Universitat Politècnica de Catalunya

Design and Evaluation of Certificate Revocation Systems

Ph.D. Dissertation

Jose Luis Muñoz Tapia

Department of Telematics Engineering

Technical University of Catalonia

Advisor

Dr. Jordi Forné Muñoz

A DISSERTATION SUBMITTED TO
THE DEPARTMENT OF TELEMATICS ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF THE TECHNICAL UNIVERSITY OF CATALONIA
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR

This document has been produced using LaTeX 2ε.

Barcelona, October 2003

Preface

Certificates are necessary but not sufficient to secure transactions between parties. The Public Key Infrastructure (PKI) has to provide its users the ability to check, at the time of usage, that certificates are still valid (not revoked). So understanding revocation is an important concern to both PKI service providers and end users. By a better understanding of the complexities of certificate revocation, certificate-using entities can improve their decision-making process in order to accept or reject a certain certificate. In this sense, this thesis presents a comprehensive survey and analysis of the main existing revocation schemes. Furthermore, the certificate revocation represents one of the hardest scalability problems of the whole PKI; so this aspect is getting more and more crucial with the development of wide spread PKIs. There are studies that even argue that the running expenses of a PKI derives mainly from administering revocation. This motivate us to propose scalable, timely, secure, and cost-effective systems to manage the revocation information. In this respect, we have three new proposals: $\mathcal{H}$-OCSP (which is a modification over the standard OCSP), AD-MHT (which is based on the Merkle Hash Tree) and E-MHT (which agglutinates several mechanisms that enhance the efficiency of traditional MHT-based systems). Our proposals are not only a set of theoretical mechanisms but they are also practical systems that have been implemented inside a Java test-bed called Cervantes (Certificate Validation Test-bed). The design of Cervantes allows it to fit any kind of revocation system without significative changes in the structure or the source code of the platform. Finally using Cervantes we are able to obtain performance results about each system developed.

Acknowledgments

I'm sorry for the English readers but they should learn Spanish to read the acknowledments ;-)

Tomando unas copas, alguien me dijo una vez que "la vida no es la meta sino el camino". La verdad es que era un tipo bastante sabio y ahora que estoy a punto de conseguir ser doctor me alegro de haber disfrutado en el camino...

Me gustaría dedicar este trabajo a mi mujer, ella sabe casi tanto como yo el tiempo que requiere hacer una Tesis. Gracias Mónica por estar a mi lado cada vez que me levanto (bueno excepto cuando estoy en la otra parte del mundo de congreso). A mis padres, a ellos les debo todo lo que soy. A mis hermanos, que casi sin quererlo me han hecho ir a parar al "mundillo" universitario. A mi familia científica (alias ISG): a mi "papa" Jordi, al "tío" Miki y a mis "hermanos" Josep, Marcel y Oscar... Ah! y a Juan (el "becario"). Que decir de vosotros... la familia es la familia. A mis amigos (no especifico para no correr el riesgo de dejarme a alguno). A mis proyectistas Marga, María José, David, Isi y Oscar, gracias chicos esto no hubiera sido posible sin vuestra ayuda. A Sertel y al departamento en general que nunca me han negado ayuda de ningún tipo. Por último me gustaría hacer una mención especial para Jordi Forné, sé que no es su mejor momento pero tengo que agradecerle todo el apoyo y la comprensión que ha mostrado durante la realización de esta Tesis. Creo que nos quedará algo más que una relación director-doctorando así que espero que el futuro sea tan bueno como el pasado.

List of Acronyms

ASN.1 Abstract Syntax Notation number One

AES Advanced Encryption Standard

AC Attribute Certificate

AD Authenticated Dictionary

CM Central Management module

CMP Certificate Management Protocols

CRL Certificate Revocation List

CRS Certificate Revocation System

CRT Certificate Revocation Tree

CA Certificate Authority

CP Certification Policy

Cervantes CERtificate VALidatioN TESt-bed

CRL-DP CRL Distribution Points

CRM Customer Relationship Management

DB Database module

D-CRL Delta CRL

DoS Denial of Service

DH Diffie Hellman key exchange

DES Digital Encryption Standard

DSS Digital Signature Standard

DER Distinguished Encoding Rules

DN Distinguished Name

DP Distribution Point

EFECT Efficient and Fresh Certification

EE End Entity

E-MHT Enhanced-MHT

FAC Frequently Asked Certificates

FTP File Transfer Protocol

HCRS Hierarchical Certificate Revocation System

HTTP HyperText Transfer Protocol

IC Identity Certificate

I-CRL Indirect CRL

I/O Input/Output module

JCE Java Cryptography Extension

JDBC Java Database Connectivity

LDAP Lightweight Directory Access Protocol

MHT Merkle Hash Tree

MD5 Message Digest 5

MIME Multipurpose Internet Mail Extensions

OID Object Identifier

OWHF One Way Hash Function

OCSP Online Certificate Status Protocol

O-CRL Overissued CRL

PGP Pretty Good Privacy

PDU Protocol Data Unit

PH Protocol Handler module

PKI Public Key Infrastructure

PKC Public Key Cryptography

RA Registration Authority

RDI Revocation Data Issuer

RSA Rivest Shamir Adleman

SCRP Simple Certificate Revocation Protocol

SCVP Simple Certificate Verification Protocol

SPKI Simple Public Key Infrastructure

SLCH Skip Lists with Commutative Hashing

SMTP Simple Mail Transfer Protocol

SCH Status Checking Handler module

TLS Transport Layer Security

TTL Time to Live

TTP Trusted Third Party

Contents

List of Figures

Chapter 1

Introduction

1.1 About this thesis

This thesis has been carried out at the Information Security Group (ISG)[1] of the Department of Telematics (ENTEL)[2] at the Technical University of Catalonia (UPC)[3]. We would like to thank the Spanish Research Council that has partially funded the development of this thesis under the following projects:

- ACIMUT: Acceso a Internet Seguro Mediante UMTS (CICYT TIC2000-1120-C03-03).

- DISQET: Distribución de Información Segura con Calidad de Servicio (QoS) para Entornos Telemáticos (CICYT TIC2002-00249).

1.2 Background

This section provides to non-expert readers enough background in security, cryptography, public key infrastructure and certificate revocation to read this thesis.

[1] ISG home: `http://isg.upc.es`
[2] ENTEL home: `http://www-entel.upc.es/`
[3] UPC home: `http://www.upc.edu`

1.2.1 A Taxonomy of Computer security

Computer security is frequently associated with three core areas:

Confidentiality– Ensuring that information is not accessed by unauthorized users.

Integrity– Ensuring that information is not altered by unauthorized parties in a way that is not detectable by authorized users.

Authentication– Ensuring that parties are the who they claim to be.

Computer security is not restricted to these three broad concepts. Additional services that are often considered part of the taxonomy of computer security include:

Access control– Ensuring that users access only those resources and services that they are entitled to access and that qualified users are not denied access to services that they legitimately expect to receive.

Non-repudiation– Ensuring that the originators of messages cannot deny that they in fact sent the messages.

Availability– Ensuring that a system is operational and functional at a given moment, usually provided through redundancy; loss of availability is often referred to as Denial of Service (DoS).

These security services are often combined, for instance, authentication is used for access control purposes or non-repudiation is combined with authentication. In order to achieve these services three basic building blocks from cryptography are used:

- Encryption is used to provide confidentiality, can provide authentication and integrity protection.
- Digital signatures are used to provide authentication, integrity protection, and non-repudiation.
- Hash Functions are used to provide integrity protection, and can provide authentication.

One or more cryptographic mechanisms can be combined to provide a security service.

1.2.2 Symmetric-key cryptography

Symmetric-key cryptography is sometimes also called secret-key cryptography. Symmetric-key encryption involves using a single key K to encrypt and to decrypt data so the sender and the recipient share the knowledge of a secret key that is used to encrypt and decrypt the messages exchanged between them.

FORMALLY. *The message M is encrypted by applying the symmetric algorithm S to M using the key K:*

$$C = S_K(M) \tag{1.1}$$

The secret message C is decrypted by applying the inverse algorithm S^{-1} to the secret message C with the key K:

$$M = S_K^{-1}(C) \tag{1.2}$$

DES [NISb] (Digital Encryption Standard) and its extended version Triple-DES [NISe] (3DES) have been the most popular symmetric-key systems during many years. Recently, the AES [NISa] (Advanced Encryption Standard) has been designated as the successor of DES.

Generally speaking, symmetric-key systems are simpler and faster than the public-key ones, but their main drawback is that the two parties must somehow exchange the symmetric key in a secure way, this problem is relevant for large scenarios and it is is known as the "key distribution problem".

1.2.3 Public-key cryptography

Public-key cryptography (PKC) is asymmetric, involving the use of two separate keys, in contrast to the symmetric conventional cryptography, which uses only one key. One of these keys is "public" (i.e. known by everybody) and the other is "private" (i.e. secret). The public-key cryptography makes easier the key distribution

problem because the public key can be distributed without keeping it secret, and the private key is never transmitted.

FORMALLY. *Diffie and Hellman postulated the conditions that a public-key system must fulfill [DH76]:*

1. *It is computationally easy for a party B to generate a pair: (public key KU_B, private key KR_B).*

2. *It is computationally easy for a sender A, knowing the public key and the message to be encrypted, M, to generate the corresponding ciphertext (1.3)*

$$C = E_{KU_B}(M) \tag{1.3}$$

3. *It is computationally easy for the receiver B to decrypt the resulting cipher text using the private key to recover the original message (1.4)*

$$M = D_{KR_B}(C) = D_{KR_B}[E_{KU_B}(M)] \tag{1.4}$$

4. *It is computationally infeasible for an opponent, knowing the public key, KU_B, to determinate the private key, KR_B.*

5. *It is computationally infeasible for an opponent, knowing the public key, KU_B, and a ciphertext, C, to recover the original message, M.*

There are also public-key systems that meet another interesting property: either of the two related keys can be used for encryption, with the other used for decryption (1.5)

$$M = D_{KR_B}[E_{KU_B}(M)] = D_{KU_B}[E_{KR_B}(M)] \tag{1.5}$$

Depending on the application, the sender uses either the sender's private key or the receiver's public key, or both, to perform some type of cryptographic function. Public-key systems are mainly used for the following purposes:

Encryption– The sender encrypts a message with the recipient's public key.

Digital signatures– A digital signature emulates a real, physical signature by generating a digital proof that only the creator/ sender of a message can make, but everyone can identify as belonging to the creator. An encryption under the private key of the creator serves as a signature that only the owner of the private key can create, but everyone with the public key can verify. The encryption (signature) can be applied to the complete message or to a small block of data that is a function of the message.

Key exchange– Two parties cooperate to exchange a session key (symmetric key). Several different approaches are possible, involving the private key(s) of one or both parties.

Some public-key algorithms are suitable for all three applications, whereas others can be used only for one or two of these applications. The most famous public-key algorithm is RSA [RSA78] (Rivest Shamir Adleman) which can be used for all three applications, whereas DSS [NISc] (Digital Signature Standard) is widely used but only can be used for signing and DH [DH76] (Diffie Hellman) can be used only for key exchange.

1.2.4 One Way Hash Functions (OWHFs)

A OWHF (One Way Hash Function) is a function that takes a variable length input (pre-image), and computes a fixed-length output string (which is usually smaller than the pre-image) called the hash value, digest or check value. Given the hash it is computationally infeasible to find a message (pre-image) with that hash; in fact one can't determine any usable information about a message with that hash, not even a single bit. For some OWHFs it is also computationally impossible to determine two messages which produce the same hash. One important role of OWHFs in cryptography is in the provision of digital signatures. Since hash functions are generally much faster than digital signature algorithms, it is typical to compute the digital signature to some document by computing the signature on the document's hash value, which is small compared to the document itself. Additionally, a digest can be made public without revealing the contents of the document from which it is derived.

FORMALLY. *A hash function H must have the following properties:*

1. *H can be applied to a block of data of any size.*

2. *H produces a fixed-length output.*

3. *$H(M)$ is relatively easy to compute for any given M, making both hardware and software implementations practical.*

4. *For any given digest m, it is computationally infeasible to find M such that $H(M) = m$.*

5. *For any given the message M, it is computationally infeasible to find another message $M' \neq M$ with $H(M') = H(M)$.*

6. *It is computationally infeasible to find any pair (M, M') such that $H(M) = H(M')$.*

Examples of well-known hash functions are MD5 (Message Digest 5) [Riv92] and SHA-1 (Secure Hash Algorithm-1) [NISd].

1.2.5 Certificates

One of the major roles of the public-key systems if to address the problem of key distribution. If there is some broadly accepted public-key algorithm, any participant can send his or her public key to any other participant or broadcast the key to the community at large. Although this approach is convenient, it has a major weakness: anyone can forge such a public announcement. Notice that the problem of key distribution has been transferred to secure delivering of public keys. Kohnfelder first introduced the concept of using a signed data structure or "certificate" to convey the public key to relying parties in his 1978 bachelor's thesis [Koh78]. Thus, over two decades ago, it was recognized that a scalable and secure method (from an integrity perspective) would be required to deliver the public keys to the parties that needed them.

Certificates can be accepted as a way of distributing the public keys because they are signed digitally by a "Trusted[4] Third Party" (TTP) called certificate "issuer". Because certificates are unforgeable (they are signed), they can be placed in a repository server without the need for the repository to make special efforts to protect them. There are several kinds of certificates[5] including:

- X.509 ICs (Identity Certificates) [HFPS99, PFS02].
- SPKI (Simple Public Key Infrastructure) certificates [Ell99, EBL+99].
- PGP (Pretty Good Privacy) certificates [Ken93].
- ACs (Attribute Certificates) [FH02].

In particular, in this thesis, we address issues related with Identity Certificates (ICs) so from here on when we refer to the term certificate we actually refer to an IC. ICs are one of the most widely used certificates. Their main function is to bind a public-key with an identity. In this sense:

> *An IC states an association between a name called a* Distinguished Name *(DN) and the user's public-key.*

Therefore, the authentication of the certificate relies on each user possessing a unique DN. DNs use the X.500 standard [X.588] and are intended to be unique across the Internet. The TTP that issues the ICs is called the Certification Authority (CA) and the X.509 standard [ITU00, X.597] defines what information can go into a certificate and its format. All X.509 certificates have the following data, in addition to the signature:

- *Version*. This field identifies which version of the X.509 standard applies to this certificate, which affects what information can be specified in it. So far, three versions have been defined.

[4] Trust in a principal can be defined as a belief that, when asked to perform an action, the principal will act according to a pre-defined description. In particular, this belief implies the belief that the principal will not attempt to harm the requestor independently of the way it fulfills the request [Nik99].

[5] Actually, any data that contains a digital signature could be considered a certificate.

- *Serial Number*. The entity that created the certificate is responsible for assigning it a serial number to distinguish this certificate from other certificates issued by the entity.
- *Signature Algorithm Identifier*. Identifies the asymmetric algorithm used by the issuer to sign the certificate.
- *Issuer Name*. The DN of the issuer.
- *Validity Period*. Each certificate is valid only for a limited amount of time: it is not valid prior to the activation date (`not-valid-before`) and it is not valid beyond the expiration date (`not-valid-after`).
- *Subject Name*. The DN of the entity whose public-key the certificate identifies.
- *Subject Public Key Information*. This is the public-key of the entity being named, together with an algorithm identifier which specifies which public-key crypto system this key belongs to and any associated key parameters.

Currently, there are three versions of X.509 ICs. Version 1 has been available since 1988. It is widely deployed and is the most generic. Version 2 introduced the concept of subject and issuer unique identifiers to handle the possibility of reuse of subject and/or issuer names over time. However, most certificate profile documents strongly recommend that names not be reused. This is why version 2 certificates are not widely used. Version 3 is the most recent (1996) and it supports the notion of extensions, whereby anyone can define additional information and include it in a certificate extension. Extensions can be marked critical to indicate that the extension should be checked and enforced. There are also non-critical extensions that can be taken into account by relying parties or that can be "silently" ignored, that is, the extension can be ignored without any further action on the part of the relying party.

1.2.6 The basic functionality of a Public Key Infrastructure

Alice wants to securely communicate with Bob. In essence, this means that Alice does not want someone else to listen to the conversation, wants the information

sent to Bob not to be altered on their way to him and finally she would possibly like a mechanism to prove that she had this conversation, in case, for some reason, Bob claim he did not. Below we describe the basic steps and the infrastructure necessary to establish a secure communication between Alice and Bob [Xen] (see Figure 1.1).

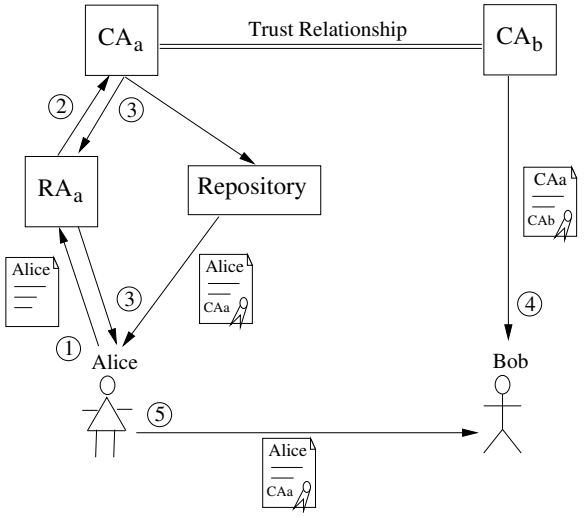


Figure 1.1: Using certificates in secure transactions.

1. Alice creates a public/private key pair using a public key algorithm. Then, she creates a certificate request, which is the certificate just prior to signing by the Certification Authority. Alice sends her certificate request to the Registration Authority (RA) for its signature.

2. Any action of approval or disapproval takes place at the RA. Then, the RA sends the request to the CA for policy approval and to be signed.

3. The result of the signing the certificate is sent back to Alice through the RA or it is often stored on a repository.

4. Alice can claim that her public key is trustworthy. Bob who wants to communicate with her, asks for her certificate. Bob, in order to verify her certificate, finds the public key of the CA that signed the Alice's public key. He needs

to do that securely. If they are both on the same CA then he has it already. If not, Bob may ask his CA to contact the Alice's CA for its public key. In order to obtain this public key, Bob has to obtain a certificate for Alice's CA from another CA whose public key is already securely obtained. CA certificates are certificates for a CA issued by another CA, which implies that there is a "trust relationship" between the two CAs. This technique can be applied recursively to obtain an increasing number of CA certificates until the public key of the CA in question is obtained. In this way, there is a certification path to the other user involving all the intermediate CAs whose public keys have to be obtained.

There are three main methods for creating systems with trust relationships between CAs. The first model is the hierarchical model, in which there is a tree of CAs with one single root certification node (see Figure 1.2-a). The second model is based on cross-certifications, where all CA certifications are bilateral (see Figure 1.2-b). The third model is the hybrid model, where both hierarchical certifications as well as cross certifications. The third model is suitable for creating trust-relationships between two different organizations that do not belong in the same hierarchy (see Figure 1.2-c).

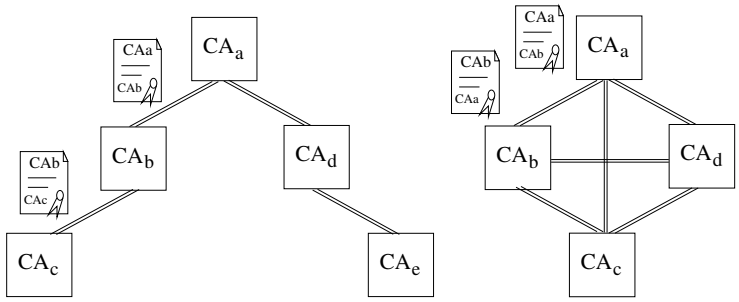
5. Finally, having the authentic public keys of each other, Alice and Bob can communicate securely.

1.2.7 What is a Public Key Infrastructure (PKI)

As we have shown, to deal with certificates, not only the CA is necessary, but also an infrastructure that ensures the validity of electronic transactions using digital certificates. The Public Key Infrastructure (PKI) can be defined as [HFPS99]

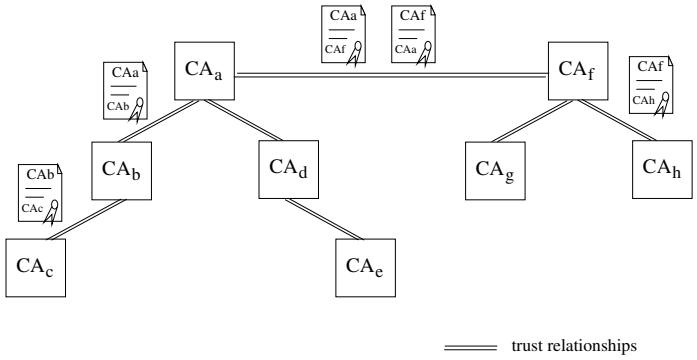
> *The Public Key Infrastructure (PKI) is defined to be the set of hardware, software, people, policies and procedures needed to create, manage, store, distribute, and revoke public key certificates based on public key cryptography*.

However, it must be noticed that the X.509 specification alone is not restrictive or specific enough to form the basis of an implementation of a PKI component.



(a) Hierarchical CAs organization

(b) Cross-certification CAs organization



(c) Hybrid CAs organization

Figure 1.2: Trust-relationships between CAs.

Realization of this fact has led to the creation of a set of mostly incompatible profiles. While X.509 defines an extensible syntax for all the data structures required to perform certificate management, a profile adds three things:

1. A restriction on the sorts of data structures that an implementation is required to understand.

2. A requirement that certain specific sorts of data structure be present, so an implementation may rely on their presence.

3. A clear definition of the behaviour of a certificate management component,

i.e. a specification of what it is supposed to do with the data structures involved.

The best example of this is seen in certificate extensions. X.509 defines the syntax of a series of extension types, and permits new extensions to be defined using object identifiers (OIDs). It does not require the presence of any extensions and does not clearly indicate what it means to support a particular extension. Thus an implementation cannot know which extensions will be present, or exactly what to do with them. A profile like PKIX defines a set of extensions that a PKIX-compliant certificate management component must be able to deal with, which extensions can be assumed to be present, and a definition of what it means to deal with them.

The distinction between X.509 and its profiles is largely one of syntax versus behaviour: X.509 defines the syntax of the objects, and the profile defines how the presence and values of objects affects the behaviour of certificate management. In addition, a profile may define extra pieces of syntax, specific to the application for which it is designed.

In this thesis we use the PKIX profile created by the IETF because it facilitates the use of X.509 certificates within Internet applications. Such applications may include WWW, electronic mail, user authentication, IPsec, etc. In order to relieve some of the obstacles to using X.509 certificates, PKIX defines a profile to promote the development of certificate management systems; development of application tools; and interoperability determined by policy. Following is the architectural model for the PKI assumed by the PKIX specifications [HFPS99]. The components in this model are (see Figure 1.3):

- *End Entity* (EE). User of PKI certificates and/or end user system that is the subject of a certificate.
- *Certification Authority* (CA). This is the entity that issues certificates.
- *Registration Authority* (RA). This is an optional entity to which a CA delegates certain management functions related to the registry.
- *Repository*. An entity or collection of distributed entities that store certificates and Certificate Revocation Lists (see Chapter 2 for further information

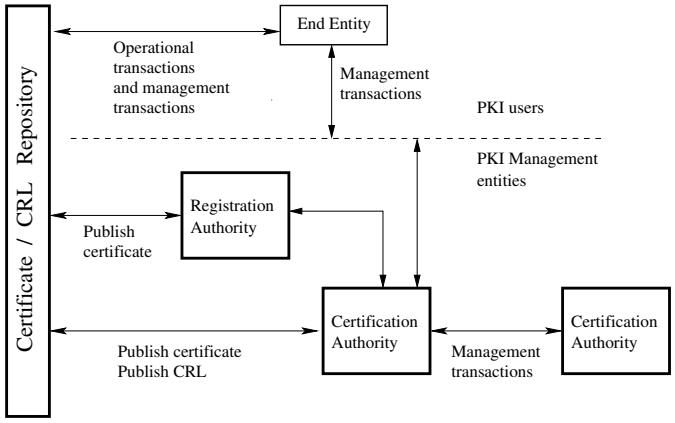


Figure 1.3: PKIX reference model.

about CRLs). Repositories serve as a mean of distributing these certificates and CRLs to EE.

Operational protocols [HFPS99] are required to deliver or transport the certificates and the status information to certificate using client systems. There are a variety of different means of certificate and CRL delivery, including distribution procedures based on raw sockets, HTTP, FTP, LDAP, SMTP, and X.500.

Management protocols [HFPS99] are required to support online interactions between PKI user and management entities. For example, a management protocol might be used between a CA and a client system with which a key pair is associated, or between two CAs which cross-certify each other. The set of functions which potentially need to be supported by management protocols include:

- Registration: This is the process whereby a user first makes itself known to a CA (directly, or through an RA), prior to that CA issuing a certificate or certificates for that user.

- Initialization: Before a client system can operate securely it is necessary to install key materials which have the appropriate relationship with keys stored elsewhere in the infrastructure. For example, the client needs to be securely initialized with the public key and other assured information of the

trusted CA(s), to be used in validating certificate paths. Furthermore, a client typically needs to be initialized with its own key pair(s).

- Certification: This is the process in which a CA issues a certificate for a user's public key, and returns that certificate to the user's client system and/or posts that certificate in a repository.

- Key pair recovery: As an option, user client key materials (e.g., a user's private key used for encryption purposes) may be backed up by a CA or a key backup system. If a user needs to recover these backed up key materials (e.g., as a result of a forgotten password or a lost key chain file), an online protocol exchange may be needed to support such recovery.

- Key pair update: All key pairs need to be updated regularly, i.e., replaced with a new key pair, and new certificates issued.

- Revocation request: An authorized entity advises a CA of an abnormal situation requiring certificate revocation.

- Cross-certification: Two CAs exchange information used in establishing a cross-certificate. A cross-certificate is a certificate issued by one CA to another CA which contains a CA public signature key used for issuing certificates.

Online protocols are not the only way of implementing the above functions. For all functions there are offline methods of achieving the same result, and the PKIX specification does not mandate use of online protocols. For example, when hardware tokens are used, many of the functions may be achieved as part of the physical token delivery. Furthermore, some of the above functions may be combined into one protocol exchange. In particular, two or more of the registration, initialization, and certification functions can be combined into one protocol exchange.

1.3 Motivation, Objectives and Contributions of this Thesis.

The Public Key Infrastructure (PKI) is responsible for the certificates not only at the issuing time but also during the whole life-time of the certificate. Notice that the certificate has a bounded life-time: it is not valid prior to the activation date (`not-valid-before`) and it is not valid beyond the expiration (`not-valid-after`) date. Typically, the validity period of a certificate ranges from several months to several years. In this context, certificate revocation can be defined as

> *Certificate revocation is the mechanism under which an issuer can revoke the binding of an identity with a public-key before the expiration of the corresponding certificate.*

Thus, the existence of a certificate is a necessary but not sufficient evidence for its validity, the PKI needs to provide applications that use certificates with the ability to check, at the time of usage, that the certificate is still valid (not revoked).

The very first objective of this thesis is to present a comprehensive survey and analysis of the main existing revocation schemes. Understanding revocation is an important concern to both, PKI service providers and PKI end users. By better understanding the complexities of certificate revocation, either of these entities can improve their decision-making process by accounting for the great quantity of variables inherent in certificate revocation.

On the other hand, many contributions have appeared in the literature on PKI in these last years. Some of them address the problem of identifying more sophisticated and efficient mechanisms to perform the various operations of a PKI. A particular emphasis is put on the certificate revocation problem. However, not much attention has been payed to the evaluation of these mechanisms. Taking into account that this is an open problem another objective of this thesis is propose a way of evaluating revocation systems. In this sense we first put some effort on creating an analytical model to evaluate revocation systems [MF02]. Our model[6] is based on "queue theory" and using this model we are able to obtain some remarkable

[6]We finally decided not to include the analytical model in this document, so you are referred to the original paper for further information.

results for the main revocation standards: CRL [HFPS99] and OCSP [MAM+99]. However, these revocation systems are quite simple and we realized that in order to evaluate more sophisticated systems we had to first simplify the number of parameters managed by the model and second complicate too much the model. For this reason we decided to develop a test-bed for evaluating certificate revocation systems. So the development of this test-bed has become another objective of the thesis. Moreover, performance evaluation of particular implementations is only possible with a test-bed. Thus, in our opinion a test-bed provides the most reliable and accurate way of performing evaluation. Another reason that make us decide on developing a test-bed is getting some practical skills and learn the "know-how" behind a revocation system.

On the other hand, it is accepted that the certificate revocation represents one of the hardest scalability problems of the whole PKI so this aspect is getting more and more crucial with the development of wide spread PKIs. There are studies that even argue that the running expenses of a PKI derives mainly from administering revocation [Stu95]. Taking this fact into account a revocation needs to be fast, efficient, timely and particularly appropriated for large infrastructures. Due to that, it is necessary to reduce the number of time-consuming calculations and to minimize the amount of data transmitted in the revocation system. So the last objective of the thesis is to develop scalable revocation systems able to timely, securely, and cost-effectively manage the revocation information. In this respect, we have three new proposals that we briefly review below.

