global CA market have admitted to wrongly issuing certificates. A well-known example is the VeriSign-Microsoft issue from March 2001[Mic01]. Two code-signing certificates were issued with a common name of "Microsoft Corp." to someone not affiliated with Microsoft. The Windows operating system can check signatures on Active-X components, Word macros, etc, before running them, and will prompt the user if the signer's certificate is not recognized. In this case, the warning would display a DN specifying "Microsoft Corp." as the common name, which is likely to trick many (even advanced) users into running the signed code.

This incident illustrates the importance of properly verifying identities before issuing a certificate. If a CA issues a certificate erroneously, we are in an even worse situation than without a PKI; a state of false security. Another point made by this incident, is the difficulty of making a positive identification.

3.5.2 CA Certificate Integrity

A big problem in PKI solutions in general is the question of *how to distribute the CA's certificate to the end-users.*

One solution which is widely in use today, is to distribute the client software with the CA certificates installed. Web browsers are good examples on this approach. They usually come with a slew of CA certificates installed. Of course, this solution requires that the web browser itself is an authenticated copy.

The problem of CA certificate integrity is not solved, though. Getting the certificates reliably to the clients is only half the job, ensuring that they remain intact is the other part.

An adversary who could insert or alter a client's CA certificates has effectively killed all security in the PKI (from that client's point of view). If the client supports more than one CA (most do), he could for example add a new, false CA certificate (for which he possesses the private key). Such an attack would almost certainly go unnoticed.

3.5.3 Private Key Integrity

The CA issues certificates tying users' public keys to their identities. It is also common for the CA/RA to obtain proof from the user that it possesses the corresponding private key. An important next question is then: *is this private key known only to that user?*.

The answer to this question depends on the PKI. Some PKIs may have a certificate policy that make requirements on how the private key should be stored (e.g. on a smartcard). Assume that the smartcard is trusted not to reveal any information about the private key. If the RA observed the user sign the certificate request using the smartcard, then the users of the PKI can trust that the person controlling the smartcard is the only one with ability to sign/decrypt with this key.

Of course, there is still the possibility that the user's computer is compromised. Even though an attacker may not have free access to the private key, it is possible that he can access the signing/decrypting function of the smartcard.

In a PKI without a strict demand to its users on how to store the private key, there is no way for a relying party to determine the trust he should put in the authenticity of another user's signatures (or decryption key).

Chapter 4

International Monitoring System

This chapter will describe the IMS and the security architecture proposed by the PTS, including the protocols used.

As we stated there, the security goals are to ensure

- secure gathering of data, and
- authenticated transmission of that data to the IDC.

The IMS addresses these two problems. Over 300 stations are deployed world-wide, acquiring data sufficient to detect all nuclear explosions of significant magnitude.[1] To protect the data from alteration before or during transfer, digital signatures are applied. To provide authenticated exchange of verification keys, a PKI is established.

Our focus is on the transfer of the data from the stations to the IDC and the mechanism for remotely issuing commands to stations.

4.1 Overview of Data Flow

The *Global Communications Infrastructure* (GCI)[Com01a] is a global communications network. It is based on Very Small Aperture Terminal (VSAT) technology. It comprises three geosynchronous satellites, through which IMS stations and NDCs can communicate with the IDC.

[1]This statement is of course difficult to prove, and certainly beyond the scope of this thesis. It is assumed that the IMS when finished will be able to detect all nuclear explosions powerful enough to be used in development of nuclear weapons. In this thesis, we will accept that assumption.

Most IMS stations are connected directly to the GCI network. This is called a *basic topology* station. Some NDCs use a different setup called *independent subnetwork*, where the NDC is connected to the GCI, and the stations are connected to the NDC via a local, private network.

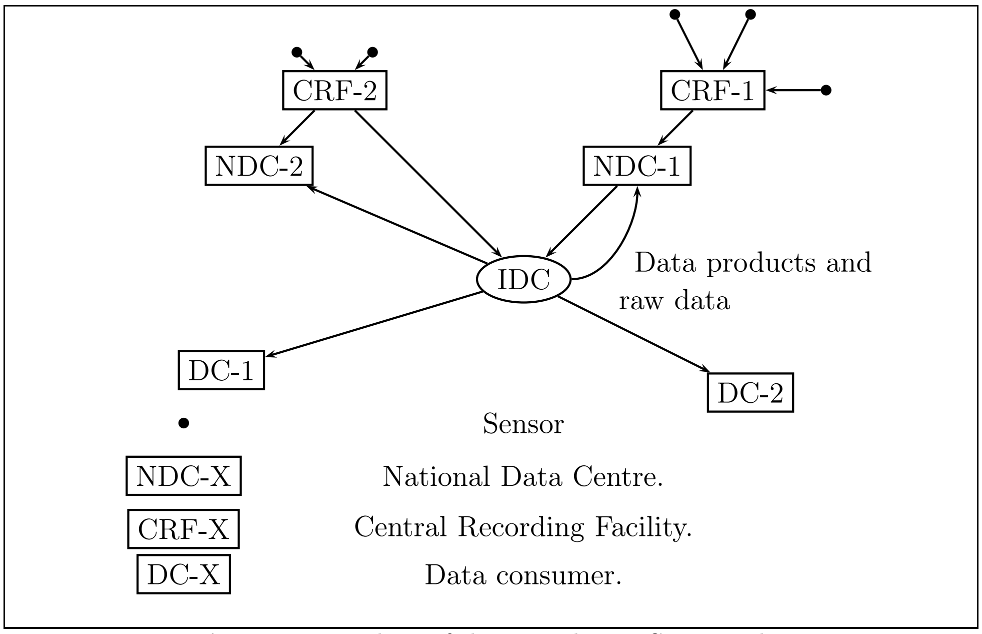
CRF-2
CRF-1
NDC-2
NDC-1
IDC
Data products and raw data
DC-1
DC-2
● Sensor
NDC-X National Data Centre.
CRF-X Central Recording Facility.
DC-X Data consumer.

Figure 4.1: Flow of data in the IMS network.

The data is received at the IDC, and data products[2] as well as the original raw data is made available to authorized data consumers (which include all signatory states).

See figure 4.1 for an illustration. In this figure, the data from NDC-1's stations are connected to an independent subnetwork, and the (signed) data is transferred to the IDC via the NDC. NDC-2, on the other hand, has a basic topology station, which is connected directly to the IDC. From the IDC, all NDCs and other data consumers have access to data products as well as the raw data from all stations.

Command and control messages to the stations will originate from either the IDC (not necessarily the IDC, but at least the PTS, which is at the same location) or the local NDC.

[2]The *data products* the IDC provide are automatically processed blocks of the raw sensor data.

4.2 The IMS Stations

The IMS stations are operated by local *station operators* (SOs), but both operation, maintenance, and manufacturing are paid for by the PrepCom. The SO is usually one or more persons at the NDC.

Recall from chapter 1 that there are four different types of stations within the IMS: seismic, hydroacoustic, infrasound, and radionuclide in addition to the radionuclide laboratories. Based on [Com01b], we give a brief description of the latter three, before concentrating on the seismic stations for the rest of the thesis.

Hydroacoustic Stations

Despite the size of the earth's oceans, only 11 hydroacoustic stations monitor the seas due to the enormously effective propagation of acoustic energy in water. 6 of them are hydrophone stations, where an underwater sensor picks up signals and transmits them to a shore-based station.

These stations are costly to install and maintain, so they are supported by 5 T-phase seismic stations situated on islands and use seismic techniques to detect when acoustic waves hit the island and are converted to seismic waves.

Infrasound Stations

There are 60 infrasound stations in the IMS network, each consisting of an array of 4 to 8 sensors. They are used to detect very low-frequency sound-waves in the atmosphere. These sound-waves can be a result of many things, some man-made (rocket launch, nuclear explosion), some natural (volcano, meteorite).

Radionuclide Stations

The 80 radionuclide stations collect air samples. Samples suspected of originating from a nuclear explosion are sent to a radionuclide laboratory for full analysis. The radionuclide stations and laboratories are the only stations to provide unambiguous evidence of a nuclear explosion.

Seismic Stations

There are 170 seismic stations in the network, more than half of the 321 stations. These stations are extensively used to detect, measure, and locate earthquakes. The seismic stations monitor (sound)waves generated by explosions, earthquakes, or other physical or man-made phenomena that travel through the earth. These stations are the most important tool for detecting and locating underground explosions.

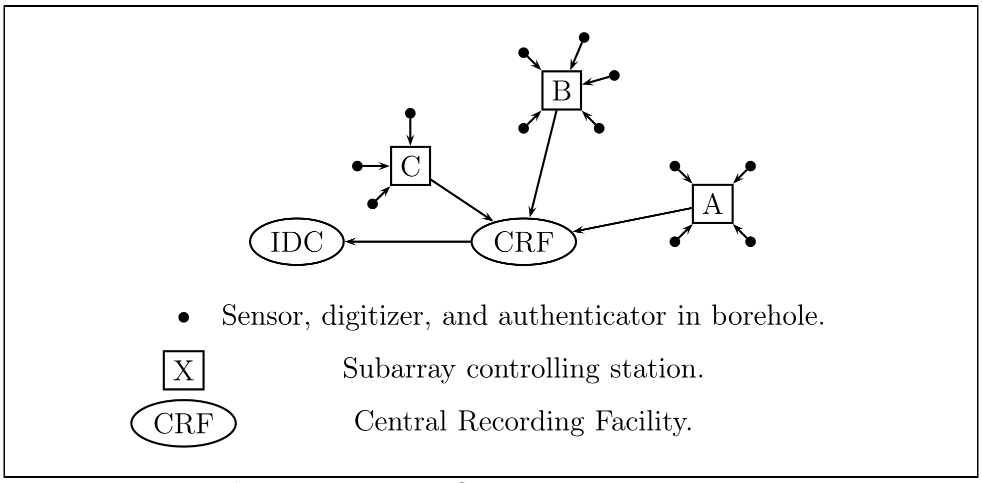
B
C
A
IDC
CRF
● Sensor, digitizer, and authenticator in borehole.
X Subarray controlling station.
CRF Central Recording Facility.

Figure 4.2: Data-flow in an array station.

Three-component and Array Stations From a technical viewpoint, there are two kinds of seismic stations: three-component stations and arrays.

A three-component station has three sensors at a single site detecting the three components of a seismic wave, up-down, east-west, and north-south.

An array station, on the other hand, has a number of sensors, spread out over an area with a diameter ranging from 1 to 60 kilometres. An array station may further have sub-arrays where a smaller number of sensors (typically less than 10), located over a relatively small area, are connected to one controlling computer, which in turn communicates with the heart of the array. See figure 4.2 for a description of a typical array station. An array station has a minimum of 12 sensors, and the largest station has over 60.

Primary and Auxiliary Stations Of the 170 seismic stations, 50 are so-called *primary stations.* These stations send data continuously in real-time to the IDC. This data is received and (automatically) analysed at the IDC.

Auxiliary stations do not automatically send data to the IDC. If data from primary stations—or some other source—indicate an event (which may or may not be a *nuclear* event), data from the same time-frame (adjusting for time of travel for the signals to the other station's sensory field) can be requested.

4.2.1 Components of an IMS Station

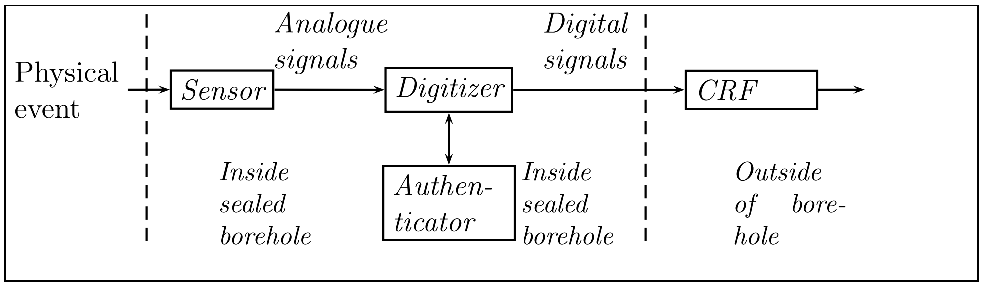

Figure 4.3: Data-flow in a typical IMS station.

Figure 4.3 shows the main components of a seismic station.

Sensor

The sensor itself has only one ability: a seismic sensor produces analogue signals that represent the movement of the ground.

Digitizer and Authenticator

A *digitizer's* job is to convert the analogue signals from the sensor into appropriately formatted digital signals. These signals are signed by an *authenticator*. It may be an integrated part of the digitizer, or a standalone device.

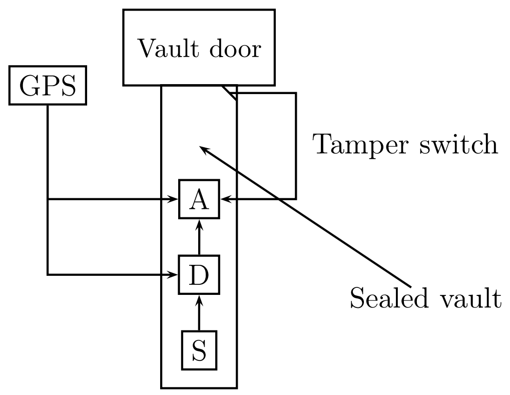

Figure 4.4: The borehole of a seismic sensor.

These two devices are put in the borehole with the sensor. This borehole is required to be closed and sealed so the data from the sensor can be tagged if the door is opened. See figure 4.4 for an illustration of a typical borehole. Here, S is the sensor, D the digitizer, and A the authenticator.

The vault door of the borehole is connected to a tamper switch, which will signal the authenticator[3] if the door is open. Additionally, both the authenticator and the digitizer are connected to a GPS device positioned outside of the borehole to synchronize clocks.

We refer to each sensor/digitizer/authenticator triple as a *channel*.

Signing Many older installations have digitizers which are not fitted with an authenticator, and hence cannot apply digital signatures to the data they produce. This is a problem which presents us with a significant security risk, which we will come back to later. In all digitizers deployed after 1st of January 2000, however, the authenticator (or, rather, the ability to digitally sign the data) is mandatory.

Central Recording Facility

The Central Recording Facility (CRF) is the heart of the station. All data transmitted from the sensors (authenticators) to other destinations go through it. The data is compiled into a single block for all the channels on the station.

All command and control messages to the station are received by the CRF. All signature verification and access control is performed here.

There are mainly two reasons for this. At the time of writing of this document (July, 2003), most digitizers in use are unable to verify signatures themselves. This may change in the future, but there is still another obstacle: The different brands of digitizers do not share a common command language. Hence, the process of sending a signed command from the IDC to a station would be much more complicated.

