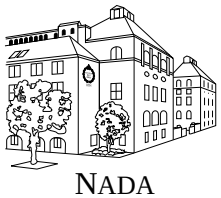
KUNGL
TEKNISKA
HÖGSKOLAN

Royal Institute of Technology
Dept. of Numerical Analysis and Computer Science

Network Application Security Using The Domain Name System

by
Simon Josefsson

TRITA-NA-E01107



NADA

Nada (Numerisk analys och datalogi)
KTH
100 44 Stockholm

Department of Numerical Analysis
and Computer Science
Royal Institute of Technology
SE-100 44 Stockholm, SWEDEN

Network Application Security Using The Domain Name System

by
Simon Josefsson

TRITA-NA-E01107

Master's Thesis in Computer Science (20 credits)
at the School of Matematisk-datalogisk linje,
Royal Institute of Technology year 2001
Supervisor at Nada was Mikael Goldmann
Examiner was Stefan Arnborg

Abstract

A major problem for a distributed security system is the management of cryptographic keys. Public key techniques are often used to overcome many of the problems. However, successful use of public key techniques in large systems such as the Internet requires a **certificate directory**, that is, a mechanism to locate and retrieve the public keys. In this thesis we explore how a common name lookup mechanism, the Domain Name System (DNS), can be used to provide this functionality. We show how the idea can be implemented in a secure mail application together with S/MIME. We compare the DNS lookup mechanism with traditional Directory Access Protocol based systems and identify weaknesses and strenghts. We also discuss and suggest a solution to privacy threats that arise because of recent security additions to the DNS, namely Secure DNS.

Säkerhet för nätverksapplikationer med Domännamnssystemet

Sammanfattning

Vid design av säkra distribuerade system är hanteringen av kryptografiska nycklar ett grundläggande problem. Publik-nyckel (PK) teknologi används ofta för att lösa många av dessa problem. För att PK-teknik ska vara praktiskt tillämpbart i stora system som t.ex. Internet krävs en **certifikatsbibliotekstjänst** som används för att lokalisera och hämta publika nycklar. Den här rapporten beskriver hur den vanliga namnuppslagningstjänsten, Domännamnssystemet (DNS), kan användas för att lösa det problemet. Vi visar hur DNS kan användas för att åstadkomma säker epost tillsammans med S/MIME. Vi jämför DNS med den traditionella bibliotekstjänsten som är baserad på Directory Access Protocol och identifierar fördelar och nackdelar. Avslutningsvis diskuterar vi, och föreslår en lösning på, hot mot personlig integritet; hot som är en följd av en nyligen förslagen säkerhetsutökning som kallas Secure DNS.

Preface

This thesis was presented to Stockholm University as partial fulfillment of the requirements for the degree of Master of Science in Computing Science.

The work was performed at RSA Security in Stockholm, Sweden. Supervisor at RSA Security was Magnus Nyström. Mikael Goldmann was supervisor at the Department of Numerical Analysis and Computer Science (NADA). Examiner was Stefan Arnborg.

Acknowledgements

I would like to thank my supervisors, Magnus Nyström and Mikael Goldmann, for advice and comments on my work, and their suggestions that helped to improve this report. All errors are of course my own.

The idea to use public key encryption of owner names in the Secure DNS "NO" record was suggested by Jonas Holmerin (the idea later developed into hashing).

This report was written in LaTeX [61] and illustrated with Dia [62]. Also, BibTeX, Emacs, ImageMagick and other free and open source software were instrumental to the creation of this document.

Contents

Preface **v**

Acknowledgements **vii**

Contents **ix**

List of Figures **xii**

List of Tables **xiii**

1 Introduction **1**
1.1 Outline of the Report 2

2 Background **3**
2.1 Cryptography 3
2.2 Internet and the Domain Name System 9
2.3 Public Key Infrastructure 11
2.4 Domain Name System 13
2.5 Electronic Messaging 14
2.5.1 Secure Electronic Messaging 14
2.5.2 Multipurpose Internet Mail Extension 15
2.5.3 Privacy Enhanced Mail 16
2.5.4 Pretty Good Privacy 17
2.5.5 Security Multiparts for MIME 17
2.5.6 Secure MIME 17

3 Use Cases **19**
3.1 Email Client 19
3.2 Certificate Publishing 24

4 LDAP and DNS as Certificate Directories **25**
4.1 Why Focus on LDAP and DNS? 25
4.1.1 How the Certificates are Used 26
4.1.2 How the Directory is Used 26

4.2 Locating Certificates . . . 26
4.2.1 Certificate Naming . . . 27
4.2.2 Lightweight Directory Access Protocol . . . 29
4.2.3 Domain Name System . . . 30
4.3 Updating Certificates in a Directory . . . 31
4.3.1 Updating in LDAP . . . 31
4.3.2 Updating in DNS . . . 32
4.3.3 Conclusions . . . 33
4.4 Performance and Overhead . . . 34
4.4.1 Caching in DNS and How it Affects Certificate Lookup . . . 34
4.4.2 The Domain Name System Protocol . . . 35
4.4.3 The Lightweight Directory Access Protocol . . . 36
4.4.4 Round Trips . . . 37
4.4.5 Packet Size . . . 40
4.4.6 Computer Resource Utilization . . . 42

5 DNS Security Considerations 45
5.1 Secure DNS . . . 46
5.1.1 Data Non-existence . . . 47
5.1.2 NXT Chaining . . . 49
5.2 Data Non-existence with Minimum Disclosure . . . 50
5.3 Implementing the Idea in DNS . . . 52

6 Conclusions 53

Bibliography 55

Index 60

Appendices 65

A NO Resource Records 65

B Sample Certificates 81

C Benchmarking Tool 89

List of Figures

2.1 Some basic cryptographic concepts 4
2.2 Simple key transfer 5
2.3 Digital Signature 6
2.4 A digital certificate 7
2.5 Secure key transfer 8
2.6 Brief example of the DNS hierarchy 10
2.7 Players of a PKI 11
2.8 Message Handling System Model 15
2.9 The PEM Public Key Infrastructure 16

3.1 A sample message 20
3.2 Selecting security functions from menu 20
3.3 Choosing the secure messaging technology to use 21
3.4 Select certificate source 21
3.5 Select encryption key to use 22
3.6 Query for more recipients 22
3.7 The original message tagged for encryption 23
3.8 Encrypted S/MIME message 23
3.9 Sample LDIF data 24
3.10 Corresponding DNS data 24

4.1 Example X.500 Directory 28
4.2 Update Certificate 31
4.3 DNS envelope 35
4.4 LDAP packet, with some structures expanded 36
4.5 Round Trip between two entities 37
4.6 Setting up a TCP connection 37
4.7 Tearing down a TCP connection 37
4.8 Round trips in a DNS Query over UDP 38
4.9 Round Trips in a DNS Query 38
4.10 Round trips in a LDAP Query 39
4.11 Bytes required to transfer a certificate with a 1024 bit RSA key with DNS and LDAP 43
4.12 Queries per second to look up a certificate 44

5.1 Naive data non-existence implementation 47
5.2 “NXT” Data-nonexistence implementation 50
5.3 Minimum information disclosure and data non-existence 51
5.4 Final example of how minimum information disclosure and data non-existence would work using NO records 52

B.1 512 bit RSA certificate 82
B.2 1024 bit RSA certificate 83
B.3 2048 bit RSA certificate 84
B.4 512 bit DSA certificate 85
B.5 1024 bit DSA certificate 86
B.6 VeriSign 1024 bit RSA certificate 87

List of Tables

4.1 Update operations supported in DNS and LDAP 33
4.2 Authentication support in DNS and LDAP 33
4.3 Number of round trips for a query using DNS and LDAP 40
4.4 Typical certificate sizes 40
4.5 Overhead of various layers 41
4.6 Bytes required to transfer a certificate that contains a 1024 bit RSA key with DNS and LDAP 42
4.7 Queries per second to look up a certificate 43

5.1 Example of (partial) DNS information for a zone josefsson.org . . 49
5.2 Example of non-existence proof data for data in table 5.1 49
5.3 Example of non-existence proof data for data in table 5.1 51

Chapter 1

Introduction

Secure communication is an increasingly important application of the Internet. Without secure communication many existing social functions cannot benefit from modern technology. The primary example is commerce. The foundation of secure communication is cryptography, which enables secure communication through the use of keys. The management of these keys has proven to be a problem when the technology is taken into use. So called public key cryptography solves several of these problems, in particular it allows the keys to be transfered, in the form of a certificate, through unprotected communication cannels.

The primary remaining problem in key management is a technical issue; namely how to locate the certificate for a certain machine or person.

Some basic requirements on the facility used to locate the certificate can easily be identified. It must be accessible everywhere. It should be practical, in the sense that it should not be extremely expensive or cause administrative hassle, to work. Preferably it should be distributed, because a central world-wide organization to store all certificates is not feasible to implement. However the facility does not need to be secure, altough if possible, it would create additional value.

So far our description is similar to how a facility for locating the address (instead of certificate), for a certain machine or person on the Internet, would work. The facility that implements this in today's world is called the Domain Name System (DNS). Our description also resembles the directory service X.500, and its more successful Internet protocol implementation which is called the Lightweight Directory Access Protocol (LDAP).

This thesis compare the Domain Name System and the Lightweight Directory Access Protocol for use as a certificate lookup service. In particular we focus on the application of secure electronic mail, used to send messages between persons using the Internet. We demonstrate that the idea of storing certificates in DNS is practical by building a prototype. We also discuss and propose solutions to a perceived privacy threat, introduced by recent additions to the Domain Name System protocol.

1.1 Outline of the Report

The report is outlined as follows. In chapter 2, we give an overview of, and a background to, Cryptography, Public Key Infrastructure (PKI), DNS and Secure Messaging, which is used throughout this report. In chapter 3 we demonstrate our implementation of a secure mail application and of a certificate publishing application. In chapter 4 we compare LDAP and DNS for certificate locating and retrieval purposes. In chapter 5 we discuss privacy threats due to Secure DNS and present a possible remedy. In the final chapter we present our conclusions and suggest topics for further investigations.

Chapter 2

Background

This chapter should give the reader an understanding of concepts used throughout this report. We first describe the foundation for our work, cryptography, and then describe the model we will be working in, namely Public Key Infrastructure (PKI). We proceed by discussing the Domain Name System, and how it relates to PKIs. We conclude by giving an overview of one major application of PKIs on the Internet, namely Secure Messaging or more specifically, secure email.

2.1 Cryptography

Cryptography is a enabler of secure communication. The word comes from the Greek words *kryptos* for hidden and *graphein* for writing. Cryptography is thus the science (or art) of "secret writing". The following is based on similar material from [89], [29] and [26].

When talking about cryptography, we refer to **senders** and **receivers** wishing to exchange **messages** or **plaintext** by exchanging **ciphertext**. It is assumed that an **eavesdropper** reading ciphertext should not be able to extract corresponding plaintext. This characteristic is called **confidentiality**. The process performed by a sender to hide plaintext is called **encryption**, the reverse operation is called **decryption**. These processes are often expressed as mathematic functions or computing algorithms. The encryption and decryption algorithms together constitute a **cipher**. Cipher algorithms intended for general use cannot be secret. So cannot the eavesdropper just invoke the decryption process to extract plaintext? Ciphers use **keys** to solve this problem. The key is used by the encryption process. A key is one element out of a large set of elements, the **key-space**. Figure 2.1 illustrates these concepts.

The usual mathematical description of ciphers uses E to the denote the encryption function and D to denote the decryption function. Plaintext is denoted by M and ciphertext by C. If the encryption or decryption functions are key dependent they are denoted by E_K and D_K respectively. Of course, E and D should have the property $D_K(E_K(M)) = M$. Ciphers where the encryption and decryption keys

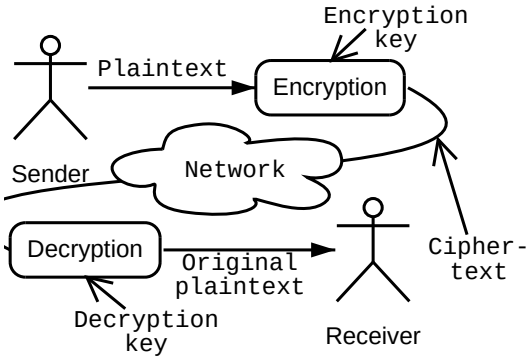


Figure 2.1. Some basic cryptographic concepts.

are equal are called **symmetric ciphers**. Such ciphers have the property that sender and receiver must have agreed on a certain key prior to the secure communication. Such keys are called **shared secret keys**. A good cipher should have the property that discovering the key, e.g., by inspecting ciphertext, should take an unreasonably large amount of time or be very expensive.

In sending and receiving messages, parties are often interested in three properties of the communication other than confidentiality. **Integrity** means that the sender and receiver should be able to verify that a message has not been modified in transit. As a consequence, this means that an intruder should not be able to substitue a false message for a legitimate one without being detected. **Authentication** means that the receiving party should be able to ascertain the origin of a message. **Nonrepudiation** means that the sender should not be able to falsely deny that she sent a message.

Using shared secret keys, it is possible to calculate **Integrity check values** from a message to achieve integrity. The integrity check value should depend on all bits of the plaintext. Should any bits of the message be changed between the sender and recipient, the recipient would calculate a different integrity check value. An adversary modifying a message might as well modify the check value too, but without knowledge of the secret key she cannot duplicate the correct integrity check value. If the receiver correctly verifies the integrity check value, she knows the message was generated by someone who knew the key. An important application of integrity check values is **Message Authentication Codes** [47] which use a symmetric block cipher (e.g., the Data Encryption Standard [91]). Integrity check values are also known as **Message Integrity Check**. Hash functions can also be used to provide an integrity check value, this mechanism is called a **Hashed Message Authentication Code** or a **keyed hash function** [25].