The first proposal, called $\mathcal{H}$-OCSP, is a modification over the OCSP Standard. $\mathcal{H}$-OCSP is fully inter-operable with OCSP (OCSP clients can operate with an $\mathcal{H}$-OCSP responder and vice-versa). The point of $\mathcal{H}$-OCSP is that it reduces the processing burden in the responder and therefore the risk of running out of processing resources. As a result, an $\mathcal{H}$-OCSP responder is better protected against DoS attacks than a standard one. $\mathcal{H}$-OCSP is based on a hash chain to update pre-produced responses at a low cost. Clients can also benefit from $\mathcal{H}$-OCSP: they can store the $\mathcal{H}$-OCSP responses of the most used certificates in their cache so that these responses can be later updated with little information and processing.

The second proposal, called AD-MHT, is based on the Merkle Hash Tree (MHT) and it uses a 2-3 tree to build the MHT. To our knowledge there are no

published implementations of such a system so we have to face important open issues for implementing the system. These issues include how to respond to a request, how to revoke a certificate, how to delete an expired certificate, the communication protocol with the end users and the verification of a response. On the other hand, the performance evaluation of the AD-MHT will show that it might be a good choice for distribution of status data among end users because AD-MHT does not require much bandwidth or processing capacity, and repositories can be used to respond to status requests.

The third proposal, called E-MHT, is based on the previous proposal but we add some mechanisms to the basic data structures of AD-MHT that allow the E-MHT to provide a response size that is close to (or even better than) typical online systems such as OCSP without degradating other resources of the system. These mechanisms include the optimization of the $\mathcal{P}$aths for non-revoked certificates, the division of the revoked certificates among multiple MHTs, the re-utilization of the $\mathcal{D}$igests and the cached responses updating at a low cost.

Finally, we would like to mention the publications related with this Thesis. They are listed below:

JCR publications and journals:

- J. Muñoz, J. Forné, O. Esparza, and M. Soriano, "Certificate Revocation System Implementation Based on the Merkle Hash Tree," International Journal of Information Security (IJIS), 2003. (To appear).
- M. Soriano, O. Esparza, M. Fernandez, J. Forné, J. L. Muñoz, and J. Pegueroles, "Secure Brokerage Mechanisms for Mobile Electronic Commerce," Electronic Commerce Research (ECR). (To appear).
- J. Muñoz, J. Forné, O. Esparza, and M. Soriano, "E-MHT. An Efficient Protocol for Certificate Status Checking," in Information Security Applications (WISA 2003), Lecture Notes in Computer Science, Springer-Verlag, 2003. (To appear).
- J. Muñoz, J. Forné, O. Esparza, and M. Soriano, "Efficient Offline Certificate Revocation," in Interactive Multimedia on Next Generation Networks, vol. 2899 of LNCS, Springer-Verlag, Nov. 2003. (To appear).

- J. Muñoz, J. Forné, O. Esparza, I. Bernabe, and M. Soriano, "Using OCSP to secure certificate-using transactions in m-commerce," in Applied Cryptography and Network Security, vol. 2846 of LNCS, pp. 280-292, Springer-Verlag, Oct. 2003.

- O. Esparza, M. Soriano, J. Muñoz, and J. Forné, "Protocols for Malicious Host Revocation," in Information and Communications Security, vol. 2836 of LNCS, pp. 191-201, Springer-Verlag, Oct. 2003.

- J. Muñoz, J. Forné, O. Esparza, and M. Soriano, "A Certificate Status Checking Protocol for the Authenticated Dictionary," in Computer Network Security, vol. 2776 of LNCS, pp. 255-266, Springer-Verlag, Sept. 2003.

- O. Esparza, M. Soriano, J. Muñoz, and J. Forné, "Host Revocation Authority: a Way of Protecting Mobile Agents from Malicious Hosts," in Web Engineering, vol. 2722 of LNCS, pp. 289-292, Springer-Verlag, July 2003.

- J. Muñoz and J. Forné, "CPC-OCSP: An Adaptation of OCSP for m-commerce," Upgrade, vol. III, pp. 22-26, Dec. 2002.

International Conferences

- J. Muñoz, J. Forné, O. Esparza, M. Soriano, and D. Jodra, "Evaluation of Certificate Revocation Systems with a JAVA Test-Bed," in DEXA Workshops 2003. Workshop on Trust and Privacy in Digital Business (TrustBus03), IEEE Computer Society.

- J. Muñoz, J. Forné, O. Esparza, and M. Soriano, "A Test-Bed for Certificate Revocation Policies," in IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM03). IEEE Communications Society.

- J. Muñoz and J. Forné, "Design of a Certificate Revocation Platform," in International Conference on Information Technology: Research and Education (ITRE 2003), IEEE Communications Society.

- J. Muñoz, J. Forné, O. Esparza, and M. Soriano, "Implementation of an Efficient Authenticated Dictionary for Certificate Revocation," in The Eighth IEEE Symposium on Computers and Communications (ISCC'2003), vol. 1, pp. 238-243, IEEE Computer Society, June 2003.

- J. Muñoz and J. Forné, "Certificate revocation policies for wireless communications," in International Conference in Communication Systems and Networks (CSN 2002), pp. 427-432, ACTA Press, Sept. 2002.

- J. Muñoz and J. Forné, "Evaluation of Certificate Revocation Policies: OCSP vs. Overissued CRL," in DEXA Workshops 2002. Workshop on Trust and Privacy in Digital Business (TrustBus02), pp. 511-515, IEEE Computer Society, Sept. 2002.

- J. Forné, J. Muñoz, and J. Castro, "Certificate revocation for e-business, e-commerce and m-commerce," in International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR 2001), Aug. 2001.

In Spanish

- E. Pallarès, J. Forné, and J. L. Muñoz, "Estudio estadístico de sistemas de revocación de certificados mediante árboles de Merkle 2-3," in IV Jornadas de Ingeniería Telemática (JITEL 2003), 2003.

- J. Muñoz and I. Bernabe, "OCSP para aplicaciones en JAVA," in II Simposio Español de Negocio Electrónico (SNE 2003), pp. 25-36, June 2003.

- J. Forné and J. L. Muñoz, "Evaluación y dimensionamiento de sistemas de revocación de certificados," Bolerín de la Red Iris, vol. 62-63, pp. 20-23, Dec. 2002.

- J. Castro, J. Forné, and J. Muñoz, "Evaluación del coste computacional de políticas de revocación de certificados," in VII Reunión Española sobre Criptología y Seguridad de la Información, vol. 1, pp. 165-178, Sept. 2002.

- J. Muñoz, J. Forné, and J. Castro, "Principios de diseño de un sistema de revocación de certificados digitales," in VII Reunión Española sobre Criptología y Seguridad de la Información, vol. 2, pp. 643-656, Sept. 2002.
- J. Castro, J. Forné, and J. Muñoz, "Políticas de Revocación de Certificados para M-Commerce," in XI Jornadas de Telecom I+D 2001, Nov. 2001.
- J. Castro, J. Forné, and J. Muñoz, "Revocación de Certificados en Aplicaciones de Negocio Electrónico," in I Simposio Español de Negocio Electrónico (SNE 2001), pp. 123-134, Oct. 2001.
- J.L.Muñoz, J. Castro, and J. Forné, "Estudio Comparativo de políticas de revocación de certificados: OCSP vs. Overissued-CRL," in III Jornadas de Ingeniería Telemática (JITEL 2001), pp. 413-420, Sept. 2001.
- J. Castro, J. Forné, and J.L.Muñoz, "Evaluación de políticas de revocación de certificados," in Congreso Iberoamericano de Telemática (CITA2001), Aug. 2001.

1.4 Thesis Organization

The layout of this Thesis is as follows:

- In Chapter 2, "The Certificate Revocation", we briefly present the certificate revocation paradigm, the reference model that we use to describe the elements and the mechanisms involved in the revocation process and the main approaches and standards that have been proposed in the literature.
- In Chapter 3, "Cervantes", we make a comprehensive description of Cervantes (Certificate Validation Test-bed). Cervantes is a client/server Java platform that has been developed by the authors to test, develop, and evaluate certificate revocation systems.
- In Chapter 4, "$\mathcal{H}$-OCSP", we review an architecture for m-commerce in which a broker is used as OCSP responder for the certificate validation. We also present a modification over OCSP called $\mathcal{H}$-OCSP. $\mathcal{H}$-OCSP is a way

to reduce the computational load and the bandwidth requirements of OCSP which is specially desirable in the wireless environment.

- In Chapter 5, "AD-MHT", we explain in detail one of the certificate revocation systems based on the Merkle Hash Tree (MHT) that we have developed. The revocation system is named AD-MHT and it uses the data structures proposed by Naor and Nissim in their Authenticated Dictionary (AD).

- In Chapter 6, "E-MHT", we present the Enhanced-MHT (E-MHT). The E-MHT is based on the AD-MHT but we add some mechanisms to the basic data structures of AD-MHT that allow the E-MHT to provide a response size that is close to (or even better than) typical online systems such as OCSP without de-gradating other resources of the system.

- Chapter 7, "Conclusions and Future Work", we conclude explaining the most remarkable results obtained from this thesis. Some guidelines for future research are given in this final chapter as well.

Chapter 2

The Certificate Revocation

2.1 Introduction

As we argued in the previous chapter when Alice wants to securely communicate with Bob, the exhibition of Bob's certificate is not a sufficient evidence for Alice. Alice also needs to "validate" the certificate. The validation of a certificate comprises two mechanisms:

- *Certification path validation.* Simplifying, this mechanism is necessary when Alice wants to establish a secure transaction with Bob and the CA of Bob's certificate is different from Alice's CA (see Section 1.2.6). Then, Alice needs a mechanism to build and validate a chain of certificates that finally certifies Bob's CA.
  The problem of verifying a certification path remains an open topic (see "Delegated Path Validation and Delegated Path Discovery Protocol Requirements" [PH02] for further information).

- *Certificate status checking.* This is the mechanism that defines how the status of a certificate (revoked/not revoked and perhaps other additional status data) must be communicated to relying parties. Among other things the status checking must define the data format to be exchanged and the entities that take part in this process. This thesis is focused on the study of this mechanism.

The rest of the chapter is organized as follows: in Section 2.2 we analyze the revocation mechanism. In Section 2.3 we review the main reasons for revocation. In Section 2.4 present the certificate revocation paradigm and the reference model that we use to describe the elements and the mechanisms involved in the revocation process. In Section 2.5 we show the main functions of the CMP protocol. In Section 2.6 we present the main approaches and standards that have been proposed in the literature regarding certificate status checking. Finally, in Section 2.7, we summarize the most remarkable features of the main status checking mechanisms.

2.2 Understanding revocation

The problem of certificate revocation is deeper than it seems at the first glance. Consider a digital certificate c_1 issued to Alice which contains Alice's public key (KU_{Alice}) and is signed by CA_1. Suppose Alice has another digital certificate c_2 with the same public key (KU_{Alice}) and issued by CA_2, different from CA_1. The problem arises if CA_1 revokes the certificate c_1. It means that c_1 is no longer valid but says nothing about c_2 which contains the same identity and public keys as that of c_1. Revocating c_1 could mean undo of any of the following:

1. KU_{Alice}: which means that Alice's public key could not be trusted anymore as they have been compromised.

2. $KU_{Alice} \leftrightarrow DN_{Alice}$: the binding between Alice's DN and her public keys have been compromised.

3. CA_1 binding on $KU_{Alice} \leftrightarrow DN_{Alice}$: the issuer could no longer vouch for the binding between the keys and the identity.

Each of these cases means something different, and chain processing in the presence of revocation information acts differently in each case. Consider the first case above, where revocation of c_1 denotes compromise of the subject public key. In this case, the fact that c_1 is revoked should cause all other certificates that involve the subject public key KU_{Alice} to be no longer valid. So, not only is c_1 now invalid, but c_2 itself is also now invalid. Ideally, if any certificate for a given subject public

key is revoked for reasons of key compromise, all such certificates would immediately be revoked, but obviously it is very difficult to guarantee this behavior. Thus, it may be argued that relying parties have a duty to check revocation status on all certificates naming a particular subject public key even if they themselves are not relying on those certificates for chain building.

The second case, direct revocation of the $KU_{Alice} \leftrightarrow DN_{Alice}$ binding by the issuer, is a fuzzier situation. As both the certificates c_1 and c_2 belong to Alice, the relying parties could reasonably choose to reject c_2 if they know c_1 is revoked (even though c_2 is not revoked explicitly).

Finally, in the third case we have revocation of certificate c_1 because the issuer of that certificate no longer has a relationship with the subject public key. Revocation here speaks not to the validity of the *name* $\leftrightarrow$ *key* binding but rather to a lack of contractual obligation. Revocation of c_1 should not in any way impact c_2 as there is no authorization statement from the issuer of c_2 concerning the validity of the subject public key KU_{Alice} itself. In this thesis we always assume the third case.

2.3 Reason for revocation

The fact that the act of revoking a particular certificate needed to be qualified with intended semantics was recognized by the authors of the X.509 standard. In X.509 it is possible to include a reason code for each revoked certificate. Reason codes are semantics modifiers and can specify situations such as:

- Key compromise.
- CA compromise (in this case all the certificates issued by this CA must be revoked).
- Affiliation change (including subject name changes).
- Superseded.
- Cessation of operation (the certificate is no longer needed for its original purpose).

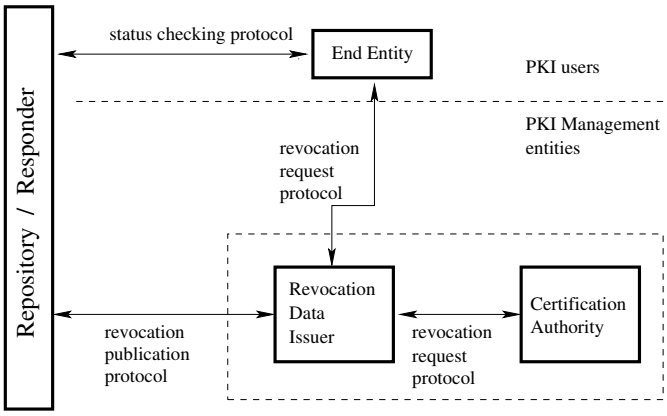


Figure 2.1: Reference Model

2.4 Certificate revocation paradigm

Figure 2.1 presents the reference model used in this thesis to describe the certificate revocation paradigm. In this reference model we have removed the entities and mechanisms that are not directly involved in the revocation process and we have added the things that were missing in the PKIX model (see Figure 1.3).

The revocation process starts with a request to revoke a certain certificate. Usually the owner of the certificate to be revoked, an authorized representative or the issuer CA can create revocation requests but, in general, the Certification Policy (CP) defines who is able to perform this task in each PKI domain. To revoke the certificate, an authorized entity generates a revocation request and sends it to the Revocation Data Issuer (RDI). RDI[1] is the term that we use to define the TTP that has the master database of revoked certificates. The RDI is also responsible for transforming the revocation records from the database into "status data". The status data has the appropriate format in order to be distributed to the EEs. The status data usually includes the following fields:

- *Certificate Issuer*. This is the DN of the CA that issued the target certificate or certificates.

[1]The CA that issued the certificate is often the one who performs the RDI functions for the certificate, but these functions can be delegated to an independent TTP.

- *Validity Period*. This period of time bounds the status data life-time (obviously this validity period is much smaller than the validity period of the certificate).
- *Issuer Name*. This is the DN of the TTP that issued the status data.
- *Cryptographic Evidence*. The evidence must demonstrate that the status data was issued by a TTP.
- *Serial Number*. This is the serial number of the target certificate or certificates.
- *Revocation Date*. This is the date when the target certificate was revoked.
- *Reason Code*. Optionally, a revocation reason for guidance can be specified. Standard revocation codes are: `unspecified`, `keyCompromise`, `cACompromise`, `affiliationChanged`, `superseded`, `removeFromCRL`, `cessationOfOperation` and `certificateHold`.

In the vast majority of the revocation systems, EEs do not have a straight connection to the RDI. Instead, the RDI publishes the status data in “repositories” or "responders". The main function of both repositories and responders is to answer the EE requests concerning the status of certificates (status checking). The difference between them is that repositories are non-TTPs that store status data pre-computed by the RDI while responders are TTPs that have a private-key and that provide a signature (that serves as cryptographic evidence) for each response.

Certificate status checking policies can be classified in different ways [Woh00]:

1. By the kind of status checking provided. The check can be performed either “offline” or “online”. Sometimes both methods are applied.
    - Offline scheme: the status data is pre-computed by an RDI and then it is distributed “offline” to the requester by a repository.
    - Online scheme: the status data is provided “online” by a responder and a cryptographic evidence is generated during each request. This can provide up-to-date information.

2. By their kinds of lists. They can be either negative or positive. Sometimes both mechanisms are combined.

   - Negative ("black") lists contain revoked certificates.
   - positive ("white") lists contain valid certificates.

3. By the way of providing evidence.

   - A "direct" evidence is given if a certificate is mentioned in a positive or negative list, respectively. Then it is supposed to be not revoked or revoked, respectively.
   - An "indirect" evidence is given, if a certificate cannot be found on a list and therefore, the contrary is assumed.

4. By the way of distributing information. It can be either via a "push" mechanism or "pull" mechanism.

   - Push mechanism: the repository or the responder periodically updates the client of the revocation.
   - Pull mechanism: the client asks the repository or the responder for status data.

A few requirements can be defined to evaluate revocation systems:

Population Size. The absolute size of the number of potentially revocable certificates can strongly influence the approach taken. Obviously, a solution intended to address a large population may require more resources and complexity as compared to a smaller group.

Latency. The degree of timeliness relates to the interval between when a RDI made a revocation record and when it made the information available to the relying parties. A more eager mechanism to update and convey this information will proportionally consume more bandwidth.

Connectivity. Does the relying party need to be online in order to ascertain the reliability? Online mechanisms create critical components in the overall security design because they make difficult to ensure that the system is operational and functional at any given moment.

Finally, it is worth mentioning that status checking is the mechanism that has the greatest impact on the overall performance of the certificate revocation system. Therefore, a status checking needs to be fast, efficient and timely, and it must scale well too. It is therefore necessary to minimize two fundamental parameters:

Processing capacity. The number of time-consuming calculations like generation and verification of digital signatures.

Bandwidth. The amount of data transmitted.

2.5 Certificate Management Protocols

The CMP (Certificate Management Protocols) [AF99] define the PDUs (Protocol Data Units) for the set of operations for all relevant aspects of certificate creation and management. These operations include CA establishment, EE initialization, registration, creation of certificates, key pair update (reissuation of certificates), certificate update, cross-certification request, cross-certificate update, key recovery operations, certificate publication, **CRL publication** and **revocation request.** Only the two last functions are relevant to this thesis. In Chapter 3 we will show how we manage these functions in the platform that we have developed.

2.6 Status Checking Protocols

2.6.1 Based on Lists

Traditional CRL (CRL)

The simplest approach for status checking is the traditional-CRL (Certificate Revocation List). CRL is the most mature approach and has been part of X.509 since its first version. A CRL is a “black” list of all revoked but not expired certificates

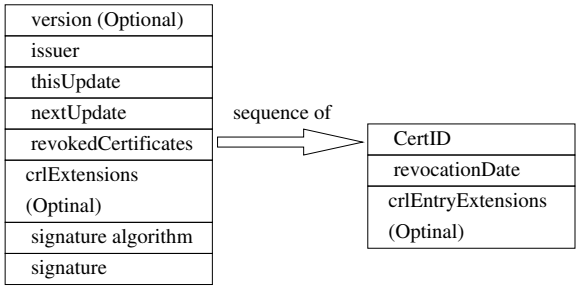


Figure 2.2: Field structure of a X.509v2 CRL.

issued by a certain CA. The integrity and authenticity of the CRL is provided by the digital signature appended to the CRL. In traditional-CRL the RDI of the CRL is the CA that signed the issued certificate. Notice that as the CRLs are signed, they can be distributed by means of repositories.

There are two versions of X.509 CRL. X.509v1 CRL was defined in the original X.509 specifications [X.588]. X.509v1 CRLs are inherently flawed due to scalability concerns (that is, the size of a X.509v1 CRL could easily grow beyond acceptable limits) and functionality limitations specifically related to the inability to extend the CRL with additional features when needed. Moreover, X.509v1 CRLs are subject to CRL substitution attacks (that is, it is possible to maliciously substitute one CRL for another without detection).

X.509v2 CRL [X.597] introduce the notion of extensions (similarly to extensions in X.509v2 ICs, see also Section 1.2.5). Certain extensions have been defined on a per revoked certificate entry basis, and others are defined on a per CRL basis. Extensions may be marked critical or non-critical. An extension marked critical should be processed and understood by the relying party while non-critical extensions may be silently ignored if they are not understood by the relying party. X.509v2 CRL has also been profiled by the IETF to realize interoperability in the Internet in [HFPS99]. The generic structure of a X.509v2 CRL is represented in Figure 2.2.

- *Version.* This is the version of the CRL (if present the value is 2, if not present it indicates that CRL is Version 1).
- *Issuer.* This is the DN of the CRL issuer.

- *ThisUpdate.* This is the the date the CRL was created.
- *NextUpdate.* Optionally, the date the next CRL is scheduled to be issued can be specified.
- *RevokedCertificates*. There is an entry for each revoked certificate. Revoked certificates are referenced by a unique identifier (that is, entries contain the unique serial numbers of the revoked certificates, not the actual certificates). Each entry also includes:
  - *RevocationDate.* This is the time that the certificate was no longer considered valid.
  - *entryCRLExtensions.* Optionally, it may include per entry extensions.
- *CRLExtensions.* Optionally, it may include CRL extensions.
- *Signature.* This is the signature of the CRL.
- *Signature Algorithm.* This is the algorithm used to generate the signature.

Some of the most relevant per entry extensions are:

- *ReasonCode.* The reason the certificate was revoked.
- *CertificateIssuer.* The name of the certificate issuer. If the Certificate Issuer extension is present, it must be marked critical in accordance with X.509 (see Indirect-CRL later in this Chapter).

Some of the most relevant per CRL extensions are:

- *CRLNumber.* Unique serial number relative to the issuer for this CRL. The serial number allows the detection of missing CRLs for any given CRL issuer. Although this extension is always marked non-critical, the generation of this field is mandated in the IETF profile [HFPS99, PFS02].
- *IssuingDistributionPoint.* This parameter indicates the name of the CRL distribution point and the types of certificates contained within the CRL (see

Distribution Points later in this chapter). When applicable, it is also used to indicate that the CRL is an Indirect CRL (see Indirect-CRL later in this Chapter). This extension, if present, must be marked critical in accordance with X.509.

- *Delta-CRLIndicator.* This extension indicates that this CRL is a Delta-CRL as opposed to a Base-CRL. This extension, if present, must be marked critical in accordance with X.509 (see Delta-CRL later in this Chapter).

Overissued CRL (O-CRL)

The traditional method of issuing CRLs involves a CA periodically publishing a CRL. Since CRLs may have a large size, they are usually cached by the client during their validity period to enhance performance. Caching of CRLs also facilitates the ability to verify certificates while working offline. There is a problem with the traditional method of issuing CRLs because the CRLs cached by every relying party expire at the same time. Immediately after the CRLs expire, and a new CRL is issued, every relying party will need to obtain a CRL from the repository in order to perform the status checking. As a result, there is a relatively high request rate when a new CRL is issued while if CRLs are valid for a relatively long period of time, then there will be also periods of time in which the repository is practically unused. *Overissued* CRL (O-CRL) [Coo99] addresses a way of reducing the peak request rate of CRLs towards the repositories and evenly distribute the requests across time. This is achieved by allowing multiple CRLs to have overlapping validity periods. O-CRL simply consists in issuing more than just one CRL during a validity period. The result will be that the CRLs in relying parties' caches will expire at different times and so requests to the repository for new CRLs will be more spread out (see performance evaluation about this in Section 3.7.1).

Indirect CRL (I-CRL)

Indirect CRL (I-CRL) enables a RDI to pick revocation records up from multiple CAs to be issued within a single CRL. I-CRLs can be used to reduce the number of overall CRLs that need to be retrieved by relying parties when performing the

certificate validation process. For example, a single PKI domain may have several CAs. Rather than force a relying party to retrieve multiple CRLs (one for each CA), the domain may decide to improve efficiency by combining all of that domain's status data into one Indirect CRL.

The I-CRL is based on the same construct that defines a traditional-CRL, but there are certain extension values that distinguish an Indirect CRL from a CRL: the Indirect CRL Boolean attribute within the *IssuingDistributionPoint* extension of the X.509v2 CRL would be set to TRUE to indicate that this CRL contains revocation information from multiple CAs. Given that the revocation information is originating from multiple sources, it is necessary to identify which CA is associated with the individual entries in the list of revoked certificates. This is accomplished with the *CertificateIssuer* extension associated with each entry.

CRL Distribution Points (CRL-DP)

In CRL Distribution Points (CRL-DP), each CRL contains the status information about a certain subgroup of certificates of the CA domain. This approach allows status data from a single CA to be posted in multiple more manageable CRLs. Each certificate has a pointer to the location of its DP (Distribution Point), so there is no need to either search through distribution points or have a priori knowledge of the revocation information locations.

The syntax of the *IssuingDistributionPoint* extension of the X.509v3 IC enables one to identify the specific location of the corresponding CRL partition. In this sense, the DP can identify a specific server, as well as the specific location within that server where the CRL partition can be found. In summary, CRL Distribution Points offer a much more scalable alternative as compared to complete CRL postings. They can also be used to alleviate the performance issue when combined with proper partitioning and caching.

One drawback associated with the use of standard CRL-DP is that once the associated certificate is issued, the CRL partition pointed to by the DP is fixed for the life of that certificate. It also implies that the issuing CA has a prior knowledge regarding how the CRL information should be partitioned, and that this partitioning cannot change over time. However, it may be desirable to make this more flexible

so the CRL partition sizes and storage locations may vary over time. Further, partitioning criteria could be based on a number of elements, including certificate serial number ranges, revocation reasons, certificate types, or any other range criteria that might apply to CRL information. In order to implement these more flexible and dynamic partitioning criteria is is necessary to define new CRL extensions.

Delta CRL (D-CRL)

Delta-CRL (D-CRL) is an attempt to reduce the size of the CRLs in order to enhance timeliness without significantly impacting performance. Delta-CRLs can be used in conjunction with full CRL postings or with CRL-DP. The idea behind Delta-CRLs is to allow incremental publications of status data, without requiring the generation of a complete, potentially voluminous CRL each time a certificate is revoked. However, D-CRLs do not eliminate the requirement for either a full CRL posting or, alternatively, a CRL-DP. Delta-CRLs are, by definition, based on some previously posted information. This previous posting is referred to as a Base-CRL , and the Delta-CRL contains status data about the certificates whose status have changed since the issuance of the Base-CRL. This allows for the publication of relatively small Delta-CRLs that can be issued on a much more frequent basis than the Base-CRL, thus optimizing the performance and timeliness.

It is possible to create and post multiple Delta-CRLs against the same Base-CRL. Each subsequently issued Delta-CRL contains the complete list of revoked certificates from the previously issued Delta-CRL plus any new certificates that have been revoked. Thus, it is only necessary to retrieve the latest Delta-CRL; it is not necessary to accumulate previously issued Delta-CRLs. Delta-CRLs can also be cached until the validity period associated with the Delta-CRL expires. Alternatively, caching can be prohibited so that a Delta-CRL would have to be retrieved every time a given certificate is validated.

Windowed Certificate Revocation (WCR)

According to [MJ00] there are two fundamental approaches to the distribution of information about revocation of certificates: explicit and implicit.

In PKIs that employ explicit revocation, each issuer explicitly states which

certificates are revoked, and indirectly which are not revoked. For instance, in systems based on the X.500 standard, each issuer periodically generates a CRL. The presence of the certificate in the list "explicitly" states revocation. In PKIs that employ implicit revocation, lack of revocation is asserted implicitly through the verifier's ability to retrieve the certificate. Any certificate retrieved from the issuer is guaranteed to be valid at or near the time of retrieval. Associated with each certificate is a time to live (TTL), which represents the maximum time the certificate may be cached. Thus, in implicit revocation, the window of vulnerability is the TTL. The performance of a system that uses implicit revocation is limited by the cost of acquiring certificates: supplying real-time information on revocation status during each acquisition is computationally expensive.

Windowed revocation involves a hybrid of explicit and implicit revocation that affords the desired scalability. In windowed revocation, the issuer asserts revocation in two different ways at two different times: (1) implicitly during initial acquisition of a certificate, and thereafter (2) explicitly through periodically published CRLs. Verifiers acquire CRLs from issuers directly. Retrieved certificates are guaranteed to be non-revoked, fresh, and authentic. Subsequent validation of the revocation statuses of certificates is effected primarily through CRLs. CRLs are generated at uniform time intervals and revoked certificates are mentioned in the CRLs that occur during possibly longer intervals denoted as "revocation windows". A revocation window is the time during which a certificate may be cached without further validation. The revocation window is specified by the issuer and documented in each certificate. By bounding the times during which each revoked certificate must be included in the periodic CRLs, revocation windows limit the sizes of CRLs and thus the costs of distributing them.