To prevent data loss in the event of a communications breakdown, the CRF is required to maintain at least 7 days of data history available for the IDC.

[3]The signal may be passed to either of the digitizer or authenticator. The party that creates the data packages will need the information. The choice is of no consequence to us.

4.2.2 Physical Security

Borehole components

There are two levels of physical security regarding the digitizers and authenticators. The first is the borehole security, and the second is tamper protection on the authenticator.

Borehole security was briefly mentioned earlier, and basically means that it should be impossible to open the vault door without having the sensor data tagged appropriately. No documentation has been found on how the borehole is protected if the digitizer is unable to produce data at the time (e.g. if it is without power).

The digitizer itself will likely not be protected against tampering, but all authenticators must be. The authenticators are required to meet FIPS 140-2[Nat01b] level 3 specifications.

The CRF

The CRF is required to be in a locked and tamper-protected environment. Access to the CRF shall be limited to the SO. There is a strict requirement on logging of the activities at the CRF, including the following:

- All logins and logouts (successful and unsuccessful);
- All superuser activity;
- All changes of (user) permissions;
- All access to critical data.

Logging should be done to a loghost (i.e. another machine), and preferably to a read-only media (like paper). There is no demand that logs should be forwarded to the PTS unless a special event dictates so.

It is assumed that the SO controls the CRF. The security measures are primarily to prevent and detect any unauthorized access.

4.2.3 Clock Synchronization

With seismic data, it is very important to have accurate clocks. In the case of an event, geophysicists would want to use data from many stations in different locations and at different times to calculate the source of the event. The sensors (actually the device, which may be the digitizer or the authenticator, which time-stamps the data) need very accurate clocks (a drift of a maximum of a few milliseconds is acceptable) and are continuously synchronized using GPS.

The CRF is also required to synchronize its clock. For logging purposes, and also for determining if a command is recent enough to process, it is important that the CRF has a correct clock. The accepted drift will be much larger than for the sensors, though.

4.3 Protocols

In this section, we will describe the protocols designed for station communication. There are two protocols, one for continuous data transfer (CD1.1) and one for command and control of stations as well as data requests to auxiliary stations (IMS2.0/IMS1.0).

It is not required that all stations must be able to receive both protocols. Primary stations (primary seismic, hydroacoustic, and infrasound) must accept the CD1.1 protocol, while auxiliary seismic stations and radionuclide stations must accept the IMS protocol. For this reason, the possibility is provided to use CD1.1 for command and control to primary stations. This is handled by embedding IMS2.0 commands in CD1.1 frames. Conversely, the IMS protocol when used for data transfer will use the same low-level data structures as CD1.1.

Our focus will be on data transfer for the first protocol and commands for the second.

4.3.1 Continuous Data Transfer

For transferring data from the primary seismic stations, the CD1.1 protocol will be used. CD1.1 is an extension of the old CD1 protocol, also known as the *alpha* protocol. Currently, both are used in the IMS network, with CD1 dominating. Converting stations to use CD1.1 is an ongoing project.

We will only describe version 1.1 of the protocol.

CD1.1 is a connection oriented protocol. The connections are opened by the *data provider* (the sender), which contacts the *receiver* with a connection request. For data transfer from station to IDC, the IMS stations (the CRF) will be the provider, and the IDC the receiver. In command and control, the roles will be reversed.

All CD1.1 transmissions are *frames*. Each frame consists of three parts: a header, a payload, and a trailer.

Table 4.1 shows the header fields. A frame is uniquely identified by the three-tuple *frame creator*, *frame destination*, and *sequence number*. Data

FIELD	FORMAT	DESCRIPTION
frame type	integer	Frame type
trailer offset	integer	The offset at which the trailer starts
frame creator	8 byte ASCII	Name of source of the frame
frame destination	8 byte ASCII	Name of intended receiver
sequence number	integer	Sequence number of Frame
series	integer	Series number of Frame

Table 4.1: Frame Header fields.

Frames (see below) do not normally have a specified destination (they may be forwarded to more than one destination), and are generally marked with 0 as their destination. This means that for each station (frame creator), no sequence numbers can be reused for data frames. The sequence numbers are 64 bit integers.

FIELD	FORMAT	DESCRIPTION
authentication key identifier	integer	The assigned identifier of the signer's certificate
authentication size	integer	Length of the authentication value field
authentication value	N data bytes	The signature
common verification	long	CRC for error detection

Table 4.2: Frame Trailer fields.

The trailer fields are listed in table 4.2. The first field is the serial number of the certificate issued by the CA on the public key required to verify the signature. If this field is non-zero, authentication is assumed to be used.

The signature is applied to *the entire header and payload.*

The payload will vary from frame type to frame type. The CD1.1 standard defines 10 different frame types. We will not go into detail with most of these types, but we will describe the most important ones: *Connection Request*, *Connection Response*, *Acknack*, and *Data.*

Connection Request A connection is established when the *data provider* sends a Connection Request Frame to a receiver on a well-known port. This receiver will typically be a *connection manager*, and not the actual receiver of the sensor data. The version of the protocol and the name of the sender are included in the frame.

Connection Response If the connection manager receiving a Connection Request Frame is able to handle the request, it will decide on a host and a port for the transmission. The host may well be the same as the manager, but in practise it is likely to be another date receiver. The Connection Response Frame includes an acknowledgment of the version specified by the sender, the name of the host which will receive the data, the IP address of the same host, and the designated TCP port to use.

Data The most important frame type is the Data type. The bulk of the transfer will normally be frames of this type.

FIELD	FORMAT	DESCRIPTION
number of channels	integer	Number of channels in this Data Frame
frame time length	integer	Time this frame encompasses, in milliseconds
nominal time	20 byte ASCII	Nominal start time of this Data Frame (UTC)
channel string	N byte ASCII	A string listing all the Channel Subframes to follow

Table 4.3: Channel Subframe Header fields.

A Data Frame consists of, in addition to the normal frame header and trailer, a Channel Subframe (CSF) Header and one or more CSFs. Table 4.3 and table 4.4 show the most important fields in a Data Frame.

Notice that each CSF includes a signature. This is the signature applied by the authenticator in the borehole. All authenticators must provide data in the CSF format. This signature is applied to all CSF fields except the subframe length, the offset of the authenticator, the certificate id, the authenticator size, and of course the authenticator itself.

Legacy stations employing digitizers which are unable to produce signatures, are for the interim allowed to apply them at the CRF. As the CRF will sign the entire Data Frame as well, the Subframe signatures are redundant. They are still required, though, because many entities will want to request data from the IDC, and will receive only the CSFs, not the entire Data Frames.

FIELD	FORMAT	DESCRIPTION
channel length	integer	Size in bytes
channel description	data bytes	24 bytes setting flags and identifiers for the channel. Among the options to be set here, are whether or not authentication is used, type of sensor, and name of site (station) and channel
time stamp	20 byte ASCII	Start time of first sample in this channel
subframe time length	integer	Time spanned in this data, in milliseconds
channel status data	data bytes	Flags describing the status of the channel. This includes security warnings (authentication seal broken, vault door open, equipment moved), data status (channel dead, channel being calibrated), power status (main power failure, battery unstable)
data size	integer	Size of data sample
data	N data bytes	The channel data
authentication key	integer	Serial number of signing certificate
signature	data bytes	DSA signature

Table 4.4: Channel Subframe fields.

Acknack A frame type called Acknack is provided to allow the receiver to let the provider inform the provider of the received (Data) frames. An Acknack Frame contains the first and last sequence number for the period acknowledged with this frame, and a number of *gaps*. There may be arbitrarily many gaps, each represented by the first and last sequence number that are missing. The frames mentioned in the gaps must be resent by the provider.

A frame is said to have been acknowledged if its sequence number is between the first and last number of an Acknack Frame and not part of a gap in that frame. The provider is required to save all data not yet acknowledged.

Security

In the CD1.1 standard, all frames have a trailer, which contains key (actually certificate) identifier and a signature. The key identifier is allowed to be set to 0, meaning that no authentication is available. The same applies to the CSF authenticator id field. The reason for this is that the protocol may be used outside of the IMS for many purposes. Additionally, data may be requested from non-IMS stations in certain situations, and they are not required to sign anything. All IMS stations are required to digitally sign every frame and subframe.

4.3.2 Commands and Data Requests

The International Monitoring System protocol (IMS) has two primary areas of use: data requests and command and control (C&C) of IMS stations. There are currently two versions of this protocol in use. Version 1.0 is used for data transfer from most stations, while version 2.0 is used for command and control of stations and radionuclide data requests. The data request part of the protocol is also used to obtain state-of-health information for the stations.

Our concern in this thesis is primarily the C&C part, so we will refer to the protocol as IMS2.0.

Command Flow

An IMS2.0 command is defined by both a command request and one or more command responses. For all commands, at least one of these command responses will act as an acknowledgment (some commands have additional responses where the results of the command are communicated). An acknowledgment is required to be returned to the transmitter within one working day.

Format

The structure of an IMS2.0 command or response is described in [WGB]. We give an overview of the parts relevant to the security of the protocol here.

A command is wrapped between a line containing `BEGIN` and one containing `STOP`. Each line contains a parameter, some mandatory and some optional. A physical line is limited to 1024 (ASCII) characters, but a logical line may be longer, as a backslash as the last character of a (physical) line may be used to continue on the next physical line. Some commands may also have a data block at the end (currently, this data block may only contain Privacy Enhanced Message (PEM) [Lin93] encoded structures).

A parameter might accept arguments, separated by whitespace. The parameters are divided into four groups:

- Base parameters;
- Control lines;
- Environment lines;
- Request/response.

The groups must succeed each other in that order, but except for the base parameters, the order within the groups is unspecified.

We will go through the most important parameters and those relevant to the security architecture of IMS2.0 here.

- Base parameters:
  - `BEGIN` This line opens every command. It takes an optional parameter, namely the IMS version number. If not present, it default to version 1.0.
  - `MSG_TYPE` This parameter is required, and takes one argument. For version 2.0, the argument must be one of `COMMAND_REQUEST` and `COMMAND_RESPONSE`.
  - `MESSAGE_ID` A message id is required for all commands, and is required to be globally unique.
  - `REFERENCE_ID` If the message is a response to a prior request, the message id of the request is required to be included in this parameter.
- Control lines:

    - The primary objective of the control lines is to inform the recipient of the command where the response should be sent. An email or an ftp address may be specified here.

- Environment lines:
    - `TIME_STAMP` Every command must include a time stamp. The time is required to be in UTC on the format "YYYY/MM/DD HH:MM:SS".
    - `STA_LIST` All command requests are required to have the name of the station it is issued to listed in this parameter. These two latter lines are the request environment lines that are mandatory for all commands.
    - `CHAN_LIST` This parameter can be used to specify the chosen channel on stations with more than one channel.
    - Each command may specify more command-specific environment lines.

- Request/response:
    - The request/response line is the line specifying which request or response this particular command is. The C&C command types are listed below.

Command Types

The C&C commands are divided into a *core commands* group and station-specific commands. The latter are not specified as they vary with the hardware at the station.

The core commands for station control are divided into two groups: operational change and PKI management. For seismic stations[4], there is only one operational change command, and three for PKI management.

- `CALIBRATE_START` Used to calibrate the digitizers. A calibration of a seismic station means that a well known analogue signal is passed to the digitizer so the proper functionality of the digitizer can be verified.
- `GENERATE_KEYPAIR` Used to instruct an authenticator or the CRF to generate a new key-pair.
- `START_KEYPAIR` Used to instruct an authenticator or the CRF to switch to using a new key-pair.

[4]For hydroacoustic and infrasound stations, the same commands apply, for radionuclide stations, there are several other commands for operational change. The key management commands are similar for all stations.

- `UPDATE_CRL` Used to instruct the station (CRF) to update its CRL. The new CRL is included in the request.

The execution of commands operational change and those for keypair management involve interaction with the digitizer and/or authenticator.

Operational Change With a `CALIBRATE_START` command, it is possible to remotely tell the digitizer to adjust its interpretation of the analogue signals from the sensor.

A `CALIBRATE_START` command defines the following environment fields (in addition to the mandatory fields described above):

```
SENSOR yes|no
TYPE random|sine
CALIB_PARAM seconds [volts] | hertz seconds [volts]
```

And the request line

```
CALIBRATE_START
```

The sensor, type, and parameter lines constitute the values to which the digitizer is to be calibrated. If the station has more than one channel, the channel the calibration is intended for must be specified. It should be noted that a calibration command with the wrong values can result in the data from this channel being completely useless.

The station should reply with a `CALIBRATE_CONFIRM` to confirm that the command is received and acknowledge the time the calibration will start on (which may be a different time from the one requested). This response is required within one working day.

After the calibration is performed, a `CALIBRATE_RESULT` should be sent with the results of the calibration.

Key Management When a sensor and a digitizer/authenticator are initialized, they are lowered into their borehole, and transmission of data is begun. There are at least two good reasons why one does not want to take either out unless absolutely necessary. First, the sensors continuously collect data, and in case of a seismic event, it is important to have all the data. The second reason is that many of them are difficult to access, both because of being located in remote areas, and because the borehole itself may be deep and/or sealed.

Additionally, sending personnel from the PTS around the world to supervise key changes will require substantial resources. The solution is remote key management.

`GENERATE_KEYPAIR` is used to tell a station (the command can specify which channel to re-key if the station has more than one) to generate new key-pairs and create certificate requests. `GENERATE_KEYPAIR` can be followed by a PEM structure specifying the DSA (system) parameters to use for the new key-pair.

After key generation, a response must be sent with the new public key. The response, `KEYPAIR_GENERATED` contains the following fields:

```
CHAN_LIST channel_name
KEYPAIR_GENERATED
SIGNATURE signature
{a PKCS #10 self-signed certificate request}
```

The channel name is included if the station has more than one channel. `signature` is the new (public) key signed using the old key (the signature should be in hex format). This field may be omitted if the entire message is signed with the old key (this is intended for updates of the CRF keys only). The certificate request in PEM format output by the authenticator is then included (authenticators unable to output certificate requests may use the public key (dname, y, g, p, q) in hex format instead).

When the certificate request is accepted by the CA and the corresponding certificate issued, it is returned to the station in a `START_KEYPAIR` command. The command will acknowledge that it begins using the new key-pair with a `KEYPAIR_STARTED` response. These two commands are very straight-forward, and we do not give any more details here.

Security

As we have mentioned earlier, the `CALIBRATE_START` command has the ability to render the station (channel) unusable. Thus, a mechanism for preventing unauthorized commands from being accepted and executed at the stations is required.

IMS2.0 commands are anticipated to be sent to the stations using standard Internet email over the private GCI network. Some SOs will accept (C&C) commands directly from the IDC, while other will require them to relay through the NDC. Although many SOs are unlikely to accept calibration commands from any other source than themselves, the security of the protocol must be considered in the light that the commands are automatically executed at the station.