In [16] Diffie and Hellman introduced the concept of **public key** cryptography, independently invented by Merkle [68]. Such ciphers are also known as **asymmetric ciphers**. Unlike symmetric ciphers, a public key cipher uses two related keys - one for encryption and one for decryption. In addition to the requirements on symmetric keys, it should also be “infeasible” (be prohibitly expensive or take

large amounts of time) to learn the decryption key given the encryption key and/or ciphertext. The encryption (public) key can be made available to anyone who wishes to securely communicate with an entity. There is no longer a need for prior key arrangement between sender and receiver.

One of the first suggested and still most commonly used public key ciphers is the RSA cipher [88]. The security of the RSA cipher depends on the difficulty of finding the prime factorization of large integers. To describe how RSA works, we begin by noting how keys are generated. Two large prime numbers p and q (of roughly equal length) are chosen randomly. In practice, the size of p, q is recommended to be on the order of 100 decimal (non-zero) digits, or larger. The encryption key e is now chosen, randomly, such that e and $(p-1)(q-1)$ are relatively prime. The decryption key d is calculated as a inverse of e modulo $(p-1)(q-1)$, in other words by solving $ed \equiv 1 \pmod{(p-1)(q-1)}$ for d. Together e and n are the public key, and d is the private key. A plaintext m is encrypted to ciphertext c by simple modular exponentiation, $c = m^e \pmod{n}$. Decryption is performed by $m = c^d \pmod{n}$. This works since $c^d = (m^e)^d = m^{ed} = m^{k(p-1)(q-1)+1} = mm^{k(p-1)(q-1)} = \{$ by Fermat's small theorem $\} = m \cdot 1 = m$, all calculations modulo n.

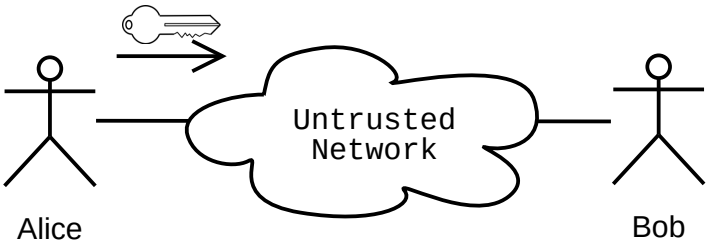


Figure 2.2. Simple key transfer. For symmetric keys, the communication must be private, integrity protected and authenticated. For asymmetric keys, the communication must be integrity protected and authenticated.

Public key ciphers place a large burden on users to somehow distribute these public keys. The straight forward method, by directly contacting the receiver and ask him for his key, is still possible. This is illustrated in figure 2.2. For symmetric keys this communication must be private (no-one can read the key), be integrity protected (no-one can modify the key) and authenticated (you know who you are talking to). For asymmetric keys, you only need integrity and authentication. The keys are, after all, public. In practice, this is a negligible advantage—current techniques to guarantee integrity and authentication also require keying material. This would create a chicken and egg problem. However, public key cryptography opens up for other forms of key distribution. Before venturing into this field we need more concepts to work with.

Written signatures are used as proof of authorship of a document, or at least proof of agreement with a document, such as a contract. Several equivalent electronic methods have been suggested. A **digital signature** is a piece of data which

accompanies a message. It is used to ascertain originator of message and the **integrity** of the message. Figure 2.3 illustrate how digital signatures are intended to operate.

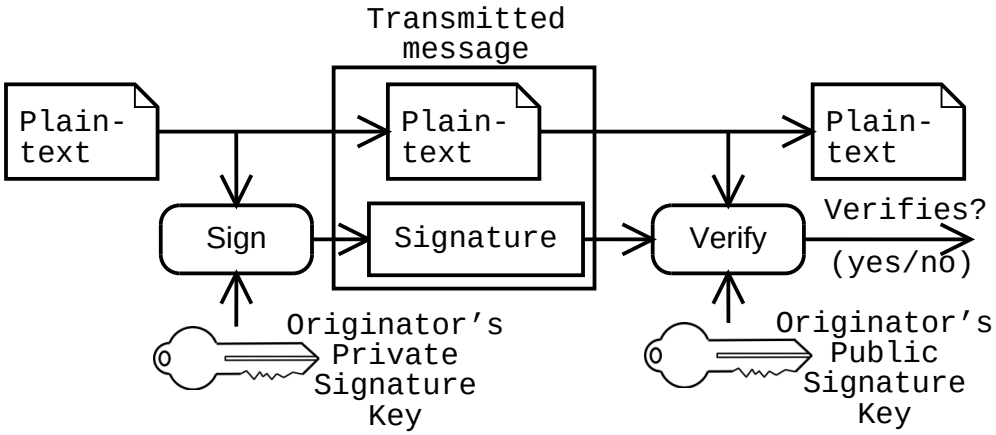


Figure 2.3. Digital Signature.

Some (but not all) public-key ciphers are able to operate as digital signature algorithms. RSA is able to operate in an **authentication mode** to provide digital signatures. The authentication mode is simply to use the private key for encryption and the public key for decryption, the inverse of regular use. This way, only the originator is able to encrypt (**sign**) the message, and everyone that knows the public key are able to decrypt (**verify**). To boost efficiency, digital signatures are rarely calculated on the entire input document but rather on a **one-way hash value** calculated from the document. Similar to ciphers, one-way hash functions are often expressed as mathematical functions or computing algorithms. One-way hash functions are one-way in the sense that they should be fast to compute but difficult to invert. In mathematical terms, calculating hash value h as $h = H(m)$ for a hash function H on a message m should be fast. But given h it should be infeasible to compute any member of the set $\{m|m = H^{-}1(h)\}$. h is uniformly distributed. Hash functions used cryptographically, for digital signatures, should have two properties:

- It must be computationally infeasible to construct an input message that hashes to a given digest.
- It must be computationally infeasible to construct two messages that hash to the same digest.

Hash functions usually produce output of a fixed length, commonly a few hundred bits. This is in contrast to documents, that may have arbitrary size. For RSA, the hash value is encrypted using the private key, which generates the signature. On the receiving end, the **verification procedure** is performed. The verification procedure consists of computing the hash value of the input document and comparing it with a decryption of the signature, using the public key of the sender. Again,

not all public-key ciphers are able to operate in this way to achieve signing, but we are satisfied in noting that other algorithms are able to achieve the same goals using other methods.

In the previous discussion, we have avoided to define what we mean by "computationally infeasible" or "hard", this is the topic of many text books on complexity theory and we refer to them (for example [37]).

Now the reader should have an understanding of techniques used to solve the problem that motivated this digresssion; namely how to distribute public keys in a secure fashion.

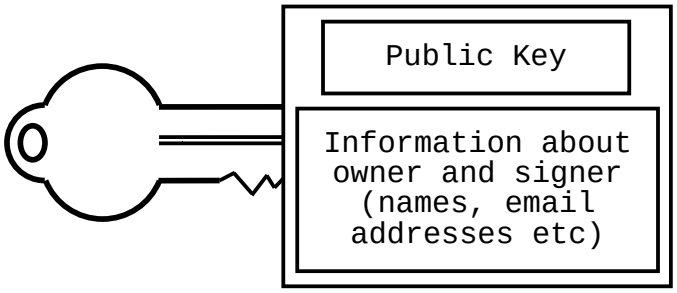


Figure 2.4. A digital certificate: digitally signed information containing a public key and some information related to that key.

Recall our discussion about direct key transfer between two entities from the beginning of this section, also illustrated in the figure 2.2. Unless this communication is integrity protected and authenticated (and encrypted in the case of symmetric keys), an adversary might be able to intercept and replace the key. Then Bob will believe the key he received belongs to Alice, and might use it to encrypt sensitive information intended for Alice's eyes only. This information might be decrypted and read by the adversary since he replaced Alice's key for his own. It might also be re-encrypted (with Alice's public key) and passed on to Alice, to reduce the risk of getting caught. This form of attack is known as a **man in the middle** attack. We will see that the use of public-key certificates is a practical method to solve this problem.

In [58] Kohnfelder introduced the notion of **public-key certificates**. A public-key certificate is a public key, digitally signed by a trustworthy entity. A certificate usually contains information about the owner and the signer, such as names or email addresses. Figure 2.4 illustrates this.

By having public keys digitally signed by a mutually trusted third party, all three problems with distributing keys are solved: Protecting the key, maintaining data integrity, and authenticating data. Since public-key technology is used, there is no need to protect (encrypt) the key. The certificate is digitally signed, and thus provides data integrity. By signing the certificate, it is later possible to authenticate the data contained in the certificate (i.e., the public key). To authentiate a certificate, the knowledge of the public key of the trusted party is required. Since this

mutually trusted third part can issues many certificates, an entity is able to strongly trust the validity of many public keys by knowing and trusting one public key.

By using the concept of certificates and a trusted third party, Bob can get Alice's public key without requiring an integrity protected and authenticated channel between Bob and Alice. Alice sends her certificate, issued by the trusted third party, to Bob. Bob knows the trusted third party's public key and is able to verify the signature of the certificate. He is then able to trust that the public key he received is correct and actually belongs to Alice. This is illustrated in figure 2.5. Of course, the man in the middle attack can now target the communication between the trusted third party and either Alice or Bob. Since this communication only takes place when certificates are issued or renewed, special care could be taken to protect that communication.

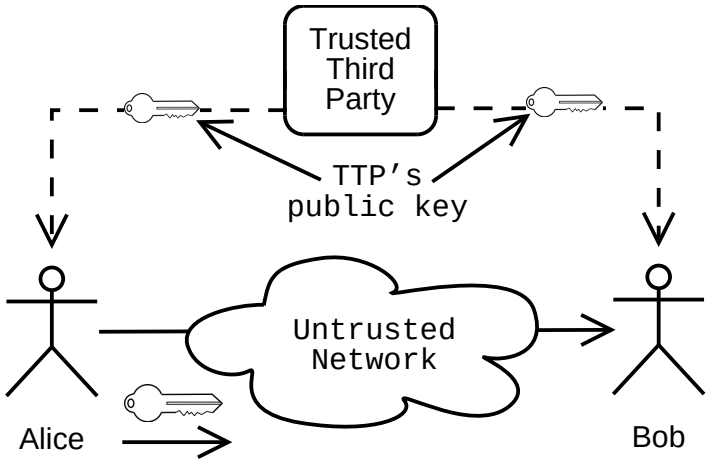


Figure 2.5. Secure key transfer. The key is actually a signed public key, a Certificate.

Using certificates also opens up for the possibility of removing the key transfer between Alice and Bob completely! This is an important feature when secure communication is being implemented between a large number of participants. This is because none of Bob's ability to trust the public key he receives now depends on whether he is actually communicating with Alice or not. His ability to trust the key depends solely upon his trust relationship with the trusted third party. This means that Bob might actually be talking to a database that stores certificates for many users, much like a phone book. Bob can use his trusted third party's public key to determine if he should trust a certificate or not.

Since the introduction of certificates, several standards have been suggested to implement this idea. The preveiling format is X.509 Certificates, as standardized by the International Standards Institute [11] in the late 1980's. The format has evolved, and has been profiled for use on the Internet by the Internet Engineering Task Force[1] (IETF). This profile is known as the Internet X.509 PKI (PKIX) pro-

[1]The IETF is a large open international community of network designers, operators, vendors, and

file. Still, to benefit from using certificates in applications, the issue of determining how the "phone book" should operate must be solved. This problem, and others, are solved under the framework of a **Public Key Infrastructure**. This thesis studies how the Domain Name System, a protocol already widely used on the Internet, can be used within a public key infrastructure to locate and retrieve certificates.

2.2 Internet and the Domain Name System

The Internet consists of loosely interconnected networks of computers located around the world. Computers communicate with each other by exchanging packets according to various protocols. Computers wishing to participate on the Internet need to follow the protocols used by other members of the Internet. The lowest level common protocol used on the Internet is named "Internet Protocol", often refered to as IP [82]. The addressing mechanism used by IP is similar to phone numbers. All entities that communicate on the Internet must have an IP address. An IP address may look like 195.42.214.244. Protocols are often layered on top of each other, to provide more specialized functions. For example, IP does not guarantee delivery of packets between entities. For those applications that require guaranteed packet delivery another protocol exists, layered on top of IP, that provides these functions. This protocol is known as Transmission Control Protocol (TCP) [83]. So how can TCP provide guaranteed packet delivery when TCP only uses IP, which does not provide guaranteed delivery? TCP accomplishes this by enumerating and acknowledging each packet. Using retransmission-timers TCP detects when acknowledgements are lost. If a packet (or acknowledgement) is lost, the packet will be re-transmitted until an acknowledgment is received. Many popular application protocols are layered on top of TCP. Examples include the Hypertext Transfer Protocol (HTTP) [28] for Web Browsing, the Simple Mail Transfer Protocol (SMTP) [13] and the Internet Mail Access Protocol (IMAP) [12] for Electronic Mail.

Using IP addresses to locate resources on the Internet has several problems. One of the most important problems is that IP addresses are hard to remember for humans. There is no obvious connection between real-world names of companies or persons that can be used to find the IP address. Returning to our phone number analogy, we observe that phone books often are used to collect phone numbers ordered by other information. They can be ordered by company name, personal names etc. If the same idea was used on the Internet, we would attach ordinary names to machines. It also must be possible to somehow convert this name into the actual IP address; this is the job for our Internet "phone book".

This problem was solved in the middle of the 1980's, and the solution is called the "Domain Name System" or DNS. The DNS organizes names of machines in a hierarchy. Universities may organize the names of their machines within a local

researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet.