2.6.2 Systems Based on Online Signatures

Some online status checking mechanisms have been proposed in the literature (OCSPv1 [MAM+99], SCVP [MHHF02], OCSPv2 [MAA00], OCSP-x [HB99] etc.) but finally only the Version 1 of the Online Certificate Status Protocol (OCSP) has become popular. The OCSP has been proposed by the PKIX workgroup of the IETF. OCSP is a relatively simple request/response protocol that allow certificate-

using applications to determine the revocation status of an identified certificate. The status data is available online from TTP referred to as an OCSP responder.

An OCSP request is comprised of the protocol version number (currently only Version 1 is defined), the service request type, and one or more certificate identifiers. The certificate identifier consists of the hash of the certificate issuer's DN, the hash of the issuer's public key, and the certificate serial number. Additional optional extensions may also be present.

Responses are also fairly straightforward, consisting of the certificate identifier, the certificate status (that is, "good," "revoked," or "unknown"), and the validity interval of the response associated with each certificate identifier specified within the original request. If the status of a given certificate is "revoked," the time that the revocation occurred is indicated and, optionally, the reason code for revocation may also be included. Like the request, the response may also contain optional extensions. OCSP also defines a small set of error codes that can be returned in the event that processing errors are encountered. The OCSP responses can contain three times:

- `thisUpdate` is the time at which the status being indicated is known to be correct.
- `nextUpdate` is the time at or before which newer information will be available about the status of the certificate.
- `producedAt` is the time at which the OCSP responder signed this response.

Whether or not the OCSP response can be cached locally will ultimately be a policy decision dictated by the governing domain.

The interaction between a relying party and an OCSP responder is illustrated in Figure 2.3. The Figure also illustrates that numerous revocation strategies can be implemented behind the OCSP responder. On the other hand, OCSP responses must be digitally signed to provide assurance that the response is originating with a trusted entity and that it is not altered in transit. The signing key used by the responder may belong to the same CA that issued the subject certificate or it could be a brand new signing key that has been approved through delegation by the CA that signed the subject certificate. In any case, the relying party must be able to

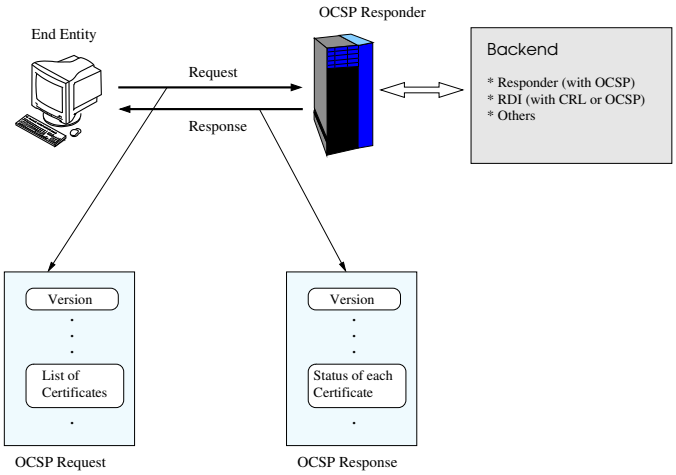


Figure 2.3: Using OCSP.

trust the response, which inherently implies that the signing public key must be trusted by the relying party. The relying party must, therefore, obtain a copy of the OCSP responder's public key certificate, and that certificate must be signed by a trusted source. Requests may also be signed, but this is an optional feature within the protocol that depends on the particular CP of the PKI domain. Location(s) of the OCSP responder(s) applicable to a particular certificate can be conveyed as part of the certificate itself (much the same as CRL-DP). Alternatively, the locations of one or more OCSP responders can be configured locally or via some other means.

On the other hand, there seems to be some confusion regarding the utility of this protocol, especially in the sense of whether or not it can offer both real-time and up-to-date information regarding the revocation status of a given certificate. While the protocol itself offers a real-time response (assuming an appropriate OCSP responder is available online to service the requests), it does not necessarily mean that the reply from the OCSP responder will comprise a zero latency response regarding the current revocation status of the certificate. Stated another way, OCSP is nothing more than a protocol. It does not specify the backend infrastructure that might be used to collect the revocation information. Thus, it does not necessarily eliminate the need for CRLs or other methods for collecting status data, and the "freshness" of the information supplied by the OCSP responder will be only

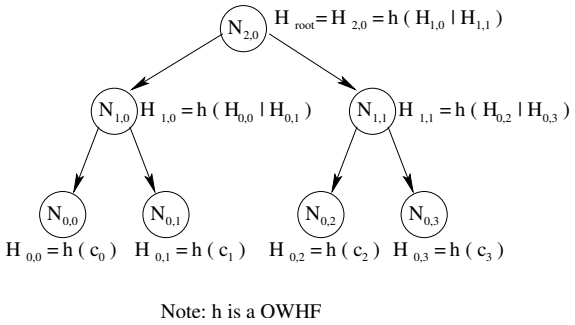


Figure 2.4: Sample MHT.

as up-to-date as the latency involved in obtaining the revocation information from their definitive source. In addition, the responses from an OCSP responder must be digitally signed, and this may result in a significant performance impact. However, it also may be possible to pre-produce at least some subset of responses, which can be used to help alleviate some of the performance overhead (see Chapter 4 for further information on this).

2.6.3 Systems based on the Merkle Hash Tree

There are some revocation systems that are based on the Merkle Hash Tree (MHT). The MHT [Mer89] relies on the properties of the OWHF. It exploits the fact that an OWHF is at least 10,000 times faster to compute than a digital signature, so the majority of the cryptographic operations performed in the revocation system are hash functions instead of digital signatures. A sample MHT is represented in Figure 2.4.

The MHT allows content to be retrieved in a trusted fashion with only a small amount of trusted data. The content is stored in the leaves of the MHT. A leaf is combined with other leaves by hashing their contents to generate a node in the upper level of the tree. By recursively applying this computation, the last level of the tree only contains one node that is called the "root". A TTP must sign the root to ensure authenticity and integrity of the MHT.

The presence of certain data in the MHT can be determined by verifying that there is a $\mathcal{P}$ath that cryptographically binds the leaf (that contains the requested

data) to the root. Thus, users need the $\mathcal{P}$ath for the leaf that contains the requested data (i.e the nodes necessary to compute the root) and the root signed by the TTP to verify a response from the MHT. A response is verified by checking that the signed root is identical to the root computed from the $\mathcal{P}$ath. Below we briefly describe the main MHT-based systems, anyway we recommend to take a look at Section 5.2.1 for further information on the MHT.

The Certificate Revocation Tree (CRT)

In CRT (Certificate Revocation Tree) [Koc98], the RDI is called the CRT-issuer. A CRT-issuer can serve more than one CA, so it must obtain the revocation records from all the CAs served. Data stored in the leaves of the MHT are expressed as a sequence of statements for each participating CA. Each statement is a condition on the serial numbers of the certificates and on which CA issued them. Statements are ordered sequentially relative to a given CA, and the set of statements for a given CA is also sequentially ordered relative to all the other known CAs. Below we show a sample CRT statement literally taken from [Koc98]:

$$c_j : \ CA_x = CA_2 \ \textit{and} \ \ 156 \leq X < 343 \tag{2.1}$$

The statement c_j indicates that the certificate with serial number $X =$ 156 issued by CA_2 has been revoked, while the certificates with serial numbers from $X = 157$ to $X = 343$ (both included) issued by CA_2 have not been revoked.

A possibility for computing $H_{0,j}$ for c_j can be

$$H_{0,j} = h(CA_2|156|343) \tag{2.2}$$

.

Where "|" means concatenation. On the other hand, CRT is a binary balanced tree: binary because each internal node within the tree has no more than two children[2] and balanced because there is the same number of levels from any leaf within

[2]If the CRT is formed by an odd number of leaves, there is a leaf $N_{0,n-1}$ that does not have a pair. Then, the single node is simply carried forward to the upper level by hashing its $H_{0,n-1}$ value.

the tree to the root. The main drawback of CRT is the management of dynamism of the binary tree. That is, searching, adding and removing a leaf might be performed in the worst case in $o(n)$ [AHU88] where n is the number of leaves. This is an important problem taking into account that the freshness requirements of the revocation may lead the MHT to be updated frequently. In [KAN00, KAN01] the authors use the MHT with random insertions and they propose a post-processing algorithm in order to balance the CRT every time that a new certificate is added.

The Authenticated Dictionary (AD)

The Authenticated Dictionary (AD) [NN00] improves the management of the tree dynamism, because the tree structure chosen to build the MHT is a 2-3 tree. A 2-3 tree is a balanced tree in which each internal node has two or three children. The main advantage of this type of tree is that management tasks such as searching, adding and removing a leaf can be performed in $o(log(n))$ [AHU88] where n is the number of leaves. The data that stores a leaf in AD is basically the serial number of a certificate that has been revoked and leaves within the tree are ordered relative to the serial number they store. The AD is further discussed in Chapter 5 were we also address many open questions to implement such a system.

Efficient and Fresh Certification (EFECT)

A CA in the EFECT (Efficient and Fresh Certification) scheme [GGM00] reissues every certificate on a periodic basis. Each certificate that has been revoked is altered in order to reflect its revocation. Frequent reissuing is not feasible using the classical signature scheme because signing millions of individual certificates takes many time if done as a serial computation. EFECT can do a more frequent reissuing because the CA does not sign individual certificates but it signs all the certificates in a single digital signature, making the computational overhead much more reasonable.

The EFECT CA does this by arranging the valid certificates as the leaves of a MHT. An EFECT certificate contains a body (the certified information), the $\mathcal{P}$ath for the authentication of the leaf that represents the certificate and the root of the MHT signed by the CA. By verifying the EFECT CA's signature on the root, and

hashing the certificate body upwards with the provided $\mathcal{P}$ath, the verifier knows that if the signed root matches her computed root, the certificate is valid. Notice that there is no revocation check since EFECT certificates are issued for a certain periods and are never revoked.

2.6.4 Other systems

Skip Lists with Commutative Hashing (SLCH)

Skip Lists with Commutative Hashing (SLCH) is based on two new ideas (see [GT00] for further information). First, rather than hashing up a dynamic 2-3 tree, the hashes are performed in a skip list [Pug90]. Second, the use of commutative hashing is introduced as a means to simplify the verification process for a user, while retaining the basic security properties of signing a collection of values via cryptographic hashing. The performance of SLCH is similar to the hash tree structures but the authors of SLCH claim that a skip list is easier to program than a hash tree structure.

Certificate Revocation System (CRS)

Micali proposed the Certificate Revocation System (CRS) recently renamed as Novomodo (see [Mic96] and [Mic02] for further information). In CRS/ Novomodo the RDI periodically sends to each repository the list of all issued certificates, each tagged with the signed time-stamped value of a OWHF that indicates if this certificate has been revoked or not. This approach allows the system to reduce the bandwidth used in the status checking: namely just a certificate identifier and a hash value indicating its status. Unfortunately, in this scheme it is necessary to publish data for every single issued certificate, revoked or not. This requires the size of the publication to increase to $o(N)$, where N is the number of all non-expired certificates issued by the certifying authority, which is typically much larger than the number of revoked certificates.

Hierarchical CRS (HCRS)

Aiello et al. suggested an improvement over CRS called Hierarchical Certificate Revocation System (HCRS) [ALO98]. HCRS focuses on reducing the transmission resources used to publish the status data. To do so Aiello et al. start with the observation that in the Micali's scheme the CA sends to the repositories exactly the same amount of information about each certificate. The authors try to reduce this by organizing a clever tree and introducing a notion of complement cover families (see [ALO98] for further information).

2.7 Summary

In this Section we summarize the main features of the three most relevant revocation systems [HMR01]: List-based, Online-based, and MHT-based systems.

	Security
List-based	In general, CRLs provide strong authentication and integrity services via their digital signature.
Online-based	The OCSP scheme assumes the responder is a TTP. Thus, firstly the the end entity must trust the responder's public key to verify the signature on the certificate status response. Secondly, the responder has to be online, but at the same time it has to protect its private key against intruders.
MHT-based	The security of the MHT-based is comparable to the other schemes. The integrity protection of the status data is excellent, and the scheme allows for untrusted repositories since the RDI signs the root of the MHT it is impossible for the repository to change status data.

	Freshness and timeliness
List-based	In order to assure maximum freshness, applications must search for the most recent CRL according to their respective CP. In a large-scale PKI, CRLs get rather large and in order to keep the timeliness mechanisms such D-CRL or CRL-DP must be applied.
Online-based	The freshness of certificate status information from the OCSP responder is only as fresh as the information from the backend mechanism upon which it relies.
MHT-based	The timeliness of the MHT-based schemes depends on how fast the RDI can update the MHT, generate a signature on the root and send this information to the repository, and how fast the repository can update the MHT and check the signature.

	Standards compliance
List-based	The X.509v2 CRL model is standards compliant. This standardization is probably the driving force behind its widespread usage today, despite obvious shortcomings.
Online-based	OCSPv1 is also an standard protocol documented by the IETF.
MHT-based	The model has not been standardized and the MHT-based systems are still “research products”. However, as we show in Chapter 5 it is possible to use this revocation scheme with X.509 certificates without problem.

	Downlink Bandwidth (Status Checking)
List-based	CRLs are the most bandwidth-intensive choice of revocation system: the number of revoked certificates can be assumed to be proportional to the number of users, and this should be multiplied with the number of users for bandwidth usage during transmission of pushed CRLs, so the downlink bandwidth usage in the status checking grows with the square of the number of users. Note: we show performance evaluation results of the downlink bandwidth in list-based systems in Section 3.7.
Online-based	Depending on the number of certificates included in a client's bulk request, the size of the response may vary widely. For most applications or users, single certificate status queries rather than bulk will be the predominant method so the response size is fixed and in general the bandwidth required by OCSP can be considered low. Note: we show performance evaluation results of the downlink bandwidth in online-based systems in Section 3.7.
MHT-based	The bandwidth for the status checking is comparable but worse than the online scheme. This is because in addition to the root signed by the RDI, the $\mathcal{P}$ath for the queried certificate must be sent. Note: we show performance evaluation results of the downlink bandwidth in MHT-based systems in Section 5.7.

Processing capacity

List-based	Traditional-CRLs require little or no processing on the part of the repositories, who only pass on information given to them by the CA. The CA only needs to sign the list periodically, and the users need only verify that signature. D-CRLs and O-CRL require more signature generations and verifications. Note: we show performance evaluation results of the processing capacity in list-based systems in Section 3.7.
Online-based	As the number of users increase, more responders must be added to meet the demand. In this respect, OCSP scales linearly. Performance of the individual responder hinges on its ability to handle the processing burdens imposed by digital signatures. However, assessing peak processing loads for ensuring optimal timeliness and scalability is a difficult task. To mitigate the difficulty, OCSP offers a feature that allows a responder to pre-produce responses and attach a validity period to each. Note: we show performance evaluation results of the processing capacity in online-based systems in Sections 3.7 and 4.8.
MHT-based	MHT-based systems scale much better than online schemes in terms of processing capacity consumed in the repository because only OWHFs and searches in the MHT are performed. Note: we show performance evaluation results of the downlink bandwidth in MHT-based systems in Section 5.7.

	Granularity
List-based	There is good granularity of information in all CRL models. All the relevant information can be placed without problems within the CRL either in the basic structure or in the extensions using the X.509v2 of the CRL.
Online-based	The OCSP standard includes all the extensions available as CRL extensions. OCSP also supports querying for ranges of certificates.
MHT-based	In Chapter 5 we show how the status data can be stored within a MHT-based system. In particular in our implementation we include the revocation date and code for each revocation entry. On another note, it is possible to query the directory for ranges of certificates. Specifically, proving that a range of certificates are valid is no different (in terms of the algorithm used as well as the bandwidth requirements) than proving that a single certificate is valid (but we have not implemented this functionality).

Chapter 3

Cervantes (Certificate Validation Test-bed)

3.1 Introduction

In this chapter we make a comprehensive[1] description of Cervantes[2] (Certificate Validation Test-bed). Cervantes is a client/server open source software project that has been developed by the author to test, develop and evaluate certificate revocation systems. Furthermore, Cervantes is not a simulator but a test-bed, in other words, Cervantes implements all the tasks that take place in a real revocation system.

Figure 3.1 shows the relationship between Cervantes and the revocation reference model presented in Chapter 2. The client of Cervantes implements the functions performed by the EE while the server of Cervantes is a single program that implements the functions of the RDI, the responders and the repositories.

Java was chosen to develop Cervantes because it is network oriented and unlike other programming tools, security mechanisms have always been an integral part of it. Another interesting feature of the Java software is that it runs almost over any operating system. Cervantes is programmed using *j2sdk1.4.1* which contains the Java Cryptography Extension (JCE) API. We take advantage of this API that

[1] An exhaustive description of Cervantes would take us too much room and it is out of the scope of the document.

[2] The Cervantes home-page is `http://isg.upc.es/cervantes`

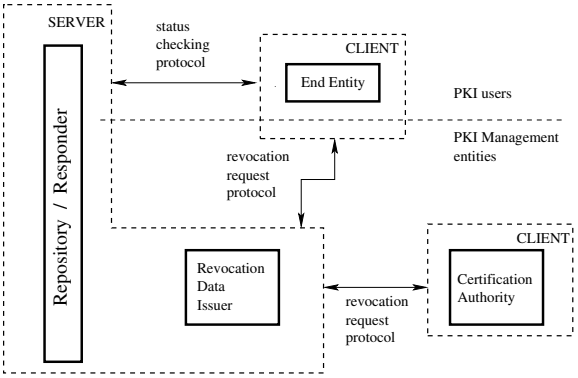


Figure 3.1: Client/Server programs of Cervantes

allows an easy use of the different security tools such as symmetric encryption, public key cryptography, certificates management, key management, etc.

The software of Cervantes is organized in three major parts: the Client, the Server and the Library. Cervantes has a modular design in each of its parts and a well-defined API between modules which allows any developer to extend the platform without re-writing the source code.

The rest of the chapter is organized as follows: in Section 3.2 we describe the Library of Cervantes. In Section 3.3 we describe the organization and the modules of the Server. In Section 3.4 we describe the Clients of Cervantes. In Sections 3.5 and 3.6 we show how CRL and OCSP are implemented in Cervantes, respectively. In Section 3.7 we use Cervantes to obtain some evaluation results for OCSP and CRL. Finally we conclude in Section 4.9.

3.2 The Library

The Library is common to the clients and the server and it is necessary to run any of them. The Library is divided in two parts: the utilities and the ASN.1 protocols.

- **The utilities** part of the Library contains classes that provide miscellaneous functions to the server and the clients such as hexadecimal conversion, HTTP encapsulation etc.

- **The ASN.1 protocols** part of the Library contains the classes necessary to manage the Protocol Data Units (PDUs) that communicate the server with the clients and vice-versa. At the moment, most of the protocols and data structures used in computer security are defined in ASN1. ASN.1 (Abstract Syntax Notation number One) is an international standard [X.695a] which aims at providing a means to formally specify data structures independent of machine or transfer syntax. Therefore, one can define with ASN.1 the PDUs to be exchanged between heterogeneous systems. An ASN.1 PDU definition consists in an ASCII file which contains the data structures and the messages used in a certain protocol. As Cervantes is built in Java, we need to convert these ASN.1 definitions into Java classes. In order to perform this conversion, the ASN.1 definition file of the PDU is used as input to an ASN.1-to-Java compiler. In particular, we use *Snacc4Java v2.3* [SNA]. The Java application fills up the fields of the objects that represent the PDU and then, these objects are encoded with one of several standardized encoding rules to produce an efficient bit-pattern representation which can be later transmitted.

  In computer security, ASN.1 data is normally encoded using the Distinguished Encoding Rules (DER) [X.695b] because these rules give exactly one way to represent any ASN.1 value as a bit-pattern. Thus, DER is appropriate for applications in which a unique bit-pattern encoding is needed, as is the case when a digital signature is computed on an ASN.1 value. In our case, the *Snacc4Java* libraries let us perform the DER encoding of the ASN.1 objects. DER encoded PDUs can be sent using many operational mechanisms such as raw sockets, HTTP, FTP, LDAP, SMTP, and X.500. So far, Cervantes only supports raw-sockets and HTTP. However, this already does not imply a great loss of inter-operability because HTTP is the most widely spread transport mechanism among the PKI products. When HTTP is used as operational protocol, the HTTP header contains the `Content-Type` set to the proper MIME type and the `Content-Length` set to the total length in bytes (including the header), while the body contains the bit-pattern corresponding to the DER encoded PDU.

3.3 The Server

3.3.1 Organization

From the logical point of view[3] [MFES], Cervantes is organized in four modules (see Figure 3.2): the Input/Output (I/O), the Database (DB), the Central Management (CM) and the Status Checking Handlers (SCHs).

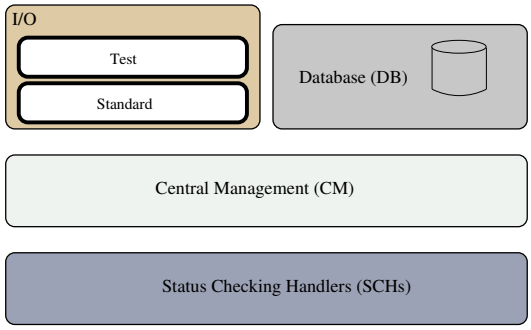


Figure 3.2: Logical structure of Cervantes

The behaviour of the modules can be controlled by means of a text file. In the following sections we will explain how each particular module works as well as the configuration parameters of each module.

3.3.2 The Database Module (DB)

This module contains the classes necessary to retrieve, add, delete or modify the information that contains the database. The database is extern to Cervantes. Any SQL-based database that can be accessed though the JDBC API can be used. In particular, we have chosen *PostgreSQL v7.2.2* because it is very stable and it is an open source project as Cervantes. Figure 3.3 shows the design of the tables used to build the database of revoked certificates and the relationship among these tables.

- *issuers*. This table contains information about the issuers of certificates (CAs) that are under control of Cervantes. Each issuer is identified by two fields: the hash of its DN and the hash of its public key.

[3]The organization of the Java classes of Cervantes is out of the scope of this document and you can find it in [MF].

- *revoked_certs*. This table contains the status data about the certificates that have been revoked. A record of a revoked certificate is uniquely identified by first the serial number, second the hash of the issuer's DN and third the hash of the issuer's public key. The status data stored for a revoked certificate consists in the revocation date and the revocation code. Finally, for each revoked certificate it is also stored the expiration date (i.e. the `not-valid-after` time stamp included in the certificate) because the certificates that have already expired should be removed from the database.
- *revocation_reasons*. This table contains the list of possible revocation codes.

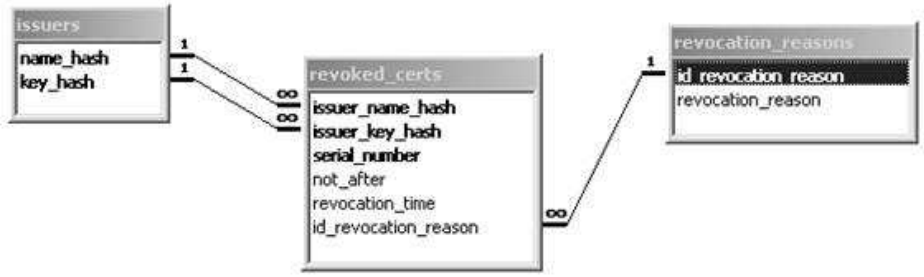


Figure 3.3: Database design

The DB module provides an API to the other modules of Cervantes. This API defines all the functions that are required by the rest of the modules of the system, these functions permit:

- Retrieve all the records from the database.
- Ask for the number of records that has the database (the number of revoked certificates currently present at the system).
- Figure out if a certain record is stored in the database.
- Delete all the records of the database (empty the database).
- Insert a revocation record (revoke a certificate).
- Delete a revocation record (remove a record of an expired certificate).

The configuration parameters of the DB module are shown in Figure 3.4.

clear database contents: When the server is started, it can keep the records from a previous execution (for instance, if you temporary stop and restart the server or if a server crash occurred) or it can be started with an empty database.
database base dir: This is the path to the directory that contains the database.
JDBC driver: This is the JDBC driver of the database (in the case of PostGreSQL this value is `org.postgresql.Driver`).
JDBC url: This is the url to the host in which it is placed the database server (this can be the host in which it is running the Cervantes server or a different host).

Figure 3.4: Configuration parameters of the DB module.

3.3.3 The Input/Output Module (I/O)

The I/O module contains the classes necessary to generate the database inputs (revocations) and outputs (expirations). The inputs and outputs can be generated in two ways depending on the server configuration: the server can run in standard mode or in test mode.

The Standard Sub-Module

In standard mode the requests to revoke a certificate are generated by the EEs or by other authorized entities using the Simple Certificate Revocation Protocol (SCRP) [MF] (Figure 3.5 shows the ASN.1 description of SCRP).

The SCRP is a lightweight version of CMP that we have developed. We decided not to use straight CMP because it agglutinates much functionality that is beyond the functionality required by a revocation system (see Section 2.5). In Cervantes, we only need a protocol to revoke certificates so SCRP is only intended for revocation purposes.

The SCRP protocol works as follows: the SCRP client sends a request to the server. The request includes the fields required to uniquely identify the target certificate to be revoked and optionally the signature of the client. The presence of this signature depends on the CP of your PKI but it will be enforced in most of the cases. Next, the SCRP server checks if the client is authorized to perform the revocation and sends the result of the operation back to the client. The result of a revocation can be `successful`, `alreadyRevoked`, `internalError`, `sigRequired`

```
SCRP DEFINITIONS EXPLICIT TAGS::=
BEGIN
IMPORTS
-- PKIX Certificate Extensions CRLReason FROM PKIX1Implicit88
{iso(1) identified-organization(3) dod(6) internet(1) security(5)
mechanisms(5) pkix(7) id-mod(0) id-pkix1-implicit-88(2) }
-- AlgorithmIdentifier, Certificate FROM PKIX1Explicit88 {iso(1)
identified-organization(3) dod(6) internet(1) security(5)
mechanisms(5) pkix(7) id-mod(0) id-pkix1-explicit-88(1) };
AlgorithmIdentifier ::= SEQUENCE {
algorithm  OBJECT IDENTIFIER,
parameters NULL OPTIONAL}
SCRPRequest ::= SEQUENCE {
tbsRequest Request,
signature  Signature }
Signature ::= SEQUENCE {
signatureAlgorithm AlgorithmIdentifier,
signature          BIT STRING,
certs          [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL }
Request ::= SEQUENCE {
certToRev            Certificate,
issuerCert           Certificate,
revocationTime       GeneralizedTime,
revocationReason [0] EXPLICIT CRLReason OPTIONAL }
SCRPResponse ::= SEQUENCE {
tbsResponse  Response,
signature    Signature }
Response ::= SEQUENCE {
certToRev  Certificate,
result     Result }
Result ::= ENUMERATED {
successful      (0), --ok
alreadyRevoked  (1), --already revoked
internalError   (2), --bad news
sigRequired     (3), --Must sign the request
unauthorized    (4)  --Request unauthorized }
END
```

Figure 3.5: ASN1 definition of SCRP.

and `unauthorized`. The configuration parameters of an SCRP server are shown in Figure 3.6.

It must be noticed that you can define as many SCRP servers as you like but

type of server: The type of server must be set to: `SCRP`.
class for server: The class of the server must be set to: `es.upc.isg.cervantes.server.scrp.SCRPPortThread`.
name: This is an arbitrary name to distinguish the different SCRP servers.
server log: This is the path to the log file in which the SCRP server stores the logs of its execution for debugging or tracing.
statistics log: This is the path to the file in which the SCRP server stores its statistics.
operational protocol. This is the underlying protocol that will transport the SCRP PDUs. This can be HTTP, raw or <auto-detect>. With the last configuration the server will detect and use the operational protocol used by the client.
SCRP server certificate: This is the path to the certificate used by the SCRP server to sign its responses.
SCRP server private key: This is the path to the file that stores the private key associated with the certificate of the SCRP server.
port: The number of TCP port used to listen to requests.

Figure 3.6: Configuration parameters of the SCRP server.

you must take into account that in standard mode it is compulsory to configure at least one SCRP server while in test mode the SCRP servers will be ignored.

Figure 3.7 shows the organization of the standard sub-module and the transactions that take place among this sub-module and the rest of the modules.

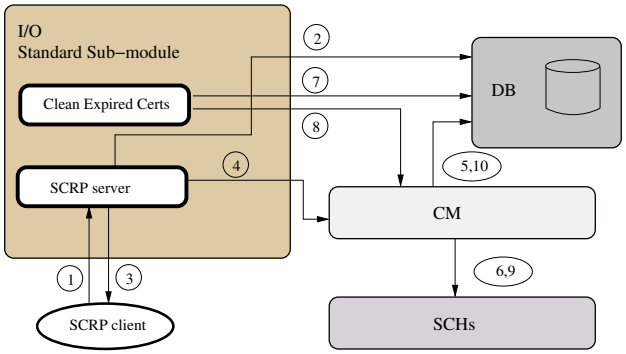


Figure 3.7: The I/O sub-modules organization

1. An SCRP client sends a request to revoke a certificate.

2. The SCRP server verifies that the target certificate has not been already revoked (the record does not already exist in the database), the client is authorized to perform the revocation request and the client's signature (if presence of the signature is mandatory).