PrepCom has chosen an Internet standard, S/MIME[DHR$^+$98], to provide message authentication of the commands. Before execution, several authentication steps must be taken at the station.

The security of the IMS2.0 protocol rests on the following:

- The security of the S/MIME protocol, and
- that the receiver checks revocation status of the certificates, verifies the signature, and verifies that the signer is authorized to submit the command.

In chapter 6, our implementation of a receiver and parser for the IMS2.0 C&C commands is described.

4.4 The PKI

4.4.1 Infrastructure and Specifications

The IMS PKI has its focus on authenticity and not confidentiality. There will only be a limited use of encryption; the main goal is to provide authenticity to and integrity of sensor data and station commands using a digital signature scheme.

The PKI is divided into two security domains, one for encryption and one for digital signatures. Both have their own Certificate Authority, the RSA CA No1 and the DSA CA No1 respectively.

In the specific cases where encryption will be used, RSA is the asymmetric algorithm of choice. We will not go into details concerning the confidentiality part of the PKI. For digital signing, DSA algorithm will be used with SHA-1 for hashing.

CA keys will be of 1024-bit length, and shall have a lifetime of no more than 10 years. End-user certificates will have the same key length, but a lifetime of between 2 and 5 years. CRL Distribution Points (CDP) will be used to accommodate easy retrieval of the relevant CRL. No documentation specifying if CDP will also be used to split the CRL in several parts.

We can divide the end-users into the following groups:

IMS stations The stations need keys for signing data and signing command responses.

PTS Personnel Parts of the staff at the PTS will be authorized to submit commands to IMS stations.

Station Operators The SOs submit commands to their own stations.

Data consumers Data consumers will need certificates to sign data requests. This includes staff from the IDC.

4.4.2 Key Generation

All station keys are required to be generated and stored in hardware devices conforming to at least FIPS 140-2[Nat01b] level 3. CA and RA keys are required to meet level 4 of the same standard. Other end-users are required to protect their keys from abuse using software vaults and good passwords.

CA and RA The keys for the CAs and RAs are to be created (in the aforementioned hardware devices) during an official "key creation ceremony" at the PTS. The number of people present at this ceremony is not specified in documents available (to the author) at the time of writing, but it seems safe to assume that it will be adequate to instill trust in the integrity of the keys.

IMS Stations Each authenticator (both those for each channel (borehole) and those for the CRF) is to be assigned a *human device manager* (HDM). This manager (there may be more than one) will typically be affiliated with the SO. The key generation process is to be supervised by a representative from the PTS.

This representative (and possibly the HDM; the documentation does not specify this) signs the certificate request output by the authenticator and forwards it to the PTS where a panel of 5 RA operators will decide on whether or not to accept it. The decision must be unanimous. If accepted, the request is passed to the CA, which issues the certificate. After being issued, the certificate will be available to anyone through the means described in section 4.4.3.

Command Sending Entities The current documentation is not clear on how keys are to be generated and certificates issued to command issuing entities (including data requesters). It is suggested that certificates may be issued when the requester visits Vienna. Many people involved with the IMS will visit Vienna from time to time, and a face-to-face registration is possible.

4.4.3 Obtaining Certificates and CRL

Once a certificate is issued by the CA, it is available to anyone through several channels. The primary means of distribution is a repository with LDAP lookup. The IDC maintain an LDAP service hosted within the PTS where the certificates of all end-users and the CRL(s) are stored. This LDAP tree will be replicated outside of the internal network and other entities will use this replication.

Some IMS stations will be able to use LDAP. For them a copy of the CRL and the certificates will be kept on a local file system and will be made available by normal file transferring protocols.

Chapter 5

Threat Analysis

5.1 Introduction

This chapter presents a threat analysis for the IMS. We begin by identifying the assets in the system. The next step is to list the threats to those assets. We also briefly comment on the impact of some of these threats. Finally, we will give some motives different entities may have for attacking the system.

This is not a risk assessment; we will not discuss the likelihood of any of these situations happening.

5.2 Assets

The IMS system has one simple goal:

That the network of IMS sensors at all times record the data that corresponds to the actual physical conditions the stations are set up to monitor, and that this data is in an authenticated manner communicated to the members of the Treaty.

We say that we have one overall asset, which is the IMS data. We further define the following as the physical assets which provide the IMS data:

1. The boreholes (sensors, digitizers, authenticators);
2. The stations (CRF, communication links from CRF to boreholes);
3. The GCI;
4. The IDC (CA, data receiving hosts, data archive).

5.3 Threats

For each of the assets listed in the previous section, we list the major threats to that asset.

We classify the threats as either one that, if realized, results in data lost or being discarded, or one that results in potentially undetected data manipulation. If an attempted data manipulation is detected, the data will naturally be discarded and thus lost (i.e. if the manipulation was detected, not if the attack was prevented).

At the end of chapter 7, we will discuss each of these threats and how they are coped with by the proposed security architecture.

Borehole

1. Physical event

A physical event in or around the borehole may damage equipment or data. The sensors are sensitive instruments, and even a relatively minor disturbance (like a truck with a running engine close by) may make the data useless. A lightning strike, flood, or an event caused by a human (e.g. an explosion) can cause harm to the equipment.

Any such event may result in damaged data or loss of data.

2. Hardware, software, or firmware malfunction

The components of the sensor, digitizer, or authenticator may malfunction and produce other results than the expected. The cause of this can be an unintentional fault in the equipment, or a result of unauthorized tampering. Faulty equipment will probably result in loss of data, unintelligible data, or not authenticated data.

If the digitizer or authenticator has been tampered with, however, undetected forgeries may happen. One problem could be that the digitizer presents the authenticator with the wrong (but perhaps carefully chosen by an adversary) data, another that the authenticator generates private keys known to an attacker.

Such tampering may be performed either by a manufacturer, a replacement of the device during storage or transport, or by an attacker with physical access to the borehole.

3. Private key leak

If the private key from the authenticator is somehow leaked to any entity, that entity will have the possibility to sign data which will be accepted as authentic by the signature verifier.

Such a leak can occur because the authenticator, by design or accident, leaks the information, because it is known (by some) to choose private keys from a limited key-space, or because it is opened while out of operation and the key is revealed.

4. Access to signing function

If any entity other than the digitizer is able to access the signing function of the authenticator while the station is operating, that entity may substitute the real, signed data with a forgery, and have it signed. The forged data may then be accepted as authentic.

Station

5. Communications or CRF breakdown

In the event of a communication breakdown between the sensors and the CRF or a full or partial stop in services at the CRF, data may be lost, unless communication is restored before the digitizer/authenticator runs out of temporary storage space (the amount of data they can store will vary). Such a breakdown can be a result of natural phenomena (lightning, storm, etc.), hardware error, software error, a deliberate human attack, or human error.

6. Unauthorized command to borehole

An unauthorized calibration command sent to one or more stations can cause loss of data and even physical damage to equipment.

An unauthorized key management command can be used to damage data. If an adversary can trick the authenticator into changing its signing key, its signatures will no longer verify, and the data must be discarded.

A command may originate from the CRF or by a device tapped into the communications link with the borehole.

7. Manipulation of data

Data from the boreholes may be manipulated before transmission to the IDC. A third party or the SO may modify the data between the borehole and the CRF, or at the CRF.

A successful data manipulation can result in false data being accepted as authentic.

GCI

8. Unauthorized command to station

An adversary with access to the GCI can send (unauthorized) commands to all stations. The adversary may have access due to being a lawful participant in the GCI (an NDC or IDC staff or an authorized data requester), or following a break-in at a node of the GCI.

The command sent may be a modification of an existing command (with or without stopping the original), a resending of an old command, a re-transfer of a command to another station than its original destination, or a command created by the attacker.

The effects are identical to those in Threat 6, except that the attack may be extended to more than one sensor. It is important to note that such an attack initiated from the GCI will have to go through the CRF. It is also different from Threat 6 in that we here talk about IMS2.0 commands, while in Threat 6 a digitizer-specific (unsigned) command.

9. Manipulation of data

Sensor data (either CD1.1 *Data* frames or IMS1.0/2.0 data request responses) may be manipulated by an adversary with the ability to alter the traffic on the GCI. An attacker with the ability to send data on the GCI, but not alter existing data, may transmit data to the IDC in an attempt to have it accepted as authentic from a sensor.

Like Threat 7, a successful attack may result in false data being accepted as authentic.

10. Communications breakdown

If an attacker can render parts of or the entire GCI unable to transport data, this could result in a data loss.

PTS/IDC

11. Certificate issued on false or exposed private key

If a certificate is issued to an entity by the CA, but the corresponding private key is known by any other entity than the signing hardware (either because the key is leaked or because it is another key than the one used by the signing device), the data from that channel/CRF can not be trusted.

If a certificate is wrongly issued to an authenticator, it may directly result in forged data being accepted as authentic.

12. Certificate issued with wrong authorizations

If a certificate is issued with a misleading (wrong) DN or with access rights the end-user was not supposed to have, that user may be enabled to bypass access control at the CRF. This may in turn lead to loss of data and possibly damage to equipment through malicious calibration commands.

If the certificate has a DN normally representing an authenticator, but issued to another entity, this threat may lead to forged data being accepted as authentic from that authenticator.

13. Certificate removed from or not posted to CRL

If a certificate where the private key is compromised, or the certificate for some other reason is revoked, is removed from the CRL, damage may be done. Similar if a certificate which should have been revoked never is, or is revoked too late.

The possible impact of this threat depends on the nature of the certificate and the reason for the revocation in the first place. If it is a certificate for an authenticator, data forgeries made by an entity with access to the private part of the (revoked) key-pair may be accepted as authentic.

A certificate issued to an entity with permission to execute operational change commands at many stations (a PTS staff member) may be used to conduct a distributed remote attack using IMS2.0.

14. Incomplete signature verification

If the hosts receiving data do not properly verify signatures, that is in itself not a source of damage. The data will be archived along with their signatures, and they may be verified later. We have not found any demands that signatures must be verified by separate entities. It is assumed that the data will be analysed by other parties in addition to the IDC, but it is unclear how likely it is that these parties will take the time to verify the subframe signatures.

A faulty verification system may still lead to forged data that would otherwise have been detected pass as authenticated (in combination with another attack).

This threat can be caused by a compromise (an attack), a software bug, or an unfaithful employee. The hosts are exposed to the GCI, and can be attacked from there. The same goes for the two following threats.

15. Faulty data receivers

If the data-receiving hosts fail to properly handle or store the continuous data, there may be data loss.

16. Data receiver(s) breakdown

If the data receiving hosts break down completely (or at least enough to not receive data), data could potentially be lost. This could be due to a software fault, a hardware problem, a human error, or an attack.

17. Manipulation of data archive

If any entity can make changes to the archived sensor data, inquiries about the data in the future may suffer.

5.4 Impact

We will not consider the impact of each of the above threats individually, but make some more general comments.

5.4.1 Data Loss

Some data loss is unavoidable in the long run. Some stations will be hit by lightning or have hardware failures that cause loss of data. It will be an important goal to minimize loss, but complete coverage at all times is not a requirement. If data is lost (simultaneously) from many stations, however, the ability to detect and locate an explosion may be drastically reduced.

Thus, if a threat may cause data loss from many stations (either because it affects many stations or because it is remote and possible to exploit in a distributed fashion) it is more severe than one which will affect a single channel or station only.

5.4.2 Data Manipulation

The fear of undetected data manipulation is the prime reason for the entire security architecture. The ambition must be to make certain that *all* deliberate and accidental manipulation of sensor data is detected and the data rejected.

We consider every possibility of an undetected manipulation to be a grave breach of security. It is still necessary to point out that there are at least two factors that mitigate the impact of such an manipulation:

- *Geophysical security* (see 7.2), makes detection of inconsistent data possible, even though other defense mechanisms (like digital signatures) fail;
- The number and distribution of IMS stations may make forgery of data from one station inadequate if the signals travel far enough to be detected by another stations as well. Even though data from other stations may not be detailed enough to pinpoint the location of the explosion, they will identify the forgery.

5.5 Trust of Station Operator

An important, and to an extent unanswered, problem in the IMS security architecture is the trust placed in the SO.

For this discussion we will use the definition of *trust* from the X.509 ITU-T recommendation[Int97b] which is also used by [AL99]:

Entity A trusts entity B when A assumes that B will behave exactly as A expects.

From the history of the nuclear test-ban treaties (see section 2.6), we see that there has been a strong demand for preventing the host country from being able to undetectedly modify the data (or later deny being the source of it). The question is the separation of the SO and the host country.

It has been made clear from the PTS that the SO is trusted to faithfully operate his station(s). With "operate", we mean doing his best to make sure the station is operative and sending data.

In this thesis, however, we assume that the SO is *not* trusted with the integrity of the data. The verifiers (the Treaty members) must be convinced that the data received and accepted as authentic has not been subject to manipulation by the SO.

5.6 Motives and Resources

The reason for the existence of the Treaty is a ban of all nuclear test explosions. The main motive for attacking this system is thus to perform an *undetected nuclear test explosion.*

This is not the only potential gain an entity could have in attacking the system, though.

Discredit Nation If an entity wants to discredit a nation in the international community, it could use the IMS to do one of two things:

- forge data to indicate that the nation has performed a nuclear explosion, or
- destroy or modify the data from that nation's sensors.

The latter could arise suspicion that the country in question has (deliberately) tampered with its own data.

Topple the Treaty If a nation or an organisation is against the Treaty, it is conceivable that they may attempt to display its shortcomings to others by breaking security.

Destabilization Terrorist organisations or similar groups could use an indication of non-compliance with the Treaty to create destabilization in the international community.

Recognition For computer criminals, breaking into high-profile systems may lead to good reputation and recognition. Breaking the security of a nuclear test-ban treaty can become a target.

Resources The participating nations form the main group of potential attackers against this system. Nations with an important motive (e.g. performing an undetected nuclear test) must be considered to have the highest resources possible available. The threats to the system must be seen in that perspective.

Chapter 6

Implementation

6.1 Introduction

The IMS network requires software solutions to implement the designed security scheme. One is the verification scheme for commands sent to stations.

As a part of this thesis, an implementation of a system for receiving, verifying, and executing commands on IMS stations has been developed. This implementation was developed for the NORSAR NDC. The application is called `imsparse`. The focus in writing the application has been to implement access control and message authentication.

6.2 Scenario

For PKI component management (keys and CRL) and operational change, the IMS stations will use the IMS2.0 protocol.[1] This command structure has two distinct uses:

- Data requests and subscriptions, and
- command and control (C&C).