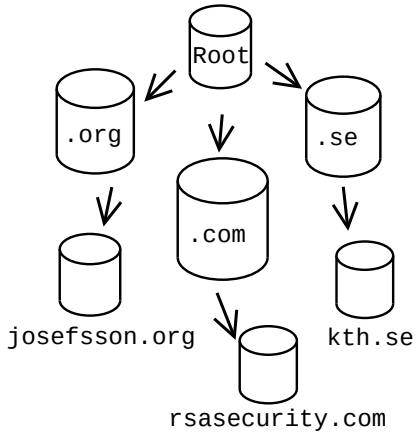


Figure 2.6. Brief example of the DNS hierarchy.

hierarchy such as departments, and the university itself may be located within the "educational hierarchy". Top-level hierarchies of the Domain Name System include "Educational", "Organizations", "Companies", "Millitary" and a hierarchy for each country around the world, such as "Sweden". The DNS uses shorter forms for brevity, "edu" for educational, "mil" for millitary, "se" for Sweden etc. Figure 2.6 illustrates a few members of the DNS hierarchy.

Because names are easier to remember than IP addresses, the Domain Name System hierarchy and the names stored in it are often used by application protocols—such as web browsing and electronic mail.

This last observation is important, and combined with the flexibility of DNS, is crucial to our work. Since the domain name hierarchy is used by many modern electronic applications, there are several advantages to being able to store related information about a domain name (such as a public key in the form of a certificate) in DNS. To see this, consider a separate infrastructure for locating certificates. This system must provide a global infrastructure, similar to how DNS works, so that it is possible to look up information from everywhere without any special knowledge of which server to query etc. Such attempts exist, and they often use their own form of addressing mechanisms. The prominent example is the ISO X.500 initiative. Now, if we wish to use another directory system with domain names (that is the usual address mechanism of the Internet) we depend on the possibility and success of locating and using domain names as an addressing mechanism within the other directory system. We will see that this is a complicated issue, and that no satisfactory solution exists to this date.

On the other hand, the Domain Name System provides the flexibility to allow us to store any data attached to a domain name. For example, it can attach "certificate" data to a domain name in the "phone book". If this is the case, we do not need to involve another directory to locate and retrieve certificates. In this report we will argue that storing application keying material, or certificates, in the Domain Name System is a promising idea.

2.3 Public Key Infrastructure

Public Key Infrastructures (PKI) consists of services required to make use of public-key based technologies on a large scale. The first service of a PKI that comes to mind is locating and retrieving certificates, but many other aspects constitute a larger part of actual PKIs. Non-technical aspects such as legal considerations are also a part of what makes up a PKI. We build on the concepts introduced in the last chapter, and continue by introducing the entities of a PKI. They are illustrated in figure 2.7.

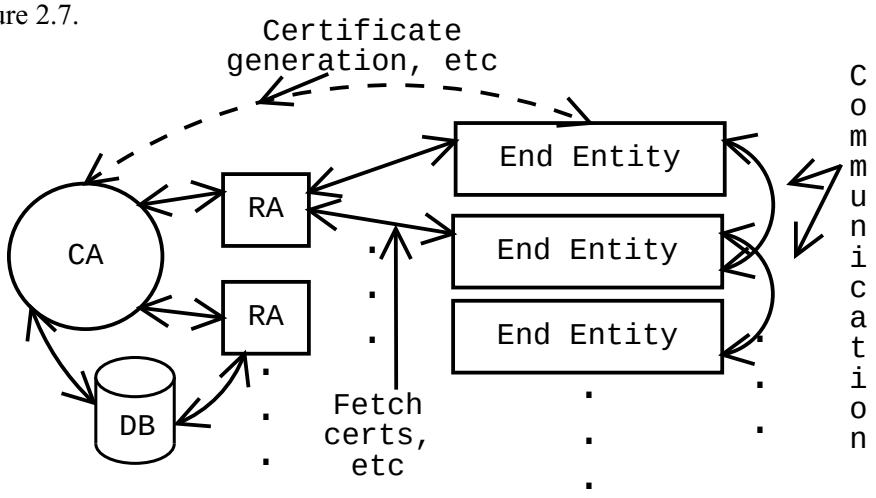


Figure 2.7. Players of a PKI.

- **Certificate Authority**

  The certificate authority, or CA, is the centre of a PKI. It issues certificates by signing public keys received from end entities together with information about the identity of the key owner, and stores the result in a database. A verification process is performed to make sure that the entity that applies for a certificate is who she claims to be.

- **Registration Authority**

  A registration authority (RA) is an optional component of a PKI. It takes over many obligations of the certificate authority. It is used to separate less security critical parts from a CA, to protect the critical parts in case of a security incident in the non-critical parts. The CA continues to sign certificates, but the registration authority may perform verification of user identities, certificate lookup and retrival etc.

- **End entities**

  End entities are the “users” of a PKI. Certificates are issued to end entities, and end entities communicate with each other securely using the certificates. End entities normally store their own private keys.

Some of the more important operations performed by the PKI entities include the following:

- **Key Generation**

  The keys used within a PKI could be created by any entity, but usually the end entity creates it using some equipment. The private part of the key must be kept protected, possibly by using some secure storage medium in the end entity.

- **Certificate Requests**

  After generating a key, an end entity usually requests a certificate to be issued by the certificate authority. This is done by sending a message to the CA (or the RA) containing the certificate request. The CA verifies the identity of the end entity and signs the public key.

- **Generating Certificates**

  After a successful certificate request, the certificate is created and stored with the CA and sent back to the entity applying for the certificate, possibly via the RA.

- **Updating Certificates**

  Certificates usually have a limited validity period, for example one month. Thus the operation of updating a certificate will be necessary on a regular interval. This is normally initiated by the end entity.

- **Revoking Certificates**

  The operation of revoking a certificate can be initiated by any of the players, depending on the reason for the revocation. End entities might revoke their certificate if they accidently reveal their private key. Certificate and registration authorities may revoke a certificate for various reasons, such as the end entity not fulfilling the requirements to be certified by that certificate authority. Information about revoked certificates can be distributed in so called **Certificate Revocation Lists (CRLs)**. CRLs are signed lists of revoked certificates. These lists are used by applications to verify that certificates are still valid. We will not consider the distribution or handling of CRLs further in this report, based on arguments similar to those presented in "Can we eliminate CRLs?" [87].

- **Publishing Certificates in Directories**

  To be useful in securing communications, certificates are often stored in public directories. These directories are used by end entities to look up and retrieve certificates of other end entities with whom they wish to communicate. Usually a certificate authority (or a registration authority) provides a directory service containing all certificates it has issued. However, end

entities may choose to publish their certificates in other directories as well. Various directory technologies exist, and we will concentrate on two: The Lightweight Directory Access Protocol and The Domain Name System.

2.4 Domain Name System

As mentioned, the DNS "phone book" looks like a hierarchical system. This is also reflected in how the actual database which holds the information is implemented. Instead of having a big database containing answers to all queries, DNS works by delegating the responsibility, or "authority", for each hierarchical component (i.e., a subtree of the DNS hierarchical tree, also called a "zone") to the servers that should be responsible for that component. These servers can further sub-delegate authority, resulting in a distributed database based on a tree topology.

To see how this works, consider looking up some piece of data attached to a DNS domain name. In our example we will look up the IP address attached to the domain name **www.nada.kth.se**. We begin with a somewhat simplified description. A client that wishes to look up a name must know the address to the "root" servers. The "root" servers are the servers located at the top of the DNS tree.[2] The client sends a query for **www.nada.kth.se** to the root server, and usually the root server only knows who is responsible for the next sub-component in the query. In our example, this means the root server is only able to tell the client the addresses of the servers responsible for **se**. The client will now forward the same query to the **se** servers, and in our example these servers do not know the answer to the query either, but have delegated the authority over the **kth.se** zone to certain servers and it informs the client of their addresses. The client repeats this procedure, asking **kth.se**, and this time it receives addresses to the servers responsible for **nada.kth.se**. As it happens, this server will know the correct answer and construct a response and sends it to the client. The client has now received the IP address of **www.nada.kth.se**.

In our example, we have made some simplifications. Specifically, if the DNS system worked as we just described it is not difficult to see that the root servers would receive an enormous amount of traffic. In practice this problem is solved in two ways. First, the "clients" in our example will cache all answers it receives. This means that once it has received the addresses for "se" from the root server it is able to query these servers directly for addresses within the "se" zone in the future.[3] Secondly, the "clients" are usually not applications or even individual workstations but rather a server located close to the workstation which usually serves many workstations. The server aggregates the DNS needs of many workstations in a smaller environment, such as a department. This means the local server holds a

[2] Currently there are 13 root servers located around the world. Advanced clients measure the round-trip time when sending queries to servers, and are thus able to select the closest server in the network topology. This increases performance and decreases network traffic.

[3] Of course, caching introduce the problem of stale data. DNS solves this by attaching a "Time To Live" value to each answer, indicating how many seconds the receiver is allowed to cache the data.

cache for several workstations at once. Since usage patterns on workstations often are quite similar this reduces traffic. Also, since the server is not re-started or turned off as frequently as workstations often are, the cache will be more effective.

We will see in more detail what the actual protocol that implements this looks like in section 4.4.2.

2.5 Electronic Messaging

Electronic Messaging is a conceptually well-known and well-understood application. The diversity of existing electronic messaging clients is, not surprisingly, very large. Many different classes of messaging exists, with many competeting technologies in each class.

The classic example is **Electronic Mail** (or **email**). While there exist several implementations such as X.400 and MEMO, the Internet standard for text messages [13] and its usual transport mechanism [84] is the dominating implementation. Other classes of messaging include **Instant Messaging**. Instant Messaging is a real-time short message service between two entities, well-known applications include the Short Message Service (SMS) on GSM Digital Mobile Phones, ICQ [46] and America On Line's AIM [2] on desktop computers. A variation of Instant Messaging is **Public Chat Groups**, which is a real-time short message service between several, often near-anonymous entities. The largest implementation of Public Chat Groups is IRC [48]. Another class is **Public Discussion Forums**, with the prominent example of Usenet [40], but still other implementations exist such as Fidonet [27] and KOM [80].

The previous discussion of Electronic Messaging technologies is far from exhaustive. The intention is to give the reader a sense of the diversity that exists in this field.

2.5.1 Secure Electronic Messaging

None of the previously mentioned messaging technologies has been designed with strong security in mind. Other features are often thought to be of more importance in the design phase. In order to accommodate professional use of messaging technology, security extensions are critical. Several security extensions for Internet Mail have been proposed, such as PEM, MSP [17], (Open-)PGP, Security Multiparts for MIME, MOSS [14], PGP/MIME and S/MIME. Before we go on and discuss some of these security extensions, we need to establish a vocabulary. It is used throughout this report.

The terminology used when talking about messaging is due to the OSI X.400 Message Handling System Model [69], the following description is due to [29, page 154] and is illustrated in figure 2.8. Messages originate from and are ultimately received by **Users**, which may be people or mail-enabled application programs. A message has one **Originator** and one or more **Recipients**. A user is

supported by software called a **User Agent**, which performs such tasks as preparing and submitting messages for its user, and receiving and preprocessing received messages for its user. A User Agent may be a stand alone software application (sometimes called a **Mailer**), or it may be integrated into another application such as a Web Browser. The message transfer backbone comprises systems called **Message Transfer Agents** (MTAs). A message is submitted at an **originating MTA**, which delivers it to a recipient user agent. MTAs may be store-and-forward message switches of a given messaging technology, or they may be mail gateways between different technologies.

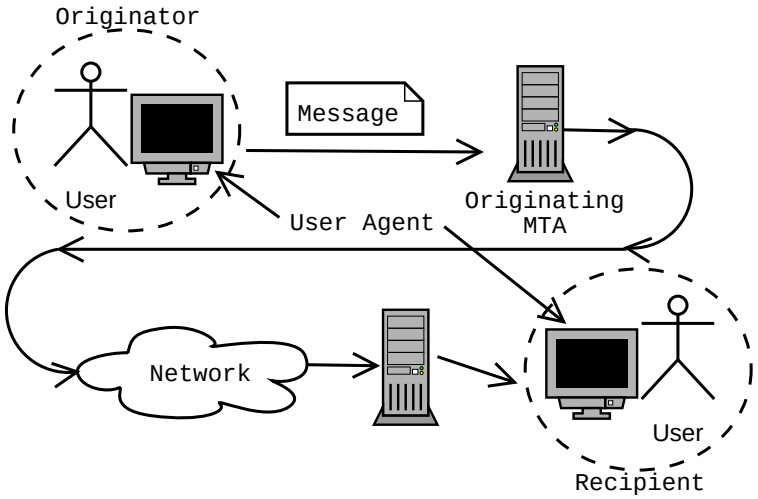


Figure 2.8. Message Handling System Model.

2.5.2 Multipurpose Internet Mail Extension

Several Secure Electronic Mail proposals, and most of the successful ones, are related to the Multipurpose Internet Mail Extensions (MIME). MIME is described in the five-part standard suite [31] [32] [72] [33] and [30]. MIME is a framework that extends the one-dimensional Internet Mail Message Format known as RFC 822 [13]. This traditional format has several drawbacks:

- It only allows for the U.S. character set.
- It does not allow for non-textual components, such as images, to be included in mail messages.
- It does not allow for structured documents, such as a mail message with one text part and one image part.

These concerns were the driving force behind development of MIME. MIME has been a success. Many standards, some even unrelated to mail, uses parts of the MIME standard. Examples include the Hypertext Transfer Protocol (HTTP) [7] and Internet Mail Access Protocol (IMAP) [12].

2.5.3 Privacy Enhanced Mail

The following overview of PEM is based on similar material from [29], [9], [26].