3. The SCRP server sends the result of the revocation back to the client.

4. For the `successful` revocations, the I/O module sends the associated data to the CM.

5. The CM inserts the new revocations in the database.

6. The CM also informs the SCHs about the new revocations (there are some status checking protocols that need to be informed every time the system has "fresh" status data).

7. The CleanExpiredCerts is a low priority execution thread that periodically traverses the database in search of records of expired certificates.

8. The CleanExpiredCerts sends the identifiers of the expired certificates it has found to the CM.

9. The CM informs the SCHs about the new revocations.

10. The CM deletes the records of expired certificates from the database.

The Test Sub-Module

The test sub-module works similarly to the standard sub-module. The difference is that now the revoked and expired certificates are randomly generated. Figure 3.8 shows the configuration parameters of the test sub-module.

Notice that the user can control the random generation process by configuring the certificates population (i.e. the number of certificates "N"), the percentage of revoked certificates "r" and the average rate of events "λ_{events}".

When a test is running, two execution threads: the revocations generator and the expirations generator, are in charge of generating radom revocations and random expirations at the correct average rate. The generation of random events is divided in two phases:

population size: This is the number of certificates issued by the CA.
%revocation: This is the percentage of the certificates population that have been revoked.
events rate: This is the average number of events (expirations/revocations) which follow an exponential probability density function.
test duration: This is the period of time that the test will last.
revocations log: This is the path to the file in which the revocations and expirations are logged.
issuer certificate: This is the path to the certificate of the Certification Authority that issued the certificates.

Figure 3.8: Configuration parameters of the Test sub-module.

Static phase. In this phase the revocations generator works at a very hight rate whereas the expirations generator remains in standby. The tests are always started with an empty database and basically the objective of this phase is fill with revocation records the database until the average number of revoked certificates "n" is reached: $n = N * r$.

Dynamic phase. Once the database has reached the average number of revoked certificates, the test sub-module reduces the rate of the revocations generator and starts the expirations generator with a proper rate in order to keep the system dynamic and stable. The objective is insert and delete records in the database at a rate λ_{events}.
As the target certificate for either a revocation or an expiration is randomly chosen, there are events that do not have any effect. For instance, nothing will happen if the revocations generator generates a revoked certificate that is already contained in the database or the expirations generator generates an expiration for a certificate that it is not in the database. It follows from the previous discussion that the average rate of events of the revocation generator "$\lambda_{revocations}$" must be

$$\lambda_{revocations} = \frac{\lambda_{events}}{(1-r)} \tag{3.1}$$

While the average rate of events of the expirations generator "$\lambda_{expirations}$" must be

$$\lambda_{expirations} = \frac{\lambda_{events}}{r} \tag{3.2}$$

The random events are generated following an exponential probability density function in time which has the following cumulative distribution function

$$F(t) = 1 - e^{-\lambda t} \tag{3.3}$$

The time between events t is obtained by means of an uniform variable "x" where $x \in [0,1]$

$$x = 1 - e^{\lambda t} \tag{3.4}$$

$$t = -\frac{1}{\lambda} ln(1-x) \tag{3.5}$$

Once an event is generated, the test sub-module sends the information related with it to the CM (identically that in standard mode). Taking into account that when running in test mode events are not real, the information of a particular event is initialized as follows:

- The serial number "s" is generated by choosing a random element among the certificate population: $s \in [0,N]$.
- The hash of the CA's DN and the hash of the CA's public key is obtained from the issuer's certificate.
- The revocation date is considered the time at which the revocation event was generated.
- The expiration date is irrelevant because the certificates will expire randomly without taking into account this date.

3.3.4 The Status Checking Handlers (SCHs)

The SCHs are the modules that contain the classes necessary to send the status data in the proper format to the EEs. Figure 3.9 depicts the organization and the transactions that take place among the different elements of a generic SCH.

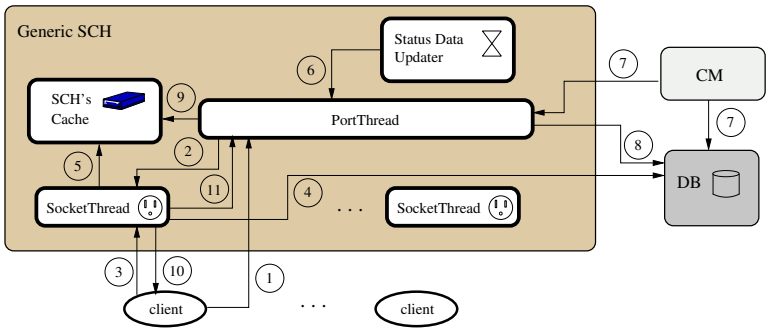


Figure 3.9: Generic Status Checking Handler (SCH) organization.

1. Each SCH has a "PortThread" which is an independent execution thread that listens for requests addressed to a certain TCP port.

2. The SCHs are able to serve several requests at the same time, to do so the PortThread creates an independent execution thread for each request called "SocketThread".

3. The SocketThread receives the data of the request and builds a proper response.

4. The SocketThread can obtain the status data required to build the PDU of the response from the database.

5. The SocketThread can also obtain the status data required to build the PDU of the response from a local cache.

6. The StatusDataUpdater is an execution thread that periodically asks the PortThread to update the cache.

7. The CM informs the database and all the PortThreads about each new event (expiration/revocation) that occurs in the system. This fresh information can be used by the PortThreads to update their caches.

8. For the PortThread another way to update its cache is to access the previously recorded data that is stored in the database.

9. The PortThread updates the cache when the StatusDataUpdater gives the order to do so.

10. Once the SocketThread has all the necessary data (from either the database or the local cache), it sends the response back to the client.

11. Finally the SocketThread informs the PortThread about the bytes and processing time required to serve the request.

The configuration parameters of a generic SCH are presented in Figure 3.10.

type of server. This is a string that identities the protocol of the status server.
class of server. This identifies the class that implements the status server. With this parameter a developer can add new status checking protocols to Cervantes without having to recompile the code.
name. This is a string that gives a name to the SCH.
operational protocol. This is the underlying protocol that will transport the status data. This can be HTTP, raw or <auto-detect>. With the last configuration the server will detect and use the operational protocol used by the client.
port. The number of TCP port used to listen to requests.
server log. This is the path to the file that stores the traces of the execution of the status server for debugging.
statistics log. This is the path to the file that stores the statistics taken by the status server.
certificate path. This is the path to the certificate used to sign the PDUs by SCH.
private key path. This is the path to the file that stores the private key associated with the certificate of the SCH.

Figure 3.10: Configuration parameters of a generic SCH server.

3.3.5 The Central Management Module (CM)

The configuration parameters of the CM are shown in Figure 3.11. Below we explain how the CM works.

The CM initiates the server (see Figure 3.12):

1. The CM reads the configuration file.

mode. This parameter indicates whether the server is running in test or standard mode.
dump statistics period. During this period of time the SCHs accumulate statistics about the bandwidth (kbps) and processing capacity (milliseconds/second). At the end of the period the SCHs will dump the values of their statistic counters to a file.
manager log. This is the path to the file that stores the traces of the execution of the server for debugging.

Figure 3.11: Configuration parameters of the CM module.

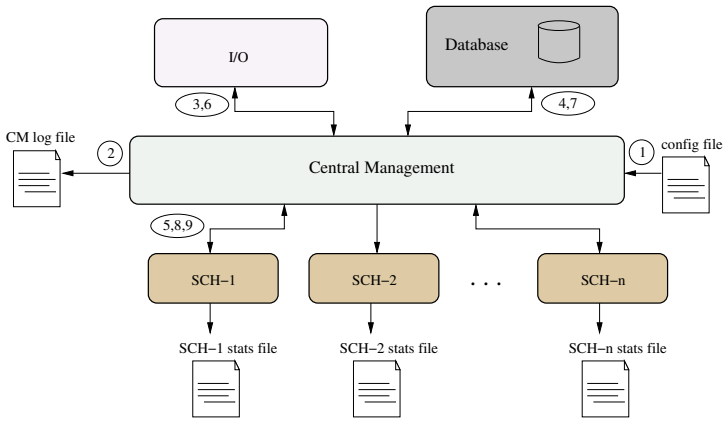


Figure 3.12: Central Management (CM) organization.

2. The CM creates a log file in which it will store all the traces of the execution of the server such as the time at which the server is started, the name of the host, the modules it has found, the running mode etc.

3. Depending on the running mode the CM starts the execution threads of the proper sub-module in the I/O.

4. The CM initiates the JDBC connection.

5. The CM creates the main execution threads (PortThreads) of each SCH found in the configuration file.

While the server is running (see Figure 3.12):

6. The CM receives the revocations/expirations from the I/O.
7. The CM inserts/deletes the records related with the new events in the database.
8. The CM informs the SCHs about the events.
9. The CM periodically orders the SCHs to dump and reset their statistic counters.

When the server must be stopped, the CM kills all the execution threads.

3.4 The clients

The clients are entities able to send requests and process responses using a particular protocol. The protocols are managed by Protocol Handlers (PHs). Each particular PH has an API that allows to set the parameters required to form the request and to retrieve the result of the response. Using the API provided by each PH any external application can perform the certificate-related operations on behalf of the user. Figure 3.13 shows an scheme of the different kind of clients that we have developed using the PHs: a GUI client and a test client.

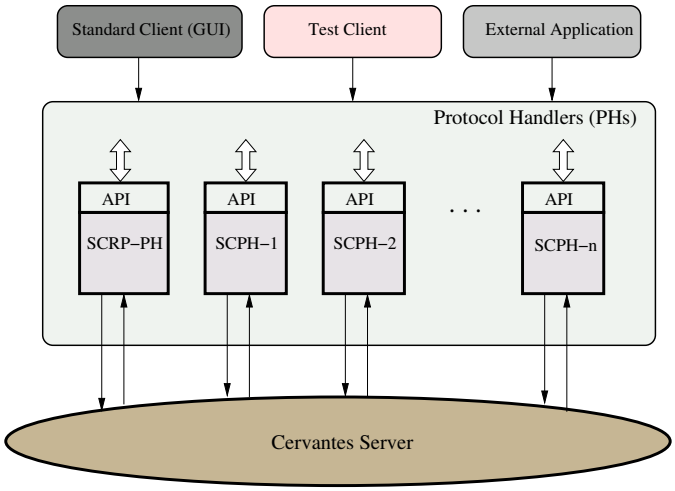


Figure 3.13: Clients organization.

There are two different kind of PHs:

- The PHs for the revocation request protocols (so far we have only one PH of this kind: the SCRP).
- The PHs for the status checking protocols.

The PHs work as follows:

- The PH sends requests with the parameters specified through its API.
- A timeout is setup for each request.
- The PH waits for a response from the server.
- The communication is closed and a "timeout fail" is reported through the API if the response does not arrive prior to the end of the timeout.
- If a response is received on time, it is verified and the result is reported through the API.

3.4.1 The GUI client

Using this GUI client the user can graphically set the parameters to send and receive status checking requests using CRL, OCSP, $\mathcal{H}$-OCSP (see Chapter 4), AD-MHT (see Chapter 5) or E-MHT (see Chapter 6). Revocation requests can also be sent using SCRP. Figure 3.14 shows a screen-shot of the main window of the GUI client and Figure 3.15 shows some possible responses.

3.4.2 The Test client

The test client is a multi-threaded application in which each execution thread represents a client that generates random status checking requests. The aim of the test client is to analyze the behaviour of each status checking protocol or a combination of them under different conditions.

In order to perform a test we define "groups of clients". A group of clients is a determinate number of execution threads (clients) with the same configuration. Figure 3.16 summarizes the configuration parameters of a group of clients.

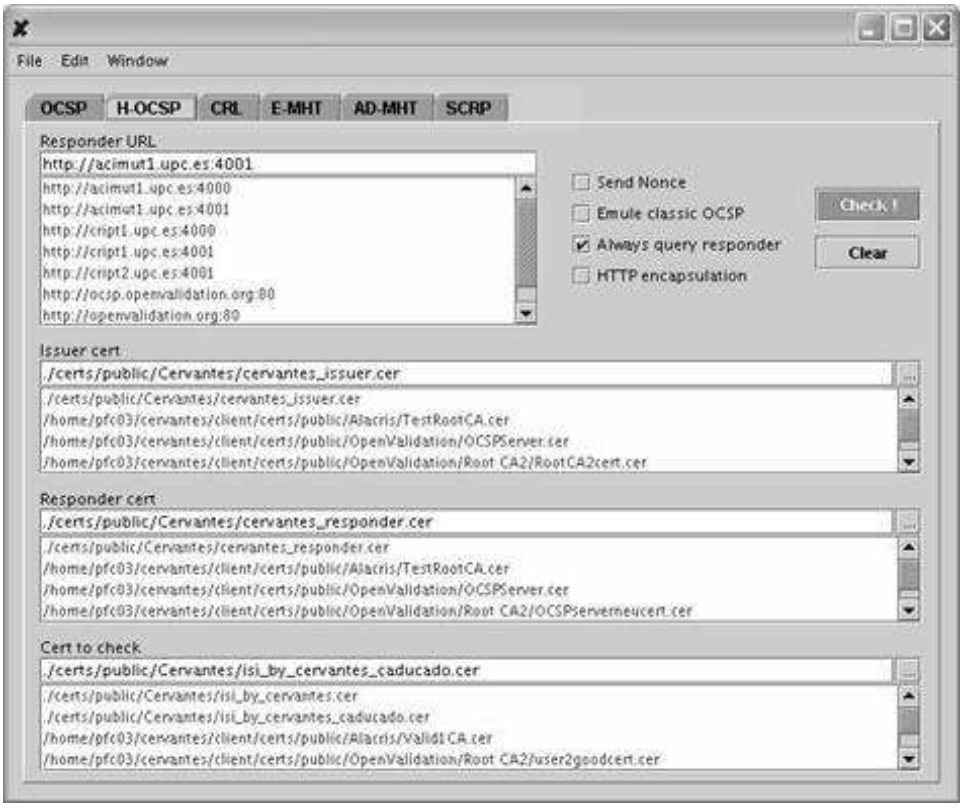
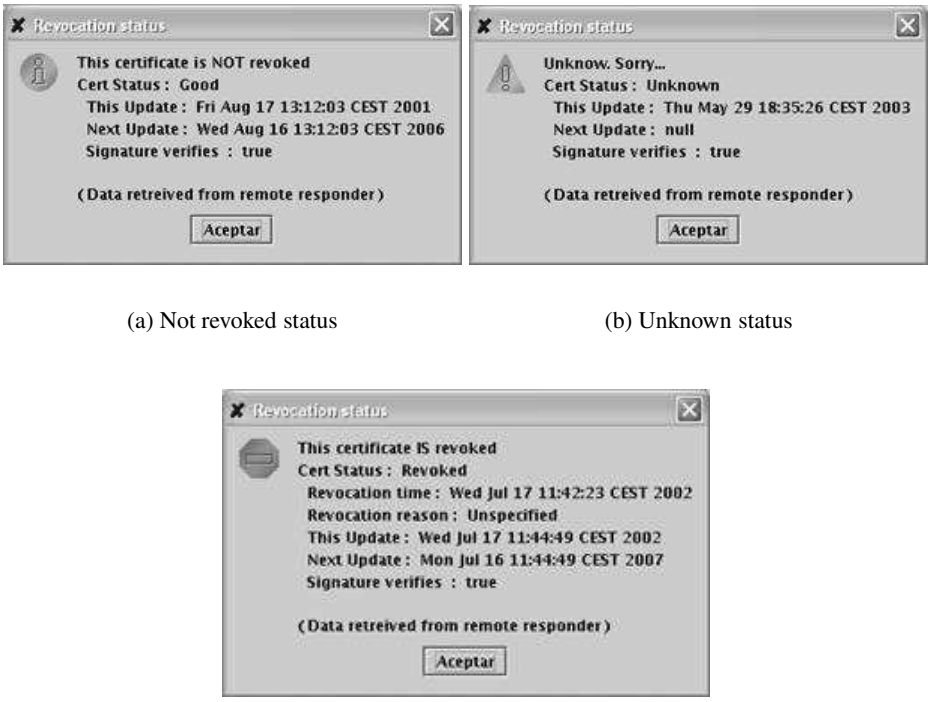


Figure 3.14: Screen-shot of the main window of the GUI client.

To define a group of clients we need to specify the URL of the server (name/@IP and TCP port) where the requests must be addressed. We also need to define the statistics of the requests. The statistic used is an exponential probability density function in time (see equations 3.3 and 3.5). In the configuration file you can set the number of clients "n" and average rate of requests per hour and client "$\lambda_{requests}$". Moreover, the configuration also allows to create a group of FAC (Frequently Asked Certificates). The FAC allows us to simulate groups of certificates that are requested many times and very often by a user. For instance, the certificate of the user, the certificate of the e-mail server of the user or the certificate of the user's bank are clear candidates to be requested very frequently. For each client a random FAC is generated at the beginning of the test. A FAC is defined first by a percentage that indicates the probability that a request goes about a certificate that



(a) Not revoked status

(b) Unknown status

(c) Revoked status

Figure 3.15: Screen-shots of responses provided by the GUI client.

belongs to the FAC and second by the number of certificates of the FAC. Finally, it can be also defined the period of time "D" in which distribute the clients: the test client waits during "t_{delay}" before starting the next the execution thread of the group of clients where

$$t_{delay} = \frac{D}{n} + 20 milisecons \tag{3.6}$$

Notice that if $D = 0$ there will be a minimum delay of 20 milliseconds between a client and the next one in order to assure random independent executions between clients.

type of client. This parameter identities the status checking protocol.
number of clients. This is the number of clients of the group of clients.
distribute clients in time. This is the period of time in which the clients must be distributed.
FAC percent. Percentage of the requests that are performed about certificates of this FAC.
FAC number of certs. Number of certificates of this FAC.
operational protocol. This is the underlying protocol that will transport the status data. This can be HTTP or raw.
requests per hour. The average number of requests per hour performed by each client.
timeout. The number of milliseconds that the client will wait for a response from the server. When the time out ends, the communication is closed. If this parameter is set to 0 the client will wait indefinitely for the response.
server name. The name or IP address of the server.
server port. The number of TCP port used to listen to requests.
client log. This is the file that stores the traces of the execution of the group of clients for debugging.
issuer certificate path. This is the path to the file that stores the certificate of the issuer of the status data.

Figure 3.16: Configuration of a group of clients.

3.5 Status Checking with CRLs

3.5.1 Server-side

Figure 3.17 shows the configuration parameters of a CRL-SCH. Our implementation of CRL allows the overissuation of CRLs so besides the validity period "VP" of the CRLs it is necessary to define the overissuation factor "O". This means that "O" CRLs will be issued during a VP.

validity period. This parameter defines the period of time in which a CRL is valid (this is equivalent to say the period of time that a CRL can be cached by a client).
overissued factor. This parameter defines the number of CRLs that will be issued during a validity period.

Figure 3.17: Additional configuration parameters of a CRL-SCH.

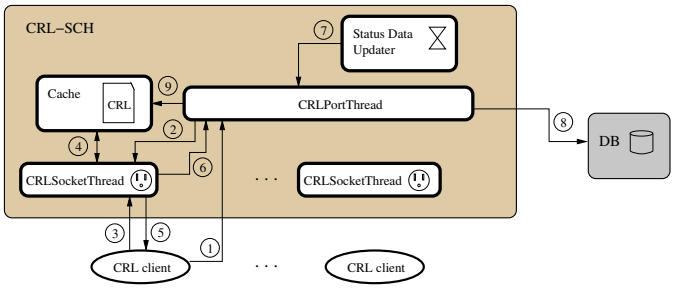


Figure 3.18: CRL-SCH

Figure 3.18 shows the behaviour of the SCH that manages the CRL (notice that this is a particular case of the behaviour of the generic SCH of Section 3.3.4).

1. The CRLPortThread listens for requests addressed to a certain TCP port.
2. For each request the CRLPortThread creates a CRLSocketThread.
3. The CRLSocketThread receives the request.
4. The CRLSocketThread retrieves the CRL from the cache.
5. The CRLSocketThread sends the CRL to the client.
6. The CRLSocketThread informs the CRLPortThread about the bytes and processing time required to serve the request.
7. Every VP/O the StatusDataUpdater asks the PortThread to update the CRL.
8. The CRLPortThread retrieves all the records from the database.
9. With all the status data retrieved, the CRLPortThread generates the list of revoked certificates, sets the validity period, signs the CRL, DER encodes the CRL and sends the result to the Cache.

3.5.2 Client-side

The CRL-PH adds new parameters in the API to let you indicate that the CRL must be stored in cache while it is valid (see Figure 3.19).

cache. This parameter indicates whether the client stores in cache the CRL or not.
local cache path. This is the path to the local directory of the client that will store the CRL.

Figure 3.19: Additional configuration parameter of a CRL-PH.

3.6 Status Checking with OCSP

3.6.1 Server-side

Figure 3.20 shows the behaviour of the SCH that manages OCSP.

1. The OCSPPortThread listens for requests addressed to a certain TCP port.
2. For each request the OCSPPortThread creates an OCSPSocketThread.
3. The OCSPSocketThread receives the request.
4. The OCSPSocketThread retrieves the status data for the requested certificate from the DB.
5. With the status data the OCSPSocketThread builds the response including the signature, DER encodes the response and sends it to the client.
6. The OCSPSocketThread informs the OCSPPortThread about the bytes and processing time required to serve the request.

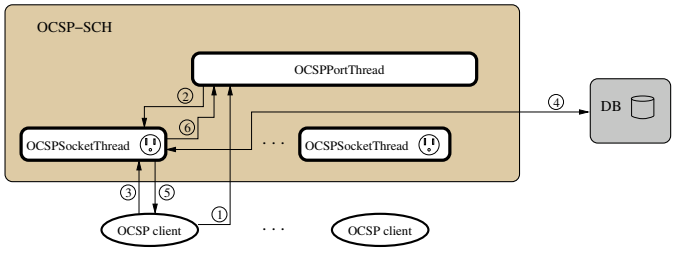


Figure 3.20: OCSP-SCH

3.6.2 Client-side

The OCSP-PH has a new parameter in the API to indicate whether the client must send a nonce or not in the request (see Figure 3.21). If the request has a nonce, then the responder must include this nonce in the signed response. Notice that this cryptographically binds the response with the request.

send nonce. This parameter indicates whether the client wants to send a nonce to cryptographically bind the request with the corresponding response or not.

Figure 3.21: Configuration of an OCSP-PH.

3.7 Evaluation with Cervantes: CRL vs OCSP

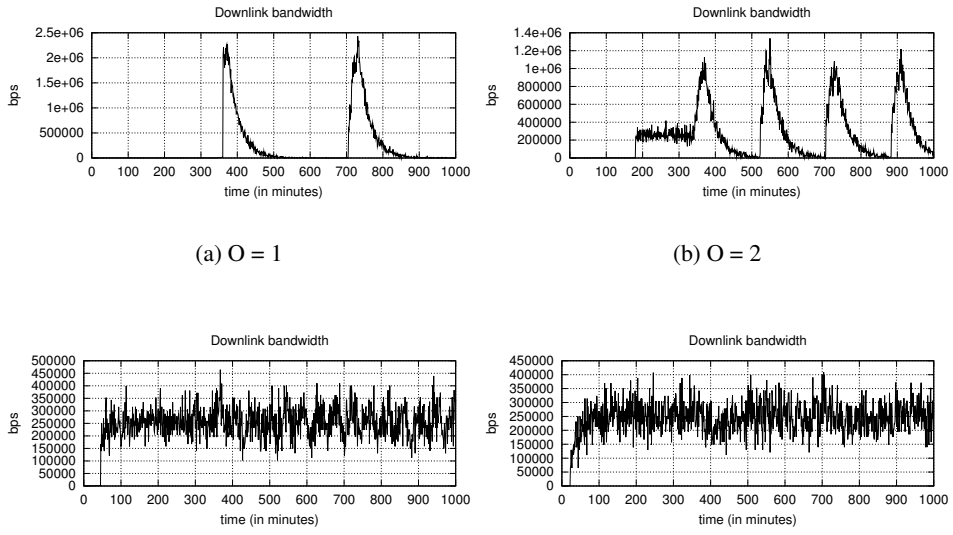
In this section we use the Cervantes platform to evaluate the two main status checking standards: CRL and OCSP. The downlink bandwidth utilization in the status checking and the processing capacity per status request are used below for evaluation. We use these two parameters for evaluation because they can considered in general the most critical bottle-necks of a revocation system (see Chapter 2).

3.7.1 CRL Caching and Overissuation

It is well-known that the downlink bandwidth is the most critical parameter in order to scale a CRL-based system [AJK+95, MF02]. This parameter is the main bottleneck of such a system because a CRL is in general a big piece of data. In order to improve the performance the clients usually store the CRL in a cache while it is valid. However, the cache must be combined with overissuation in order to be effective. Figure 3.22 shows the temporal evolution of the downlink bandwidth when using cache and different overissuation factors. The experimental results have been obtained under the following conditions:

- The Cervantes server runs in a Pentium III (800 MHz).
- The clients generate 2 status checking requests per hour following an exponential probability density function.

- There are 10,000 clients.
- Each client has a certificate.
- There is an average of 10% revocation.
- The validity period of a CRL is 6 hours.
- The CRLs are cached by clients during their validity period.
- Clients will request their local cache instead of the repository if they have a cached CRL.
- The launching of the test clients is distributed along the first validity period.



(a) O = 1

(b) O = 2

(c) O = 8

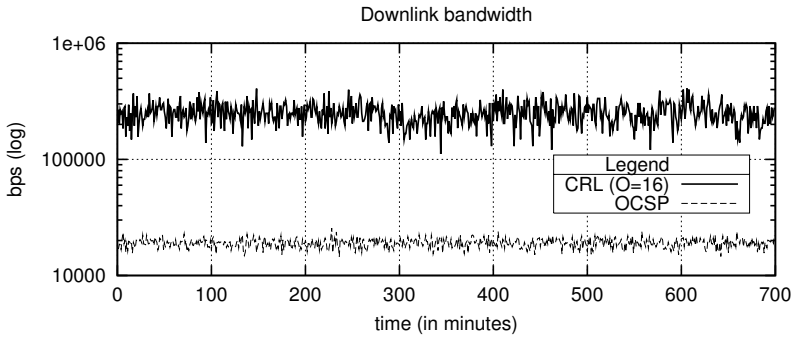
(d) O = 16

Figure 3.22: Overissuing CRLs.

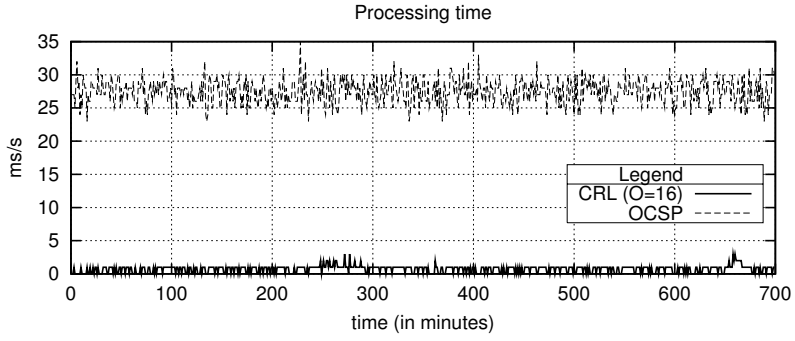
Notice that the bandwidth utilization has peaks when we do not use overissuation ($O = 1$). Without overissuation all the clients have the same CRL copy in their cache so the CRL expires at the same time for everybody. That is why the

bandwidth peaks are localized around the expiration dates (every 6 hours in this scenario). Moreover, around the expiration dates the bandwidth utilization reaches the same level as without using cache so actually we do not make a real profit of the cache. On the other hand, notice that these peaks become smoother when overissuation is increased (see $O = 2$, $O = 8$ and $O = 16$).

3.7.2 CRL vs OCSP



(a) Bandwidth



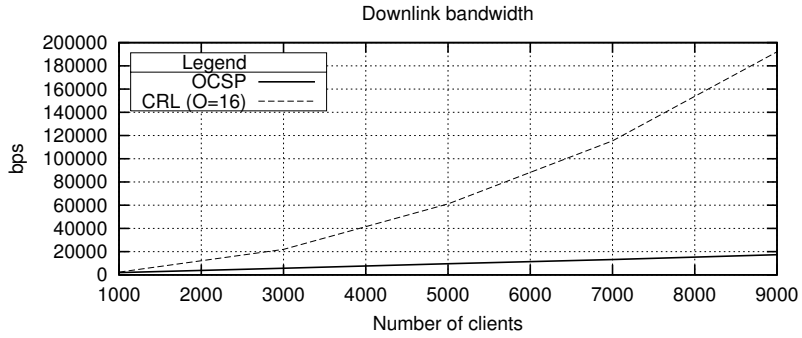
(b) Processing time

Figure 3.23: Temporal evolution of OCSP vs CRL.