For data transfer, an application called AutoDRM is used throughout the IMS network. Our implementation is solely for C&C of the stations.

Commands will be received by signed S/MIME[DHR+98] messages. They will be received at the Central Recording Facility (CRF) of the station. Before executing a command, the CRF must verify the signature. A command

[1]As we described in chapter 4, primary stations are not required to be able to receive IMS2.0 command, and instead receive their C&C messages as CD1.1 requests. `imsparse` does not currently handle that scenario, but it should be simple to write an input function that can read the embedded IMS2.0 commands from the CD1.1 receiver.

must never be executed unless the signer is an authorized issuer of the command in question.

6.3 Requirements

Upon receiving a signed email, the application must do the following:

- Obtain the signer's public key certificate;
- Verify the validity of the certificate (verifying that the certificate is not expired, that the CA's signature on it is correct, and that it has not been revoked);
- Perform access control by verifying that the signer of the command is authorized to issue it at this station.
- Verify the signature on the email;
- Verify that the command is properly formatted and recent;
- Verify that the values in the command are reasonable;
- Execute the command;
- Send the required response to the return address;
- Provide a full log of the received command and the actions taken.

An infrastructure is required to carry out these steps. To verify the validity of the certificate (and subsequently the signature), we need the CA's certificate installed. We also need a recent, valid CRL to check the certificate in question against. Thirdly, access control is required. Depending on how we wish to solve the access control problem, we may need to protect an ACL as well.

6.4 Data Structures

Before describing the implementation, we describe the protocols and data formats involved:

X.509 PKI certificates are represented by X.509 data structures. The standard is described in RFC 2459[HPFS02].

X.509 CRL CRLs are of the X.509 CRL format, described in the same RFC as the certificates.

S/MIME and PKCS #7 Secure Multipurpose Internet Mail Extensions [DHR$^+$98] are used to provide an authentication mechanism to standard Internet email[Pes01]. S/MIME is an adaption of the Public-Key Cryptography Standards (PKCS) #7[Kal98b].

Abstract Syntax Notation 1 (ASN.1)[Int98], is used to describe these formats.

6.4.1 X.509

X.509 is an offspring of the X.500 directory service[Int97a]. It originates from ITU[Int97b], and is also adopted as an Internet Standard in RFC 2459[HPFS02].

Syntax of X.509

The RFC defines an X.509 certificate in ASN.1 syntax as described below:

```
Certificate ::= SEQUENCE {
  tbsCertificate       TBSCertificate,
  signatureAlgorithm   AlgorithmIdentifier,
  signatureValue       BIT STRING
}

TBSCertificate ::= SEQUENCE {
  version              [0] EXPLICIT Version DEFAULT v1,
  serialNumber         CertificateSerialNumber,
  signature            AlgorithmIdentifier,
  issuer               Name,
  validity             Validity,
  subject              Name,
  subjectPublicKeyInfo SubjectPublicKeyInfo,
  issuerUniqueID       [1] IMPLICIT UniqueIdentifier OPTIONAL,
                       -- If present, version shall be v2 or v3
  subjectUniqueID      [2] IMPLICIT UniqueIdentifier OPTIONAL,
                       -- If present, version shall be v2 or v3
  extensions           [3] EXPLICIT Extensions OPTIONAL
                       -- If present, version shall be v3
}

SubjectPublicKeyInfo ::= SEQUENCE {
  algorithm            AlgorithmIdentifier,
  subjectPublicKey     BIT STRING
}
```

```
Extensions ::= SEQUENCE (1..MAX) OF Extension

Extension ::= SEQUENCE {
  extnID                   OBJECT IDENTIFIER,
  critical                 BOOLEAN DEFAULT FALSE,
  extnValue                OCTET STRING
}
```

We have left out some of the ASN.1 types, but they should be self-explanatory. See the RFC for a detailed description.

A few notes on the fields:

- Version. v1, v2, and v3 are currently defined. v3 should be used when one or more extensions are needed, otherwise v1 or v2 are required. The difference between v1 and v2 is the inclusion of the subjectUniqueIdentifier and issuerUniqueIdentifier in v2.
- Serial number. The serial number must be unique within the CA's domain, and shall never be re-used in the lifetime of the CA. The serial number is the unique identifier of the certificate.
- Signature. The algorithm identifier of the algorithm used by the CA while signing this certificate. This field must match the identifier supplied outside of the data block.
- Issuer. The DN of the CA issuing the certificate. This field can not be empty.
- Validity. Two dates, one describing the first date and one the last date this certificate is valid. Both must be present.
- Subject. The DN of the owner of the certificate. This field may be empty, but only if the Subject Alternative Name extension is included (and marked critical and non-empty).
- Subject public key info. The public key of the subject, namely the identifier of the algorithm as well as any parameters (e.g. p, q, and g with DSA), and the key itself (y in DSA).
- Unique identifier of issuer. This field is optional and only defined in X.509 v2 and v3; it is a bit string identifying the issuer.
- Unique identifier of subject. Analogous to the previous field, but identifying the owner of the certificate.

- Extensions. This field is only defined in v3. If it is present, it contains a sequence (at least one) of certificate extensions on the following form:
  - Extension ID. A unique identifier of the extension. If, in a PKI, there is a need for custom extensions, an ID must be defined.
  - Critical. True or false. If an extension is marked critical, every implementation is required to understand and process it, and to reject the certificate otherwise. Non-critical extensions may be ignored.
  - Extension value. The value as an octet string.

The data block is signed and the signature and the algorithm identifier is included in the certificate.

Extensions

Although many end-user certificates do not need any extensions, some do. All CA certificates do, in order to allow a client to distinguish between an end-user and a CA certificate. It is possible to add custom extensions. Several extensions are defined by the RFC, we describe four of them here:

Basic Constraints If the Basic Constraints extension is present, it must be marked critical. The extension allows two values to be set: CA constraint and Path Length Constraint. Both are optional, the former has a default value of false, the absence of the latter says there is no limit to the length of the tree spanning from this CA. If the CA constraint is set to true, that means this certificate is a CA certificate and is allowed to issue end-user certificates. The Path Length says how many intermediate CAs are allowed below this CA, a value of 0 means the CA is allowed to issue end-user certificates only.[2]

[2]The lack of verification of this extension is the source of a very serious bugs in Microsoft's Internet Explorer versions 5.0, 5.5, and 6.0 (the deficiency was actually in Microsoft's CryptoAPI, so several other applications were affected as well; the browsers and the application that verifies signatures on security updates etc were the most serious) all failed to properly verify the value of the Basic Constraints extension (IE6 did actually deny verification if the Basic Constraints field was present and set to false, but this does not help much as it is uncommon to include the field in end-user certificates). The effect of this vulnerability is that anyone with a (end-user) certificate issued by a CA accepted by IE (and which has a Path Length of minimum 1) was able to act as an intermediate CA and issue his own end-user certificates. These will in turn be accepted by IE as authentic. It is easy to see how this failure can result in rendering secure browsing by affected browsers completely insecure. This incident is a good demonstration of how dependent a PKI system is on good implementations. See [Ben02] and [Mic02] for more information on this bug.

Key Usage The Key Usage extension allows the CA to issue certificates intended only for specific purposes, for example digital signatures only, or encipherment only.

Subject Alternative Name This extension allows the issuer to specify an alternative name for the subject. As mentioned above, the subject field of a certificate may be empty, in that case this extension must be set (to a non-empty value) and marked critical. This extension allows names for the subject ranging from DNS names to IP addresses or email addresses.

CRL Distribution Points CDP is, as mentioned in chapter 3, implemented as an extension on the certificates. The extension is a list of locations where the CRL is (supposed to be) available. All locations should point to the same CRL.

6.4.2 Syntax of X.509 CRL

The ASN.1 syntax of an X.509 CRL is as follows:

```
CertificateList  ::=  SEQUENCE  {
  tbsCertList             TBSCertList,
  signatureAlgorithm      AlgorithmIdentifier,
  signature               BIT STRING  }


TBSCertList      ::=  SEQUENCE  {
  version                 Version OPTIONAL,
                              -- if present, MUST be v2
  signature               AlgorithmIdentifier,
  issuer                  Name,
  thisUpdate              Time,
  nextUpdate              Time OPTIONAL,
  revokedCertificates     SEQUENCE OF SEQUENCE  {
       userCertificate         CertificateSerialNumber,
       revocationDate          Time,
       crlEntryExtensions      Extensions OPTIONAL
                             }  OPTIONAL,
  crlExtensions           [0] Extensions OPTIONAL }
```

The fields are mostly self-explanatory. The extensions are similar to certificate extensions. Notice that an extension may be both on a specific (revoked) certificate or on the entire CRL. The nextUpdate field tells the users when the CRL no longer is valid. Delta CRLs are implemented as a CRL extension indicating the CRL number of the base CRL. The base CRL must include an extension stating its CRL number.

6.4.3 PKCS #7

PKCS #7 version 1.5 is described in RFC 2315[Kal98b]. It is a standard for handling data that may have cryptographic functionality attached to it. The standard also permits enveloping of data so that data can be first signed and then encrypted, signed by several parties, doubly encrypted, or otherwise wrapped.

The RFC defines six content types divided into two content type classes; base and enhanced. The base class is intended for raw data, and contains only one of the content types: data. The enhanced class contains the other five: signed data, signed and enveloped data, enveloped data, digested data, and encrypted data.

In our applications, the use of PKCS #7 is limited to a regular data content within a signed data type. We will explain all the types, but our focus is on the signed data type.

Data. Data of this type is just an octet string.

Signed data. This type consists of an item of any type and zero or more "encrypted message digests". Each encrypted digest [3] is a digital signature for one signer. For each signer, a SignerInfo is collected, including the encrypted message digest, the digest algorithm identifier, and the ID of the signer's certificate (DN of issuer plus unique (within the issuer's domain) serial number of certificate) plus some optional attributes.

Enveloped data. With the enveloped data content type, data of other types can be encrypted for any number of recipients. An enveloped data item consists of a content-encrypting key, encrypted with the recipients' public keys, and the contents, encrypted with the content-encrypting key.

Signed and enveloped data. This type is, as the name indicates, a combination of the two previous types.

[3]The RFC describes the process of creating the signature as "message digest and associated information are encrypted with the signer's private key". We have called, and will continue to call, this key the "signing key" and the creation of the signature as "signing the digest", as it may be fundamentally different from encrypting, see section 2.3.5 for details. Throughout the description of PKCS #7, we will for simplicity use the same terminology as the RFC does.

Digested data. This type is intended for use with data to add integrity to the content. It is simply the content, the identifier of the chosen digest algorithm, and the digest, produced by applying the content to the digest algorithm.

Encrypted data. Similar to enveloped data, this type is designed for encryption. The difference is that keys are not managed.

The ASN.1 syntax for a Signed Data PKCS #7 structure is as follows:

```
SignedData ::= SEQUENCE {
  version               Version,
  digestAlgorithms      DigestAlgorithmIdentifiers,
  contentInfo           ContentInfo,
  certificates          [0] IMPLICIT
             ExtendedCertificatesAndCertificates OPTIONAL,
  crls                  [1] IMPLICIT
             CertificateRevocationLists OPTIONAL,
  signerInfos           SignerInfos
}

DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier

SignerInfos ::= SET OF SignerInfo

SignerInfo ::= SEQUENCE {
  version               Version,
  issuerAndSerialNumber IssuerAndSerialNumber,
  digestAlgorithm       DigestAlgorithmIdentifier,
  authenticatedAttributes
                        [0] IMPLICIT Attributes OPTIONAL,
  digestEncryptionAlgorithm
                        DigestEncryptionAlgorithmIdentifier,
  encryptedDigest       EncryptedDigest,
  unauthenticatedAttributes
                        [1] IMPLICIT Attributes OPTIONAL
}

EncryptedDigest ::= OCTET STRING
```

6.4.4 S/MIME

S/MIME is an attempt to answer the requirement for secure email. Standard Internet email, as defined by RFC 2822[Pes01], is limited in what it accepts.

Lines may not exceed 1000 characters, and only 7 bit ASCII is allowed. To allow sending binary data, like a digital signature, over standard email, MIME, Multipurpose Internet Mail Extension[FB96], was developed, and has been a working standard for many years.

S/MIME is basically PKCS #7 and MIME combined. MIME-types for signed, encrypted, etc. messages are defined (e.g. multipart/signed).

A multipart/signed S/MIME message includes *one* MIME-part that is the signed part, and one for the signature (optionally including certificates).

An important aspect with S/MIME is that the *mail headers* (To, From, Subject, etc.) are *not* signed.

6.5 Security Considerations

`imsparse` will run at the CRF, which is defined as a restricted environment. Access to the CRF should be limited to the SO(s).

6.5.1 Trusted Computing Base

The *trusted computing base* (TCB) is the parts of a computer system that provides a secure environment.

The following external environment is defined as the TCB for `imsparse`:

- All security-critical functions relevant to signature verification from the cryptographic library used by `imsparse`.
- The CA certificate.
- The parts of the C library used in security-critical functions in `imsparse` and the cryptographic library.

We must assume that the `imsparse` binary is not modified in an unauthorized way. We also assume that the private keys used to sign the ACL (if used) and the configuration file are accessible only to trusted users.

6.5.2 Logging

It is important that the command handler provides logs for the SO and/or the PTS about the events it observes. `imsparse`'s primary logging tool is the UNIX `syslog` utility. Additionally, it will log to local files.

`imsparse` is not an application that will run constantly, only when called upon to handle an incoming command (email). `imsparse` will thus only log an event that triggers it. On every occasion when `imsparse` is called to read an incoming message, it will write to the log.

Additionally, certain events are considered *auditable events*, and will be logged. In all cases, we log the current time as well. Every incoming email that is fed to `imsparse` should be securely stored in another location as well. This can easily be accomplished by forwarding the email to several locations.

- An unsigned email will be logged as such and the execution terminated.
- In the case of a bad signature, the incident will be logged.
- A good signature, but a failure in the access control verification will cause the attempted command and the DN of the signer to be logged.
- A command from an authorized sender which is rejected because of a too old time stamp or a reused message id will be logged with time stamp and message id.
- An otherwise accepted command which is rejected because the values in it (this applies to operational change commands only) are outside of the limits required by the SO is logged (the entire command written to file).
- Any executed command will be logged in full. The results of the execution will be logged by the specific command parser.

6.5.3 PKI Components

Two PKI components are needed on the CRF: the CA's certificate and the CRL. We can retrieve the CRL from LDAP. It may also be transferred to the station using the `UPDATE_CRL` command.

If, due to an attack or an accident, the station is unable to retrieve the latest CRL and the CRL it has is too old, it is imperative that *no action* is taken on the command, except possibly storing it for processing when a valid CRL is obtained.