Privacy Enhanced Mail (PEM) was the first serious effort to secure Internet mail. The Internet Resources Task Force (IRTF) Privacy and Security Research Group (PSRG) did the initial design. The Internet Engineering Task Force (IETF) PEM Working Group continued development for three years, resulting in a four-part Proposed Internet Standard published in early 1993 [64] [56] [5] [55]. PEM is a broad standard suite, it provides encryption, authentication, message integrity and key management. PEM supports both symmetric and asymmetric (public-key) key management schemes. PEM uses DES for encryption, MD2 or MD5 for authentication and X.509v1 with RSA for public-key management. The standard also allows for different suites of algorithms to be defined later. PEM is designed to be taken into use selectively, by site or by user, without affecting other parts of the network.

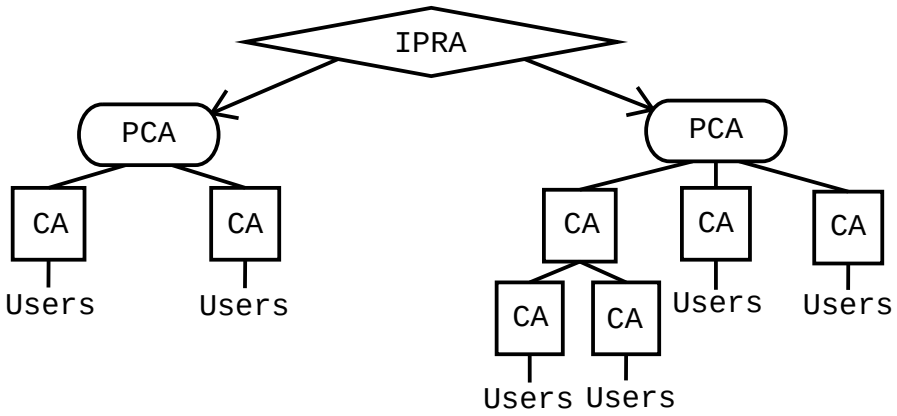


Figure 2.9. The PEM Public Key Infrastructure.

Even though PEM is a landmark protocol in the development of secure messaging, and is also generally considered to be of sound technical design [29], it did not catch on. This was mainly due to two reasons. First, the message syntax that PEM describes was incompatible with the widely successful MIME message syntax that emerged at the same time [29, p. 156]. Secondly, the public-key management described by PEM restricted the Certificate structure [9, p. 51]. Namely, it required a top-down Certificate Authority (CA) approach. An entity, the Internet Policy Registration Authority (IPRA), establishes global certification policies by certifying Policy Certification Authorities (PCAs). Each PCA in turn certifies CAs that will follow the certificate policy of that PCA. This hierarchy is illustrated in figure 2.9. A strict hierarchical approach works well in strict hierarchical organizations, but this is a feature that the Internet lacks. Two other complications with the public-key management approach also turned out to cause problems. First, PEM required the IPRA to maintain a database of unique Distinguished Name (DN), that all PCAs were supposed to query before certifying Certificate Authorities. Secondly, the

X.509 version 1 certificate format does not contain fields for certificate policies, forcing all applications to be aware of PCA policies by other means, which PEM did not provide for.

2.5.4 Pretty Good Privacy

Pretty Good Privacy (PGP) was developed during the same period as PEM, in the early 1990's. PGP was originally designed for securing Internet mail. PGP shares most technical features, such as digital signatures and public-key based encryption, with PEM. Like PEM it uses a proprietary, non-MIME-compatible, message format [3]. However, later MIME-compatible variations have evolved [22]. PGP's main difference from other proposals is its key management system. It does not use X.509 Certificates, but rather a proprietary syntax. Also, it uses a non-hierarchical certification model known as "web of trust". We will not study PGP further, a good reference is [98], and an account of PGP History can be found in [4].

2.5.5 Security Multiparts for MIME

Security Multiparts for MIME [35] is a simple framework for adding security enhancement to Internet Email by using MIME. Basicly, it describes how you combine a text message (or other data) with cryptographic information. It does not describe the cryptographic operations themselves, that is left for other specifications. It has gained wide popularity. Security Multiparts for MIME is used by at least three protocols; MOSS [14], PGP/MIME [22] and S/MIME, of which the latter two have been successfully deployed and are widely used today.

2.5.6 Secure MIME

Secure MIME (S/MIME) [86] [85] combines the previously mentioned Security Multiparts for MIME framework with the Cryptographic Message Syntax (CMS) [44] standard. CMS is derived from the Public-Key Cryptographic Standards 7 (PKCS#7) [60]. The differences between PKCS #7 and CMS are minor. Added features to CMS include support for key agreement techniques. As stated in the previous section, the Security Multipart framework does not define cryptographic operations or specific cryptographic message syntax. S/MIME supports both signed and encrypted messages. We now turn to a brief overview of the CMS standard because an S/MIME messages essentially is a CMS messages.

CMS defines a syntax for data that has cryptographic operations applied to it. Cryptographic operations include digital signatures and encryption. The syntax is described using OSI Abstract Syntax Notation One (ASN.1) [49], a language often used to describe data structures. The details of how ASN.1 works are not essential here, a good introduction to ASN.1 can be found in [54]. Returning to CMS, it describes how plain text (a file or network stream) is wrapped into data, after signing or encryption operations have been performed. The data structure contains

information used by the receiver to understand what treatment the data has been subjected to. This enables the receiver to restore the original data, and to properly verify or decrypt the content. The CMS format is compatible with the message format used in PEM, in the sense that CMS messages can be converted from and to PEM messages without any cryptographic operations. CMS does not require a certain key management procedure or a special security infrastructure. Further, CMS can be used either inside a public key infrastructure, or in an infrastructure using shared symmetric keys.

Chapter 3

Use Cases

This chapter describe important use cases when certificates stored in DNS can be used, and where DNS provides additional advantages over traditional certificate directories such as LDAP.

3.1 Email Client

Electronic mail on the Internet has been an important form of communication for some years now. Various methods of securing electronic mail have been suggested (see our overview in section 2.5). One of the solutions that has seen commercial success is S/MIME. A major problem in using S/MIME to secure mail is locating certificates. Clients often support several methods to locate certificates, e.g., from LDAP servers or from white-pages on the Internet. Most of these services have the drawback of requiring configuration by the user; configuration of the LDAP server hostname, the address to the whitepage service, of the LDAP base objects, etc.

DNS provides a solution to this problem. DNS is already used as a ubiquitous lookup service to look up mail exchangers and their IP addresses, something all email application need to do. Thus DNS is already an integral part of email applications today. The costs of adding support in the application to be able to look up other data are small. And unlike LDAP, additional configuration by the user will not be required.

We have implemented S/MIME functionality in a email client, with certificate fetching from DNS, as a proof of concept. The implementation is based on the extensible mail client Gnus [90]. The following figures illustrate how a user "Some One" might send an encrypted email to "simon@josefsson.org". Unlike other applications, no prior configuration of the recipient's certificate, of LDAP servers, or similar parameters are required. Signing is possible, but less complicated, so we omit that description.

Figure 3.1 shows a message window. The user has typed in the recipient's address and some text.

Figure 3.1. A sample message.

Now the user wishes to encrypt this message, figure 3.2 shows how the encryption function is chosen from the menu.

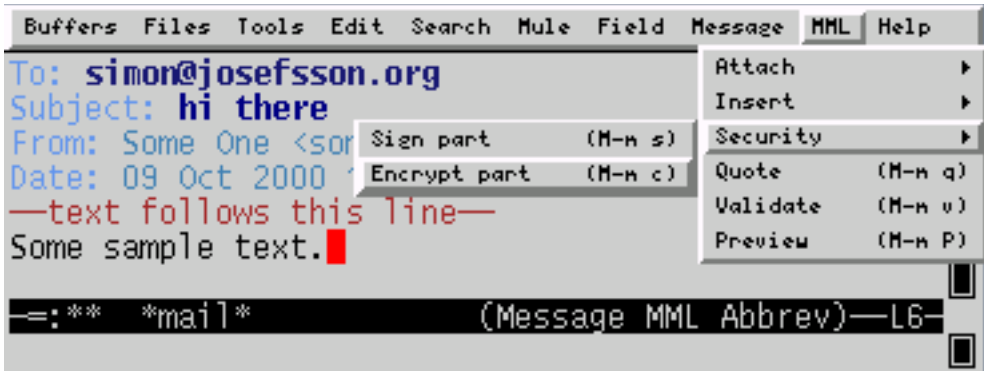


Figure 3.2. Selecting security functions from menu.

The next step, shown in figure 3.3, illustrates how a user selects what encryption technology to use. One of the few alternatives to S/MIME that are in current

use, is PGP/MIME. It might be available as an option here. This step might be removed for novice users.

Figure 3.3. Choosing the secure messaging technology to use.

Now the client needs to locate public keys used to encrypt the message. There might be a number of sources available: DNS, an LDAP server, a local file, etc. See figure 3.4. This step might also be removed for novice users, by making the client automatically try various sources in order. First try to locate the certificate in DNS, then from preconfigured LDAP servers (if any), then query the user for a filename, etc.

Figure 3.4. Select certificate source.

After selecting DNS as the source for certificates, it is possible to chose whom the message should be encrypted to. Normally, you would encrypt a message to the same person you are sending it to, but there may be situations when you need to use a middle-man. The intended receiver of the encrypted message is queried by the client in figure 3.5. This is also a somewhat esoteric feature. It should be

removed for novice users. The recipient of the mail should be used as the receiver of the encrypted message.

Figure 3.5. Select encryption key to use.

A message can be encrypted to more than one person, and the client queries the user if she wishes to add more recipients in figure 3.6. Likewise, this could be removed for novice users, only using whatever recipients are in the message.

Figure 3.6. Query for more recipients.

By answering “n” (for “no”) to the previous question, the client is finished and inserts tags into the message. The mark-up language is called MIME Meta Language (MML), and is used to mark-up MIME articles. It is an SGML/XML based language. Figure 3.7 shows MML tags added to the message.

Figure 3.7. The original message tagged for encryption.

The article is now ready to be sent, encrypted. To illustrate what the encrypted content looks like, it is possible to preview the message. This is done by selecting "Preview" from the menu shown in figure 3.2. The resulting display is shown in figure 3.8.

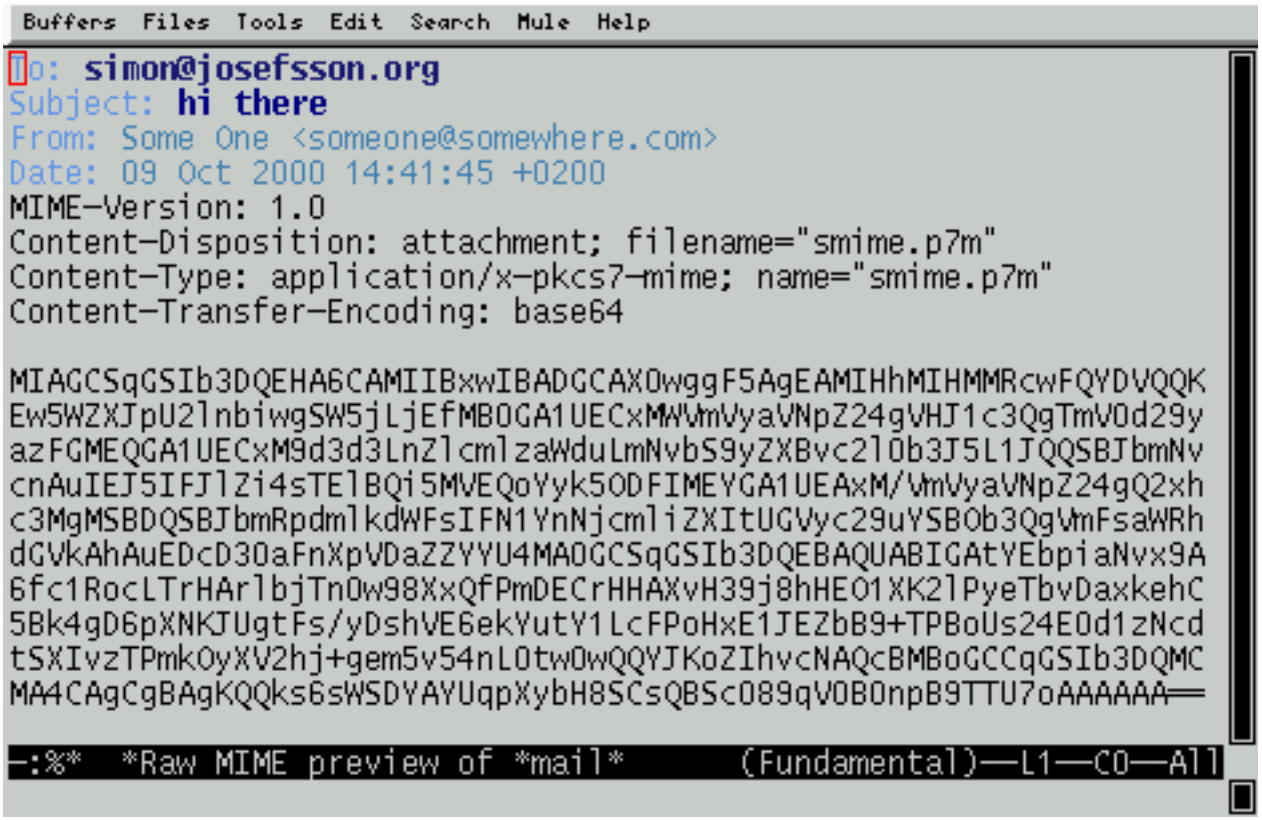


Figure 3.8. Encrypted S/MIME message.