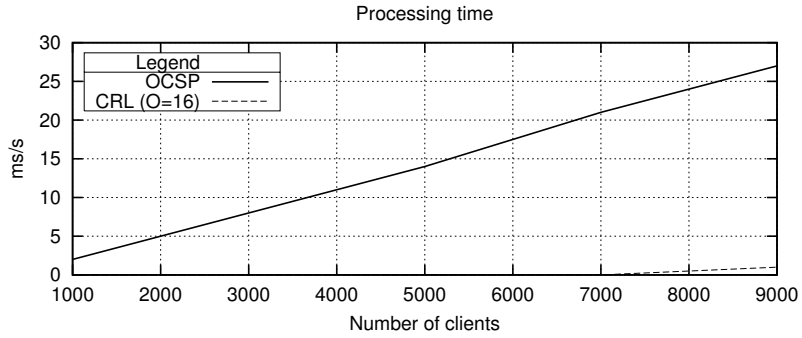
Using the same configuration as in section 3.7.1 in Figure 3.23 we show the

temporal behaviour of the Overissued-CRL (with an overissuation factor: $O = 16$) and the OCSP in terms of downlink bandwidth and processing time.

The results show evidence of the bottlenecks of each system: while in the CRL system the figure of the downlink bandwidth is over an order of magnitude bigger than in the OCSP, it happens the contrary in the figure of the processing time.



(a) Bandwidth



(b) Processing time

Figure 3.24: Scalability regarding the number of clients.

Figure 3.24 shows how scalable is CRL and OCSP when the number of clients is increased. It can be observed that Overissued-CRL is not bandwidth-scalable when growing the number of users "n" because on one hand the CRLs become bigger and on the other hand the CRL downloads are increased. The result is that

bandwidth grows with n^2. The processing capacity requirements of CRL can be considered negligible.

In OCSP the bandwidth and the processing capacity both grow linearly with the number of users because the processing capacity and the communication overhead of OCSP does not depend on the number of revoked certificates. The bandwidth figure has a reasonable value, but the processing time might be a bottleneck in the case of relatively large populations that have a hight request rate or when the responser is attacked by a flood of queries[4].

3.7.3 Validity Period in CRL

Using the same configuration as in section 3.7.2 in Figure 3.25 we show the downlink bandwidth behaviour with regards the users request rate for different validity periods: $VP = 2h$ and $VP = 4h$.

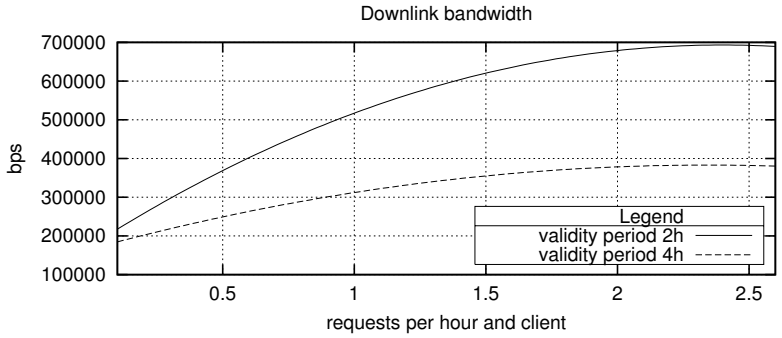


Figure 3.25: CRLs with different Validity Periods.

It can be observed that in general increasing the validity period implies a decreasing the downlink bandwidth figure, that is to say there is a trade-off between risk (freshness) and bandwidth. Other thing to be noticed is that the bandwidth utilization tends to a threshold when increasing the status requests. The existence of this threshold was expected since when the request rate per user grows, the user starts benefiting from her cached CRL and the request rate towards the repository reaches a threshold and therefore the downlink bandwidth too. However, for each

[4]In Chapter 4 we present more performance evaluation on this and several modifications that make OCSP more robust to this kind of attacks.

validity period the bandwidth threshold is reached around a different request rate. This is because for large validity periods the cache is effective even for little request rates and therefore bandwidth reaches the threshold earlier, whereas for low validity periods it is necessary to have higher request rates to take advantage of the cached CRL.

3.7.4 CRL-Distribution Points

In CRL-DP the status data is distributed among multiple CRLs and each CRL contains the status information about a certain group of certificates. Using the same configuration as in section 3.7.2 Figure 3.26 shows the CRL-DP behaviour for different number of distribution points: $DP = 1, 2$ and 4.

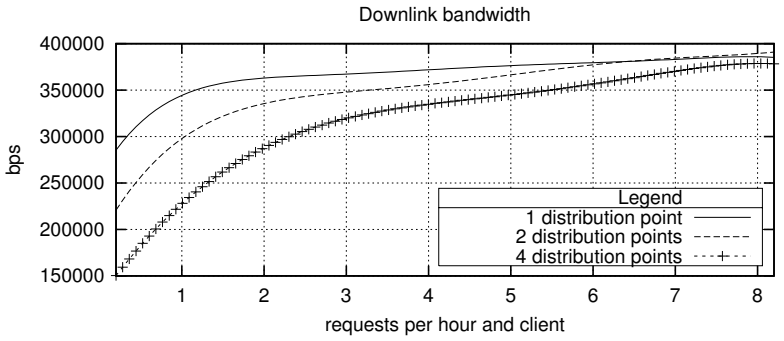


Figure 3.26: CRLs with different Distribution Points.

It can be observed that having many DPs is very effective for low request rates while high requests cause to deteriorate the downlink bandwidth utilization. This behaviour was expected because for high request rates the users finally gather the CRLs of all the DPs and therefore the overall bandwidth is not reduced (actually it gets slightly worse due to the additional overhead of each CRL).

3.8 Conclusions

In this chapter we presented Cervantes which is a platform to evaluate certificate revocation systems. The platform is very flexible because of its modular design.

Its design allows Cervantes to fit any kind of status checking protocol without significative changes in the structure or the source code of the platform. In particular we have shown how to implement in Cervantes the two main standards: CRL and OCSP. In the following chapters we will show also how to develop other (more complex) status checking protocols in Cervantes.

Finally we have used Cervantes to obtain performance results about OCSP and O-CRL. The results obtained were expected and they serve us to check that the platform is behaving correctly.

As future work, we are planning to split the program of the server in different programs for the RDI, the responders and the repositories.

Chapter 4

$\mathcal{H}$-OCSP

4.1 Introduction

In [SEF+] we presented an architecture for m-commerce. In this chapter we review this architecture in which a broker is used as OCSP responder for the certificate validation. We also present a modification over OCSP called $\mathcal{H}$-OCSP [MFE+03]. $\mathcal{H}$-OCSP is a way to reduce the computational load and the bandwidth requirements of OCSP which is specially desirable in the wireless environment.

4.2 An architecture for m-commerce

The access to the Internet by means of mobile devices potentially increases the number of users of e-commerce. One of the novelties of m-commerce is the possibility of attracting clients in the neighborhoods of commercial and/or service centers by providing them with appropriate information. A way to ease and make more efficient the access to information from mobile terminals is to use a broker between the terminal and the wired network. Figure 4.1 shows the broker-based architecture presented in [SEF+, SP02]. The broker re-uses the information and sends useful data in a predictive mode to wireless users, reducing data traffic in the wireless link.

The main functions of the broker are detailed below:

- *Intermediate storage*: the broker stores a copy of the information that it re-

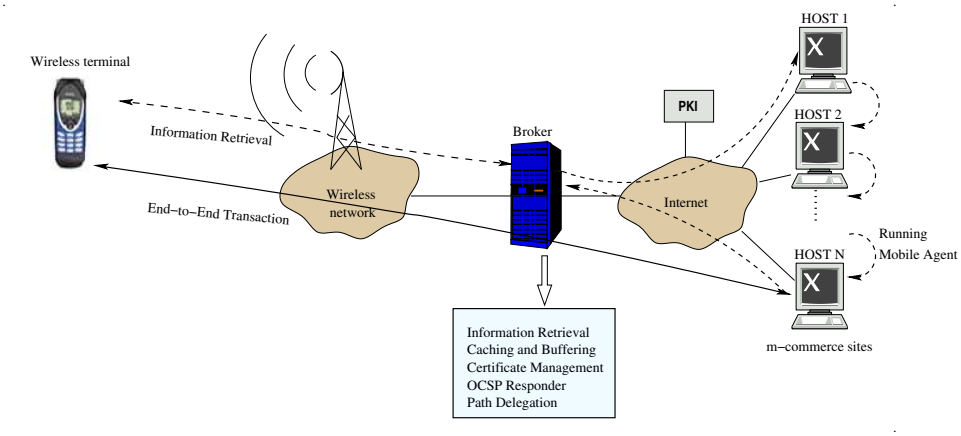


Figure 4.1: Broker Architecture for a Wireless Scenario

ceives. This will allow its re-use by other users. Therefore, the information must be organized inside proxy and cache systems.

- *Customer Relationship Management infrastructure*: the knowledge of the user profile allows to manage the user's needs for information, and to deliver information in a predictive fashion and dynamic tracking of clusters of data. A user profile could be managed and updated dynamically, processing information from different sources. In wireless environment, CRM (Customer Relationship Management) can profit from the facilities of the push mechanism of mobile communication protocols.

- *Information search and retrieval*: tasks such as searching, advising, contacting, comparing, filtering and facilitating access to databases perfectly fit for mobile agent technology. In this sense, agents can perform some tasks on behalf of a user while the mobile device remains off-line, which is very desirable on a noisy weak link with unpredictable disconnection.

- *Certificate management*: a PKI is required to provide the m-commerce transactions with the security services such as integrity, privacy, non-repudiation, authentication and access control. The broker is appropriate for storing and managing digital certificates, and combined with OCSP timely information about the status of certificates can be obtained.

Digital certificates are usually used to identify parties involved in e-commerce and m-commerce transactions, as well as to provide end-to-end security in these transactions.

Wireless users ask for information or services in the Internet and their requests arrive to the broker that will obtain this information on behalf of the user. After receiving the information, depending on the required service, the user may perform a transaction with a selected m-commerce site. When a user wishes to establish a transaction with a particular m-commerce site, an end-to-end authenticated and private channel is required in order to transfer sensitive data such as a credit card number. Technologies based on ICs, like TLS (Transport Layer Security) [DA99] are being widely used for establishing this kind of secure channels. However, prior to perform a transaction using a certain IC, the user must be sure that the certificate is valid. It must be stressed that the certificate management is a heavy process and that the clients in our environment are resource-limited. For this reason, clients delegate to the broker the processing of the certificates. Notice that the broker is a TTP and it is in general not resource-limited, therefore it is appropriate for storing and managing certificates.

The rest of the chapter is organized as follows: in Section 4.3 we show the most common drawbacks of using OCSP for status checking. In Section 4.4 we propose $\mathcal{H}$-OCSP as a way to reduce the computational load in the broker. In Section 4.5 we define the ASN.1 add-on for $\mathcal{H}$-OCSP that makes it inter-operable with the standard OCSP. In Section 4.6 we present the security discussion for the $\mathcal{H}$-OCSP. In Section 4.7 we show the implementation of $\mathcal{H}$-OCSP in Cervantes. In Section 4.8 we evaluate the behavior of $\mathcal{H}$-OCSP compared to standard OCSP. Finally, we conclude in Section 4.9.

4.3 Preliminaries

Remember that OCSP enables certificate-using applications to determine the revocation state of an identified certificate. The status of certificates is available online through a responder that signs online each response produced. An OCSP client issues a status request to an OCSP responder and suspends acceptance of the certificate in question until the responder provides a response. For e-commerce, online

technologies are interesting not only because they can provide the status data in real-time but also because of billing issues (requests can be used as the basis for billing). OCSP Responses can contain three times in them:

- `thisUpdate` is the time at which the status being indicated is known to be correct.
- `nextUpdate` is the time at or before which newer information will be available about the status of the certificate.
- `producedAt` is the time at which the OCSP responder signed this response.

The client may also send a nonce in its request. In this case, the responder has to include the nonce in the signature computation, and thereby the request and the response are cryptographically binded. However, a denial of service vulnerability is evident with respect to a flood of queries. The production of a signature significantly affects response generation cycle time, thereby exacerbating the situation. Unsigned error responses open up OCSP to another denial of service attack, where the attacker sends false error responses.

In order to alleviate these denial of service vulnerabilities, the OCSP responders may pre-produce signed responses specifying the status of certificates at a certain time [MAM+99]. The time at which the status was known to be correct shall be reflected in the `thisUpdate` field of the response. The time at or before which newer information will be available is reflected in the `nextUpdate` field, while the time at which the response was produced will appear in the `producedAt` field of the response. However, the use of precomputed responses allows replay attacks in which an old (good) response is replayed prior to its expiration date but after the certificate has been revoked. Deployments of OCSP should carefully evaluate the benefit of precomputed responses against the probability of replay attacks. In this sense, notice that it exists the following trade-off:

> *The online signature consumes much processing time. To reduce the possibility of falling into denial of service, the responder may precompute the responses and store them in a cache. But pre-produced*

responses are susceptible of generating reply attacks. To avoid the replay attacks, the responder needs to generate pre-produced responses within a short period of time which consumes many processing resources and this fact may lead the responder again to denial of service.

4.4 $\mathcal{H}$-OCSP basics

The result of the previous discussion is that the responder would benefit from a mechanism to pre-produce responses with low processing resources utilization. Below we outline a mechanism that reaches this target. Furthermore, under certain circumstances, the exposed mechanism has also many important benefits for the wireless clients that use OCSP.

The mechanism that we propose is called $\mathcal{H}$-OCSP and it exploits the fact that a OWHF faster to compute than a digital signature. When a pre-produced response needs to be updated because its `nextUpdate` has become obsolete, a OWHF is performed to update this response instead of a new signature. Using an OWHF will permit the repository to update the responses more frequently without falling into denial of service.

$\mathcal{H}$-OCSP is based on the Even et al. algorithm [EGM96] and it works as follows: when a response is going to be pre-produced, the responder adds a hash-chain to it. The hash chain permits the repository to update the pre-produced response in successive periods with a scarce resources utilization. The hash chain results from applying $d+1$ times a OWHF h over a secret nonce (4.1)

$$R \xrightarrow{h} R_d \xrightarrow{h} R_{d-1} \xrightarrow{h} \cdots \xrightarrow{h} R_i \xrightarrow{h} \cdots R_2 \xrightarrow{h} R_1 \xrightarrow{h} R_0 \tag{4.1}$$

Let us define the parameters involved in the process:

`primaryUpdateValue` **(R)** is the secret nonce. R is only known by the responder (broker) and it is generated for each new pre-produced response.

`maximumUpdateIndex` **(d)** is the maximum number of periods that a pre-produced response can be updated.

`baseUpdateValue` (R_0) is the last value of the hash chain and it is included in the signature computation of the pre-produced response. R_0 is computed by applying $(d+1)$ times h over R

$$R_0 = h^{d+1}(R) \tag{4.2}$$

`currentUpdateValue` (R_i) is computed by applying $(d+1-i)$ times h over R

$$R_i = h^{d+1-i}(R) \tag{4.3}$$

Where i is the number of periods "Δ" elapsed from the documented one (the documented validity period is the period included in the response). Δ is defined as

$$\Delta = \texttt{nextUpdate} - \texttt{thisUpdate} \tag{4.4}$$

A relying party can verify the validity of a pre-produced response that it is living beyond its documented life-time, say, at time t, where t is included within the period $[\texttt{nextUpdate} + (i-1)\Delta, \texttt{nextUpdate} + i\Delta]$, by checking the equality of equation (4.5)

$$R_0 = h^i(R_i) \quad with\ i \leq d \tag{4.5}$$

It must be stressed that to forge a `currentUpdateValue` with the information provided by a previous update value an attacker needs to find a pre-image of a OWHF which is by definition computationally infeasible.

It is obvious that if a pre-produced response becomes stale because the status of the certificate it refers changes, a new signature must be performed, in other words, $\mathcal{H}$-OCSP is only applicable to responses that need to update their documented life-time. However, notice that the majority of the response updates are due to the fact that the documented life-times become obsolete, hence the $\mathcal{H}$-OCSP will have indeed a great impact over the responder performance.

Notice also that $\mathcal{H}$-OCSP produces the desired behavior of pre-produced responses:

For one thing, the responder can use pre-produced responses with

> *a small life-time which reduces the risk of replay attacks. For another, the repository can update its pre-produced responses at low cost which reduces the risk of denial of service.*

Finally, it is worth mentioning that when the responder recovers from a crash or is restarted after being down for some reason, the most secure and easiest way to implement the startup environment is to erase all the pre-produced responses from cache. This is the way we perform the startup environment in our $\mathcal{H}$-OCSP responder. However, other implementations may keep the pre-produced responses but do so at their own peril.

Below, we show how to save bandwidth between the client and the responder using $\mathcal{H}$-OCSP. This feature is very interesting in the wireless environment where one of the biggest constrains is the scarce bandwidth that it is available.

Pre-produced responses can be also cached by clients. Let us assume that a previous $\mathcal{H}$-OCSP response for a certain certificate is stored in the client's cache. Then, if the client needs to check the status of the same certificate later, she can ask the responder for a `currentUpdateValue` instead of downloading a standard OCSP response which has a much bigger size. Moreover, if a client performs the majority of his requests for a small set of certificates while other certificates are more rarely requested, it becomes likely to have cached responses for these FAC. In this case, the status checking can be performed only sending the `currentUpdateValue` and the best performance of $\mathcal{H}$-OCSP related to bandwidth utilization can be clearly appreciated (see Section 4.8).

On the other hand, a client may wish to keep an OCSP response as a consumer protection issue. If a client performs an important or a delicate transaction, it may later arise a conflict about the content of the exchanged messages. Usually, digital signatures performed during the transaction are used to solve these kind of non-repudiation trials, therefore it is necessary to have a proof of the status of the involved certificates in the precise moment that the transaction was performed. Notice that the hash chain permits to store old responses for a certain certificate with a reduced storage capacity. This storage can be performed by the broker or even by the client (for the certificates she considers important).

4.5 ASN.1 add-on for $\mathcal{H}$-OCSP

In this section we address important implementation issues in order to make $\mathcal{H}$-OCSP inter-operable with the standard OCSP. In order to deploy $\mathcal{H}$-OCSP, we need to slightly modify the OCSP protocol. The additional parameters that we introduce are included in extensions. This provides compatibility because support for any specific extension is optional and unrecognized extensions are silently ignored by either the clients or the responders. The protocol is designed to permit inter-operability among standard OCSP clients and responders and $\mathcal{H}$-OCSP clients and responders with any combination among them. The ASN.1 add-on for $\mathcal{H}$-OCSP is presented in Figure 4.2.

An $\mathcal{H}$-OCSP client can include an extension in the `singleRequestEx- tensions` with OID `id-pkix-ocsp-base-update-value` to let the responder know that she understands $\mathcal{H}$-OCSP. If the $\mathcal{H}$-OCSP client has an $\mathcal{H}$-OCSP response for the target certificate in its cache, the extension includes the `baseUpdateValue`. Otherwise, the extension is filled with an array of 0 bytes. Upon receipt of a request, a responder determines if the message is well formed, if the repository is configured to provide the requested service and if the request contains the compulsory information.

If an $\mathcal{H}$-OCSP client requests a standard OCSP responder, the extension is silently ignored and the responder responds with a standard OCSP response. If an $\mathcal{H}$-OCSP responder receives a request, it looks for the correspondent extension. Depending on the request extension, the $\mathcal{H}$-OCSP responder will respond with different types of responses:

type-A is an $\mathcal{H}$-OCSP response that includes the `currentUpdateValue`. It is sent to the client if she understands $\mathcal{H}$-OCSP and if the `baseUpda- teValue` provided in the request extension matches the one currently stored by the responder.

type-B (basic) is a standard OCSP basic response. It is sent either if the client does not understand $\mathcal{H}$-OCSP or if it is the first request for a pre-produced response.

```
HOCSP DEFINITIONS EXPLICIT TAGS::=
BEGIN
IMPORTS

--PKIX Certificate Extensions
AlgorithmIdentifier, BasicOCSPResponse, id-pkix-ocsp FROM OCSP

AuthorityInfoAccessSyntax, GeneralName, CRLReason FROM PKIX1Implicit88
{iso(1) identified-organization(3) dod(6) internet(1) security(5)
mechanisms(5) pkix(7) id-mod(0) id-pkix1-implicit-88(2)}

Name, Extensions,Certificate, -- AlgorithmIdentifier, id-kp,
id-ad-ocsp FROM PKIX1Explicit88 {iso(1) identified-organization(3)
dod(6) internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
id-pkix1-explicit-88(1)};

--Type of H-OCSP responses
TypeAResponse ::= SEQUENCE { currentUpdateValue OCTET STRING }
TypeCResponse ::= SEQUENCE {
  basicResponse BasicOCSPResponse,
  typeAResponse TypeAResponse }
baseUpdateValue ::= OCTET STRING
maximumUpdateIndex ::= INTEGER

-- Object Identifiers (proposal)
id-pkix-hocsp-type-a OBJECT IDENTIFIER ::={ id-pkix-ocsp 8 }
id-pkix-hocsp-type-c OBJECT IDENTIFIER ::={ id-pkix-ocsp 9 }
id-pkix-hocsp-base-update-value OBJECT IDENTIFIER::={ id-pkix-ocsp 10
}
id-pkix-hocsp-maximum-update-index OBJECT IDENTIFIER ::=
{id-pkix-ocsp 11}
```

Figure 4.2: ASN.1 add-on for $\mathcal{H}$-OCSP

type-C contains a basic response plus a `currentUpdateValue`. The basic response includes also the `maximumUpdateIndex` parameter in one of the `singleExtensions` of the response. It is sent if the client understands $\mathcal{H}$-OCSP but it has not a previous $\mathcal{H}$-OCSP response in cache for the target certificate or if the cached response she has is obsolete (i.e. the `baseUpdateValue` does not match the one in the responder).

Finally, we have checked inter-operability of our clients and responders with:

- Euro PKI (*http://ocsp.europki.org\:8026*).
- Open Validation (*http://ocsp.openvalidation.org\:80*).
- Alacris (*http://ocsptest.alacris.com\:3080/ocsp*).

4.6 Security discussion

Next, we present an informal discussion (rather than formal proofs or demonstrations) about the main security aspects of $\mathcal{H}$-OCSP.

Remember that OCSP can be used with:

- *Cryptographically binded requests and responses*. If the client wants a response cryptographically binded to her request, the repository must take into account the nonce that goes in the request when computing the signed response.
- *Pre-produced responses*. These responses are not bound to any particular request, so the information that a response contains can be re-used to respond as many requests as desired.

Cryptographically binded requests and responses are open to denial of service attacks because of the cost of computing the signatures. In this case the effectiveness of $\mathcal{H}$-OCSP avoiding this kind of denial of service attack fails for the following reasons:

- This mode of operation is very costly to the responder because it has to produce a different response per each client for the same information and store all these data in its cache.
- If the majority of the clients are honest and they collaborate with $\mathcal{H}$-OCSP by using their cache entries for previously asked data, the $\mathcal{H}$-OCSP responder will be more robust than the standard one. However, active adversaries can claim that they do not have previously asked for data (e.g. omitting their cache entries) and they can ask each time for a new response. In this case, the denial of service protection of $\mathcal{H}$-OCSP fails if attackers flood the responder

with their requests ($\mathcal{H}$-OCSP will have the same asymptotic behaviour that OCSP).

As a result of the previous discussion, we discourage the use of $\mathcal{H}$-OCSP with cryptographically binded responses (or at least the administrator must be concerned about the risks of using this mode of operation).

On the other hand, when using pre-produced responses, the $\mathcal{H}$-OCSP responder is protected from denial of service attacks for data contained in its cache during (at the most):

$$\Delta_p = \Delta * d \tag{4.6}$$

We denote Δ_p as the "maximum protection period". Notice that any new request over data already contained in a cached response will not involve much processing capacity usage during the protection interval because the Even et al. algorithm can be executed almost in real time. Therefore the denial of service attack intended by the active adversary can be avoided. Even though, the responder may have minor denial of service problems at start-up since at that time it must sign each produced response.

4.7 The Cervantes module of $\mathcal{H}$-OCSP

4.7.1 Server-side

Figure 4.3 shows the behaviour of the SCH that manages $\mathcal{H}$-OCSP.

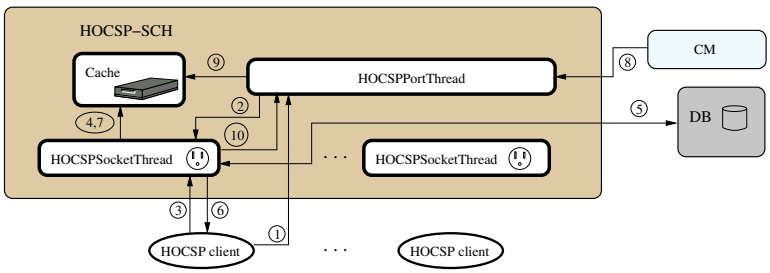


Figure 4.3: $\mathcal{H}$-OCSP-SCH

1. The HOCSPPortThread listens for requests addressed to a certain TCP port.

2. For each request the HOCSPPortThread creates an HOCSPSocketThread.

3. The HOCSPSocketThread receives the request.

4. The Cache stores previous pre-produced responses and their respective `primaryUpdateValues`. So before generating a new response the HOCSPSocketThread tries to find a previous response in cache for the requested certificate.

5. If the HOCSPSocketThread does not find a previous response in cache, it must obtain the status data from the database.

6. The HOCSPSocketThread can send different types of responses to the client depending on the data that it has gathered and what the client has in its cache:

   (a) *Conditions:* The HOCSPSocketThread could acquire a pair (basic response, `primaryUpdateValue`) from the cache for the target certificate and the client has the same basic response in her cache.
   Type of response: type-A. To build the type-A response the HOCSPSocketThread computes the `currentUpdateValue` from the ca- ched `primaryUpdateValue`, DER encodes the resultant response and sends it to the client.

   (b) *Conditions:* The status data was acquired from the database.
   Type of response: type-B. To build the type-B response from scratch the HOCSPSocketThread generates the basic PDU of the response with the status data, generates the `primaryUpdateValue` and includes this parameter as a response extension, produces the signature, DER encodes resultant response and sends it to the client.

   (c) *Conditions:* The HOCSPSocketThread could acquire a pair (basic response, `primaryUpdateValue`) from the cache for the target certificate but the client does not have the same basic response in her cache.
   Type of response: type-C. To build the type-C response the HOCSPSocketThread computes the `currentUpdateValue` from the ca- ched `primaryUpdateValue`, DER encodes the resultant response using the basic response of the cache and sends it to the client.

7. The HOCSPSocketThread sends the new responses it produces to the cache so that other requests can benefit from this work and possible DoS attacks can be alleviated.

8. The CM informs the HOCSPPortThreads about any change in the status data.

9. When the HOCSPPortThread is informed about a change in the status data it must update the cache, that is delete the affected responses from the cache.

10. The HOCSPSocketThread informs the HOCSPPortThread about the bytes and processing time required to serve the request.

4.7.2 Client-side

The $\mathcal{H}$-OCSP-PH adds new parameters in the API to indicate that you want to store in cache the $\mathcal{H}$-OCSP responses (see Figure 4.4).

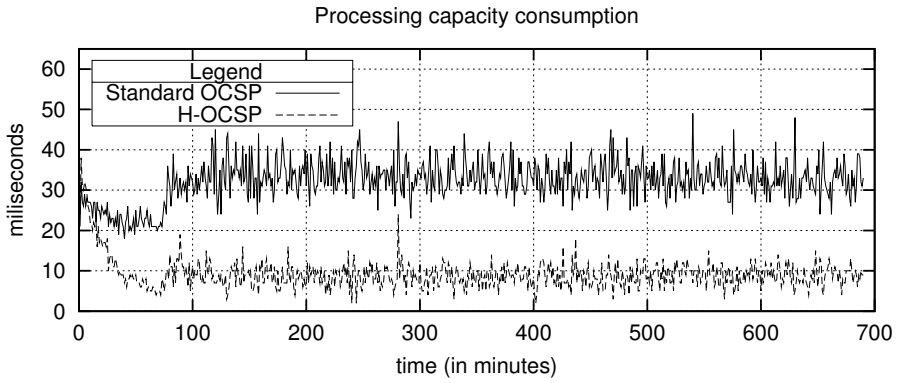
cache. This parameter indicates whether the client stores in cache the $\mathcal{H}$-OCSP responses or not.
local cache path. This is the path to the local directory of the client that will store the responses.

Figure 4.4: Configuration of an $\mathcal{H}$-OCSP-PH.

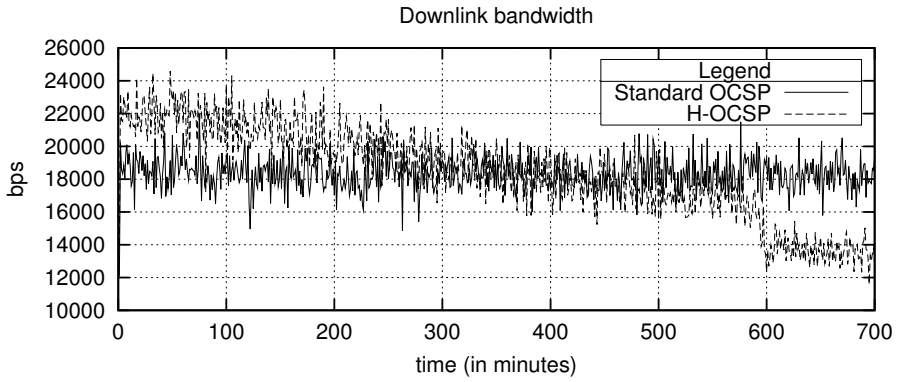
4.8 Evaluation

In this section, we compare the performance of standard OCSP versus $\mathcal{H}$-OCSP in terms of the downlink bandwidth consumption (responder-to-clients) and the processing capacity utilization in the responder. The experimental results have been obtained using Cervantes under the following conditions:

- The responder runs in a Pentium III (800MHz).
- The clients generate 2 status checking requests per hour following an exponential probability density function.
- There are 10,000 clients.