The CA certificate is part of our TCB. Having access to an authenticated copy of the CA's certificate is a prerequisite for performing a trustworthy signature verification.

6.6 Access Control

A major focus of this implementation is on access control (AC) ; how to restrict execution of different commands to authorized entities only, is a challenging task.

In chapter 2, we gave two specific examples on AC mechanisms: *access control lists* (ACLs) and *certificate-based access control.* We have implemented support for ACLs and we outline how certificate-based AC will be implemented.

In designing a system for AC in our scenario, decisions must be taken on the following questions:

- How detailed to make our access control.
- Which (text) format to use in the certificate extensions and in the ACL.
- If using ACL, how to insure that the ACL is authentic?
- By whom the AC is managed.

Granularity Our scenario is that we have named entities (named by the distinguishing name on their certificate) requesting permission to run a command on our station.

A station may have several sites (sub-arrays), and each site several channels (sensors). We must therefore make a decision on the level of granularity.

There are three distinctions to be made:

- Whether access is granted for a subset of the commands or always all or none,
- whether it is granted for one or more channel or always for all or none, and
- whether the parameters accompanying the command are significant for whether or not access is granted.

The first problem is easy to answer: we must separate between commands. It is explicitly stated that the PTS is the only authorized source of key management commands. The SO, however, will normally be authorized to submit operational change commands to its own stations.

The second problem is not equally obvious. It is given that access can be granted to only a subset of the *stations*, but the question is whether it is useful to allow an entity to issue commands to only *parts* of a station. Our decision has been to always grant access to all channels or not at all for a given command. We assume that an entity will not be granted privileges at only a part of the station.

The third question is often a difficult one. We could picture that an entity was allowed to perform *some* operational change on a station, but not all. There are, however, no commands in the IMS2.0 specifications that open for this. The decision is therefore to always either fully allow or disallow an entity to run a specific command.

Format The format chosen to define the access control is important. If ACLs are used, it is relatively simple to update them to reflect the addition of a new station, a new command, or a change in privileges, but with a certificate-based solution, such a change will imply that a new certificate must be issued to everyone affected.

There has been no decision from the PrepCom on how to implement access control, and hence there is no previously defined format for ACLs/certificate extensions.

In the `STA_LIST` (and channel/site list) parameter in IMS2.0, the station code may be given as a string where using '*' as a wildcard is allowed. The string `noa` would represent only the station `noa`, while `no*` would cover all stations starting with `no`.

This format may be used for access control as well, but it is not much help. There is, to the author's knowledge, no defined *group names* for the stations. An SO will often operate several stations, and have the same privileges on all of them. The naming scheme does not allow the wildcard to be used to represent all of the SO's stations (and no other stations).

We have decided to use the following simple format to represent access level:

```
station,station,...:command,command,...
```

Here, `station` may include wildcards, and command may be a specific command (`CALIBRATE_START`, `UPDATE_CRL`) or a *command group.* We define the following command groups:

- `OPCHANGE`: `CALIBRATE_START`
- `KEYMANAGE`: `GENERATE_KEYPAIR`, `START_KEYPAIR`, `UPDATE_CRL`.

Additionally, a group for data requests can be defined.

An ACL would then have lines beginning with a DN followed by a delimiter and the access control string. Each entity may be affected by several lines. When using certificate extensions, the extension value would be a number of lines each containing one of these access definitions.

An SO controlling stations `sta, stb`, and `stc` will typically have

```
sta,stb,stc:OPCHANGE
```

A PTS staff member authorized to perform key management on all stations will have

```
*:KEYMANAGE
```

Authenticity When using ACLs, as stated in chapter 2, the (message) authentication of the ACL is vitally important. We will discuss methods for protecting our ACL against unauthorized changes later.

Certificate-based solutions do not require additional authentication, since the certificates are already signed by the CA.

Management To have AC, it must be *managed.* An entity must be trusted with the task of deciding who has access to what. There can not be more trust in the AC than in the managing entity, so choosing this entity is an important task.

The contract between the local SO and the IDC will describe what entities are allowed to perform which actions. It is likely that the AC management can be placed with the PTS, in cooperation with the SO. Alternatively, since the SO is trusted to faithfully run the station, the responsibility of AC management may be decentralized.

6.6.1 Access Control List

Our focus is on how to protect the ACL from unauthorized manipulation, and to allow the applications and users needing to consult the ACL to verify its integrity. As opposed to file access control, which is inherent in the OS, our ACL will be implemented on top of it.

Alternatives to said implementation could be a file in the directory hierarchy, an entry in a directory service or a database. Our focus will be that the ACL, regardless of the choice of low-level implementation, can be subject to unauthorized change. Technical personnel in the PTS have stated that the ACL (and certain configuration files) must be “protected against any unauthorized external manipulation”.

Plain data One obvious possibility is to keep the ACL as plain data. The advantage is that administration is very simple, but there is a massive drawback: there is no security apart from that offered by the OS (e.g. by making the file read-only).

Hash Verification An alternative supplement to the plain data ACL, is to add a checksum verification. If we are able to either protect the configuration file or to compile the checksum of the ACL directly into the binary (we assume the ACL is changed relatively rarely), a simple comparison of the hashes—produced by a strong, cryptographic hash function—will provide integrity.

A hash provided at compile-time only is inflexible. However, since the configuration file must be equally well protected, it makes sense to include the ACL's checksum there.

Signature Verification Another alternative is to have the ACL signed by a responsible entity (the PTS or the SO) and let the command interpreter verify the signature on the ACL before accepting it. This adds good security. The advantage of demanding signed ACLs, is that access control management can be placed where it is desired. It may be desirable to let the SO alter the configuration file, but not the ACL. In that case, ACL signing is the best choice.

If the SO is trusted to maintain the ACL, signing the ACL is functionally identical to using hash verification.

`imsparse` has implemented verifying ACL authenticity both by using signatures and hash (in config file or compiled in the source).

6.6.2 Certificate-based Access Control

The obvious way to implement a certificate-based access control system in the IMS system, is to use certificate extensions.

As we have seen, it is possible with X509 certificates version 3 to add extensions to the certificates. If this technique is used, we will not need to maintain ACLs; it will be adequate to check on signature verification that the signing certificate is issued with the required privileges.

It is an attractive approach, but requires central management. We will now have the certificate to a greater extent signify the *role* of the owner, which is a nice side-effect.

Interoperability If certificate extensions are to be used, there is another well-known problem which must be discussed: Interoperability. Recall from section 6.4.1 that an extension on an X.509 certificate can be marked either critical or non-critical. If the extensions used to enable authority verification on the IMS stations are marked critical, the consequence is that these certificates are incompatible with other applications.

The certificates issued to command issuers in the IMS PKI are meant to be used for that purpose only. Using a critical extension can be a way of stating that these certificates are for IMS commands only. If the certificates are intended for any actions other than submitting commands to stations, the extension should be marked non-critical.

6.7 Integrity of Configuration File

As mentioned earlier, it is required that the config file as well as the access control mechanism should be protected from tampering.

In `imsparse`'s case, the configuration file includes several parameters important to security:

AC The path to and method of access control is defined here. If the config file was not protected, though, this could easily have been moved to the source (i.e. using signed ACLs and have the DN of the authorized signed in the source).

CA certificate The location of the CA certificate may be in the config file.

Limits The limits for which values are acceptable for different operational change parameters are set here.

Command handling The application to run or file to write to after successfully parsing a command is in the config file.

The alternatives from the discussion on integrity of the ACL apply to the configuration file as well. The applicability of the methods differ, though. While the ACL can be expected to be very static, and cause little problem if compiled into the binary, configuration files may change relatively often. There is no real difference between compiling in the hash of the configuration file and compiling in the entire configuration file.

A better alternative is to include the DN of a trusted entity in the source. This entity should be the local operator or the IDC/PTS. The utility can then be instructed to accept only configuration files when also presented with a (valid) digital signature, signed by the entity specified at compile-time. In this case, the certificate of the CA which issued the certificate to the config file signer (which may be a different CA than the one used for verifying command issuers' certificates) must also be specified in the source.

This latter approach appears to be the most attractive, providing good flexibility and low management costs. It is also the default way to protect the config file in `imsparse`.

6.8 Tools

`imsparse` is a relatively simple application, and currently requires only a single third-party software (in addition to the standard C library), namely a cryptographic library.

The demands on portability are not clearly stated, and our decision has been to at least make certain the application runs under Solaris and Linux.

6.8.1 Programming Language

Scripting languages like Perl or Python could have been a good choice due mainly to their simplicity in string manipulation. They are not pre-installed on many older platforms, though. C++ would have offered object orientation, and also a more robust interface to strings and string manipulation, but again availability was a concern.

C was an obvious alternative. It is always there, the author was already familiar with the language, and it is “standard”. The need for object orientation was not considered to be that great, and the problems of parsing the command structure not anticipated to be overwhelming.

It had been indicated from several sources that limiting the number of patches and packages required to be installed at the host would be an important goal. This was a good argument for C, which was also our decision.

6.8.2 Third-party Software

A library for cryptographic functionality is required, and the host running `imsparse` would also benefit from being able to obtain data from LDAP, but this is not yet implemented.

There are several cryptographic libraries available. We considered the following:

libgcrypt A GNU library, based on the code from GPG, the GNU Privacy Guard[Koc].

cryptlib A cryptographic library available for many platforms maintained by Peter Gutmann[Gut].

OpenSSL A widely used library implementing both standard cryptographic functionality, and the SSL/TLS protocol. Developed by a team of volunteers, and available on many platforms[The].

The most important criteria for us, were

- availability on relevant platforms,
- support for relevant cryptoalgorithms, and
- documentation.

When choosing a library for cryptographic operations, efficiency is often an important factor. For this task, however, it is not. This is because we do not anticipate very many commands, and certainly not enough to make verification of an S/MIME signature a bottleneck.

As we require only DSA signature verification and creation (we need to sign the replies), X.509 certificates (v3) and CRLs (v2), and S/MIME (PKCS #7) support, our demands for algorithm and protocol support did not rule any library out.

libgcrypt `libgcrypt` is a GNU tools. As most GNU tools, it is possible to compile on most UNIX systems. It is not available as prepackaged binaries for Solaris, though. It comes with some documentation, but not very extensive.

cryptlib `cryptlib` is well documented, and also with a high availability. Its license states that it may be freely distributed with open source projects, but another, more restrictive, license applies for commercial applications. `imsparse` is not intended to be a commercial application, so this is not a problem. `cryptlib` is highly portable, and is available for all relevant platforms, but does not come prepackaged for Solaris.

A 270-page comprehensive user's manual is supplied.

OpenSSL OpenSSL is a free cryptographic library implementing Secure Sockets Layer (SSL) version 2 and 3 and Transport Layer Security (TLS) version 1 as well as providing access to most widely deployed cryptographic algorithms.

OpenSSL's low-level functions are not intended for developer use, and not well documented, but the high-level functions—which we will primarily be using—are adequately documented. Additionally, the book "Network Security with OpenSSL"[VMC02] from O'Reilly provides many examples and guidelines.

OpenSSL is available as packages for Solaris and Linux. It is also available for Windows. The library is used in many wide-spread applications.

A problem with OpenSSL is that prior to version 0.9.7 (which was released on New Year's Eve of 2002), it had no automatic CRL checking. Starting with version 0.9.7, OpenSSL can be told to check the revocation status of all certificates before verifying a signature.

Decision We decided to go with OpenSSL. The main reason was the author's familiarity with the library; another that it is available as a ready package for Solaris (which is likely to be the most used platform). `cryptlib` was a good alternative.

When `imsparse` was started, OpenSSL 0.9.7 had not yet been released. A mechanism for checking revocation status was therefore developed. When the functionality emerged in 0.9.7, we decided to drop the support for 0.9.6 instead of maintaining the revocation code ourselves.

6.9 Program Design

There are several different models of digitizers and authenticators used in the IMS. These devices do not share a common command language, so writing a portable application that can communicate with every different model is time-consuming. Our decision has been to design `imsparse` to output the contents of the command it received in a configurable manner, and the to optionally run another program.

The command response will be generated and sent by this second application, as the results will not be available until it has been run.

6.9.1 Program Flow

`imsparse` is a C program comprising some 5000 lines of code. We will now describe it step by step, using pseudo-code and English.

```
main() {
  signer = verify_file_signature("/etc/imsparse.conf",
                                 "/etc/imsparse.conf.sign",
                                 cadir);
  if (! authorized_config_signer(signer)) {
    exit();
  }
  contents = read_signed_input(stdin);
  signers = verify_smime_signature(contents, cadir);
  if (!handle(contents, signers))
    exit(1);
  else
    exit(0);
}
```

`main()` first verifies and reads the configuration file before reading the input. The signed contents and the DNs of all signers (it is uncertain if the IMS2.0

protocol will allow more than one signer, but we do; in that case it is enough that one of them is authorized to execute the command).

The `cadir` variable must at first be compiled into the source. It is an `OpenSSL`-style directory where both the CA certificate and the CRL is located.

The OpenSSL functions for reading and verifying PKCS #7 signed material are used.

```
int
handle(contents, signers) {
  struct command comm;
  for (line in contents) {
    parse_line(comm, line);
  }
  if (!syntax_ok(comm)) {
    /* Syntactically wrong */
    return 0;
  }

  if (! match_station_code(comm->station, conf->station_code)) {
    /* Wrong target for command */
    return 0;
  }
  if (! access_control(signers, comm)) {
    /* Signer(s) not authorized to submit command */
    return 0;
  }
  if (! verify_recent(comm)) {
    /* Command too old */
    return 0;
  }
  if (mid_used(comm->mid)) {
    /* Message-ID already used */
    return 0;
  }

  switch (comm->type) {
  CALIBRATE_START:
    return handle_calibrate_start(comm);
  UPDATE_CRL:
    return handle_update_crl(comm);
  /* etc... */
  default:
    /* Command unknown */
    return 0;
  }
}
```

The `handle()` function parses all the lines of the command (actually, all the signed data). When finished, it verifies that the command is syntactically correct before insuring that:

- The command is intended for this station;
- The signer(s) are authorized to execute the command (at this station);
- The command is not a replay (it is not older than 24 hours, and the message id is not in our logs of previous ids).

```
int
access_control(signers, comm) {
  char **aclines;
  switch (config->ac_method) {
  ACL_FLAT_FILE:
    ac_acl_flat_file(config->acl_file, signers, aclines);
    break;
  /* Future work: */
  ACL_CERTIFICATE_EXT:
    ac_cert_ext(config->ac_extension, signers, aclines);
   break;
  /* etc... */
  }

  if (match_ac(aclines, config->station, comm->type))
    return 1;

  return 0;
}
```

The idea with the access control function is to use either method, ACL (either from file, database, or some other backend) or certificate extension, to obtain all the access control statements concerning the signer(s) of the command. These lines are then finally investigated to see if they match the station and command type. The format of the access control statement lines are as described in 6.6.

```
int
```

```
handle_calibrate_start(comm) {
  /* Extract values from comm structure and verify
     that they are reasonable.  The values which are
     considered reasonable are in the configuration.
  */

  if (! values_ok(comm))
    return 0;

  /* Write a line representing the command values to
     this variable.  The format can be configured.
  */

  char *calstline = write_calibrate_start_line(
                      config->calibrate_start_format,
                      comm);

  char *run  = config->calibrate_start_run;
  char *file = config->calibrate_start_file;

  if (file)
    write_file(file, calstline);
  if (run)
    ret = run_cmd(run, calstline))

  /* If the values were okay and the run_cmd was
     /NOT/ unsuccessful (it may not have been run,
     and that would be okay, we return 1.
  */
  if (ret != 0)
    return 0;
  else
    return 1;
}
```

The `handle`-functions for the different commands primarily consist of checking that the values are sane (most important for operational change) and writing a representation of the command to a customizable line. At this point, all access control and authentication have been carried out, and there is the matter of actually executing the function.