3.2 Certificate Publishing

The previous email client retrieved certificates from DNS to secure email. Of course, someone needs to publish certificates in DNS for this to be possible. Traditionally, publishing has often been a service of the trusted third party, where the CA provides an LDAP interface to look up certificates (only issued by that CA, naturally). To publish certificates in DNS we have developed a proof-of-concept PKI utility with the following features.

- Generate cryptographic keys (e.g., RSA), generate PKIX certificate requests and sign PKIX certificates.
- Convert PKIX certificates into the LDAP Data Interchange Format (LDIF). This format is used to represent LDAP data in ASCII format.
- Convert data in LDIF format to the ASCII representation of DNS CERT resource records. This data is read by DNS servers.

This allowed us to experiment with the various elements of a PKI, and specifically to experiment with storing PKIX certificates in DNS. Data in DNS is stored in text files using a special syntax, these files are called "zone files". The syntax is a simple textual encoding of the various elements in the DNS protocol (see section 4.4.2). To illustrate how a certificate in LDIF format, figure 3.9, is converted into a DNS resource records for storage in a DNS zone file, we show the output in figure 3.10. The process to convert the data does not require any cryptographic operations, only a Base 64 [51] decoding, insertion of a identifying byte sequence [21, section 2.3], and a Base 64 encoding. One of the three integers in the DNS figure are used to identify the data as being a PKIX certificate (the other two are not used).

```
dn: cn=User 0, dc=josefsson, dc=org
cn: User 0
objectClass: pkiUser
mail: user0@josefsson.org
userCertificate;binary:: MIICAzCCAa2gAwIBAgIBATANBgkqhkiG9w...
```

Figure 3.9. Sample LDIF data.

```
user0.josefsson.org. IN CERT 1 0 0 A1UEJDCCAgMwggGtoAMCAQIC...
```

Figure 3.10. Corresponding DNS data.

Chapter 4

LDAP and DNS as Certificate Directories

This chapter extensively compare the model and implementation of the LDAP and the DNS, for use as a certificate directory.

4.1 Why Focus on LDAP and DNS?

LDAP and DNS are not the only solutions available. We first motivate our decision to compare these two. Our study concentrates on the use of PKIs in global, open network such as the Internet. Our basic needs are to be able to, interactively or automatically, look up application keying material or certificates using email addresses or hostnames as search keys. We do not need any additional cryptographic or security operations in the lookup service. Usually not even complex search capabilities are needed. Alternative protocols such as HTTP and FTP [45] do not meet our first requirement. However, both LDAP and DNS can meet this requirement.

Another way to see why LDAP and DNS are the two most relevant choices to consider is by looking at the currently most used certificate implementation on the Internet, the X.509-derived PKIX. X.509 certificates were designed to be used with the X.500 directory service. LDAP can be seen as "X.500 for the Internet". It is based on the X.500 protocol and the X.500 directory service model, but updated for use in the Internet. Thus LDAP is a good candidate for our study. X.509 certificates are often used by Internet hosts and Internet email users. DNS, via its security extensions, tries to solve similar goals, e.g., how to attach a public key to a host or email user. However, the security extensions of DNS have not been as successful as X.509 certificates for this purpose to this date. Thus, since X.509 certificates are used by entities addressed in the DNS model, we chose to study how to distribute the certificates through the same system. It is an appealing thought to combine a successful certificate format (X.509) with a less successful directory service (X.500), with a successful name lookup service (DNS) with a to this date rarely adopted security service (Secure DNS).

4.1.1 How the Certificates are Used

Certificates may be used in very different environments, from proprietary solutions within small companies with a handful of entities that need a security service, to world wide networks consisting of a multitude of protocols and applications without any organization in full control of everything. Different environments place different restrictions on what services are expected and needed. The environments we consider share the following characteristics:

- Certificates will be looked up frequently when needed, as opposed to being stored at clients. Thus protocol considerations such as size and overhead become important. (This is especially true for mobile applications.)
- The directory services should support identifying a certificate by using Internet addresses such as email addresses. This is in contrast with the world X.509 was developed for, where the messaging system used the same addressing scheme as the X.509 certificates.
- Certificate extensions need to be documented and well known, to work in a open network.

We see that PKIX certificates stored in LDAP or DNS systems fulfill our needs.

4.1.2 How the Directory is Used

Just like certificates, directories can be used in a variety of ways. Again, in open networks certain characteristics are interesting and others are not. For example, being able to locate and retrieve a certificate for a given email address is very important. Being able to list certificates for people named Adam is not very interesting for two reasons. First, it can be abused to collect information. Second, it does not scale; there are millions of Adams out there. More complex queries are, where deemed necessary at all, better handled by more specialized agents in local environments that need them (e.g., a company wide phone book).

Conclusion 1 *The primary use of the directory services we study is fetching one certificate at a time. The queries consist of a complete email address, a domain name, a URL etc, rather than more complex queries.*

Even though one of our protocols, LDAP, provides much more complex directory operations than DNS, we will not consider most of them for this reason. Both DNS and LDAP satisfy our needs here as well.

4.2 Locating Certificates

Perhaps the most important service of a Public Key Infrastructures in open networks (such as the Internet) is locating certificates. The problem is easy to understand; if you want to send encrypted email to **kalle@josefsson.org**, how would

you get the keying material (his certificate)? If you want to establish a secure IP connection to **www.seb.se** using PKIX Certificates, how would you get their certificate?

There is no single generic solution to this problem today. One widespread solution for the email case, are large "white-pages" accessible through the world wide web [97] [92] [75]. A whitepage service is a centrally controlled directory of information, much like a phone book. Adding, deleting and updating information is done by contacting the service provider. This approach lacks several important features though, including:

- **Up-to-date information**

  Since organizations are not in direct control of their own data, there will be a delay when data is changed.

- **Data integrity**

  Most white-page services do not provide any means to protect the integrity of data. This means that it might be possible for an adversary to modify data at the white-page service, or in transit to or from the white-page service.

- **Data origin authentication**

  Most white-page services lack guarantees of the origin of data. Thus you do not know if the information you learn about X is what X, and no-one else, actually wants to have published.

- **Administrative requirements**

  For security related material, many organizations would prefer to be in control of what kind of information is published, and not to rely on an external partner.

This makes whitepage services unsuitable for storage of security sensitive information, such as certificates. Some of these, and other, problems can be solved by using directory technologies such as LDAP and DNS. However, before data can be located using these protocols, we must have a mean to identify data. The next section discusses this issue.

4.2.1 Certificate Naming

PKIX certificates are based on X.509 certificates. X.509 certificates were developed to secure X.500 directory systems. However, these certificates are not restricted for use only within the X.500 directory system. This is fortunate, as the X.500 directory system itself has not been successful (at least compared to information stored on the Internet). PKIX is an effort to make the original X.509 certificate more suitable for use on the Internet. Since X.509 defines an identity

based certificate[1] these certificates must include names of entities. Entities include certificate owners and certificate issuers. Version 1 and 2 of X.509 use X.500 names exclusively to identify these entities. An understanding of names used in X.509 certificates thus requires a basic understanding of the X.500 directory service, which we now present.

The X.500 directory is much like a telephone directory. Given a name, you can find information stored under this name. Data in an X.500 directory consists of a set of **entries** each having an unambiguous **Distinguished name** (DN). Each entry represents a real-world object, such as a person, an organization, a computer, or a nation. The directory entries consist of **attributes** that have a **type** and one or more **values**. An entry can contain several attributes, such as phone number and addresses. Types are often abbreviated, "C" for country, "O" for organization, "CN" for common name, etc. All entries are stored in a tree-like structure called the **Directory Information Tree** (DIT), where each level in the hierarchy can introduce a **Relative Distinguished name** that, when combined, form the **Distinguished name**. Figure 4.1 illustrate these concepts.

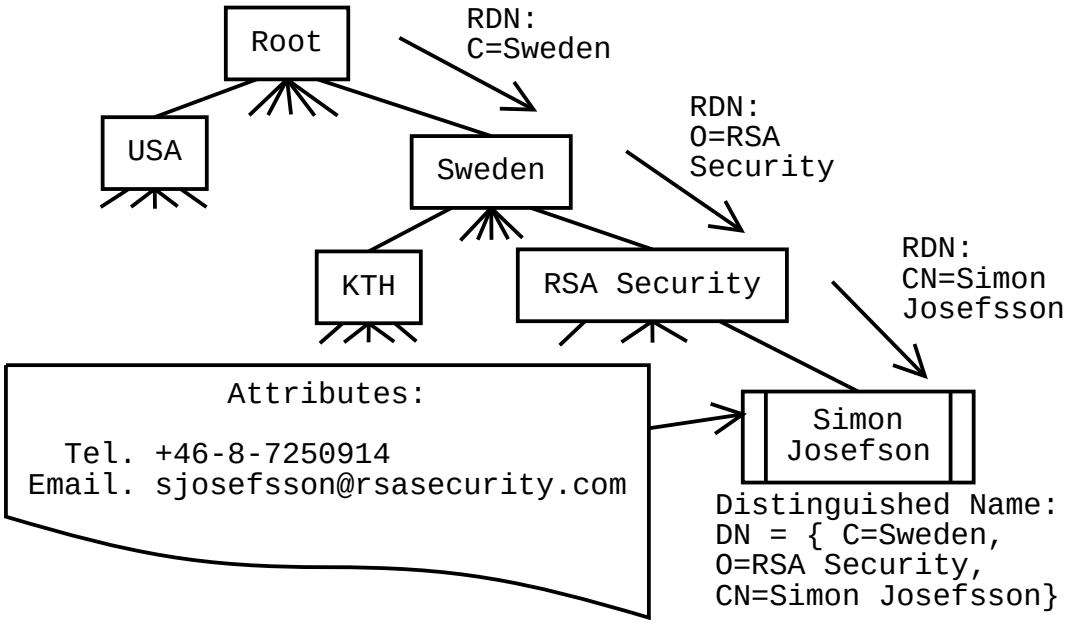


Figure 4.1. Example X.500 Directory.

[1] Other kinds of PKIs that are not identity based exists. SDSI and SPKI are efforts for creating "credential based" public key infrastructures.

This naming standard has prevailed even in current PKIX certificates. This is unfortunate, since most modern applications never use X.500 names or X.500 technology but rather internet names when addressing objects. By "Internet names" we mean IP addresses (**130.237.72.201**), domain names (**www.kth.se**), email addresses (**someone@kth.se**), URLs and similar names. However, X.509 version 3 certificates allow for "alternative" names in addition to X.500 names. This new standard defines how IP addresses, domain names, email addresses, URLs etc can be stored as names in certificates. This naming complexity can of course cause problems when certificate lookups are implemented in internet applications. If **someone@kth.se** is not used to locate a certificate for sending mail to **someone@kth.se**, how do I find out what should be used for locating the certificate?

4.2.2 Lightweight Directory Access Protocol

LDAP is closely related to the X.500 model, and the Directory Access Protocol (DAP, the "L" in LDAP stands for Lightweight). LDAP is targetted for use on the Internet though. However, LDAP does not provide the global infrastructure that DAP was intended to operate in. To see how this is a serious problem, consider that we only replace the problem of locating certificates by locating the LDAP server! A number of techniques for solving that problem exist [42]:

- **Well known DNS alias**

  One solution is to use a common and well-known domain name for LDAP servers [39]. When an application is searching for a certificate for **kalle@josefsson.org** the application contacts **ldap.josefsson.org** and looks up the certificate. Of course, this scheme relies on DNS to work.

- **Using Location of service DNS Records**

  This is a recent idea that also relies on DNS, but in a different way than the previous idea. Instead of using "well known names" as the previous idea, it uses the concept of "well known services" which is more robust. A well known service is "LDAP", "HTTP" or similar. It is possible to look up the well known services (e.g., the service "LDAP") for a domain, in order to find the server name [38]. This information is also stored in DNS. To illustrate, a client asks the **josefsson.org** domain server "Give me the name of your LDAP servers" and contact server hostnames are pointed out by the answer. In short, a layer of indirection is added.

- **LDAP Referrals**

  LDAP version 3 allows LDAP servers to re-direct clients to other LDAP servers, based on the query [96]. This is similar to the previous idea in that it uses indirection, but it does not require other protocols than LDAP itself.

- **Client Configuration**

  The simplest solution is to require administrators to configure applications to use the "correct" LDAP server. This is prone to errors and does not work well when the number of servers grows. A better approach might be to combine this approach with the previous, LDAP referrals.

We should also note that LDAP uses X.500 attributes when looking up data. Fortunately, common Internet names (email addresses, domain names, etc) have well defined X.500 types so this is not a serious problem.

4.2.3 Domain Name System

Using DNS to store certificates is an appealing method. DNS is already an integral part of practically all computers on the Internet, used to look up addresses of World Wide Web sites, of mail servers for email, etc. Thus there is no need to introduce new technology that users (and applications) everywhere will have to adopt to be able to look up certificates. Also, the DNS protocol is more lightweight than LDAP (see section 4.4).

We should mention two serious drawbacks of DNS compared to LDAP as a certificate query protocol. However, both of these drawbacks might be considered as advantages in the scenario we are interested in, we include some rationale for this below.

- **No advanced search functions**

  We have discussed this earlier; searching for people named "Adam" is not productive on a global scale and may even open up for abuses.

- **Inability to provide different answer sets in response to different query sources**

  The reason DNS cannot provide this functionality is that it is designed to contain public information without any access control. Thus this argument is moot, since the intended use of directories we are interested in are public by assumption. The ability to restrict access to resources would harm Internet wide usage of the system for certificate look up.

Both issues stem from DNS being a less complex protocol than LDAP.

4.2.3.1 Internal DNS and PKI hosting

We should mention two specialized uses of DNS as a certificate directory.