(a) Processing capacity consumption



(b) Downlink bandwidth

Figure 4.5: $\mathcal{H}$-OCSP versus Standard OCSP

- Each client has a certificate.
- There is an average of 10% revocation.
- The test is configured with a database dynamism of one event (revocation/expiration) per hour and random revocations and random expirations are used.
- The target certificates are randomly chosen.

- It is assumed that each client has a FAC of 10 certificates that take the 50% of the status checking requests.
- Pre-produced responses are used (non-cryptographically binded requests and responses).
- The responses are cached by clients during their validity period.

Figure 4.5-a shows the computational load of the responder. It can be observed that the computational load at the $\mathcal{H}$-OCSP responder decreases and becomes steady after a few minutes (approx. after 70 min, when the $\mathcal{H}$-OCSP responder starts to take advantage of the pre-computed responses).

In the steady state, the $\mathcal{H}$-OCSP responder consumes around five times less processing capacity than the standard OCSP responder. This is a substantial improvement taking into account that the CPU measurements include not only cryptographic operations but all the operations performed by the java virtual machine (notice that many of these operations such as the garbage collection are very time-consuming).

Figure 4.5-b shows the measured downlink bandwidth utilization. In the steady state, when the client's cache is fully working, the better performance of $\mathcal{H}$-OCSP can be also observed. With less percentage of requests for these frequently asked certificates the performance decreases but in the worse case is approximately as good as in standard OCSP.

4.9 Conclusions

In this chapter we introduce the problem of certificate validation in m-commerce transactions. We have shown that many of the validation functions can be delegated to a broker in order to alleviate the resources utilization in the client.

$\mathcal{H}$-OCSP has been proposed as a way to reduce the computational load of the responder. $\mathcal{H}$-OCSP is a hash chain-based mechanism to update pre-produced OCSP responses at a low cost. As a result, an $\mathcal{H}$-OCSP responder is better protected against denial of service attacks than a standard one. Wireless clients can also benefit from $\mathcal{H}$-OCSP by storing the responses of the most used certificates

in their cache. However, there is a trade-off between the storing capacity of the wireless terminals and the benefits that $\mathcal{H}$-OCSP can achieve. We are currently developing efficient cache updating policies for terminals with reduced storing capacity.

On the other hand, we have defined the ASN.1 add-on for $\mathcal{H}$-OCSP that makes it inter-operable with the standard OCSP and we have shown the implementation of $\mathcal{H}$-OCSP in Cervantes. Finally, we have evaluated the behavior of $\mathcal{H}$-OCSP compared to standard OCSP and we have shown that $\mathcal{H}$-OCSP requires in general less resources. Although $\mathcal{H}$-OCSP has been designed with the wireless scenario in mind, most of its benefits also apply for the wired internet.

Chapter 5

AD-MHT

5.1 Introduction

MHT-based systems seem to avoid some of the OCSP and CRL scalability drawbacks, but to our knowledge there were no published implementations of such a system. In this Chapter we explain in detail one of the certificate revocation systems based on the MHT that we have developed. The revocation system is named AD-MHT and it uses the data structures proposed by Naor and Nissim in [NN00]. We address important issues that were not addressed in the original proposal, such as the algorithms to efficiently respond to a request, revoke a certificate and delete an expired certificate. We have also developed the status checking protocol for communicating the AD-MHT repository with the End Entities and an algorithm to verify the responses [MFES03b]. Moreover, AD-MHT has been implemented as part of the Cervantes platform and several details of the implementation were presented in [MFES03a]. Finally, an extended version of all togheter has been published in [MFES03e].

The rest of the chapter is organized as follows: in Section 5.2 we show the basics of the AD-MHT. In Section 5.3 we present a status checking protocol for the AD-MHT. In Section 5.4 we discuss the issues related to the response verification. In Section 5.5 we present a security discussion for the AD-MHT system. In Section 5.6 we show the implementation of AD-MHT in Cervantes. In Section 5.7 we compare the AD-MHT with the main standards, OCSP and CRL, in terms of

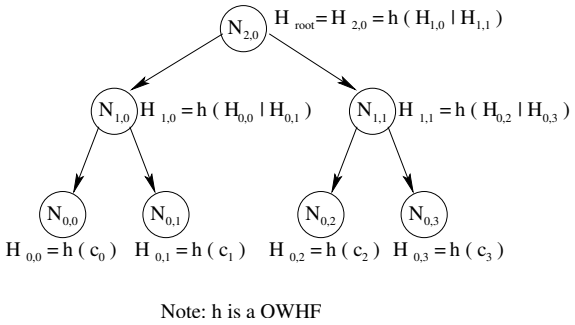


Figure 5.1: Sample MHT.

down-link bandwidth and processing capacity consumption. Finally, we conclude in Section 5.8.

5.2 AD-MHT Basics

The AD-MHT is based on the MHT and the 2-3 Tree.

5.2.1 The Merkle Hash Tree

The MHT [Mer89] relies on the properties of the OWHF. It exploits the fact that an OWHF is at least 10,000 times faster to compute than a digital signature, so the majority of the cryptographic operations performed in the revocation system are hash functions instead of digital signatures. A sample MHT is represented in Figure 5.1.

We denote by $N_{i,j}$ the nodes within the MHT where i and j represent respectively the i-th level and the j-th node. We denote by $H_{i,j}$ the cryptographic variable stored by node $N_{i,j}$. Nodes at level 0 are called "leaves" and they represent the data stored in the tree. In the case of revocation, leaves represent the set Φ of certificates that have been revoked,

$$\Phi = \{c_0, c_1, \ldots, c_j, \ldots, c_n\}. \tag{5.1}$$

Where c_j is the data stored by leaf $N_{0,j}$. Then, $H_{0,j}$ is computed as (5.2)

$$H_{0,j} = h(c_j)\,. \tag{5.2}$$

Where h is a OWHF. To build the MHT, a set of t adjacent nodes at a given level i; $N_{i,j}$, $N_{i,j+1}$, ...,$N_{i,j+t-1}$, are combined into one node in the upper level, which we denote by $N_{i+1,k}$. Then, $H_{i+1,k}$ is obtained by applying h to the concatenation of the t cryptographic variables (5.3)

$$H_{i+1,k} = h(H_{i,j}|H_{i,j+1}|\ldots|H_{i,j+t-1}). \tag{5.3}$$

At the top level there is only one node called the "root". H_{root} is a digest for all the data stored in the MHT.

The sample MHT of Figure 5.1 is a binary tree because adjacent nodes are combined in pairs to form a node in the next level ($t = 2$) and $H_{root} = H_{2,0}$.

Definition 5.1. $\mathcal{D}igest = \{DN_{RDI}, H_{root}, Validity\ Period\}_{SIG_{RDI}}$

Definition 5.2. *The $\mathcal{P}ath_{c_j}$ is defined as the set of cryptographic values necessary to compute H_{root} from the leaf c_j.*

Remark 5.1. *Notice that the $\mathcal{D}igest$ is trusted data because it is signed by the RDI and it is unique within the tree while $\mathcal{P}ath$ is different for each leaf.*

Claim 5.1. *If the MHT provides a response with the proper $\mathcal{P}ath_{c_j}$ and the MHT $\mathcal{D}igest$, an End Entity can verify whether $c_j \in \Phi$.*

Example 5.1. Let us suppose that a certain user wants to find out whether c_1 belongs to the sample MHT of Figure 5.1. Then,

$\mathcal{P}\text{ath}_{c1} = \{H_{0,0}, H_{1,1}\}$

$\mathcal{D}igest = \{DN_{RDI}, H_{2,0}, Validity\ Period\}_{SIG_{RDI}}$

The response verification consists in checking that $H_{2,0}$ computed from the $\mathcal{P}\text{ath}_{c_1}$ matches $H_{2,0}$ included in the $\mathcal{D}$igest,

$$H_{root} = H_{2,0} = h(h(h(c_1)|H_{0,0})|H_{1,1})\,. \tag{5.4}$$

Remark 5.2. *Notice that the MHT can be built by a TTP (RDI) and distributed to a repository because a leaf cannot be added or deleted to* Φ *without modifying* H_{root}[1] *which is included in the Digest and as the Digest is signed, it cannot be forged by a non-TTP.*

5.2.2 The 2-3 Tree

A 2-3 tree is a balanced tree in which each internal node has two or three children ($t \in \{2,3\}$). The main advantage of this type of tree is that management tasks such as searching, adding and removing a leaf can be performed in $o(log(n))$ [AHU88] where n is the number of leaves. Certificates are distinguished in the MHT by their serial numbers and each leaf within the tree represents a certificate. On the other hand, leaves are ordered by serial number. Figure 5.2 shows a sample 2-3 that represents a set of revoked certificates $\Phi = \{2,5,7,8,12,16,19\}$.

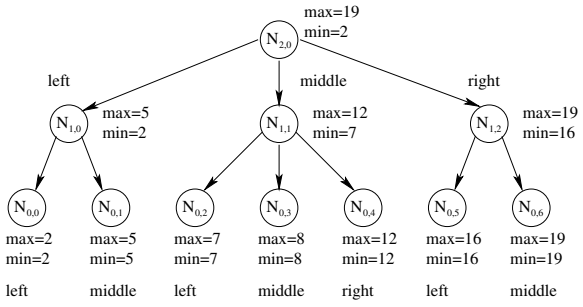


Figure 5.2: Sample 2-3 tree

Notice that an internal node has only two or three children. If it has two children, these are the "left" and "middle" ones, and if it has three children these are the "left", "middle" and "right" ones. In other words, an internal node always has "left" and "middle" children. A leaf has no children and $min = max = c_j$. Leaves are ordered in the following way: leaves on the left represent smaller serial numbers than leaves on the right.

At this point a new question arises: how to demonstrate that a certain certificate has not been revoked. In other words, how to prove that a certain target certificate

[1] To do such a thing, an attacker needs to find a pre-image of a OWHF which is computationally infeasible by definition.

identified by serial number c_{target} does not belong to the set of revoked certificates Φ. To prove that $c_{target} \notin \Phi$, as the leaves are ordered, it is enough to demonstrate the existence of two leaves, a **minor adjacent** (c_{minor}) and a **major adjacent** (c_{major}) that fulfill:

1. $c_{major} \in \Phi$.
2. $c_{minor} \in \Phi$.
3. $c_{minor} < c_{target} < c_{major}$.
4. c_{minor} and c_{major} are adjacent nodes.

Apart from the data that identifies the certificate that has been revoked, revocation systems provide the reason and the date of revocation. To add this information to the MHT, we need to include it in the computation of the cryptographic value of the corresponding leaf (5.5).

$$H_{0,j} = h\{CertID|Reason|Date\} \tag{5.5}$$

5.2.3 How to respond to a request

As pointed out in the previous section the response varies according to whether or not the requested certificate belongs to the MHT. If $c_{target} \in \Phi$, we need to provide the user with the $\mathcal{P}$ath from the target leaf to the root. Next, we propose a recursive algorithm that starts from the root and goes across the tree until the target leaf is reached. During this trip through the tree, the algorithm finds the $\mathcal{P}$ath for the target leaf. To sum up, when the algorithm has reached a certain internal node denoted by N_i , it decides the next node to go to (denoted by N_{i-1}) and adds the siblings of N_{i-1} to the $\mathcal{P}$ath. The algorithm is presented below in pseudo-code and it is illustrated by Example 5.2.

```
While (N_i ≠ leaf){
  If (N_i has two children){
    If (c_target < N_i.middle.min){
      N_{i-1} = N_i.left
```

```
      #N_i.middle is included in Path
      N_i.middle ≫Path
      }
    Else {
      N_{i-1} = N_i.middle
      N_i.left ≫Path
      }
}
If (N_i has three children){
  If (c_target < N_i.middle.min){
      N_{i-1} = N_i.left
      N_i.middle ≫Path
      N_i.right ≫Path
      }
  Else if (c_target < N_i.right.min){
      N_{i-1} = N_i.middle
      N_i.left ≫Path
      N_i.right ≫Path
      }
  Else {
      N_{i-1} = N_i.right
      N_i.left ≫Path
      N_i.middle ≫Path
      }
```

Example 5.2.

1. Start on the root. [see Figure 5.3# $root = N_{2,0}$, $c_{target} = 16$]
2. Choose next node. [see Figure 5.3# $N_{1,2}$]
3. Add siblings to $\mathcal{P}$ath. [see Figure 5.3# $\{N_{1,0}, N_{1,1}\} \gg \mathcal{P}$ath]
4. Choose next node. [see Figure 5.3# $N_{0,5}$]

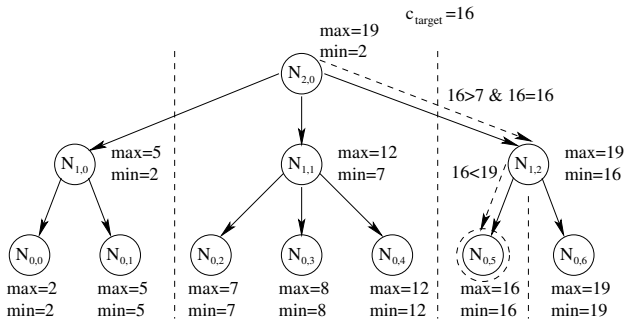


Figure 5.3: Example: searching a revoked certificate

5. Add siblings to $\mathcal{P}$ath. [see Figure 5.3# $N_{0,6} \gg \mathcal{P}$ath]

6. The algorithm ends because the target leaf has been reached. [see Figure 5.3# $c_{target} = 16$]

If $c_{target} \notin \Phi$, we need to find the two adjacent leaves to the target certificate. Notice that if $c_{target} \notin \Phi$ and we follow the algorithm previously described, we will get the $\mathcal{P}$ath of the minor adjacent of c_{target}. To find the major adjacent we need a similar algorithm using other border-lines, which is why we use the *max* parameter.

5.2.4 How to revoke a certificate

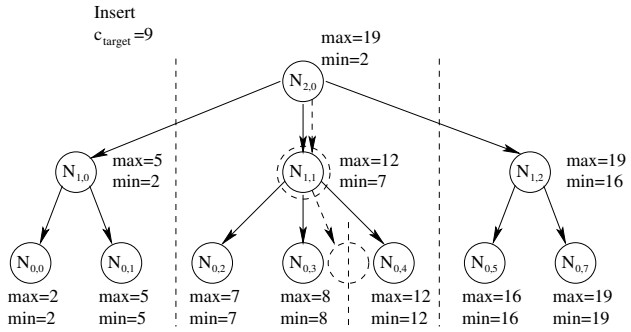
When a certificate has been revoked, it must be inserted in the MHT. The algorithm that we propose for inserting a revoked certificate in the MHT is depicted below and it is also illustrated by an example in Figure 5.4.

1. Start searching the target leaf. [see Figure 5.4-a# $c_{target} = 9$]

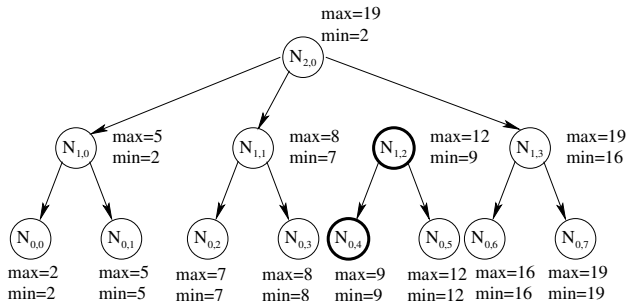
2. The search is stopped at level 1, denoted by $N_{1,j}$. [see Figure 5.4-a# $N_{1,j} = N_{1,1}$]

3. If $N_{1,j}$ has "2" children, then c_{target} is inserted as child of $N_{1,j}$ in the correct position.

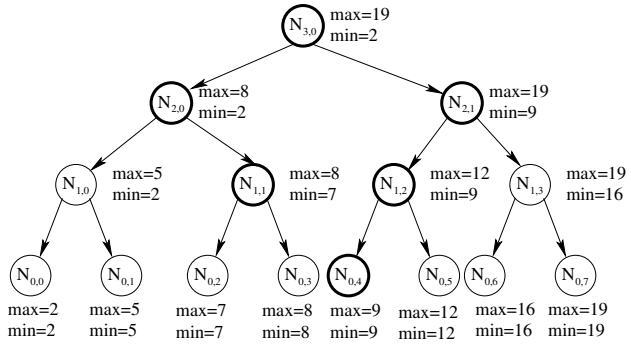
   (a) $N_{1,j}.max$, $N_{1,j}.min$ and $H_{1,j}$ are updated.



(a) Step 1



(b) Step 2



(c) Step 3

Figure 5.4: Example: inserting a revoked certificate (with root splitting)

(b) Now, the resulting tree is balanced. We recalculate the $H_{i,j}$ from the leaf we are at to the root and the algorithm ends.

4. If $N_{1,j}$ has "3" children, c_{target} would be the forth child, which is not possible in a 2-3 tree by definition. Then,

   (a) $N_{1,j}$ is split and a new node is created. We denote it by $N_{1,j+1}$. [see Figure 5.4-b# $N_{1,j+1} = N_{1,2}$]

   (b) The two leaves with the smaller serial number remain as children of $N_{1,j}$, while the other two leaves become children of $N_{1,j+1}$.

   (c) $N_{1,j}.max$, $N_{1,j}.min$, $H_{1,j}$, $N_{1,j+1}.max$, $N_{1,j+1}.min$ and $H_{1,j+1}$ are updated.

The father of $N_{1,j}$ is denoted by $N_{2,k}$. [see Figure 5.4-b# $N_{2,k} = N_{2,0}$]

5. $N_{1,j+1}$ must be inserted as a child of $N_{2,k}$. To insert the new child, the algorithm is applied recursively.

In the last instance, the root node may be split. In this case, a new root is created whose children will be the old root and the new node. The root splitting is how the tree grows. [see Figure 5.4-c#]

5.2.5 How to delete an expired certificate

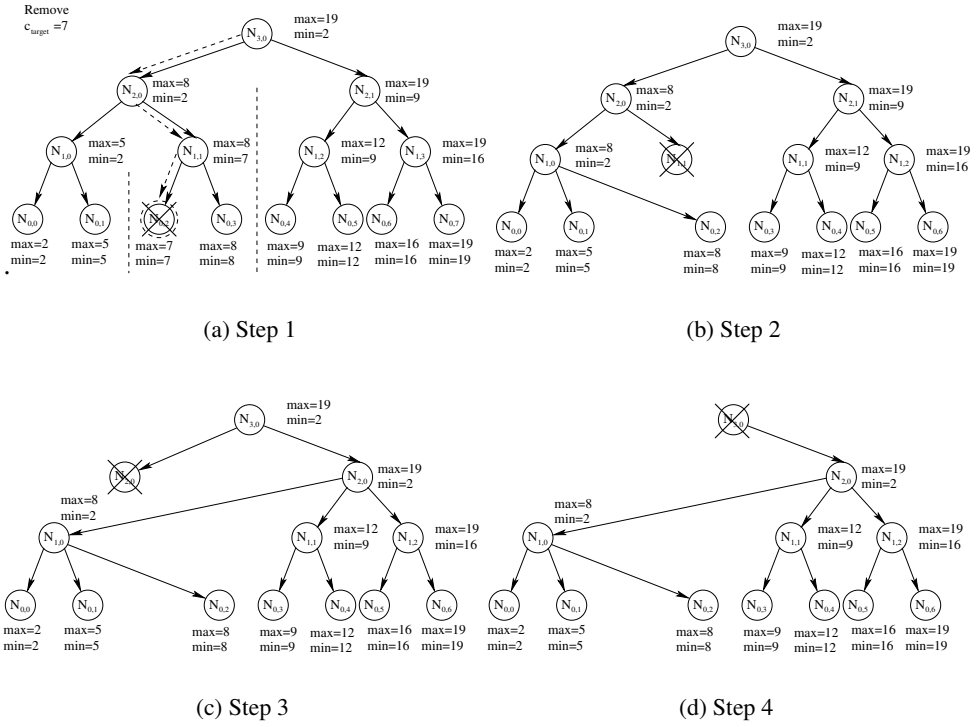
It makes no sense to have expired certificates in revocation databases since they are not valid. The algorithm for deleting an expired certificate is rather complicated: we illustrate an example in Figure 5.5.

1. Start searching the target leaf. We denote it by $N_{0,j}$. [see Figure 5.5-a# $c_{target} = 7$, $N_{0,j} = N_{0,2}$]

2. If the target leaf is not found, the algorithm is aborted.

We denote by $N_{1,k}$ the father of $N_{0,j}$. [see Figure 5.5-a# $N_{1,k} = N_{1,1}$]

3. If $N_{1,k}$ has "3" children, then



(a) Step 1

(b) Step 2

(c) Step 3

(d) Step 4

Figure 5.5: Example. Deleting an expired certificate

(a) $N_{0,j}$ is deleted.

(b) The children of $N_{1,k}$ are placed in their correct position.

(c) $N_{1,k}.max$, $N_{1,k}.min$ and $H_{1,k}$ are updated.

(d) Now, the resulting tree is balanced. We recalculate the $H_{i,j}$ from the leaf we are at to the root and the algorithm ends.

4. If $N_{1,k}$ has "2" children, then

(a) $N_{0,j}$ is deleted. Notice that $N_{1,k}$ is left only with one child, which is not possible in a 2-3 tree by definition. Thus, $N_{1,k}$ must be reallocated within the tree. [see Figure 5.5-a#]

The father of $N_{1,k}$ is denoted by $N_{2,m}$. [see Figure 5.5-a# $N_{2,m} = N_{2,0}$]

5. If $N_{1,k} = N_{2,m}.left$, then its adjacent node is $N_{1,k+1} = N_{2,m}.middle$. Then

   (a) If $N_{1,k+1}$ has "3" children, then

      i. $N_{1,k+1}.left$ is reallocated as $N_{1,k}.middle$.

      ii. The children of $N_{1,k+1}$ are placed in the correct position.

      iii. $N_{1,k}.max$, $N_{1,k}.min$, $H_{1,k}$, $N_{1,k+1}.max$, $N_{1,k+1}.min$ and $H_{1,k+1}$ are updated.

      iv. Now, the resulting tree is balanced. We recalculate the $H_{i,j}$ from the leaf we are at to the root and the algorithm ends.

   (b) If $N_{1,k+1}$ has "2" children, then

      i. The child of $N_{1,k}$ is reallocated as $N_{1,k+1}.left$.

      ii. The children of $N_{1,k+1}$ are placed in the correct position.

      iii. $N_{1,k}$ is deleted.

      iv. If $N_{2,m}$ has been left with "2" children, the resulting tree is balanced. We recalculate the $H_{i,j}$ from the leaf we are at to the root and the algorithm ends.

      v. If $N_{2,m}$ has been left with only "1" child, the algorithm must be applied recursively. [see Figures 5.5-b,5.5-c and 5.5-d#]

6. If $N_{1,k} = N_{2,m}.middle$, its adjacent nodes are $N_{1,k-1} = N_{2,m}.left$ and $N_{1,k+1} = N_{2,m}.right$.

   (a) If $N_{1,k-1}$ has "3" children, then

      i. $N_{1,k-1}.right$ is reallocated as $N_{1,k}.left$.

      ii. The children of $N_{1,k-1}$ are placed in the correct position.

      iii. $N_{1,k}.max$, $N_{1,k}.min$, $H_{1,k}$, $N_{1,k-1}.max$, $N_{1,k-1}.min$ and $H_{1,k-1}$ are updated.

      iv. Now, the resulting tree is balanced. We recalculate the $H_{i,j}$ from the leaf we are at to the root and the algorithm ends.

   (b) Else if $N_{1,k+1}$ exists and has "3" children,

      i. $N_{1,k+1}.left$ is reallocated as $N_{1,k}.middle$.

ii. The children of $N_{1,k+1}$ are placed in the correct position.

iii. $N_{1,k}.max$, $N_{1,k}.min$, $H_{1,k}$, $N_{1,k+1}.max$, $N_{1,k+1}.min$ and $H_{1,k+1}$ are updated.

iv. Now, the resulting tree is balanced. We recalculate the $H_{i,j}$ from the leaf we are at to the root and the algorithm ends.

(c) Else if $N_{1,k-1}$ and $N_{1,k+1}$ have both "2" children, then

i. The child of $N_{1,k}$ is reallocated as $N_{1,k-1}.right$.

ii. $N_{1,k}$ is deleted.

iii. $N_{1,k-1}.max$, $N_{1,k-1}.min$ and $H_{1,k-1}$ are updated.

iv. Now, the resulting tree is balanced. We recalculate the $H_{i,j}$ from the leaf we are at to the root and the algorithm ends.

(d) Else if $N_{1,k+1} = null$ and $N_{1,k-1}$ has "2" children, then

i. The child of $N_{1,k}$ is reallocated as $N_{1,k-1}.right$.

ii. $N_{1,k}$ is deleted. Notice that $N_{2,m}$ is left only with one child, which is not possible in a 2-3 tree by definition. To reallocate $N_{2,m}$, the algorithm must be applied recursively.

7. If $N_{1,k} = N_{2,m}.right$, then its adjacent node is $N_{1,k-1} = N_{2,m}.middle$.

(a) If $N_{1,k-1}$ has "3" children, then

i. $N_{1,k-1}.right$ is reallocated as $N_{1,k}.left$.

ii. The children of $N_{1,k-1}$ and $N_{1,k}$ are placed in the correct position.

(b) $N_{1,k}.max$, $N_{1,k}.min$, $H_{1,k}$, $N_{1,k-1}.max$, $N_{1,k-1}.min$ and $H_{1,k-1}$ are updated.

(c) Now, the resulting tree is balanced. We recalculate the $H_{i,j}$ from the leaf we are at to the root and the algorithm ends.

(d) If $N_{1,k-1}$ has "2" children, then

(e) The child of $N_{1,k}$ is reallocated as $N_{1,k-1}.right$

(f) $N_{1,k-1}.max$, $N_{1,k-1}.min$ and $H_{1,k-1}$ are updated.

(g) $N_{1,k}$ is deleted.

(h) Now, the resulting tree is balanced. We recalculate the $H_{i,j}$ from the leaf we are to the root and the algorithm ends.

5.3 AD-MHT status checking protocol

Below, the authors propose a request/response protocol devised in ASN.1 to perform the status checking in the AD-MHT[2]. The protocol imports ASN.1 definitions from [MAM+99] and [HFPS99].

5.3.1 The AD-MHT request

Figure 5.6 shows the ASN.1 description for an AD-MHT request. Each request contains:

- The protocol version (currently version 1).
- A unique identifier for each target certificate (`CertID`).
- Optionally the request might be signed by the client.

Upon receipt of a request, the repository determines whether the message is well formed, whether it is configured to provide the requested service and whether the request contains the compulsory information. If any one of the prior conditions are not met, the repository produces a response with an error message that is indicated in `MHTResponseStatus` (see Figure 5.7). Otherwise, it returns a response with the appropriate status data.

5.3.2 The AD-MHT response

Figure 5.7 shows the ASN.1 description for an AD-MHT response. The response syntax is more complex than the request since it must include the $\mathcal{D}$igest of the tree and one or two $\mathcal{P}$aths for each target certificate (remember that we need to prove the existence of the minor and major adjacent leaves to ensure that a certificate is not revoked).

The `BasicADMHTResponse` contains:

[2]We propose to reserve the following MIME types for the HTTP transport of the AD-MHT requests and responses:

- `application/admht-request`
- `application/admht-response`

```
ADMHTRequest ::= SEQUENCE {
tbsRequest              ADMHTTBSRequest,
optionalSignature  [0] EXPLICIT OCTET STRING OPTIONAL }
ADMHTTBSRequest ::= SEQUENCE {
version             [0] EXPLICIT Version OPTIONAL,
requestList             SEQUENCE OF ADMHTCertRequest }
ADMHTCertRequest ::= SEQUENCE {reqCert   CertID }
CertID ::= SEQUENCE {
issuerName    [0] EXPLICIT OCTET STRING OPTIONAL,--Hash of the issuer
(CA) DN
issuerKeyHash [1] EXPLICIT OCTET STRING OPTIONAL,--Hash of the issuer
(CA) public-key
serialNumber      CertificateSerialNumber }
CertificateSerialNumber ::= OCTET STRING
```

Figure 5.6: ASN.1 description of the AD-MHT Request

- A `SignedTreeDigest` that is common for all the target certificates.
- A `SingleADMHTResponse` per target certificate.

The `SignedTreeDigest` includes:

- The `issuer`, that is, the DN of the RDI.
- The `validityPeriod`.
- The `rootHash` inclusion is optional because the client can calculate it from the $\mathcal{P}$ath, even though the RDI must include the `rootHash` in the signature computation.