6.10 Future Work

`imsparse` is a functional application, but with more time, some additional features would have been added.

- Support for all IMS 2.0 commands.
- Support for fetching the new CRL when it is outdated.
- Support for fetching missing certificates.
- Capabilities for sending acknowledgement response.
- Support for certificate-based access control.

All IMS2.0 Commands So far, our implementation accepts `CALIBRATE _START` commands only. As this application is mainly written to demonstrate one part of the security architecture at the stations, supporting all the commands was not a priority. Adding support for the key management commands would not show any new thoughts or solutions.

Fetching CRLs and Certificates `imsparse` has no `LDAP` capabilities, nor can it make `ftp` or `http` connections.

If an S/MIME command does not contain all the required certificates, `imsparse` will not be able to verify the signature. The PTS has indicated that that could be solved by requiring the signer's certificate to always be included with the S/MIME message.

CRLs, however, are a much more important issue. We *must* have an up-to-date CRL when verifying a signature.

If `UPDATE_CRL` commands are sent regularly(i.e. every time a new CRL is posted), there may not be a use for CRL retrieval. This is not likely to happen for some time, though.

The simplest approach is to use an external program. `imsparse` has the framework for supporting running an external application to get the newest CRL, but the support is not finalized.

Sending Acknowledgment Response Currently, `imsparse` is not capable of responding to the command. Creating support for the initial acknowledgement should does not demand a structural change. It does, however, demand support for using the cryptographic hardware token the CRF uses for signatures.

Support for Certificate-based Access Control Currently, our implementation only accepts an ACL for access control mechanisms. As indicated, we intend to use the same format for the extensions as we use for the ACL. The reason this has not been done, is that time was limited and setting up a CA and issuing certificates for testing was required.

Adding this support will need a minor change in the program design to allow the access control code to get the contents of the certificates as well as the DNs.

An important aspect which must be kept in mind when implementing support for certificate-based access control is that it is possible to include arbitrarily many certificates in the PKCS #7 structure. The access control software must consider access rights from the certificates corresponding to the keys used to sign the command only.

6.11 Experiences

This application grew from initially not being included in the thesis at all to become an important part of it.

With the `CALIBRATE_START` command, almost every part of the parsing is critical to security, because a misinterpretation of a command parameter can cause problems. For the other commands, a more explicit division of the application in security-critical parts and non-critical parts would have been useful.

The use of C as the programming language has worked well. The biggest problem has been the low-level parsing of the commands, where C's string handling made the functions messy. The memory debuggers `valgrind` and `purify` have been used to assist the memory management. Further, string functions less prone to buffer overruns (e.g. `snprintf` instead of `sprintf`) have been used in an attempt to avoid exploitable bugs.

OpenSSL has worked well. The library is complicated to use for low-level cryptographic and/or mathematic (i.e. the bignum-part of the library) operations, but the higher-level functions we have used are straight-forward and well documented.

Due to the lack of testing environments, `imsparse` has only been compiled and run on Solaris (5.7 and 8) and Linux (different distributions).

Chapter 7

System Analysis

This chapter discusses the quality and appropriateness of the security architecture proposed by the PrepCom. We first comment on two aspects we have more or less ignored in our analysis: other verification mechanisms and legacy systems.

The next, and important, step is to analyse the security of the two communication protocols used in the system, IMS2.0 and CD1.1. We then discuss the problems of physical security on the stations, and in particular its application to key management.

The two last sections discuss the quality of the PKI solution and finally how the system copes with the threats we defined in chapter 5.

7.1 Security Goals

There are two separate security goals in the IMS network. The first goal is to prevent *sabotage of the system.* The second is to *prevent undetected manipulation of the data.*

An aspect which separates them is the role of the SO. In reaching the first goal, the SO is an important ally; in reaching the second he is a potential threat. The receiver of the data must be able to be confident that the received data is the exact same data the digitizer received from the sensor (message authentication). The source of the data (the station, represented by the SO) must later be unable to claim that the data must be a forgery (non-repudiation).

Proposed Security Architecture

Two primary communication protocols have been defined by the PTS:

- CD1.1 - for continuous sensor data traffic, and
- IMS2.0 - for commands to stations and explicitly requested data.

With both these protocols, digital signatures are to be applied. To facilitate the verification of the digital signatures, a PKI is set up. For communication between the stations and the IDC, a private closed network, the GCI, has been set up.

7.2 Geophysical Authentication

An aspect that is not in the primary scope of this thesis is that of *geophysical authentication.* According to [Com01b], the IMS network detects up to 50000 earthquakes every year. There is no way to accurately predict the time, magnitude, or place of all these earthquakes.

If data from one or more sensors are successfully forged (i.e. the forged data is accepted by the receiver as authentic) and an earthquake occurs that the sensors in question *should* have detected, the forgery will with high probability be discovered. Analysis of the data from the entire IMS will show the inconsistency.

Geophysical authentication is an important part of the security of the verification regime, but it is not relevant to the security of the signature-based security scheme. We analyse the security of that scheme independently of other authentication mechanisms.

7.3 Time of Signing

In section 4.2, we saw that the (analogue) data gathered by the sensors are sent through a digitizer before being signed. Newer digitizers have an authenticator producing digital signatures, but there are still older models in action which are unable to sign data. Many of them might stay operative for some years.[1] In this case, the CSFs are signed by the same key as the Data Frames at the CRF.

There are two big problems with this approach:

- The SO, who we do not trust with the data, has full access to the data *before* it is signed;
- A compromise of the CRF could result in undetected data forgery.

[1]There does not appear to be a clear indication as to when all older (non-signing) digitizers are required to be replaced.

Our conclusion with regard to these legacy systems is that the data from them can be no more trusted than the SO. Fortunately, they will not be part of the IMS forever.

7.4 International Monitoring System Protocol

The IMS2.0 protocol is used for two objectives: remote management of stations and requested data traffic from auxiliary stations.

Our focus regarding this protocol has been its role in station management. For data transfer, the basic elements of the continuous data transfer protocol (CD1.1) will be used.

Key management will be achieved through the use of IMS2.0, but we will discuss this separately in section 7.6.

7.4.1 Goals of IMS2.0

The IMS2.0 protocol is a part of the *first security goal* outlined earlier. If an attacker could successfully issue a malicious calibration command to a digitizer, it would be immediately detected because the data from the channel would be destroyed.

The security goal of this protocol is therefore

To allow authorized, and only authorized, entities to remotely calibrate and manage PKI components at the IMS stations, while insuring that these commands not have been subject to unauthorized manipulation.

7.4.2 Protocol Format

Command Type The command type of an IMS2.0 command is specified with a line in the message body. Unlike most other lines, such a line does not have a keyword, but only the command type alone on the line, e.g.: `CALIBRATE_START` or `GENERATE_KEYPAIR`. This approach does nothing to simplify command parsing. In practice, it means an implementation must check each line not recognizes immediately as something else, and see if it is a command type specifier. The reason this approach was chosen seems to be to be compatible with the previous (and widely used) version of the protocol, used for data transfer.

Remember that many command lines will be meaningful in a specific command only (for example will a `CALIB_PARAM` line be an error if present in an `UPDATE_CRL` command). Since the command type may be specified anywhere within the command, an implementation will have to test each line not immediately recognized as another keyword to see if it is a command type identifier.

One simple solution to this problem would be to introduce a new keyword, e.g. `CMD_TYPE`, which specify the command type (for example `CMD_TYPE CALIBRATE_START`).

Data Part Another complicating part of the command structure are data parts. A `GENERATE_KEYPAIR` command, for example, may include the DSA (system) parameters the authenticator should use in generating the keys. These parameters are sent in PEM format. Similarly, the `UPDATE_CRL` command includes the CRL, in PEM format.

An approach which would have simplified parsing of these commands would have been to begin-end keywords for them. That would also provide support for several such data fields. An example could be

```
START_KEYPAIR
DATA_BEGIN X.509_CERTIFICATE
-----BEGIN CERTIFICATE-----
X.509 certificate in PEM format
-----END CERTIFICATE-----
DATA_END X.509_CERTIFICATE
STOP
```

as opposed to today's

```
START_KEYPAIR
-----BEGIN CERTIFICATE-----
X.509 certificate in PEM format
-----END CERTIFICATE-----
STOP
```

For PEM structures only, there would not be much difference with this adaption, because a PEM structure is simple to detect. The current solution is not very flexible, though.

7.4.3 Command Processing

Commands should be processed automatically on reception at the CRF. It is likely that many SOs will refuse to accept operational change commands from any external source without manual verification. This can be handled by not performing calibration until manually accepted by the SO or by assigning only the SO as an authorized source of operational change commands.

7.4.4 Replay Attacks

Recall from section 2.4.2 that in a replay attack an attacker can resend a previously signed message to the same recipient who might verify it as valid again.

This is an attack the protocol must defend against. There are two command parameters that can prevent such an attack: the time stamp and the message id.

Time Stamp The time stamp field—mandatory for all command requests and responses—is in UTC time and a station will check this field to determine whether the request is "recent and that it is reasonable to process".

The problem with this field as a security means is that for many stations, the commands may need to pass through the SO on its way from the IDC to the station, and it might stay there for a while before someone relays it to the station. The consequence is that a station must accept a time stamp that is not completely recent. The standard does not specify how old messages that should be accepted, but the acknowledgment should be sent within one working day. To simplify the decision, a limit of 24 hours can be used.

Message Id To prevent a replay attack within the time frame not covered by the time stamp (i.e. 24 hours in our case), the message id must be used. The message ids are required to be globally unique. The receiver must maintain a list of message ids seen during the past 24 hours, and reject a message with a repeated id.

The use of the ids is not currently described in the protocol documents, and should be clarified.

As a conclusion, the protocol does have the necessary tools to prevent replay attacks, but their use is currently not very well described.

7.4.5 Entity Forgery

Verification of the command issuer is handled by the signature scheme. The recipient (station) is not, though. An adversary must not be able to intercept a command from the PTS to one station and subsequently transmit it to another station and have it accepted as valid.

Such an attack is prevented by including a unique identification of the recipient in the signed part of the message.

Recall that in S/MIME[DHR+98], only the body of the mail is signed, not the mail headers.

In earlier drafts of the IMS2.0 standard, the identification of the receiving station(s) was specified as a required parameter to most commands. For example:

```
CMD_TYPE CALIBRATE_START <station list>
```
[2]

There are two unfortunate sides to this solution. One is that the standard did not mark the station list parameter as mandatory for all possible commands (should the language be extended). The second, and more important, is that this relegates an important decision to later in the parsing. The command parser can not start looking for the To-field to decide if the address is correct.

It can be seen as a trivialization of an important matter.

This deficiency was discovered, and commented on, during the thesis work. It was also commented on by other sources. Our comments did not reach the PTS in time for the comment deadline on the draft, but similar comments from other sources did, and the next draft included a keyword `STA_LIST` which all commands require.

Now, any IMS2.0 parser should look up the `STA_LIST` option and verify the validity of the value at an early point in the processing. This amendment makes it simple for a parser to prevent entity forgery.

7.4.6 Other Attacks

In many ways, the IMS2.0 command structure is a simple protocol. Although parsing it may be complicated, its design makes it resistant to cryptanalytic attacks like insertion, deletion, and reordering attacks. Each command is signed as a block, and only one command is sent in each message (S/MIME will only accept one signed part). An IMS2.0 parser must be certain to accept as input *only the signed data* from an S/MIME message.

Conclusively, due to the—from a cryptographic point of view—simplicity of the IMS2.0 protocol, it is highly resistant to cryptanalysis.

Its security relies on the security of the S/MIME protocol. If it is possible to trick the S/MIME verification command to output more than the actual signed text (or to forge a signature completely), then the IMS2.0 protocol would be vulnerable. The strength of the S/MIME protocol is outside of the scope of this thesis.

[2]Note that the `CMD_TYPE` syntax was used then, and is dropped now. This does not affect the argument.

7.5 Continuous Data Protocol (CD1.1)

The continuous data from the primary stations is (will be, when stations are upgraded) sent to the IDC and other participants using the CD1.1 protocol, as explained in 4.3.1.

7.5.1 Goals of CD1.1

It should be noted that the CD1.1 protocol is a protocol not only for digitally signed data transfer. It is a protocol for data transfer, with the optional possibility of applying digital signatures. Our interest is in the security of the protocol when signatures are applied.

This protocol is a part of the solution of the *second security goal* from 7.1. The primary threat is that an attacker (and in this case, that includes the SO) is able to manipulate the CD1.1 traffic at any point between the borehole and its destination, and have the changes accepted as authenticated.

We formulate the goal—from a security concern—of the CD1.1 protocol:

To prevent any attacker from alter, add to, or delete any of the data output by the authenticator, and have the result accepted as authentic by the receiver.

Given that we consider the SO to be untrusted in this situation, and the fact that the CRF is operated by the SO, we will only trust the signatures created in the sealed borehole, i.e. those within the CSFs.

In other words, the security mechanisms in other elements of the CD1.1 protocol than the CSFs can function as a supplement, but it must be possible to insure authenticity of the data from the information in the signed CSFs alone.

With this in mind, we formulate the following requirements for the CSFs, and then argue why they are sufficient:

Each CSF must contain adequate information for the receiver to

- identify the source of the data (the channel),
- identify the exact time this data collection began, and
- determine the length, in time, the data spans.

The requirement that the receiver must be able to tell who the transmitter (the channel) of the data is, is obvious. As stated in section 2.1.2, without entity authentication of the source, no message authentication is possible.