- **Internal DNS**

  It is possible to set up a private DNS infrastructure within a company. This is known as an *Internal DNS*. All of the protocol benefits (small overhead, low

latency, etc) remain, but the infrastructure of the Internet DNS is lost. Our study is intended towards the use of DNS as a certificate directory in open networks, so we will not discuss this kind of operation further.

- **PKI Hosting**

  Certain organizations may not want to handle the administrative burden of maintaining a certificate directory. Instead, they may purchase this service from a third party. This is possible to support with DNS, since it supports cross-domain aliases. As an example, consider an alias for the DNS domain **jas.nada.kth.se** that points at **jas.nada.kth.se.trustedthirdparty.com**. Since this mostly is an operational policy decision, we are satisfied with noting that this is possible.

4.3 Updating Certificates in a Directory

Something related to looking up certificates from a directory, is to update certificates in a directory. This is illustrated in figure 4.2. Both LDAP and DNS provide this functionality.

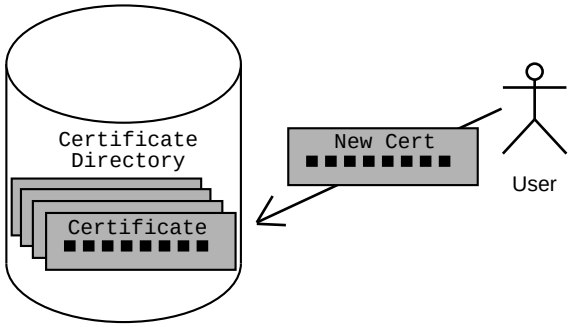


Figure 4.2. Update Certificate.

However, neither LDAP nor DNS are primarily designed to store certificates. As a consequence, neither of them supports the cryptographically "natural" way of authenticating a certificate update: Proving that you posses the private key, corresponding to the public key in the certificate, allows you to update the certificate. Rather, both LDAP and DNS use other authentication mechanisms. The following sections describe how updating works in DNS and LDAP, focusing on authentication of updates.

4.3.1 Updating in LDAP

LDAP supports addition, deletion, modification, and renaming of entries. Authentication is performed, in LDAP version 2, by means of clear text passwords or Kerberos [70]. The Secure Socket Layer (SSL) [43] [34] is often used to protect

clear text passwords during transit. LDAP version 3 supports an extensible authentication framework called Simple Authentication and Security Layer (SASL) [73]. This is a successful security framework, used by other protocols such as IMAP4 [12] and POP3 [74]. It supports several authentication methods, which are more secure than password based schemes.

4.3.2 Updating in DNS

Updating data in DNS has traditionally only been done by manual editing of static "zone files" located at each DNS server. This works well when the frequency of changes is fairly low. If the frequency of changes increases, it can sometimes be solved by other means. One solution is to store data in generic databases. The information can be updated in the database, and trigger automatic generation of zone files.

Simply put, the DNS protocol was not designed for remote data updates. Recently, in 1997, a standard was published called "Dynamic DNS" [93]. It made updating possible within the DNS protocol. Dynamic DNS supports addition, deletion and modification of entries. Our concern here is how the updates are authenticated.

"Dynamic DNS" assumes that authentication should be handled by other means (such as IPSEC), or by a security framework described in [19]. The IPSEC case is not interesting to us, as we wish to study DNS itself. The other alternative proved to be a failure after implementation experience. Thus, some new ideas had to be developed. Today, two solutions exist to authenticate DNS updates:

- **Secure DNS Transaction and Request Authentication**

  This standard, known informally as SIG(0), uses public keys stored in DNS to accomplish authentication of data. Data is digitally signed with a private key that has a corresponding public key stored in DNS. The digital signature is sent together with the data, and the remote system retrieves the public key from DNS and verifies the signature. This specification is as part of the Secure DNS specification [20], later updated by [1].

- **Secret Key Transaction Authentication**

  Transaction signatures (TSIG) [94] do not mandate any specific cryptographic operations, but is used to transport any kind of authentication data. It is intended to work with a variety of algorithms. One of the reason behind developing transaction signatures was that SIG(0) requires computationally expensive public-key operations and complex authentication logic. Transaction signatures, on the other hand, use message authentication codes (MACs). Currently two specifications make use of transaction signatures:

  - **Hashed Message Authentication with MD5 (HMAC-MD5)**

    This is a simple algorithm to calculate message authentication codes. It uses shared preconfigured symmetric keys. [59]

– **GSS Algorithm for TSIG**
This uses the extensible security framework of GSS-API [65] to perform message authentication. It establish shared secret keys that are used temporarily. GSS is a security framework, often used in combination with Kerberos [70], which provides security services such as privacy and integrity. It enables DNS updates to be securely updated using an already installed security infrastructure.

4.3.3 Conclusions

LDAP is without question the most flexible protocol when it comes to updating data and authenticating updates. In table 4.1 we list supported operations in DNS and LDAP. It should be noted that renaming is not an essential function as it can be emulated by deleting a record and adding it by another name.

	DNS	LDAP
Add	Yes	Yes
Delete	Yes	Yes
Modify	Yes	Yes
Rename	No	Yes

Table 4.1. Update operations supported in DNS and LDAP.

Support for commonly used authentication methods in DNS and LDAP are shown in table 4.2. Clearly, authentication in LDAP version 3 is more complete than in DNS. (The authentication methods are Kerberos [70] GSSAPI [65], HMAC-MD5 [59] CRAM-MD5 [57], DIGEST-MD5 [63], RSA SecurID [78], and PKIX/TLS [15] [95].)

	DNS	LDAP
Kerberos	Yes, GSSAPI	Yes, SASL
CRAM/HMAC-MD5	Yes, native	Yes, SASL
DIGEST-MD5	No	Yes, SASL
RSA SecurID	No	Yes, SASL
PKIX (TLS)	No	Yes, SASL

Table 4.2. Authentication support in DNS and LDAP.

4.4 Performance and Overhead

Looking up and retrieving certificates over a network will of course have a impact on both network and servers involved. In this section we will study the DNS and LDAP protocols. After a description of the protocols, we proceed with theoretical discussions of three different aspects of performance. The discussions are illustrated with real-world benchmarks.

The three different measurements we considered to be of importance are:

- **Network latency:** Number of round trips.
- **Network bandwidth:** Packet size.
- **Overall performance:** Queries per second.

Before we begin, we wish to make a comment about the inherent caching in the Domain Name System.

4.4.1 Caching in DNS and How it Affects Certificate Lookup

The nature of DNS, being a caching distributed database, makes both measuring and predicting real-world numbers difficult. To be able to make good predictions, one would need to simulate a network consisting of, say, a hundred thousand servers and millions of clients. Clients and servers are connected by different types of networks, so different latencies and bandwidth would need to be simulated. This is not trivial. Instead of simulating a network, one can measure performance on a real network though, one such study is presented in [53].

In our DNS benchmark we will limit ourselves to studying one single server, connected to several clients. This is not without justification. An organization setting up a DNS environment is concerned with how their own server is performing. If we disregard caching in DNS, our comparison between DNS and LDAP would be on equal terms. We must consider caching though, since it is a key component of DNS. So, how much does it affect performance? First, at least it does not degrade performance. This means our results for DNS will reflect a worst case, while our LDAP results reflect the normal case. The discrepancy between our benchmark and the real world thus depends on how large the impact of caching is. We can make some conjectures about this, since the caching mechanism is well-defined. The main application we focus on, electronic mail, is such that the time between lookups is fairly long. You usually do not send more than a few messages to an individual per day. The size of these savings are thus only a small multiple of the differences in latency and bandwidth between the remote, official, server and your local, caching, server. For end users, the savings are hardly noticeable compared to the time to send the entire message. All in all, by studying the behavior of one DNS server we get a “fair” comparison between LDAP and DNS by ignoring the more complex issues of the whole system.

4.4.2 The Domain Name System Protocol

The DNS protocol is designed to be compact and simple. It runs over a datagram transport service, and does not require it to be reliable (e.g., DNS handle packet loss and packet size limits internally). The datagram service normally used is the Internet Protocol User Datagram Protocol (IP/UDP). UDP packets have a maximum size in practice, and the Internet Protocol Transmission Control Protocol (IP/TCP) is used as a fall-back when the maximum UDP packet size is exceeded. The protocol is binary, unlike many other Internet protocols that are text based. The binary encoding is used for compactness.

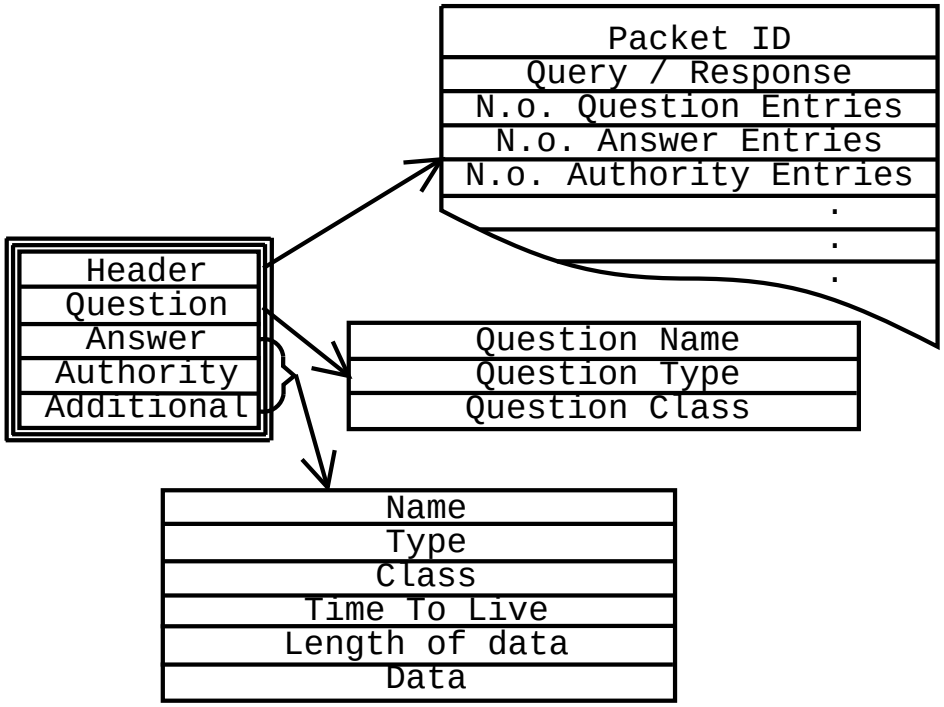


Figure 4.3. DNS envelope.

The DNS protocol uses the same envelope for all communication. The envelope is illustrated in figure 4.3. The envelope's first member, the header, is present in all packets and contains control information. Besides packet identity and packet length information, it contains information on whether the packet is a query or a response; if the packet is truncated (indicating that the question should be re-sent via TCP); etc. For query packets, the "question" field is present (e.g., has non-zero size) and the remaining fields are empty. For "responses" the question field is empty and remaining fields contains data. The "question" field, if present, contains a domain name (ASCII string), a domain type (integer) and a domain class (integer). The remaining three sections have the same structure, and contain Resource Records (RRs). Resource Records consists of a domain name, a type and a

class, together with the corresponding data. (For comparison, the question section contains all fields except the data.)

Again, not all sections need to contain data. The header section specifies whether there is data or not in each section.

This is a brief overview of the DNS protocol. The protocol is described in [71], with a good hands on introduction in [8].

4.4.3 The Lightweight Directory Access Protocol

The Lightweight Directory Access Protocol (LDAP) [96] is based on the Directory Access Protocol, which is a part of the comprehensive online directory developed through the standardization process of ISO and ITU. This original standard and service is known as X.500 [10]. The LDAP protocol was designed to run over a connection-oriented, reliable transport. Both DAP and LDAP are described using ASN.1. We will not go into the details of ASN.1 but rather use familiar terms when talking about data structures.

Like DNS, all protocol operations are encapsulated in a common envelope. This envelope is shown in figure 4.4, where only some structures have been expanded. As can be guessed by the figure, the LDAP protocol is much more complex than DNS.

This is an overview of LDAP, [96] is the definitive source.

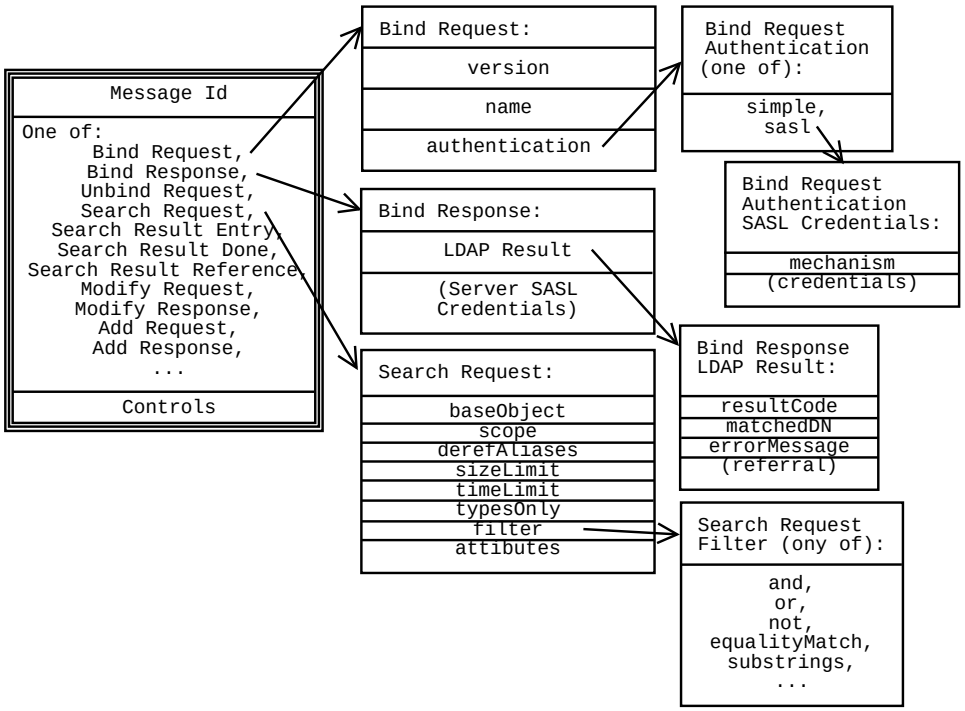


Figure 4.4. LDAP packet, with some structures expanded.