The `SingleADMHTResponse` includes the information necessary to check, whether or not the target certificate has been revoked:

- If `minorAdjacent= majorAdjacent`, the certificate has been revoked. Then, only the `minorAdjacent` is included in the $\mathcal{P}$ath.
- If `minorAdjacent`$\neq$ `majorAdjacent`, the target certificate has not been revoked. Then, $\mathcal{P}$aths from both adjacent leaves must be included in the response.

```
ADMHTResponse ::= SEQUENCE {
responseStatus             MHTResponseStatus,
basicResponse          [0] EXPLICIT BasicADMHTResponse OPTIONAL }
BasicADMHTResponse ::= SEQUENCE {
signedTreeDigest           SignedTreeDigest,
singleResponse             SingleADMHTResponse }
MHTResponseStatus ::= ENUMERATED {
successful            (0),  --Response has valid confirmations
malformedRequest      (1),  --Illegal confirmation request
internalError         (2),  --Internal error in issuer
tryLater              (3),  --Try again later
                            --(4) and (5) are not used
unauthorized          (6) } --Request unauthorized
SignedTreeDigest ::= SEQUENCE {
tbsTreeDigest             TBSTreeDigest,
signature                 OCTET STRING } --SHA1 with RSA is used
TBSTreeDigest ::= SEQUENCE {
issuer                Name, --DN of the RDI
validity              Validity,
rootHash          [0] EXPLICIT OCTET STRING OPTIONAL,
extensions        [1] EXPLICIT Extensions OPTIONAL }
SingleADMHTResponse ::= SEQUENCE {
minorAdjacent        TreePath,
majorAdjacent    [0] EXPLICIT TreePath OPTIONAL } --Only needed for
not revoked certificates
TreePath ::= SEQUENCE {
adjacentID      CertID,
status          RevokedInfo,
firstPathStep   PathStep }
PathStep ::= SEQUENCE { --SHA1 is used
leftHash         [0] EXPLICIT OCTET STRING OPTIONAL,
middleHash       [1] EXPLICIT OCTET STRING OPTIONAL,
rightHash        [2] EXPLICIT OCTET STRING OPTIONAL,
nextPathStep     [3] EXPLICIT PathStep OPTIONAL }
RevokedInfo ::= SEQUENCE {
revocationTime         GeneralizedTime,
revocationReason   [0] EXPLICIT CRLReason OPTIONAL }
```

Figure 5.7: ASN.1 description of the AD-MHT Response

The `TreePath` includes:

- The `adjacentID` that uniquely identifies the target certificate.
- The `status`, which includes the revocation date and the revocation reason.
- The `PathSteps` that allow H_{root} to be computed recursively.

Each `PathStep` contains:

- The cryptographic value(s) necessary to compute a cryptographic value in the upper level.
- The next `PathStep` (in the last instance, the root is reached).

The algorithm that allows H_{root} to be computed is depicted below:

1. The *i*-th `PathStep` allows H_{i+1} to be computed. H_{i+1} can be a `leftHash`, a `middleHash` or a `rightHash` in the $(i+1)$-th level.
2. If H_{i+1} is a `leftHash`,
   (a) If the $(i+1)$-th level has "2" nodes, then the `nextPathStep` will include only a `middleHash`.
   (b) If the $(i+1)$-th level has "3" nodes, then the `nextPathStep` will include a `middleHash` and a `rightHash`.
3. If H_{i+1} is a `middleHash`, then
   (a) If the $(i+1)$-th level has "2" nodes, then the `nextPathStep` will include only a `leftHash`.
   (b) If the $(i+1)$-th level has "3" nodes, then the `nextPathStep` will include a `leftHash` and a `rightHash`.
4. If H_{i+1} is a `rightHash`, then the `nextPathStep` will include a `leftHash` and a `middleHash`.

5.4 Response verification

In this Section, we solve some open issues related to the response verification that were not addressed in the original AD proposal.

5.4.1 Adjacent node checking

First of all, the client must check that each `TreePath` included in the response is correct, that is, that the `rootHash` computed from the $\mathcal{P}$ath matches the `rootHash` included in the $\mathcal{D}$igest.

If a target certificate has not been revoked, this is not enough: the client also needs to ensure that the `TreePath`s provided belong to real adjacent nodes (remember that the repository is a non-TTP, so the user can be misled into believing that a certain pair of nodes within the tree are adjacent leaves).

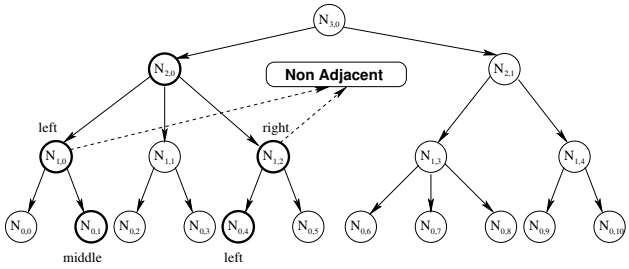
Example 5.3. Let us suppose that we want to perform a transaction using a given certificate. The certificate is identified by c_{target}. Using the example in Figure 5.2, let us assume that $c_{target} = 16$. Notice that $c_{target} \in \Phi$, but let us suppose that a malicious repository provides us with the $\mathcal{P}$ath for a couple of leaves that belong to the MHT, claiming that they are adjacent. For instance, let us assume that these leaves are $c_{minor} = 8$ and $c_{major} = 19$. If we only check that $\{c_{minor}, c_{major}\} \in \Phi$, we will think that c_{target} is valid and we will perform the fraudulent transaction.

Next, the authors propose a recursive algorithm that given a certain couple of `TreePath`s, verifies whether they actually belong to "real" adjacent leaves. The algorithm works without adding any extra information to the protocol or the data structures. The alleged adjacent leaves are denoted by $N_{0,j}$ and $N_{0,j+1}$.

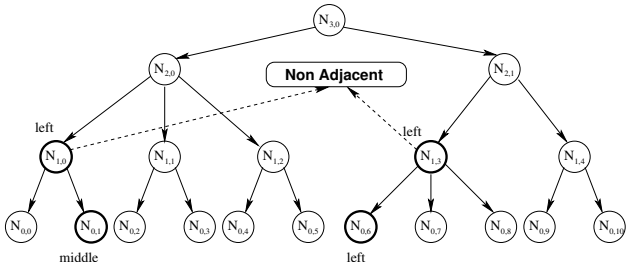
1. The client computes $H_{1,m}$ and $H_{1,n}$, which denote respectively the cryptographic values of the fathers of $N_{0,j}$ and $N_{0,j+1}$.

2. If $H_{1,m} = H_{1,n}$, then both leaves have the same father. Then,

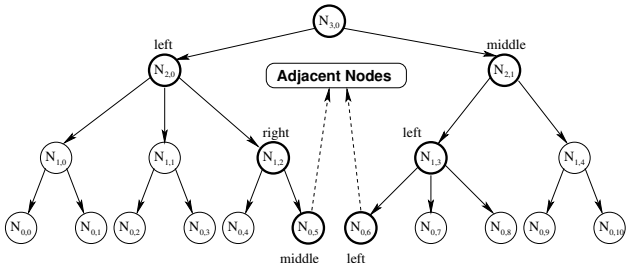
   (a) If $N_{0,j} = N_{1,m}.left$ and $N_{0,j+1} = N_{1,m}.middle$, then **they are adjacent nodes**.



(a) Example 1



(b) Example 2



(c) Example 3

Figure 5.8: Examples of adjacent node checking

(b) If $N_{0,j} = N_{1,m}.middle$ and $N_{0,j+1} = N_{1,m}.right$, then **they are adjacent nodes**.

(c) Else, **they are not adjacent nodes**.

3. If $H_{1,m} \neq H_{1,n}$, then the leaves do not have the same father. Then,

(a) If $N_{1,m}$ has "2" children and $N_{0,j} \neq N_{1,m}.middle$, then **they are not adjacent nodes**.

(b) If $N_{1,m}$ has "3" children and $N_{0,j} \neq N_{1,m}.right$, then **they are not adjacent nodes**.

(c) If $N_{0,j+1} \neq N_{1,n}.left$, then **they are not adjacent nodes**.

(d) Else computes $H_{2,p}$ and $H_{2,q}$, which denote respectively the cryptographic values of the fathers of $N_{1,m}$ and $N_{1,n}$, and applies the algorithm recursively. In the last instance, the root is the unique common father between the pair of nodes.

We illustrate some examples of the previous algorithm in Figure 5.8.

It must be pointed that the strength of the above algorithm resides in the position that a certain node occupies relative to its father, in other words whether a certain node is "left", "middle" or "right". Notice that the end user can trust this information since the relative node positions cannot be swapped by a malicious repository because we use a non-commutative hash function. If the malicious repository modifies the concatenation order, then it changes the cryptographic value of the next step (5.6)

$$H_{i+1,k} = h(H_{i,j}|H_{i,j+1}) \neq h(H_{i,j+1}|H_{i,j})\,. \tag{5.6}$$

5.4.2 MHT bounds

On the other hand, notice that in some cases minor adjacent, major adjacent or both are missing. For instance,

- If $\Phi = \{\emptyset\}$, i.e. the MHT is empty, then both adjacent nodes are missing.

- If $c_{target} < c_j \; \forall j$, i.e. the serial number of the target certificate is smaller than the smallest leaf within the MHT, then there is no minor adjacent.
- If $c_{target} > c_j \; \forall j$, i.e. the serial number of the target is bigger than the biggest leaf within the MHT, then there is no major adjacent.

A serial number is nothing more than an array of bits. The serial number with all its bits set to 0 and the serial number with all its bits set to 1 are reserved (not assigned to "real" certificates) to bound the MHT. These "special" serial numbers represent 0 and $+\infty$ respectively, so now each possible serial number has two adjacent nodes independently of the certificates contained by the MHT.

5.5 Security discussion

For AD-MHT to be effective, certificate-using applications must connect to the any of the AD-MHT repositories available. In the event that such a connection cannot be obtained, certificate-using applications could implement other processing logic (CRL, OCSP etc.) as a fall-back position.

Another important aspect that the AD-MHT administrators must take into account when deploying the system is that if you do not rely on HTTP as the transport mechanism, you might get in trouble when going across firewalls because many of them will not allow anything but HTTP to pass through, so there could be a problem to communicate end-points from either side of a firewall. Furthermore, AD-MHT administrators should not forget that the HTTP transport makes it possible for firewall administrators to configure them to selectively block out messages using specific MIME types. Administrators of the AD-MHT system should also take the reliance of HTTP caching into account because it may give unexpected results if the AD-MHT requests or responses are cached by intermediate servers and these servers are incorrectly configured or are known to have cache management faults. Therefore, AD-MHT deployments should take the reliability of HTTP cache mechanisms into account when AD-MHT over HTTP is used.

Possible attacks on the AD-MHT system include the following:

- *RDI Masquerade Attack*: An attacker or a malicious repository could attempt to masquerade a trustworthy RDI.

Countermeasures: This attack is avoidable if the client verifies the signature included in the $\mathcal{D}$igest using the correct certificate of the RDI.

- *Response Integrity Attack*: An attacker or a malicious repository could modify part or the whole of a response sent by legitimate repository.
  Countermeasures: This attack cannot be successfully carried out if the response is verified according to the procedure described in Section 5.4. Notice that the inherent structure of the MHT together with the response verification algorithm make it infeasible to alter an AD-MHT response without making it invalid: the MHT cannot be modified without modifying the root which is signed, and fake adjacent nodes are detected by the protocol presented in Section 5.4.

- *Replay Attack*: An attacker or a malicious repository could resend an old (good) response prior to its expiration date but after the $\mathcal{D}$igest has changed.
  Countermeasures: Decreasing the validity periods of the responses will decrease the window of vulnerability.

- *Denial of Service Attack*: An attacker could intercept the responses from a legitimate repository and delete them or the attacker could delay the responses by, for example, deliberately flooding the network, thereby introducing large transmission delays. Notice that requests do not contain the repository they are directed to, which allows an attacker to replay a request to any number of repositories. Finally, unsigned error responses open up the protocol to another denial of service attack, in which the attacker sends false error responses.
  Countermeasures: The only way to prevent this attack is redundancy of repositories, which is easy to deploy since repositories are non-TTPs.

5.6 The Cervantes module of AD-MHT

Figure 5.9 shows the behaviour of the SCH that manages AD-MHT.

1. The ADMHTPortThread listens for requests addressed to a certain TCP port.

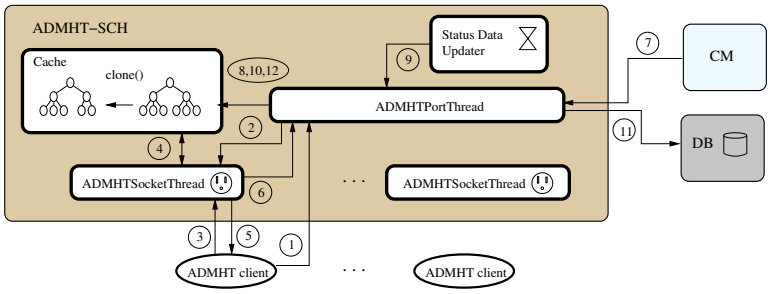


Figure 5.9: ADMHT-SCH

2. For each request the ADMHTPortThread creates an ADMHTSocketThread.

3. The ADMHTSocketThread receives the request.

4. The Cache stores two MHTs: the "listening tree" and the "management tree". The listening tree is used by the SCH to respond for status checking requests and it is immutable during the validity period of the $\mathcal{D}$igest. The management tree is updated for each expiration/revocation and after a validity period "VP" the management tree is cloned and the listening tree is replaced with the clone. Thus, the ADMHTSocketThread retrieves the status data ($\mathcal{D}$igest+$\mathcal{P}$ath) from the listening tree stored in Cache.

5. The ADMHTSocketThread sends the response to the client.

6. The ADMHTSocketThread informs the ADMHTPortThread about the bytes and processing time required to serve the request.

7. The CM informs the ADMHTPortThread about any change in the status data.

8. When the ADMHTPortThread is informed about a change in the status data (revocation/expiration) it must update the management tree.

9. Every VP the StatusDataUpdater asks the ADMHTPortThread to clone the management tree.

10. The ADMHTPortThread clones the management tree and signs the $\mathcal{D}$igest of the new listening tree.

11. If Cervantes is restarted and it must keep the revocation records from a previous execution, the ADMHTPortThread has to build the "first" MHT based on these records. So it retrieves all the recorded status data from the database and builds this first listening tree. If Cervates is started with an empty database, the listening tree will be also empty.

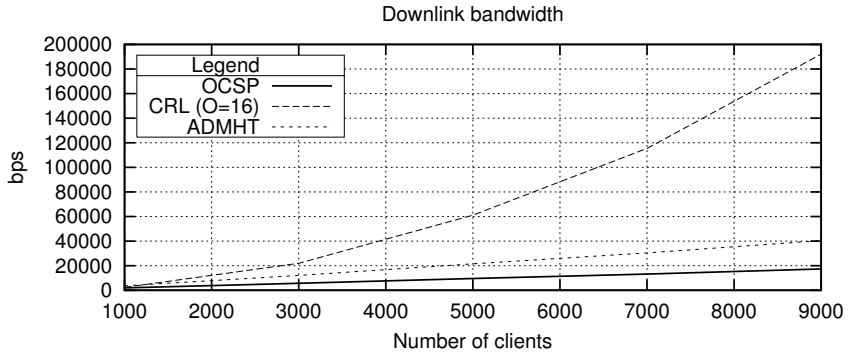
12. The ADMHTSocketThread sends the first listening tree to the Cache.

5.7 Evaluation

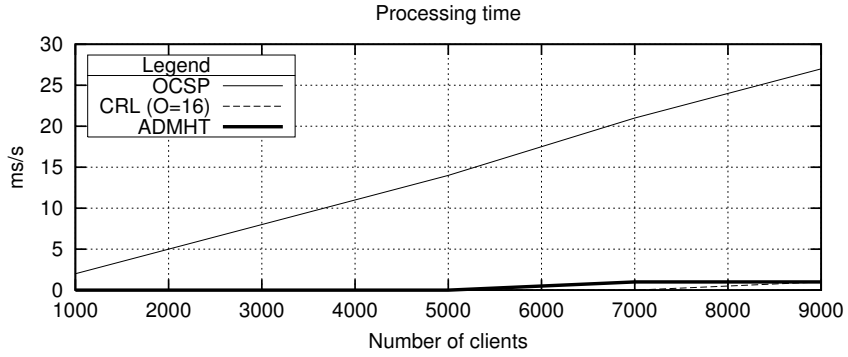
In this section O-CRL, OCSP and AD-MHT are evaluated by means of Cervantes in terms of down-link bandwidth utilization and processing capacity consumed per request. The experimental results have been obtained under the following conditions:

- The Cervantes server runs in a Pentium III (800 MHz).
- The clients generate 2 status checking requests per hour following an exponential probability density function.
- There are 10,000 clients.
- Each client has a certificate.
- There is an average of 10% revocation.
- The validity period of a CRL is 6 hours.
- The CRLs are cached by clients during their validity period.
- Clients will request their local cache instead of the repository if they have a cached CRL.
- 16 CRLs are issued within a validity period ($O = 16$).

Figure 5.10 shows the comparison among these systems. Despite the use of cache, the CRL performance in terms of down-link bandwidth is very poor compared to OCSP or AD-MHT. As a result, CRLs do not seem a good choice for distribution of



(a) Bandwidth



(b) Processing time

Figure 5.10: Scalability regarding the number of clients: AD-MHT versus O-CRL and OCSP

status data among end users and they should be only used as the distribution mechanism for intermediate entities. OCSP is a good choice in terms of bandwidth but the processing capacity resources it uses are the highest of the evaluated systems. Also, responders are needed in order to distribute the OCSP data. AD-MHT bandwidth performance is slightly worse than that of OCSP, but taking into account the overall performance, the AD-MHT might be a good choice for distribution of status data among end users because it does not require much bandwidth or processing capacity and a repository can be used to respond to AD-MHT requests.

5.8 Conclusions

The chapter discusses in detail the design and implementation of a certificate revocation system based on the MHT. The system, which is named AD-MHT, uses the data structures proposed in [NN00]. This chapter also addresses some important open issues that are necessary for implementing such a system and that were beyond the scope of the original proposal, such as how to respond to a request, how to revoke a certificate, how to delete an expired certificate, the communication protocol with the end users and the verification of a response. On the other hand, AD-MHT has proven to be resistant against malicious behaviors such as RDI masquerading, AD-MHT response modification, replay attacks or denial of service.

The AD-MHT system has been implemented as part of the Cervantes platform and finally an evaluation with Cervantes of AD-MHT versus CRL and OCSP has shown that taking into account the overall performance, the AD-MHT system might be a good choice for distribution of status data among end users because it does not require much bandwidth or processing capacity, and repositories can be used to respond to AD-MHT requests.

Chapter 6

E-MHT

6.1 Introduction

This chapter presents the Enhanced-MHT (E-MHT). The E-MHT is based on the AD-MHT but we add some mechanisms to the basic data structures of AD-MHT that allow the E-MHT to provide a response size that is close to (or even better than) typical online systems such as OCSP without degradating other resources of the system. These mechanisms include the optimization of the $\mathcal{P}$aths for non-revoked certificates, the division of revocation status data among multiple MHTs and a low cost mechanism for re-utilization of the MHT digests and E-MHT responses [MFES03d]. In [MFES03c] we presented a request/response protocol for status data retrieval from the E-MHT and we presented its ASN.1 definition.

The rest of the chapter is organized as follows: in Section 6.2 E-MHT is presented as a system that provides offline status responses with a reduced size. In Section 6.3 we present the E-MHT status checking protocol. In Section 6.4 we discuss the issues related to the response verification. In Section 6.5 we present the security discussion for the E-MHT. In Section 6.6 we show the implementation of E-MHT in Cervantes. In Section 6.7 E-MHT is evaluated regarding OCSP [MAM+99] and AD-MHT [MFES03e]. Finally, we conclude in Section 6.8.

6.2 The Enhanced-MHT basics

In this Section, we propose E-MHT which arises from a performance analysis of the AD-MHT. Remember that MHT-based systems are offline because the MHT can be pre-computed by the RDI and distributed to repositories. Offline systems are more robust than online systems in the sense that it is more complex to maintain the level of security of a responder than of a repository: a responder has to be online, but at the same time, it has to protect its private key against intruders. Regardless, AD-MHT repositories share the following characteristics with online responders:

- A MHT provides status data for particular certificates rather than data of a whole set of certificates.
- The repository computes part of the cryptographic proof for each response (i.e. the $\mathcal{P}$ath or $\mathcal{P}$aths).

Taking into account the previous features, it is easy to understand why the size of an AD-MHT response is usually some orders of magnitude smaller than a classical CRL. However, the size of an AD-MHT response is larger than an OCSP one because of the cryptographic values. For instance, we have observed in our tests with AD-MHT that for a population of 1,000 revoked certificates, the AD-MHT response doubles the size of an OCSP response. E-MHT agglutinates several mechanisms for enhancing the efficiency of traditional MHT-based systems. These mechanisms, which are explained below, include the optimization of the MHT $\mathcal{P}$aths for non-revoked certificates, the division of the revoked certificates among multiple MHTs, the re-utilization of the tree $\mathcal{D}$igest and the cached responses updating at a low cost.

6.2.1 Optimization of $\mathcal{P}$aths

As discussed in the previous section, the response for non-revoked certificates in the AD-MHT must include the $\mathcal{P}\text{ath}_{c_{minor}}$ and the $\mathcal{P}\text{ath}_{c_{major}}$. Taking into account that only a small percentage of the certificates are revoked (usually a 10% is considered in similar studies), the majority of the requests will be performed over non-revoked certificates. Therefore, a size reduction over this kind of response will

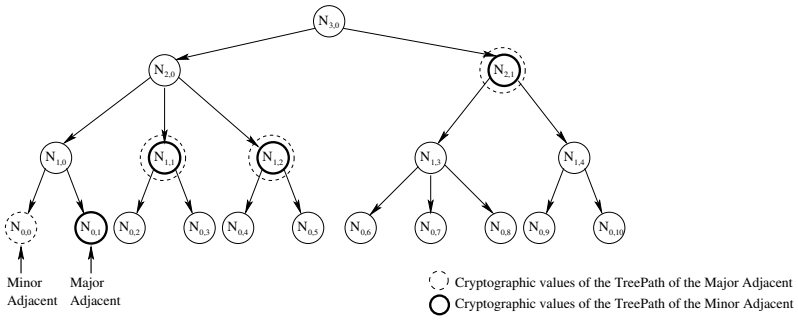


Figure 6.1: $\mathcal{P}$ath's optimization

have a great impact over the system performance. With this idea in mind, we propose below a way of reducing the number of cryptographic values for non-revoked certificates.

The repository builds the responses for non-revoked certificates as follows:

- It provides the complete $\mathcal{P}$ath for c_{minor}.
- To build the $\mathcal{P}\text{ath}_{c_{major}}$, the repository omits the cryptographic values that fulfill that $H_{i,j} \in \mathcal{P}\text{ath}_{c_{minor}}$ (i.e. redundant values that were already included in the $\mathcal{P}\text{ath}_{c_{minor}}$).

Example 6.1. Figure 6.1 shows an example where the repository can omit $H_{1,1}$, $H_{1,2}$ and $H_{2,1}$ in the $\mathcal{P}\text{ath}_{c_{major}}$. In this example, the number of values necessary to verify the response is reduced from 8 to 5, within a bigger MHT the saving can be considerable.

6.2.2 Multi-MHT

We borrow the philosophy of the X.509 CRL-DP as another way of reducing the response size: the group of revoked certificates Φ is divided into k subgroups and each subgroup is used to build a smaller MHT. We can use two ways of performing this division:

- Similarly to CRL-DP we can use a certificate extension to point to the correct MHT.

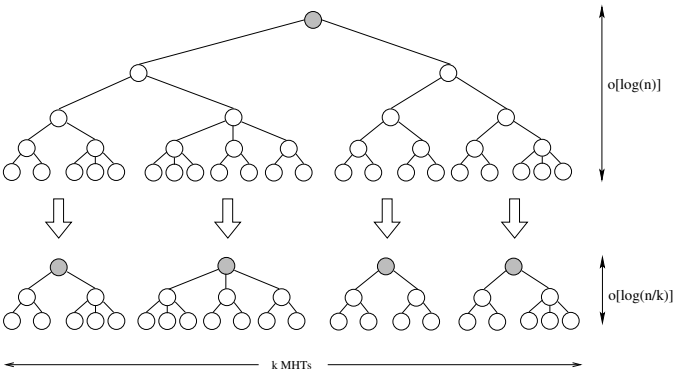


Figure 6.2: Multi-MHT division

- We can also perform the division without using any certificate extension. This can be achieved by using the least significant bits of the serial number. These bits will determinate which tree the certificate belongs to. The only constrain that this method introduces is that the number of trees k must have the form $k = 2^m$ where m is the number of bits used to perform the division (the protocol depicted in Section 6.3 is based on this kind of division).

A priori, the MHT division is not as beneficial as the CRL division. Notice that with k Distribution Points, the CRL size is divided by k while the number of cryptographic values per $\mathcal{P}$ath in the MHT is only divided by $o(log(k))$ (see Figure 6.2).

Notice also that spreading information among multiple MHTs increases the issuer's resources utilization in the publication process: the RDI has to keep updated k $\mathcal{D}$igests instead of just one. However, the effects of this update can be minimized if we choose the appropriate instances of time to perform the updating. Remember that each $\mathcal{D}$igest includes a validity period which is bounded by two time stamps: `not-valid-before` and `not-valid-after`. A naive way of setting these time-stamps up is to choose the same values for all the MHTs. Notice that this will lead the system to demand many resources in a short period of time in order to update the changes of each MHT, sign all the $\mathcal{D}$igests and distribute all this data to the repositories. The more clever way of performing the MHT's update is to distribute this process in time as much as possible. This can be achieved by using overlapping

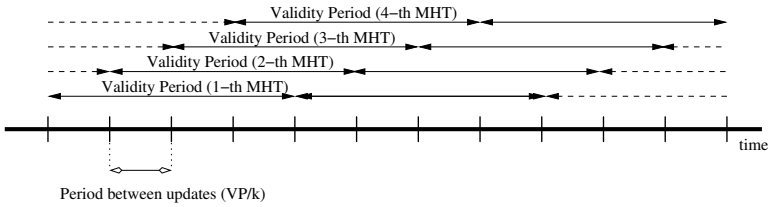


Figure 6.3: 4 MHTs with overlapping validity periods.

validity periods between the MHTs (Figure 6.3 shows 4 MHTs with overlapping periods).

Notice that overlapping the MHT validity periods is to some extent similar to overissuing CRLs so it is possible under certain circumstances to obtain similar benefits too. Remember that the main benefit provided by overissued CRL is the distribution of the requests in time, in other words, the reduction of the peaks in the request rate towards the repository. Something similar can be achieved in an E-MHT with overlapping periods:

- In a revocation system there are usually users that ask many times and very often about particular certificates (FAC). For instance, the certificate of the user, the certificate of the user's e-mail server or the certificate of the user's bank are clear candidates to be requested very frequently.
- There are also automatic platforms (such as security proxies) that send requests to the revocation system to periodically check the status of a particular group of certificates.

In all these situations, many requests are performed a little time after a previous response for that certificate has expired. If certificates belong to different MTHs with different expiration dates (i.e. `not-valid-after`), notice that the requests will be performed around different times and therefore the peak requests rates will be reduced as in O-CRL.

6.2.3 Re-utilization of the $\mathcal{D}$igest

Dividing the MHT on its own is not a spectacular breakthrough, but combined with $\mathcal{D}$igest re-utilization and caching, the overall performance can be considerably

improved as we will show in Section 6.7.

Because AD-MHT is composed by a single MHT, a new cryptographic value for the root has to be computed for any revocation or expiration during the validity period. On the contrary, data is fragmented among multiple MHTs in E-MHT. This circumstance favours that many MHTs stay unchanged after a validity period[1]. Although a tree does not change its shape, the RDI must sign its $\mathcal{D}$igest with a new validity period due to the revocation freshness requirements, that is to change the validity period time-stamps. The $\mathcal{D}$igest re-utilization offers an alternative to signature in case that we need to setup a new validity period of an unchanged $\mathcal{D}$igest. The point of the re-utilization mechanism is that resources consumed to update a $\mathcal{D}$igest are drastically reduced in comparison with conventional signature. To implement the $\mathcal{D}$igest re-utilization we use again a hash chain. From now on, the validity period included in the $\mathcal{D}$igest will be denoted as the "documented" validity period and `nextUpdate` will denote the end of this period. Let's see the parameters involved in the process:

`primaryUpdateValue` (R) is the secret nonce. R is only known by the RDI and it is generated each time a MHT is created or its root has changed.

`maximumUpdateIndex` (d) is the maximum number of periods that a $\mathcal{D}$igest can be re-utilized.

`baseUpdateValue` (R_0) is the last value of the hash chain and it is included in the signature computation of the $\mathcal{D}$igest. R_0 is computed by applying $d+1$ times h over R: $R_0 = h^{d+1}(R)$.

`currentUpdateValue` (R_i) is computed by applying $d+1-i$ times h over R: $R_i = h^{d+1-i}(R)$. Where i will denote the number of periods "Δ" elapsed from the documented one.

A relying party can verify the validity of a $\mathcal{D}$igest that it is living beyond its documented life-time, say, at time t, where t is included within the period $[\texttt{nextUpdate}+(i-1)\Delta, \texttt{nextUpdate}+i\Delta]$, by checking that $R_0 = h^i(R_i) \quad with\, i \leq d$.