Although the data stream from the station (i.e. the Data Frames) is identified by serial numbers, this is not possible[3] at the channel-level.

The nature of the data provides a different opportunity, though. Since each channel at a given time produces only a single signal, the pair *channel identifier/exact time* uniquely identifies a sample of data.

7.5.2 Security Analysis of CD1.1

Recall that the CSF signatures are applied to every field of the subframes except the total length, the "subframe body" length (the authenticator offset), the certificate id, and the authenticator (signature) itself. See table 4.4 for the fields of a CSF.

Source Identification

The *channel description* field of the CSF contains the 5-byte site name and 3-byte channel name. These two fields uniquely identify the channel, preventing source forgery.

Subframe Information

Each CSF include its start time in UTC and the time spanned by the subframe in milliseconds. The start time is on the format *YYYY/MM/DD HH:MM:SS.MMMM*, so the time granularity is in milliseconds. These two values combine with the channel name to allow the recipient to uniquely identify the data.

However, having the required information present does not help much if it is not used. To avoid packet loss in a CD1.1 transmission, the serial number from the Frame Header (the header of the Data Frames) will normally be used (the Acknack frames use this value). This 64 bit integer is designed to allow a connection to stay up for decades without having to reuse the serial number. This mechanism is good from a technical point of view, where the fear mainly is lost packages due to network or sensor/digitizer problems. To prevent deliberate removal/substitution of entire subframes, it is important that the receiving application also verifies that the information within the subframes is correct (and, of course, the subframe signatures).

[3]Some digitizers provide a serial number on the CSFs as well, which could be used for this purpose, but this feature is seemingly intended for future use only, as documentation currently states it will not be used or checked.

Other Attacks

When the basis is that only the CSF signatures are trusted, the CD1.1 protocol has the same property as the IMS2.0 protocol: simplicity. Each subframe contains enough authenticated information to allow the receiver to verify that all the expected data, and nothing but the expected data, is received, and that no data has been modified posterior to signature application. Because of this property, CD1.1 is resistant to any attack not modifying the CSFs.

7.5.3 Checking Revocation Status

It is obvious that the receiver of the data must consult the CRL before accepting a station's key, but how often should the list be consulted?

The key which was considered uncompromised at the time of connection might not stay that way forever. It is imperative that a procedure for the CD1.1-receiver to update its CRL and check it regularly is decided upon.

7.6 Key Management

It has been decided that it should be possible for the PTS to order a station to change its key-pair. The reason is that a private key will have a certain life-cycle, and therefore needs replacing once in a while. If there has been a private key compromise, the station needs to be reinitialized. Note that our only concern here is the authenticator keys, not the signing key on the CRF. As we argued in section 7.5 above, the CRF's signature is not required for the receiver to detect manipulations of the data flow.

Key update will be carried out using the command and control protocol, IMS2.0. The key management commands will be sent from the PTS exclusively. The two commands are `GENERATE_KEYPAIR` and `START_KEYPAIR`.

Private Key Confirmation A `GENERATE_KEYPAIR` command is responded to by `KEYPAIR_GENERATED`. It is required to contain a self-signed PKCS #10 certificate request. Since the request is signed with the newly created private key, it enables the CA to verify that the requester is in possession of the private key. For legacy digitizers/authenticators, it is possible to provide the public key in hexadecimal instead of a PKCS #10 request. In that case there will be no private key confirmation.

Identification of Private Key A bigger problem than private key confirmation, is proving the integrity of the private key. It is obvious that when the PTS requests a key change at a station, they want to be certain that the new key is as trustworthy as the old key.

The IMS2.0 protocol requires that the new public key is signed using the *old* key. This makes sense, since we now use the old key, which is trusted, to prove that the new key is authentic. The question is: *how to verify that the signed public key is the authentic key from the authenticator?*

The `KEYPAIR_GENERATED` response includes the new public key signed by the old private key. There is no additional information like date or response message id included in the part signed with the old key. If the SO at any time in the past has had access to having data of his choosing signed by the authenticator, he could have signed the public key of a key-pair to which he is in possession of the private key. This key could then be used in the `KEYPAIR_GENERATED` response.

To complete this attack, the SO must replace the authenticator's signature for all traffic from the affected channel using the key he knows, as this key will *not* be the one the authenticator uses. This property in practice means that the SO is the only one who can carry out this attack, as he with high probability would discover any other attempted such attack.

A problem with the attack, is that it is easily discovered by an on-site inspection (the inspector would notice that the data from the borehole is signed with the wrong (old) key).

Conclusion The problem of trustworthy key updates is directly related to that of physical security (of the authenticator). To insure that the new key-pair, on which the CA issues the new certificate, is the key-pair the authenticator uses, it is required that the SO at no time has had unsupervised access to the signing function of the authenticator.

There are currently two ways to achieve that: One, the authenticator is built so that when out of normal operation, access to the signing function must either be detected (when restored to operation) or difficult to obtain. Alternatively, *every* time the authenticator has been exposed (i.e. the borehole has been opened), it must be reinitialized.

A third alternative would be to alter the key management protocol. Today, the public key which is signed (using the old private key) is signed with *no additional information.* If the protocol had demanded that the signature was to be applied to the new public key *and* a time stamp, exploiting the problem would have been difficult. The SO would then have to be able to predict the time of the next key update at the time he had access to the signing function.

7.7 Physical Security

7.7.1 IMS Stations

In 1991, Gustavus Simmons wrote[Sim91]: “It is not difficult to physically secure the seismic sensor package in subsurface emplacements [...] since the seismic sensors themselves would detect any attempt to gain physical access to them long before they were in jeopardy. Hence only the data stream sent through an open communications channel would be subject to possible manipulations.”.

This result is based on the fact that it is impossible to approach anything in the borehole without the sensor data reflecting it; this is the nature of a seismic sensor. An assumption made here is that the station is working. A different problem is the quality of the physical security when it is not. If a problem requires a sensor to be shut down for maintenance, the SO will have access to the equipment. It is important that any tampering with the equipment will either be detected or not reduce security.

Authenticator It is evident that the signatures can be no more trusted than the equipment that makes them.

For the authenticators, we list the following important requirements:

- It must be difficult to extract the private key from the authenticator;
- It must be difficult to install unauthorized upgrades to the firmware or other security-critical components;
- It must be difficult to access the signing function.

PrepCom documents demand that the authenticator must comply with level 3 of the requirements put forth in FIPS 140-2[Nat01b]. Level 3 security requires that the device shall have a high probability of detecting and reporting physical access to, use of, or modifications of critical security parameters (CSPs).

Information from PTS personnel does however indicate that the signing function *will* be available to anyone with physical access to the authenticator. There are no demands that the authenticator maintains a counter of the signatures it has made to date. If there was, this could be used to prove to the PTS that the signing function was not accessed while the authenticator was out of operation.

Digitizer In cases where the digitizer and the authenticator are two separate objects, the digitizer itself is not a cryptographic module, but it is a security-critical module. It should be difficult to replace or make modifications to the digitizer.

No documents have been found that describe policies on this subject.

The CRF Access restriction to the CRF is important. The host will be accessible from the GCI (in the case of a independent subnetwork, only from the NDC), and should be considered at risk. Procedures for keeping remotely accessible software up to date and for choosing secure passwords must be in place. Physical access to the CRF must be limited to the SO.

7.7.2 The GCI

The GCI is a closed private network, making access for unauthorized third parties more difficult. The network has many nodes, though, and access to it is no more protected than the weakest of those nodes.

We have no detailed information about the security of the GCI network, but we will consider the security architecture in the perspective that all traffic on the GCI is subject to malicious alteration.

7.8 PKI

In this section, we will discuss the suitability of the PKI proposed by the PTS. We will first address the question of whether a PKI, as described in chapter 3, is a good solution in this scenario. We then continue looking at some decisions taken on how to build the PKI.

7.8.1 Appropriateness of the PKI

An important question when analysing the appropriateness of a security solution, is whether the specific solution was chosen because it fit the problem at hand, or whether the problem was altered to fit the solution.

Our first question should therefore be: “What do we want to achieve?”. The answer:

- Authentication and non-repudiation of data from stations, and
- authentication of commands to stations.

This is achieved through the use of digital signatures. Recall from 2.6 that digital signatures are well suited for this particular task. When we accept Gustavus Simmons' argument that keeping the private key safe in the borehole is simple, digital signatures give us what we need; authentication without secrecy *and* non-repudiation.

Entity Naming As we wrote in chapter 3, choosing an appropriate naming scheme for the entities in a PKI is often difficult. The entities in the IMS PKI consist of two groups:

- IMS stations, and
- command issuers (including data requesters).

There are two types of signing bodies at the stations: the digitizers and the CRF. Both these groups are already subject to well-defined naming schemes, which easily translate into DNs.

The command issuers are slightly more difficult, but they are limited in number. They are also provided certificates for their roles in the IMS, not as private persons. It should be easy to relate the name to their affiliated department (NDC, PTS, IDC, etc.).

CA Key Distribution Authenticated transfer of the CA certificate to every end-user is a big obstacle in most PKI solutions.

In the IMS PKI, signatures will primarily be verified by three different groups:

- Stations;
- Command issuers;
- Data receivers.

The command issuers and the stations will for the purpose of CA certificate distribution be the same group. Commands may originate from the PTS and the local SO (data requests may come from other sources, but these will naturally be data receivers). We can assume that the PTS personnel can get the authenticated CA certificate easily. When an SO has an authenticated copy of the CA certificate, the stations he controls will also have one.

Distributing the CA certificate to SOs should not be difficult as most NDCs regularly meet PTS personnel face to face. The same argument applies to the data requesters. Most entities will get all their data from the IDC.

In short, the relatively small size of the user-base, the well-defined naming scheme, and the level of contact between the CA (PTS) and the end-users are all good arguments for using a PKI as described in chapter 3 to distribute the public keys of the IMS network.

7.8.2 CRL Variant

In section 3.3.3, we described two alternatives for implementing a PKI's CRL. We will now make some remarks on their suitability for the IMS PKI.

The entities in the PKI which will need to consult the CRL often are the IMS stations and the data receivers. The IMS stations need to verify the validity of the certificates of the entities to which they send data, and vice versa.

Although there are many stations certificates, it is perceived that there will be relatively few revocations. It will happen, though, as sensors and digitizers are destroyed from time to time (in particular by lightning). In the case of a digitizer (and with it, the private key) being destroyed, the corresponding certificate must be revoked.

The two CRL extensions discussed in 3.3.3 are CRL Distribution Points (CDP) and Delta CRLs. CDP will be used in the IMS PKI.

CDP is advantageous in the IMS PKI for two reasons. The first is that it simplifies the location of the relevant CRL for a user of a certificate, the second that it allows splitting the CRL in two. Due to the anticipated low traffic on the CRL, this may not be necessary. However, as IMS stations rarely or never will need to verify each others' certificate, and command issuers and data receivers normally will need only the CRL containing station certificates, splitting the two could be useful.

Delta CRLs are most useful in a PKI with a large CRL, but where revocations are relatively rare (in particular one where certificates have a long life-time, so they stay on the CRL for a long time). This CRL will probably not grow very big, so Delta CRLs do not seem necessary.

7.8.3 Choice of Algorithms

The IMS PKI does have support for both certificates for digital signatures and for encryption, but, again, we will only consider the digital signatures part.

The PrepCom has chosen the Digital Signature Standard (DSS) to produce digital signatures, and SHA-1 for hashes. Keys shall be 1024 bits long.

Security

An important question when discussing the security of a digital signature algorithm is for how long the data integrity must be protected.

For the IMS data, we will assume this period is relatively short. It is unlikely that 20 year old data will need to be verified for integrity.

The maximum field of 1024 bits specified in the DSS makes it an algorithm which is not perceived to be secure against attackers with a high resource level for a very long time. In [LV01], Lenstra and Verheul argue that DSS with a field size of 1024 bit is useful for commercial applications only until 2002. There has been no reports of a calculation of a 1024 bit discrete logarithm, though, and DSS is widely used. The same article suggest that a replacement for DSS where longer keys (and hashes) are allowed is in the making.

Performance

The question of performance is a question of whether or not the operational goals can be met with the available hardware. For C&C, we can assume that time required for signature creation and verification are negligible because of the (anticipated) low number of commands.

For the continuous data, a significant number of signatures must be created and verified.

There are three involved parties:

- The authenticators;
- The CRF;
- The data gathering facilities at the IDC.

The Authenticator The authenticators are the lowest on computational power, but are limited to signing, and then only the data from one channel. Each channel will produce approximately 1600 bytes of data, plus less than 100 bytes overhead for the CSFs. There will normally be one CSF each 10 seconds. This limited number of small packets should not be a computational problem for the authenticators.

The CRF The CRF will both create and verify signatures. The verifications are (in addition to the C&C requests and the connection establishment frames, which we can ignore) the *Acknack* frames from the data receiver. These will likely be relatively rare (the interval is up to the receiver, but there will at least an order of magnitude less than the *Data* frames), so they will not be significant in the total time required.

The CRF will sign one *Data* frame each tenth second (plus resends due to packet loss, if required). These frames vary in size (from CRF to CRF) depending on the number of channels they comprise. With one CSF at around 1700 bytes per channel and a maximum of about 60 channels plus the *Data* frame overhead (less than 100 bytes), the CRF will sign a message of up to approximately 100kB every ten seconds.

With only one signing every ten seconds, performance should not be a problem at the CRF.

The IDC The data receiving hosts at the IDC will both create and verify signatures. Creation is limited to connection establishment and *Acknack* frames. Both these actions are too rare to be significant.

Private communication with PTS personnel indicate that approximately 130 CSFs will be received every second at the IDC. These will be encapsulated in one *Data* frame from each of the stations continuously sending data. These stations are the 50 primary seismic stations, the 11 hydroacoustic, and the 60 infrasound stations; a total of 121 stations, or approximately 12 *Data* frames per second.

The operations to be executed at the IDC each second are therefore:

- 130 SHA-1 hashes of 1.6kB blocks;
- 12 SHA-1 hashes of blocks ranging from 5kB (3 channels; three-component stations) to 100kB (60 channels), with an average of about 16kB (10 channels);
- 142 DSA verifications.

All the verifications are planned performed in software.

Timings made on a Sun Ultra10 workstation show that one DSA verification using OpenSSL takes about 30 milliseconds. This included 300 microseconds—or 1% of the total time—for computing the SHA-1 hash of a 1.6kB block.

A run of the `openssl speed` application for timing (for example) SHA-1 shows us that the number of bytes processed per second increases if the blocks given to SHA-1 increases. As the total size of the data is almost the same for the 130 1.6kB blocks as the 12 6-64kB blocks, we can conclude that hashing the latter will take less time than the former. Summing up, the hashing of all the data should take less than one tenth of a second $(260 * 300\mu s = 0.078s)$.