4.4.4 Round Trips

A “round trip” denotes the time spent between sending one packet from an entity A (“client”) to an entity B (“server”) and back to the entity A. This is illustrated in figure 4.5.

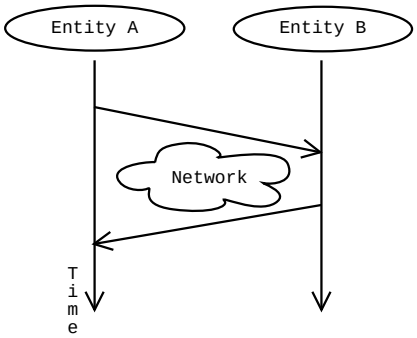


Figure 4.5. Round Trip between two entities.

This measurement is of special importance in high latency environments, such as mobile application. Protocols that accomplish the same thing, by trading space, CPU processing or other resources to reduce the number of round trips, will be more efficient under these circumstances.

Since DNS can use both UDP and TCP, while LDAP only uses TCP, we first compare the “round trip” characteristics of these underlying transport protocols. UDP [81] does not incur any round trips other than those of the application protocol (e.g., DNS). Recall from the introduction section that TCP [83] on the other hand provides reliable connections on top of an unreliable transport layer. This requires some overhead, mostly due to round-trip costs of acknowledgments. If a packet or an acknowledgement is lost, it is re-transmitted. Retransmission only occur if packets are lost, this is relatively uncommon so we ignore this additional complication. Figure 4.6 and 4.7 illustrates the packets sent in setting up and tearing down a TCP connection.

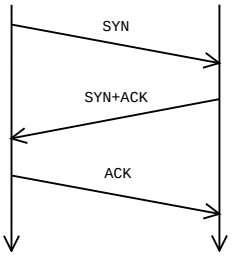


Figure 4.6. Setting up a TCP connection.

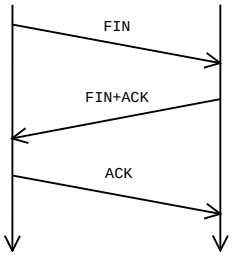


Figure 4.7. Tearing down a TCP connection.

4.4.4.1 Round Trips in a DNS query

A DNS query using UDP has the simplest characteristics, one packet is sent from the client A to the server B. The server B responds to the query by sending a packet to A. This is illustrated in figure 4.8.

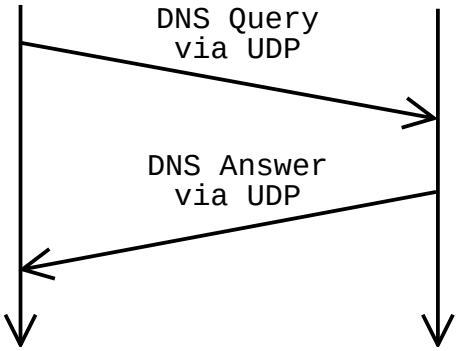


Figure 4.8. Round trips in a DNS Query over UDP.

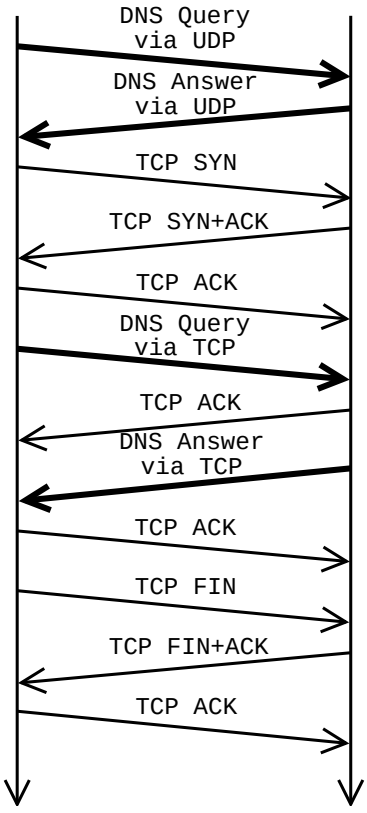


Figure 4.9. Round Trips in a DNS Query.

A complication occurs when any of the packets exceed the size limit of the User Datagram Protocol. When UDP is transported via IPv4, this limit is around 600 bytes. For IPv6 this limit is around 1500 bytes. The protocol handles this by returning a truncated packet (via UDP), with control information in the header informing the client that the packet was indeed truncated. If the data the client was interested in is contained within the first, non-truncated, part, all is fine.

However, it is more likely that the client needs the entire response packet. DNS solves this problem by describing a fall-back mechanism to use TCP. Now, a client receiving a truncated response will query the server again, using TCP. Of course, this adds the round trip costs associated with TCP connections. This case is illustrated in figure 4.9, where the DNS packets are highlighted using thick lines.

4.4.4.2 Round Trips in a LDAP query

An LDAP query is more complicated. The protocol is transported with TCP. This means that the round trip costs of setting up and tearing down TCP connections are always added. Also, the LDAP protocol itself is more "heavyweight" than DNS. Figure 4.10 illustrates round trips involved in a complete LDAP query. Thin arrows are TCP packets, thick arrows are LDAP packets. We now describe each of the thick arrows, the application level.

LDAP uses one round trip to set up the connection, sending a "bind request" packet in the forward direction, and returning a "bind result" packet. The query itself is contained in one "search request" packet. The answer (e.g., certificates) to this query is sent in "search entry" packets. There may be zero or more such tokens sent, depending on how many records matched the search criteria. The status of the whole search is sent from the server to the client in a "search response" packet. The client closes the connection with by sending a "unbind request" and closing the TCP connection.

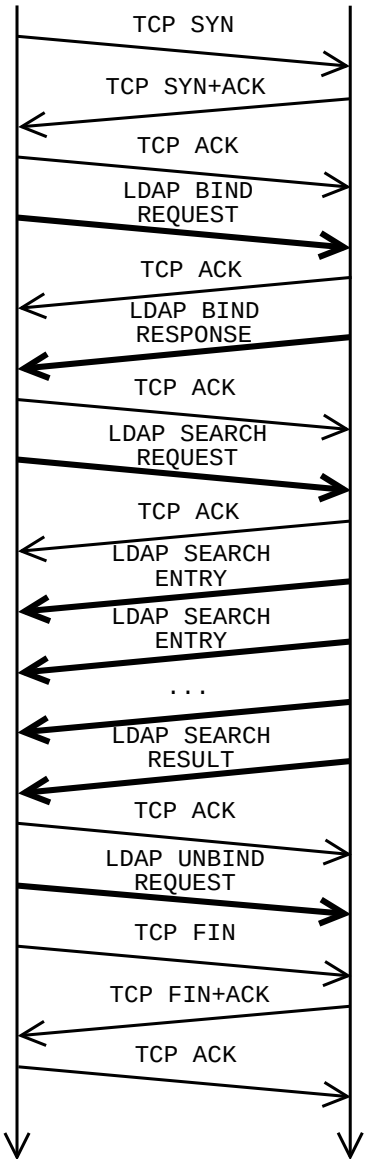


Figure 4.10. Round trips in a LDAP Query.

4.4.4.3 Conclusions

Practical experimentation, using the network packet sniffer Ethereal [24], confirms our theoretical discussion. We summarize this sections' results in table 4.3.

Protocol and Transport mechanism	Number of packets	Round trips on Application level	Round trips on Transport level
DNS on UDP	2	1	1
DNS on UDP+TCP	12	2	5
DNS on TCP	10	1	4
LDAP on TCP	≥ 15	3	5

Table 4.3. Number of round trips for a query using DNS and LDAP.

4.4.5 Packet Size

We are studying the usual representation of PKIX, but we should note that there are others which are more compact [66]. Of course, the choice of representation affects certificate size. A comparison between different representation formats is outside the scope of this section.

Typical sizes of PKIX certificates for common key lengths are shown in Table 4.4. The choice of algorithm and key length influence the size of certificates. There are other factors that affect certificate size as well. Certificates usually store other information, such as names and addresses too. In our examples, we are storing a name (User 0001, User 0002 etc) and email address (user0001@josefsson.org, user0002@josefsson.org etc). However, in practice these addresses are usually longer (complete X.500 addresses) and addresses exceeding 200 characters are known to exist. In practice, certificates also include pointers to CRL distribution points and a Certificate Policy. In our size comparison, we include a commercially available certificate. A detailed printout of the contents of these certificates can be found in Appendix B.

RSA 512 bit	519 bytes
RSA 1024 bit	587 bytes
VeriSign RSA 1024 bit	1160 bytes

Table 4.4. Typical certificate sizes.

Table 4.4 is of special interest to our study. It shows that all common certificates are **larger** than the IPv4 UDP packet limit. This indicates that applications using DNS to look up certificates should instruct their DNS libraries to immediately use TCP, instead of letting it first try UDP and then fall back to TCP.

Observation 1 *Use of UDP is **not** sufficient when looking up certificates in DNS in an IPv4 environment. Applications should use TCP.*

Table 4.4 also show that all common certificates are **smaller** than the corresponding IPv6 UDP packet limit. This is fortunate, and make the argument for using DNS to look up certificates stronger. This is especially true in mobile application, where two factors work together. The first factor is that bandwidth and latency savings are noticeable, and a small number of round trips is often a design requirement. The second factor is that IPv6 is likely to gain faster acceptance in mobile applications than in traditional networks. This is partly because of addressing issues, IPv6 make it possible to assign an Internet address to practically all electronic devices.

Observation 2 *Use of UDP is sufficient when looking up certificates in DNS in an IPv6 environment.*

The previous discussion does not apply to LDAP. LDAP is "designed to run over connection-oriented, reliable transports" [96, paragraph 5.2] such as TCP. However, there exists a datagram version of LDAP as work in progress [41] that uses UDP. However, this is experimental and also does not support authentication. We were unable to find a open implemention of it.

4.4.5.1 Packet size of DNS and LDAP queries

Comparing the two protocols in a real-world situation, we used the "RSA 1024 bit" certificate. We stored the certificate in both DNS and LDAP directories. We fetched the certificate from the directories using a simple client. We measured the amount of data each of the protocols, DNS and LDAP, required to transfer the certificate. All tests were carried out using IPv4.

We did *not* measure the amount of overhead added by physical layers, the IP layer, nor the UDP/TCP layer. This overhead does not vary depending on the size of data packets[2], thus the size of overhead from these layers is merely a function of number of round trips, refer to section 4.4.4 where we study this aspect. Table 4.5 illustrate header sizes of involved layers.

Ethernet header	14 bytes
IP header	20 bytes
UDP header	8 bytes
TCP header	32 bytes

Table 4.5. Overhead of various layers.

[2] Assuming no fragmentation occurs because of oversized packets.

Both DNS and LDAP are open and widely implemented protocols. There are no differences in network characteristics depending purely on the implementation. This means we are actually benchmarking the protocols, and not the implementations. For reference, however, we note that the applications used were BIND [8] and Open LDAP [79].

Our results are listed in table 4.6. "DNS over UDP/TCP" depicts the case were DNS first tries UDP and falls back to TCP. For comparison, it also includes the "raw" certificate size. These values were collected using Ethereal [24]. The raw data is available electronically [52].

	Data from client	Data from server	Total data
Certificate (data)	-	-	587
DNS over TCP	37	691	728
DNS over UDP/TCP	74	728	802
LDAP over TCP	80	772	852

Table 4.6. Bytes required to transfer a certificate that contains a 1024 bit RSA key with DNS and LDAP.

4.4.5.2 Conclusions

Figure 4.11 illustrate table 4.6 and summarizes this section's results. LDAP uses almost twice as much overhead as DNS.

4.4.6 Computer Resource Utilization

The previous sections discuss facts and figures related to network bandwidth utilization and network latency, but in reality there are many factors combined that affect performance. This section makes a simple measurement of number of queries per second one can expect with common server implementations. Measurements on actual implementations must be considered carefully before anything can be said about the technology they implement, though.

The tests consisted of running a simple benchmark tool, developed for this purpose, that queries a server for a certificate and retrieves the certificate. The tool asks 5000 queries. The tool itself was run three times (with similar results) to assure that no external factors influenced the results. Since our LDAP implementation only runs on TCP, we are only comparing it to DNS over TCP. The source code of the benchmark tool used can be found in Appendix C.

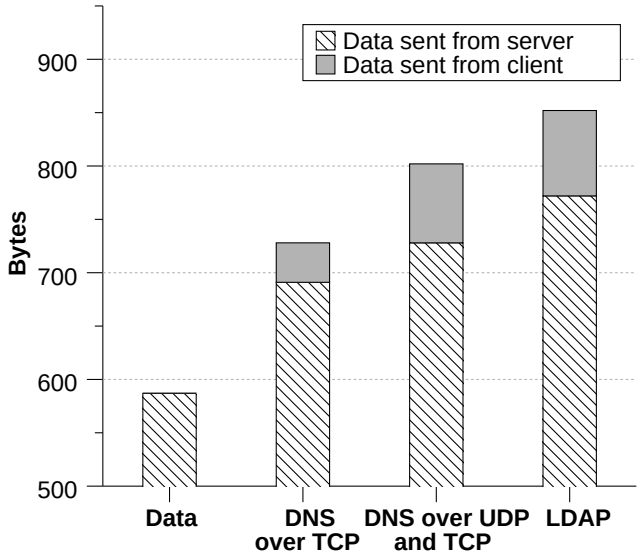


Figure 4.11. Bytes required to transfer a certificate with a 1024 bit RSA key with DNS and LDAP.