[1] Actually, the probability that a MHT does not change during a validity period gets multiplied by k.

Notice that the resources increase in the revocation publication due to the division can be compensated with $\mathcal{D}$igest re-utilization because to update an unchanged MHT, the RDI just needs to send the appropriate `currentUpdateValue` instead of a new $\mathcal{D}$igest[2].

6.2.4 E-MHT responses update

An E-MHT response can also be updated beyond its documented life-time with a `currentUpdateValue` if the $\mathcal{D}$igest included in it has not changed. In this sense, previous E-MHT responses can be cached by clients, so that if the client needs to check the status of the same certificate later, the client can ask for a `currentUpdateValue` instead of downloading a complete E-MHT response which is larger[3]. Moreover, if a client usually requests for a given set of certificates (FAC), then, the responses of these certificates will be likely cached and they might be updated by means of a `currentUpdateValue` parameter. On the other hand, response re-utilization permits the relying parties to retrieve more timely status data from the system without perceptibly affecting the scalability, due to the following reasons:

- A small validity period makes less probable that a MHT changes during this period of time.
- With small validity periods, the responses need to be updated more often but this can be performed at a very low cost (using response re-utilization) if the corresponding MHT has not changed.

6.3 E-MHT status checking protocol

The request-response protocol for the E-MHT status checking has been designed and implemented by the authors in ASN.1 and it is actually an extension of the AD-MHT protocol.

[2] A $\mathcal{D}$igest is about 10 times larger than a `currentUpdateValue` for 1000 revoked certificates (using our implementation).

[3] A E-MHT response is about 20 times larger than a `currentUpdateValue` for 1000 revoked certificates (using our implementation).

```
EMHTRequest ::=     SEQUENCE {
 tbsRequest            EMHTTBSRequest,
 optionalSignature [0] EXPLICIT OCTET STRING OPTIONAL }
EMHTTBSRequest ::=    SEQUENCE {
 version           [0] EXPLICIT Version OPTIONAL,
 requestList           SEQUENCE OF EMHTCertRequest }
EMHTCertRequest::=    SEQUENCE {
 reqCert               CertID,
 baseUpdateValue  [0] EXPLICIT OCTET STRING OPTIONAL }
CertID ::= SEQUENCE {
 issuerName    [0] EXPLICIT OCTET STRING OPTIONAL,--Issuer DN hash
 issuerKeyHash [1] EXPLICIT OCTET STRING OPTIONAL,--Issuer pub key hash
 serialNumber      CertificateSerialNumber }
```

Figure 6.4: ASN.1 description of the E-MHT Request

6.3.1 The E-MHT request

Figure 6.4 shows the ASN.1 description for an E-MHT request.

Each `EMHTRequest` contains:

- The protocol version.
- An unique identifier for each target certificate.
- The request signature that is optional.
- The `baseUpdateValue` is also optional and it is included in the request if the client has an entry in its cache for the target certificate and the response re-utilization has not overflow ($i \leq d$), i.e. the inclusion of `baseUpdateValue` means that client wants a re-utilization of its cache entry if possible.

6.3.2 The E-MHT response

The response syntax is presented in Figure 6.5. Upon receipt of a request, the repository determines whether the message is well formed, whether it is configured to provide the requested service and whether the request contains the compulsory information. If any one of the prior conditions are not met, the repository produces a response with an error message that it is indicated in `MHTResponseStatus`. Otherwise, it returns a response with the corresponding status.

```
EMHTResponse   ::=   SEQUENCE {
 responseStatus      ResponseStatus,
 basicResponse   [0] EXPLICIT BasicEMHTResponse OPTIONAL }
BasicEMHTResponse   ::=   SEQUENCE { singleResponse SingleEMHTResponse }
SingleEMHTResponse ::= SEQUENCE {
 currentUpdateValue [0] EXPLICIT OCTET STRING OPTIONAL,
 emhtResponseData   [1] EXPLICIT EMHTResponseData OPTIONAL }

ResponseStatus ::= ENUMERATED {
 successful         (0), --Response has valid confirmations
 malformedRequest   (1), --Illegal confirmation request
 internalError      (2), --Internal error in issuer
 tryLater           (3), --Try again later
                         --(4) and (5) are not used
 unauthorized       (6) }--Request unauthorized

EMHTResponseData ::= SEQUENCE {
 signedTreeDigest    SignedTreeDigest,
 minorAdjacent       TreePath,
 majorAdjacent   [0] EXPLICIT TreePath OPTIONAL } --Only for not revoked
SignedTreeDigest ::= SEQUENCE {
 tbsTreeDigest       TBSTreeDigest,
 signature           OCTET STRING }
TBSTreeDigest ::= SEQUENCE {
 issuer            Name,
 validity          Validity, --Validity Period
 numberOfTrees     INTEGER,
 treeNumber    [0] EXPLICIT OCTET STRING OPTIONAL,
 baseUpdateValue   OCTET STRING,
 maxUpdates        INTEGER,
 rootHash      [0] EXPLICIT OCTET STRING OPTIONAL}
TreePath ::= SEQUENCE {
 adjacentID        CertID,
 status            RevokedInfo,
 firstPathStep     PathStep }
PathStep ::= SEQUENCE {
 leftNode           [0] EXPLICIT OCTET STRING OPTIONAL,
 mediumNode         [1] EXPLICIT OCTET STRING OPTIONAL,
 rightNode          [2] EXPLICIT OCTET STRING OPTIONAL,
 nextPathStep       [3] EXPLICIT PathStep OPTIONAL }
RevokedInfo ::= SEQUENCE {
 revocationTime       GeneralizedTime,
 revocationReason [0] EXPLICIT CRLReason OPTIONAL }
```

Figure 6.5: ASN.1 description of the E-MHT Response

The `BasicEMHTResponse` contains a `SingleEMHTResponse` for each target certificate included in the request and the `SingleEMHTResponse` is formed by two optional fields:

- The `emhtResponseData` which contains all the information necessary to check the status of a certain target certificate during the documented validity period.
- The `currentUpdateValue` is necessary to check the status of a target certificate whose $\mathcal{D}$igest has been signed prior to the current period (i.e. it is necessary to verify a re-utilization).

If the client has a cached `emhtResponseData` that can be updated with a re-utilization only the `currentUpdateValue` is included in the response, if not, an `emhtResponseData` and perhaps a `currentUpdateValue` are sent to the client. The `emhtResponseData` is formed by:

- The `signedTreeDigest` which includes the `issuer`, the `validityPeriod`, the `numberOfTrees`, the `treeNumber`, the `baseUpdateValue` and the `maxUpdates`.
- The `minorAdjacent`, actually, represents the target certificate if it has been revoked.
- The `majorAdjacent` is optional and it is included only when the target certificate has not been revoked.

Each `TreePath` is formed by:

- The `adjacentID` which is an unique identifier of a certain certificate.
- The `status` that contains the revocation date and reason.
- The `PathStep`(s) that allow to compute the H_{root} recursively.

6.4 Response verification

The client must check that the `TreePath` of the minor adjacent is correct, that is, that the `rootHash` computed from the $\mathcal{P}$ath matches the `rootHash` included in the $\mathcal{D}$igest.

If a target certificate has not been revoked, this is not enough, the client must also check the `TreePath` of the major adjacent. To do so if we are using optimized responses it is enough to reach a cryptographic value already computed in the verification of the $\mathcal{P}\text{ath}_{c_{minor}}$ (in the worst case H_{root} might be the only common cryptographic value between the two adjacent leaves).

When E-MHT is deployed with more than one MHT ($k > 1$ and $k = 2^m$), the client must check that the "m" least significative bits of each adjacent math the `treeNumber` included in the $\mathcal{D}$igest.

Finally, remember that the repository is a non-TTP, so the user can be misled into believing that a certain pair of nodes within the tree are adjacent leaves therefore the client also needs to ensure that the `TreePath`s provided belong to real adjacent nodes. This check is achieved by applying the adjacent node checking algorithm explained in the Chapter 5.

6.5 Security discussion

If the E-MHT responses are verified according to the procedure described in Section 6.4 the level of security of E-MHT is equivalent to the level of security of the AD-MHT (see Section 5.5).

6.6 Implementation of E-MHT in Cervantes

Figure 6.6 shows the behaviour of the SCH that manages E-MHT.

1. The EMHTPortThread listens for requests addressed to a certain TCP port.
2. For each request the EMHTPortThread creates an EMHTSocketThread.
3. The EMHTSocketThread receives the request.

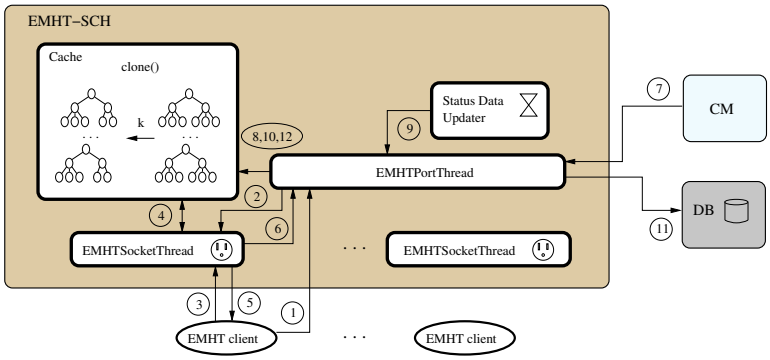


Figure 6.6: EMHT-SCH

4. The EMHTSocketThread retrieves the status data ($\mathcal{D}$igest and/or $\mathcal{P}$ath and/or `currentUpdateValue`) from the proper listening tree.
   The Cache stores $2k$ MHTs: k "listening trees" and k "management trees". Besides the trees, the cache stores the k `currentUpdateValues` (one per listening tree). The listening trees are used by the SCH to respond for status checking requests and they are immutable during the validity period "VP" of their $\mathcal{D}$igest. When a new expiration/revocation occurs, the proper management tree is updated. On the other hand, if after the validity period "VP" of a certain management tree, its $\mathcal{D}$igest has changed, this tree is cloned and the respective listening tree is replaced with the clone. Otherwise, a new `currentUpdateValue` is computed for the tree.

5. The EMHTSocketThread sends the response to the client.

6. The EMHTSocketThread informs the EMHTPortThread about the bytes and processing time required to serve the request.

7. The CM informs the EMHTPortThread about any change in the status data.

8. When the EMHTPortThread is informed about a change in the status data (revocation/expiration) it must update the proper management tree.

9. Every VP/k the StatusDataUpdater asks the EMHTPortThread to clone a management tree.

10. The EMHTPortThread checks if the $\mathcal{D}$igest of the involved management tree has changed. If the $\mathcal{D}$igest has changed the management tree is cloned and the corresponding listening tree is replaced with the clone. Otherwise (the management tree has not changed) the new `currentUpdateValue` for the tree is sent to the Cache.

11. If Cervantes is restarted and it must keep the revocation records from a previous execution, the EMHTPortThread has to build the "first" k MHTs based on these records. So it retrieves all the recorded status data from the database and builds these k listening trees. If Cervantes is started with an empty database, the listening trees will be also empty.

12. The EMHTSocketThread sends the first k listening trees to the Cache.

6.7 Evaluation

In this section we present a performance evaluation of E-MHT versus AD-MHT and OCSP in terms of downlink bandwidth utilization and we show what benefits E-MHT may provide. The experimental results have been obtained under the following conditions:

- The Cervantes server runs in a Pentium III (800 MHz).
- The clients generate 2 status checking requests per hour following an exponential probability density function.
- There are 10,000 clients and each client has a certificate.
- There is an average of 10% revocation.
- The test is configured with a database dynamism of one event (revocation/expiration) per hour and random revocations and random expirations are used.
- It is assumed that each client has a FAC of 50 certificates that take the 50% of the status checking requests.
- E-MHT is configured with 8 MHTs and 100 `maxUpdates`.

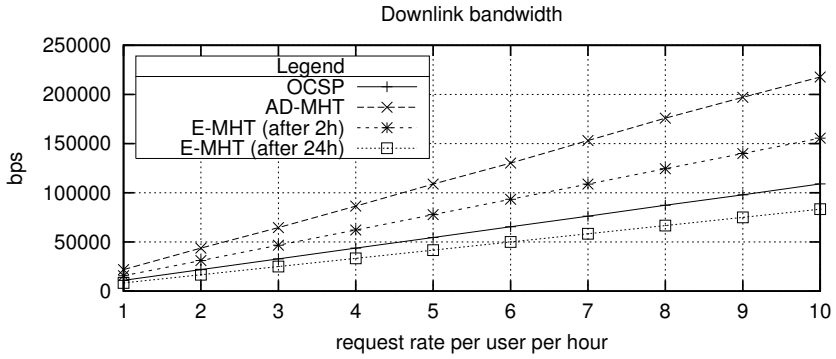


Figure 6.7: Downlink utilization for OCSP, AD-MHT and E-MHT status checking.

- The responses are cached by clients during their validity period.

It can be observed from Figure 6.7 that the AD-MHT is the system that requires the highest downlink bandwidth among the evaluated systems. On the other hand, the bandwidth used by E-MHT is measured after 2 hours and after 24 hours. Notice that after 2h the bandwidth reduction is mainly due to the division mechanism and the optimization of the MHT $\mathcal{P}$aths for non-revoked certificates (with this little time elapsed, the response re-utilization is practically not working). In this case, obviously the bandwidth required is higher than in OCSP, because without response re-utilization, MHT-based systems require always a higher bandwidth than OCSP. However, after 24h, the combination of the previous mechanisms and the response updating makes the E-MHT performance even better than OCSP.

6.8 Conclusions

This chapter introduces a new MHT-based revocation system named E-MHT. E-MHT agglutinates several mechanisms to enhance the efficiency of traditional MHT-based systems, such as CRT or AD. These mechanisms include the optimization of the MHT $\mathcal{P}$aths for non-revoked certificates, the division of the revoked certificates among multiple MHTs, the re-utilization of the tree $\mathcal{D}$igest and the cached

responses updating at a low cost.

E-MHT has been implemented as part of the Cervantes platform. As real revocation system, E-MHT requires additional elements beyond the merely definition of the data structures and mechanisms involved in. An important element to take into account is the protocol for status data retrieval. In this sense, a request/response protocol for the status checking has been defined in ASN.1. The main goal of E-MHT was to design an enhanced MHT-based system with laxer bandwidth requirements than the AD without deteriorating other aspects of the system. As shown in Figure 6.7 this goal has been by far achieved. Furthermore, under certain circumstances, E-MHT has proven to be even more efficient than OCSP due to the combined effect of the proposed mechanisms.

Chapter 7

Conclusions and Future Work

In this thesis we have presented a comprehensive survey and analysis of the main existing revocation schemes. This is important because understanding revocation is an important concern to both, PKI service providers and PKI end users. By better understanding the complexities of certificate revocation, either of these entities can improve their decision-making process by accounting for the great quantity of variables inherent in certificate revocation.

The certificate revocation represents one of the hardest scalability problems of the whole PKI, so this aspect is getting more and more crucial with the development of wide spread PKIs. There are studies that even argue that the running expenses of a PKI derives mainly from administering revocation. Thus, a revocation needs to be fast, efficient, timely and particularly appropriated for large infrastructures. Due to that, it is necessary e.g. to reduce the number of time-consuming calculations and to minimize the amount of data transmitted in the revocation system. In this respect, we have presented three new proposals to efficiently manage the certificate revocation. We briefly review them below.

The first proposal, called $\mathcal{H}$-OCSP, is a modification over the OCSP Standard. $\mathcal{H}$-OCSP is fully inter-operable with OCSP (OCSP clients can operate with an $\mathcal{H}$-OCSP responder and vice-versa). The point of $\mathcal{H}$-OCSP is that it reduces the processing burden in the responder and therefore the risk of running out of processing resources. As a result, an $\mathcal{H}$-OCSP responder is better protected against DoS attacks than a standard one. $\mathcal{H}$-OCSP is based on a hash chain to update

pre-produced responses at a low cost. Clients can also benefit from $\mathcal{H}$-OCSP: they can store the $\mathcal{H}$-OCSP responses of the most used certificates in their cache so that these responses can be later updated with little information and processing.

The second proposal, called AD-MHT, is based on the Merkle Hash Tree (MHT). AD-MHT uses a 2-3 tree to build the MHT. To our knowledge there are no published implementations of such a system so we have faced important open issues for implementing the system. These issues include how to respond to a request, how to revoke a certificate, how to delete an expired certificate, the communication protocol with the end users and the verification of a response. On the other hand, AD-MHT has proven to be resistant against malicious behaviors such as RDI masquerading, response modification, replay attacks or denial of service. As a conclusion the AD-MHT system might be a good choice for distribution of status data among end users taking into account the overall performance because it does not require much bandwidth or processing capacity, and repositories can be used to respond to status requests.

The third proposal, called E-MHT, is based on the previous proposal but we add some mechanisms to the basic data structures of AD-MHT that allow the E-MHT to provide a response size that is close to (or even better than) typical online systems such as OCSP without degradating other resources of the system. These mechanisms include the optimization of the $\mathcal{P}$aths for non-revoked certificates, the division of the revoked certificates among multiple MHTs, the re-utilization of the $\mathcal{D}$igests and the cached responses updating at a low cost.

Our proposals are not only a set of theoretical mechanisms but they are also practical systems that have been implemented inside a Java test-bed called Cervantes (Certificate Validation Test-bed). In our opinion a test-bed provides the most reliable and accurate way of performing evaluation. Moreover, performance evaluation of particular implementations is only possible with a test-bed. Cervantes is very flexible due to its modular design. Its design allows the platform to fit any kind of status checking protocol without significative changes in the structure or the source code of Cervantes. In particular we have shown how the two main standards (CRL and OCSP) have been implemented in Cervantes. Besides, $\mathcal{H}$-OCSP, AD-MHT and E-MHT have also been implemented in Cervantes. Finally, Cervantes has been used to obtain performance results about each system developed.

Expected results have been obtained for CRL and OCSP while our proposals have proven to be suitable for managing certificate revocation.

As future work, more development on Cervantes should be done[1] such as split the program of the server into different programs: one for the RDI and other programs for the responders and the repositories. This will imply developing efficient publication protocols for each status checking protocol.

On the other hand, efficient cache updating policies for terminals with reduced storing capacity should be developed for $\mathcal{H}$-OCSP and E-MHT.

The natural way of continue this thesis is to study the other topic involved in certificate validation: the certificate path validation. This topic remains an open field and important contributions in this area can be expected in short.

One approach to eliminating the certificate validation problems is to issue certificates with a very short life time. In this approach the CA automatically reissues the certificates for a defined number of periods or until a predefined time. The drawback of this approach is that distributing these certificates to many of the platforms where they will be used can be difficult and not user friendly since it requires involvement of the subscriber each time. This approach can also tax the CA or the TTPs involved if they are required to reissue certificates too frequently or for too many subscribers. However, this approach deserves a deep study.

Finally, the revocation mechanisms developed in this thesis could be extended to other environments such as mobile agent protection against malicious host behaviour. In this respect some work has been carried out in [ESMF03b, ESMF03a]. The platform proposed there aids to solve the problem of malicious hosts by using a TTP, the Host Revocation Authority (HoRA). The HoRA controls which are the hosts that acted maliciously in the past. The agent sender must consult the HoRA before sending an agent in order to remove from the agent's itinerary all the "revoked hosts". The HoRA can also revoke a malicious host if the agent sender detects and proves that this malicious host did not act honestly.

[1] Apart from making the platform more user-friendly :-)

Index

Bibliography

[AF99] C. Adams and S. Farrell. Internet X.509 Public Key Infrastructure Certificate Management Protocols, 1999. RFC 2510.

[AHU88] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1988.

[AJK+95] Andre Arnes, Mike Just, Svein J. Knapskog, Steve Lloyd, and Henk Meijer. Selecting revocation solutions for PKI. In *NORDSEC '95*, 1995.

[ALO98] W. Aiello, S. Lodha, and R. Ostrovsky. Fast digital identity revocation. In *Advances in Cryptology (CRYPTO98) Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[Coo99] D.A. Cooper. A model of certificate revocation. In *Fifteenth Annual Computer Security Applications Conference*, pages 256–264, 1999.

[DA99] T. Dierks and C. Allen. The TLS protocol version 1.0, 1999. RFC 2246.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. In *IEEE Transactions on Information Theory*, pages 644–654, 1976. IT-11(6).

[EBL+99] C. Ellison, B.Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI certificate theory, 1999. RFC 2693.

[EGM96] S. Even, O. Goldreich, and S. Micali. Online/offline signatures. *Journal of Criptology*, 9:35–67, 1996.

[Ell99] C. Ellison. SPKI requirements, 1999. RFC 2692.

[ESMF03a] O. Esparza, M. Soriano, J.L. Muñoz, and J. Forné. Host Revocation Authority: a Way of Protecting Mobile Agents from Malicious Hosts. In *Web Engineering*, volume 2722 of *LNCS*, pages 289–292. Springer-Verlag, July 2003.

[ESMF03b] O. Esparza, M. Soriano, J.L. Muñoz, and J. Forné. Protocols for Malicious Host Revocation. In *Information and Communications Security*, volume 2836 of *LNCS*, pages 191–201. Springer-Verlag, October 2003.

[FH02] S. Farrell and R. Housley. An Internet Attribute Certificate Profile for Authorization, 2002. RFC 3281.

[GGM00] I. Gassko, P.S. Gemmell, and P. MacKenzie. Efficient and fresh certification. In *International Workshop on Practice and Theory in Public Key Cryptography (PKC2000). Lecture Notes in Computer Science*, pages 342–353. Springer-Verlag, 2000.

[GT00] M. Goodrich and R. Tamassia. Efficient authenticated dictionaries with skip lists and commutative hashing. Technical report, Johns Hopkins Information Security Institute, 2000.

[HB99] P. Hallam-Baker. OCSP extensions, 1999. Expired IETF Internet Draft:draft-ietf-pkix-ocspx-00.txt.

[HFPS99] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile, 1999. RFC 2459.

[HMR01] A. Hagstrom, C. Michelsen, and D. Rowe. Cryptographic support for certificate revocation. Technical Report ECE636/INFT931, Secure Telecommunication Systems, 2001.

[ITU00] ITU/ISO Recommendation. X.509 Information Technology Open Systems Interconnection - The Directory: Autentication Frameworks, 2000. Technical Corrigendum.

[KAN00] H. Kikuchi, K. Abe, and S. Nakanishi. Performance evaluation of public-key certificate revocation system with balanced hash tree. In *Second International Workshop on Information Security (ISW 99)*, pages 103–117. Springer-Verlag, 2000.

[KAN01] H. Kikuchi, K. Abe, and S. Nakanishi. Certificate revocation protocol using k-Ary Hash Tree. *IEICE Transactions on Communications*, 8:2026–2032, 2001.

[Ken93] S. Kent. Privacy Enhancement for Internet Electronic Mail, Part II: Certificate-Based Key Management, 1993. RFC 1421.

[Koc98] P.C. Kocher. On certificate revocation and validation. In *International Conference on Financial Cryptography (FC98). Lecture Notes in Computer Science*, number 1465, pages 172–177, February 1998.

[Koh78] L.M. Kohnfelder. Towards a practical public-key cryptosystem. Master's thesis, MIT Laboratory for Computer Science, 1978.

[MAA00] M. Myers, R. Ankney, and C. Adams. Online Certificate Certificate Status Protocol, Version 2, 2000. Expired IETF Internet Draft. draft-ietf-pkix-ocspv2-00.txt.

[MAM+99] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP, 1999. RFC 2560.

[Mer89] R.C. Merkle. A certified digital signature. In *Advances in Cryptology (CRYPTO89). Lecture Notes in Computer Science*, number 435, pages 234–246. Springer-Verlag, 1989.

[MF] J.L. Muñoz and J. Forné. Design of a Certificate Revocation Platform. In *International Conference on Information Technology: Research and Education (ITRE 2003)*. IEEE Communications Society.

[MF02] J.L. Muñoz and J. Forné. Evaluation of Certificate Revocation Policies: OCSP vs. Overissued CRL. In *DEXA Workshops 2002. Work-*

shop on Trust and Privacy in Digital Business (TrustBus02), pages 511–515. IEEE Computer Society, September 2002.

[MFE+03] J.L. Muñoz, J. Forné, O. Esparza, I. Bernabe, and M. Soriano. Using OCSP to secure certificate-using transactions in m-commerce. In *Applied Cryptography and Network Security*, volume 2846 of *LNCS*, pages 280–292. Springer-Verlag, October 2003.

[MFES] J.L. Muñoz, J. Forné, O. Esparza, and M. Soriano. A Test-Bed for Certificate Revocation Policies. In *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM03)*.

[MFES03a] J.L. Muñoz, J. Forné, Oscar Esparza, and Miguel Soriano. Implementation of an Efficient Authenticated Dictionary for Certificate Revocation. In *The Eighth IEEE Symposium on Computers and Communications (ISCC'2003)*, volume 1, pages 238–243. IEEE Computer Society, June 2003.

[MFES03b] J.L. Muñoz, J. Forné, O. Esparza, and M. Soriano. A Certificate Status Checking Protocol for the Authenticated Dictionary. In *Computer Network Security*, volume 2776 of *LNCS*, pages 255–266. Springer-Verlag, September 2003.

[MFES03c] J.L. Muñoz, J. Forné, O. Esparza, and M. Soriano. E-MHT. An Efficient Protocol for Certificate Status Checking. In *Information Security Applications (WISA 2003)*, Lecture Notes in Computer Science. Springer-Verlag, 2003. (To appear).

[MFES03d] J.L. Muñoz, J. Forné, O. Esparza, and M. Soriano. Efficient Offline Certificate Revocation. In *Interactive Multimedia on Next Generation Networks*, volume 2899 of *LNCS*. Springer-Verlag, November 2003. (To appear).

[MFES03e] Jose L. Muñoz, J. Forné, O. Esparza, and M. Soriano. Certificate Revocation System Implementation Based on the Merkle Hash Tree. *International Journal of Information Security (IJIS)*, 2003. (To appear).

[MHHF02] A. Malpani, P. Hoffman, P. Housley, and T. Freeman. Simple Certification Validation Protocol (SCVP), December 2002. Expired IETF Internet Draft: draft-ietf-pkix-scvp-11.txt.

[Mic96] S. Micali. Efficient certificate revocation. Technical Report TM-542b, MIT Laboratory for Computer Science, 1996.

[Mic02] S. Micali. NOVOMODO. Scalable Certificate Validation and Simplified PKI Management. In *1st Annual PKI Research Workshop*, pages 15–25, 2002.

[MJ00] P. McDaniel and S. Jamin. Windowed Certificate Revocation. In *IEEE INFOCOM 2000*, pages 1406–1414, March 2000.

[Nik99] P. Nikander. *An Architecture for Authorization and Delegation in Distributed Object-Oriented Agent Systems*. PhD thesis, Helsinki University of Technology, 1999.

[NISa] NIST. Advanced Encryption Standard (AES). FIPS PUB 197.

[NISb] NIST. Data Encryption Standard (DES). FIPS PUB 46-1.

[NISc] NIST. Digital Signature Standard (DSS). FIPS PUB 186.

[NISd] NIST. Secure Hash Algorithm (SHA-1). FIPS PUB 180-1.

[NISe] NIST. Triple DES. FIPS PUB 46-3.

[NN00] M. Naor and K. Nissim. Certificate Revocation and Certificate Update. *IEEE Journal on Selected Areas in Communications*, 18(4):561–560, 2000.

[PFS02] W. Polk, W. Ford, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, 2002. RFC 3280.

[PH02] D. Pinkas and R. Housley. Delegated Path Validation and Delegated Path Discovery Protocol Requirements, 2002. RFC 3379.

[Pug90] William Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, June 1990.

[Riv92] R. Rivest. he MD5 Message-Digest Algorithm, 1992. RFC 1321.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. In *Communications of the ACM*, number 21, pages 120–126, 1978.

[SEF+] M. Soriano, O. Esparza, M. Fernandez, J. Forné, Jose L. Muñoz, and J. Pegueroles. Secure Brokerage Mechanisms for Mobile Electronic Commerce. *Electronic Commerce Research (ECR)*. (To appear).

[SNA] SNACC for JAVA. http://www.alphaworks.ibm.com/tech/snaccforjava.

[SP02] M. Soriano and D. Ponce. Security and usability proposal for mobile electronic commerce. *IEEE Communications Magazine*, 40:62–67, 2002.

[Stu95] Stuart Stubblebine. Recent-secure authentication: Enforcing revocation in distributed systems. In *IEEE Symposium on Research in Security and Privacy*, pages 224–234, May 1995.

[Woh00] P. Wohlmacher. Digital certificates: a survey of revocation methods. In *2000 ACM workshops on Multimedia*, pages 111–114. ACM Press, March 2000.

[X.588] CCITT Recommendation X.500. The directory overview of concepts, models and services, 1988.

[X.597] ITU/ISO Recommendation X.509. Information technology Open Systems Interconnection - The Directory: Public Key and Attribute Certificate Frameworks, 1997.

[X.695a] ITU-T Recommendation X.680. Abstract syntax notation one (asn.1): Specification of basic notation, 1995.

[X.695b] ITU-T Recommendation X.690. ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), 1995.

[Xen] Symeon Xenitellis. *The Open-source PKI Book*. http://ospkibook.sourceforge.net.