If we assume $30ms$ per signature, verification totals over 4 ($0.03*142 = 4.260$) seconds; 98% percent of the total time. This means that for verification alone, computing power corresponding to a little over 4 Sun Ultra10 workstations is the minimum.

The hardware specifications planned for the signature verification hosts (there will probably be more than one) are not known to us, but these numbers show that even though there are many signatures, the hardware requirements are significant, but not overwhelming.

These values do not take into account that verification may be optimized when the system parameters of the DSA public key are common to all entities. With common parameters, one of the modular exponentiations in the verification algorithm (α^{u1}) may be optimized by some precomputation. Table 20.3 in [Sch96] indicates an improvement of 37.5% when using a common public component. Such an optimization has an upper bound of (less than) 50% speed-up, since only one of two exponentiations is affected.

7.9 Threats

We will now go through the threats summarized in chapter 5 and comment on to which extent they are met by the proposed security architecture. Many of those threats are of such nature that mechanisms protecting against them are outside of the main scope of this thesis. In those cases, we will comment briefly on the nature of the possible defense mechanisms.

We list the threats from chapter 5 here for convenience:

1. Physical event near station.
2. Hardware, software, or firmware malfunction.
3. Private key leak.
4. Access to signing function.
5. Communications or CRF breakdown.
6. Unauthorized command to borehole (from the CRF).
7. Manipulation of data (at the CRF).
8. Unauthorized command to station (from the GCI).
9. Manipulation of data (in the GCI).
10. Communications breakdown (GCI).

11. Certificate issued on false or exposed private key.
12. Certificate issued with wrong DN or authorization.
13. Certificate removed from or not posted to CRL.
14. Incomplete signature verification.
15. Faulty data receivers.
16. Data receiver breakdown.
17. Manipulation of data archive.

Threat 1 A physical event is impossible to properly defend against. Firstly, there may be natural events (e.g. lightning strike or flood) over which we have no control, and secondly, constantly guarding the sensor's vicinity is economically completely infeasible.

The chances of damages to very many stations simultaneously is perceived to be small enough to accept.

Threat 2 is different. Although there may be accidental malfunctions (e.g. buggy firmware, faulty hardware), a greater risk is deliberate tampering. First of all, it is important to have procedures for monitoring production, transport, and storage of both the digitizer and the authenticator. Secondly—and more relevant to this thesis—there is a need for procedures on what happens if the borehole has been opened. This scenario is discussed in 7.7 and 7.6, and our conclusion is that the for a channel to remain trustworthy after being off, it should be reinitialized.

Threat 3 is a threat that is met primarily by the demands that the authenticator conforms to FIPS 140-2 level 3, and by having supervision procedures to make sure that is the case. This includes evaluation of the firmware of the authenticator (e.g. random number generator).

Threat 4 should be impossible to carry out as long as the authenticator is in the sealed borehole. It is obvious that the communication channel provided by the authenticator to handle commands and requests cannot allow arbitrary data to be signed since that would also allow forged CSFs to be signed.

Threat 5 is a threat that must be handled by insuring that proper redundancy and disaster recover procedures work. To avoid data loss, it is also important that the authenticator can store as much data as possible while waiting for the communication to come back up.

Threat 6 is another threat that cannot be prevented by the PKI or the command protocol, but by securing the CRF and the communication links.

Threat 7 and 9 are perhaps the two most severe threats, with the latter being the gravest. In our analysis of CD1.1 earlier in this chapter, we argued that the receiver, when verifying all available information, will be able to detect any change in the data from the authenticators. The receiver will be able to do this regardless of where the change happened, as long as it was after the signature was applied.

If we assume that the correct data is signed in the CD1.1 CSFs, and that the proper verification procedure is followed, we claim that undetected manipulation of data in transit between the authenticator and the verifier is as difficult as forging the signature.

Threat 8 is a threat dealt with by the IMS2.0 protocol. We have shown that the protocol is resistant to attacks like replay attacks or entity forgery. The digital signature on the command provides protection against a malicious modification of an authorized command in transit. Our implementation from chapter 6 shows access control mechanisms for insuring that only commands from authorized command issuers are accepted.

Threat 10 is met by the demand that stations must keep a backlog of 7 days of data. A breakdown, whether as a result of an attack or an accident, will be quickly discovered, and steps will be taken to make sure alternative transfer of the data to the IDC is commenced.

Threats 11-13 are not primarily prevented by technical solutions, but by procedural ones.

Initial certificate issuing for stations are handled by a trusted individual from the PTS staff observing the key generation process. Certificates to data requesters and command issuers will likely be completed with visits to Vienna. A certificate request must be accepted by a 5-man panel before being processed by the CA. Logs must be kept of all CA activity, and auditing carried out continuously.

A more severe problem is key updates. When a certificate is about to expire, a new one must be issued. This procedure is discussed in 7.6. The current procedures do not address the problem on how to insure the authenticity of the new keys if the authenticator has been exposed to the SO.

We have not found a description of the procedures for certificate revocation. It is important that any report of a key compromise or any other reason for revocation is quickly addressed. Again, logging and auditing must make sure that no mistaken or unauthorized changes are made to the CRL.

Threats 14-16 are met primarily by software and redundancy. In 7.5.2, we listed what a CD1.1 receiver must do to prevent an attack that was made after the data left the authenticator. As this software appears to be the only part of the system that is *required* to verify the signatures, they are critical to security.

A breakdown in these services will be far less severe. Stations will not delete data until it is acknowledged to have been received by the IDC. This is a similar situation to Threat 10. If the receiving hosts acknowledge the reception of data, but lose or modify it later, there is a greater risk of data loss.

The potential impact of an undiscovered (but exploited) malfunction in the data receivers implies that these hosts must be subject to strict access control, logging, and auditing.

Threat 17 is also on the outside of the main scope of this thesis. Once the data is received, and verified to be correct, it is archived for future use.

It is important that write access to the archive is limited to inserting new data only, and that redundancy is good enough to survive a disaster (fire, flood, tape and disk corruption).

The DSS signatures made today will not be considered trustworthy forever. In some scenarios, it is possible to demand the data to be re-signed when keys expire, but this is infeasible for the sensor data. It is obviously not possible to return the data to the authenticators for a new signature.

It is possible to introduce signatures made by IDC staff. These signatures can be added to the old ones (not necessarily one for each CSF, it is probably more practical to sign data for e.g. a full day in one block), and they can be replaced before their keys expire.

Chapter 8

Conclusion

We have described the security solution presented by the PTS to tackle the problems encountered in the implementation of a global, comprehensive nuclear test-ban treaty. We have also provided an implementation of parts of that solution.

Using our implementation, we have described two different solutions to handle access control of commands.

Our investigation of the quality of the protocols suggested for command and control and data transfer have not shown any security concerns. We consider the IMS2.0 protocol to be unnecessarily complex and difficult to parse.

We listed a series of threats to the system in chapter 5. We closed the analysis by giving our opinion on to which extent the proposed security architecture answers to these threats.

A challenging task is to decide on procedures regarding sensors which have been inoperative. It is our conclusion that if a sensor has been out of operation, restored to operation, and then at any time later been through a key update, it can no longer be fully trusted. Other than the key update problem, it is our opinion that the security architecture meets the most severe threats against the verification regime.

Bibliography

[ACL$^+$99] Ross J. Anderson, Bruno Crispo, Jong-Hyeon Lee, Charalampos Manifavas, Vaclav Matyas Jr., and Fabien A P Petitcolas. *The Global Internet Trust Register.* MIT Press, 1999.

[AL99] Carlisle Adams and Steve Lloyd. *Understanding Public-Key Infrastructure: Concepts, Standards, and Deployment Considerations.* Technology Series. New Riders Publishing, 1999.

[Ame86] American National Standardization Institute. ANSI X9.9: Financial institution message authentication. Technical report, American National Standardization Institute, 1986.

[And01] Ross J. Anderson. *Security Engineering.* John Wiley & Sons, Inc., 2001.

[BB03] David Brumley and Dan Boneh. Remote timing attacks are practical. `http://crypto.stanford.edu/~dabo/abstracts/ssl-timing.html`, 2003. To be published in the 12th USENIX Security Symposium.

[Ben02] Mike Benham. IE SSL vulnerability. http://lists.insecure.org/lists/bugtraq/2002/Aug/0111.html, August 2002.

[CF99] S. Chokhani and W. Ford. Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework (RFC 2527). Technical report, Internet Engineering Task Force, Mar 1999.

[Com94] Comprehensive Nuclear-Test-Ban Treaty. CTBT treaty text. URL:http://www.ctbto.org/treaty/treatytext.tt.html, 1994.

[Com01a] Comprehensive Nuclear-Test-Ban Treaty. The global communications infrastructure and the international data centre. `http://www.ctbto.org/reference/outreach/booklet4.pdf`, Aug 2001.

[Com01b] Comprehensive Nuclear-Test-Ban Treaty. The global verification regime and the international monitoring system. http://www.ctbto.org/reference/outreach/booklet3.pdf, Aug 2001.

[COS86] Don Coppersmith, Andrew M. Odlyzko, and Richard Schroeppel. Discrete logarithms in GF(p). *Algorithmica*, 1(1):1–15, 1986.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[DHR$^+$98] S. Dusse, P. Hoffman, B. Ramsdell, L. Lundblade, and L. Repka. S/MIME version 2 message specification. (RFC 2311). Technical report, Internet Engineering Task Force, March 1998.

[Dif88] Whitfield Diffie. The first ten years of public-key cryptography. *Proceedings of the IEEE*, 76(5):560–577, May 1988.

[ElG85] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithm. *IEEE Transactions on Information Theory*, IT-31:469–472, 1985.

[Ell00] Carl M. Ellison. Naming and certificates. In *Proceedings of the tenth conference on Computers, Freedom and Privacy*, pages 213–217. ACM Press, 2000.

[FB96] N. Freed and N. Borenstein. Multipurpose internet mail extensions (MIME) part one: Format of internet message bodies. (RFC 2045). Technical report, Internet Engineering Task Force, November 1996.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ron L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[Gut] Peter Gutmann. cryptlib Encryption Toolkit. http://www.cs.auckland.ac.nz/~pgut001/cryptlib/.

[HPFS02] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. (RFC 3280). Technical report, Internet Engineering Task Force, April 2002.

[Int97a] International Telecommunications Union - Telecommunications standards section. ITU-T recommendation X.500. "information technology–open systems interconnection–the directory: Overview of concepts, models, and services.". Technical report, International Telecommunications Union - Telecommunications standards section, June 1997.

[Int97b] International Telecommunications Union - Telecommunications standards section. ITU-T recommendation X.509. "information technology–open systems interconnection–the directory: Authentication framework.". Technical report, International Telecommunications Union - Telecommunications standards section, June 1997.

[Int98] International Telecommunications Union - Telecommunications standards section. X.208 - ASN.1. Technical report, International Telecommunications Union - Telecommunications standards section, 1998.

[Kah67] David Kahn. *The Codebreakers - The Story of Secret Writing*. Scribner, 1967.

[Kal98a] B. Kaliski. PKCS #1: RSA Encryption Version 1.5 (RFC 2313). Technical report, Internet Engineering Task Force, March 1998.

[Kal98b] B. Kaliski. PKCS #7: Cryptographic message syntax version 1.5. (RFC 2315). Technical report, Internet Engineering Task Force, March 1998.

[KBC97] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication (RFC 2104). Technical report, Internet Engineering Task Force, February 1997.

[Ker83] Auguste Kerckhoffs. La cryptographie militaire. *Journal des Sciences Militaires*, IX:5–38, January 1883.

[Koc] Werner Koch. Libgcrypt. `http://www.gnu.org/directory/security/libgcrypt.html`.

[Lin93] J. Linn. Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures (RFC 1421). Technical report, Internet Engineering Task Force, February 1993.

[LV01] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 14(4):255–293, 2001.

[Mic01] Microsoft Corp. Microsoft security bulletin MS01-017. `http://www.microsoft.com/technet/security/bulletin/MS01-017.asp`, March 2001.

[Mic02] Microsoft Corp. Microsoft security bulletin MS02-50, September 2002. http://www.microsoft.com/technet/security/bulletin/MS02-050.asp.

[MvOV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[Nat77] National Bureau of Standards. Data encryption standard (DES). Technical Report FIPS 46, National Institute of Standards and Technology, 1977.

[Nat80] National Bureau of Standards. DES modes of operation. Technical Report FIPS 81, National Institute of Standards and Technology, Dec 1980.

[Nat94] National Institute of Standards and Technology. Digital signature standard (DSS). Technical Report FIPS 186, National Institute of Standards and Technology, May 1994.

[Nat95] National Institute of Standards and Technology. Secure hash standard. Technical Report FIPS 180-1, National Institute of Standards and Technology, April 1995.

[Nat01a] National Institute of Standards and Technology. Advanced encryption standard (AES). Technical Report FIPS 197, National Institute of Standards and Technology, Nov 2001.

[Nat01b] National Institute of Standards and Technology. Security requirements for cryptographic modules. Technical Report FIPS 140-2, National Institute of Standards and Technology, Nov 2001.

[Pes01] P. Pesnick. Internet message format. (RFC 2822). Technical report, Internet Engineering Task Force, March 2001.

[Pre] Preparatory Commission for the Comprehensive Nuclear-Test-Ban Treaty Organisation. CTBTO - Preperatory Commission for the Comprehensive Nuclear-Test-Ban Treaty Organisation. http://www.ctbto.org/.

[Riv92] R. Rivest. The MD5 message-digest algorithm. Technical report, Internet Engineering Task Force, April 1992.

[RSA78] R.A. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. In *Communications of the ACM*, volume 21, pages 120–126, 1978.

[Sch96] Bruce Schneier. *Applied Cryptography - Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, Inc., 2nd edition, 1996.

[Sim91] Gustavus J Simmons. How to insure that data acquired to verify treaty compliance are trustworthy. In Gustavus J Simmons, editor, *Contemporary Cryptology*, pages 615–630. IEEE Press, 1991.

[Sti95] Douglas R. Stinson. *Cryptography - Theory and Practice.* CRC Press, 1995.

[The] The OpenSSL Project. OpenSSL: The Open Source toolkit for SSL/TLS. `http://www.openssl.org/`.

[VMC02] John Viega, Matt Messier, and Pravir Chandra. *Network Security with OpenSSL.* Cryptography for Secure Communcations. O'Reilly and Associates Inc., June 2002.

[WGB] WGB. The structure of commands to ims stations. Non-Paper.

[YHK95] W. Yeong, T. Howes, and S. Kille. Lightweight Directory Access Protocol (RFC 1777). Technical report, Internet Engineering Task Force, March 1995.