4.4.6.1 Conclusions

Figure 4.12 illustrates table 4.7 and summarizes this section's result. The discrepancy between DNS and LDAP performance is quite large, to DNS's advantage. Some of it can be explained by our previous discussions on packet sizes, and especially round trips. A protocol using more than one round trip requires servers to somehow remember states in the protocol, which results in more complex software. But the large discrepancy cannot be explained by this alone. Rather, the reason has much to do with the design of, and expected use of, the protocols. The LDAP protocol was not designed for small, fast, simple queries but rather for sessions were programs (or users) may ask many questions, or search among answers interactively. Hence not much effort has gone into optimizing existing LDAP server software for the former case. DNS on the other hand has been designed with efficiency of implementations in mind, and this shows.

	Implementation	Queries per second
DNS	BIND version 9.0.0	485
LDAP	OpenLDAP version 1.2.9	33

Table 4.7. Queries per second to look up a certificate.

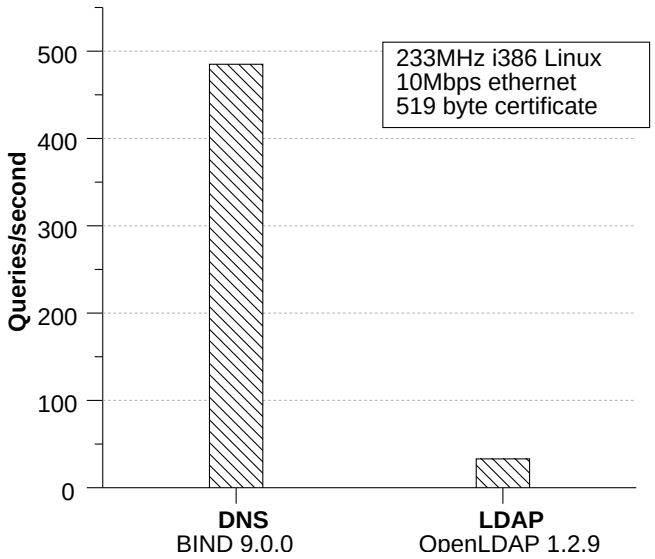


Figure 4.12. Queries per second to look up a certificate.

Chapter 5

DNS Security Considerations

This chapter is about protecting your Certificate Directory, which is assumed here to be a public directory, from privacy abuses. We first need to discuss this concept on an abstract level. How can public information be of privacy concern? On the surface, this might look like a contradiction. This introduction borrows some ideas from "The Ethics of Information: Protecting Privacy in the Computer Age" [36].

The concepts of **private information** and **public information** are key to this discussion. The traditional notation of **privacy** is that of protecting private information. Here, we will try to argue that, in the light of computer technology, the notation of privacy needs to be reconsidered.

Information about a person such as name, address, telephone number, employment, passport photo, etc are by the strictest sense of the definition **public**. These seemingly unrelated pieces of information, taken together, present a **privacy** concern to most people. Traditionally the "taken together" part means costly intelligence work. It would cost a lot of time and money. Traditionally, you are safe. But with public databases and high-speed connections, it is easy to collect information about a vast number of people at the same time and process it locally. All such databases with "public" information are a privacy concern.

Today we thus have to replace the "time and money" factor that secured privacy in older times, with a new factor that makes "data mining" infeasible.

Now, if we return to our application, we have a certificate directory. Certificates often carry additional information, used to authenticate the certificate holder for certain purposes. Examples of additional information are qualification, licenses (attorney, doctor, ...), official approvals (vehicle driving licenses, ...). One can imagine a credit card vendor using certificates containing users credit card numbers. This information presents the privacy concern.

Secure DNS [20] contains a serious problem in this regard, when used as a Certificate Directory. This chapter presents this problem in detail, and a new idea to solve this problem.

5.1 Secure DNS

Secure DNS is a recent development in the DNS field. It is currently in testing for deployment by various organizations [67] [76]. The following three distinct services are the goal of Secure DNS [20]:

- **Data Origin Authentication**

  Authentication of data is provided by cryptographically signing data stored in DNS. Both keys and signatures are stored in DNS, together with the data it authenticates. Since keys are used in authenticating these signatures, they need to be authenticated just as other DNS data. Of course, this chain of keys and signatures must have a locally trusted root, or the data cannot be trusted at all. Secure DNS aware clients are usually configured with a (small) number of trusted keys.

  The keys used to sign data in DNS are attached to each collection of DNS data (a "DNS zone"). This is in contrast to attaching keys to individual servers involved in querying the information. This design allows for "off line signing", so that even a compromise of the servers involved does not have to affect the security of the data.

  Data non-existence services are also provided. This means that it is possible to strongly secure that a certain name does *not* exist. E.g., a client asking for **maria.josefsson.org** should be able to trust that this domain does not exist (if that is the case).

- **Key distribution**

  To support Data Origin Authentication, Secure DNS defines a method of storing keys in DNS known as "KEY records". Thus DNS is used as a public key distribution mechanism (a "Public Key Infrastructure"). This PKI is used to support the public-key cryptographic operations required by Secure DNS itself, but it can also be used for other protocols.

- **Transaction and request authentication**

  Each individual transaction can also be protected by means of Secure DNS. This is in contrast to only protecting the data involved. This makes it possible for a DNS client to be sure it is at least getting responses from the server it thinks it queried, and that responses are to the query it sent.

Outside the goals of Secure DNS, and hence *not* implemented, are means for confidentiality of queries or responses (e.g., information is considered to be public), and protection against denial of service attacks.

5.1.1 Data Non-existence

This section gives an abstract description of how Secure DNS achieves strong data non-existence. Again, "data non-existence" is how clients (securely) trust that a certain domain, say **maria.josefsson.org**, does not exist.

We illustrate this by presenting a scenario, a naive first attempt at a solution to this scenario, and then discuss some problem with that solution. We proceed to describe the currently implemented solution, NXT records [20, section 5], that overcomes these problems. Since we have found further problems with this solution, we describe these problems. We conclude by presenting our own solution, that overcomes these new problems [50].

Scenario: Consider a set of **keys**[1] that uniquely identify a domain name (e.g., **kalle.josefsson.org**) from an alphabet (strings of alphabetic characters, "a"-"z"[2]) that each is attached to one or more **data** item(s) (e.g., certificates). Client entities (DNS resolvers) can send queries to server entities (DNS servers) for a key, expecting the corresponding data back. The data is cryptographically **signed** to provide authentication of data origin. However, if a certain key does not have any attached data, a message need to be returned with that information. The problem of **data non-existence** is to strongly authenticate this last piece of information.

The naive implementation, illustrated in figure 5.1, is for a client A to query a server B and expect a cryptographically signed answer back. To prevent replay-attacks to occur after new names have been introduced, the signature should have limited life length.

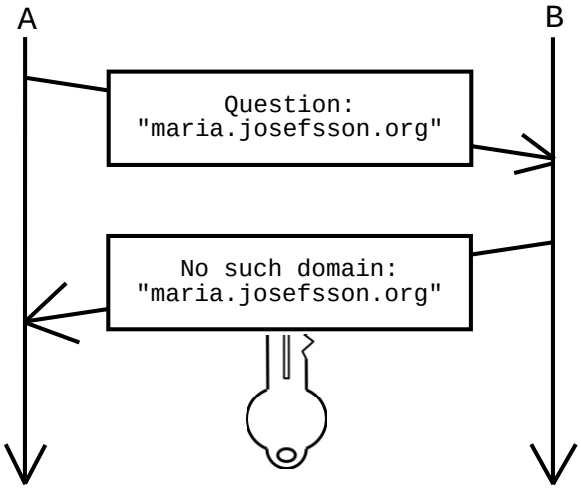


Figure 5.1. Naive data non-existence implementation.

[1] Do not confuse this with cryptographic keys. In this chapter, we will explicitly mention "cryptographic key" when we talk about them.

[2] DNS today is US and ASCII centric. Work on supporting internationalized domain names is under way

First we convince ourself that the naive implementation does indeed work. It does, since we have:

- The client can verify the signature with the server's key (stored in DNS).
- If the signature is valid, the client knows that **maria.josefsson.org** indeed does not exist.

Three major limitations that this naive implementation faces, and that Secure DNS had to overcome, are the following:

- **It requires the signing key at server B to be "online"**

  That is, available for signing operations when answering queries. This would be a security concern, since breaking into the servers would mean compromising the security of Secure DNS.

- **Cryptographic signing operations are slow**

  This brings up two related problems. First, it would be easy to overload a server by repeatedly asking questions, a so called "Denial of Service" attack. Secondly, today large servers may answer several thousand queries per second, cryptographic signing hardware with this kind of performance is not commonly available (today).

- **Vulnerability to Playback attacks**

  A negative answer for a domain may be recorded, and if the domain is later added, it may be possible to use the signed negative response to deny existence for a domain. (This may be solved by adding a challenge/response scheme for each DNS query, but DNS does not include one today.)

These problems are easily translated into *requirements* for a better solution:

- Off line signing.
- Because of off line signing, we need to find some piece of data that can be signed when off line that works as a "data non-existence proof" when online.

We first mention that a naive approach of signing all possible keys is not feasible, since this is practically a infinite set. We are now going to study the currently implemented solution, NXT records, but first we need som data to work with. Consider table 5.1.

By introducing a *canonical ordering*[3] of keys, we can enumerate all keys that have data attached to them. By ordering keys we can construct links between each key in the sorting order. This piece of information can be signed and used as a "data non-existence proof" (discussion below). To illustrate this, consider table 5.2 where some new keys and data are added to our previous table.

[3]The canonical ordering used in Secure DNS are simple byte-by-byte comparison of the ASCII encoding of strings. See [20, Section 8].

Key	Data
kalle.josefsson.org	[Kalle Josefsson's Certificate]
simon.josefsson.org	[Simon Josefsson's Certificate]
lotta.josefsson.org	[Lotta Josefsson's Certificate]

Table 5.1. Example of (partial) DNS information for a zone josefsson.org.

Key	Data
kalle.josefsson.org	[Kalle Josefsson's Certificate]
kalle.josefsson.org	Next key: lotta.josefsson.org
lotta.josefsson.org	[Lotta Josefsson's Certificate]
lotta.josefsson.org	Next key: simon.josefsson.org
simon.josefsson.org	[Simon Josefsson's Certificate]
simon.josefsson.org	Next key: kalle.josefsson.org

Table 5.2. Example of non-existence proof data for data in table 5.1.

We now describe how this new information is used in a clever way to achieve a "non-existence proof". Entity A queries for a non-existent key **maria.josefsson.org**. The server replies with a tuple, **(lotta.josefsson.org, Next key: simon.josefsson.org)**. This is illustrated in figure 5.2. The "next" tuple is cryptographically signed, to provide authentication of data non-existence. These "Next" tuples can be calculated in advance, and be cryptographically signed off line. Thus it fulfills our two earlier requirements.

Now, how does the entity A know that **maria.josefsson.org** does not exist? By using the canonical ordering process, it knows that **maria.josefsson.org** sort later than **lotta.josefsson.org** and earlier than **simon.josefsson.org**. Since these "next" records were created using all existing keys, the entity A now can be certain that **maria.josefsson.org** indeed do not exist. (Of course, assuming that the cryptographic signature was correctly verified.)

5.1.2 NXT Chaining

This form of non-existence proof raises an immediate problem. From the previous paragraph, the entity A that queried for **maria.josefsson.org** learned more than that it does not exist. Explicitly, it learned that both **lotta.josefsson.org** and **simon.josefsson.org** do exist. Continuing by asking for a domain that sorts earlier than **lotta.josefsson.org**, it is easy to see how this can be abused to learn the entire content of the entire data set.

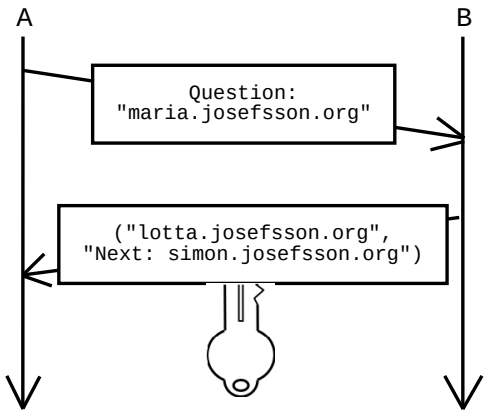


Figure 5.2. "NXT" Data-nonexistence implementation.

Observation 3 *Secure DNS's non-existence proofs "NXT" can be used to collect all data in a DNS zone. We call this "NXT chaining".*

Besides privacy concerns, gaining knowledge of a victim's DNS information opens up more direct forms of attacks [6]. In short, the knowledge of a certain organization's DNS information provides clues that help an intruder. Although most of the dangers in the scenarios described in the document have become obsolete (like allowing zone transfers), NXT chaining in Secure DNS re-introduces one of the concerns.

An alternative that reduces information revealed by non-existence responses is needed. The next section presents our idea.

5.2 Data Non-existence with Minimum Disclosure

The idea is to replace the sensitive information of NXT records of figure 5.2 with something that serves the same purpose, but cannot be used to discover names of other keys in the data collection. This is illustrated in figure 5.2.

Obviously the function "f" of figure 5.2 should not be possible to invert, or the idea of replacing the original data is lost. We suggest using a well-known cryptographic hash function, SHA-1 [77]. We illustrate the corresponding DNS data in table 5.3.[4] We see encoded SHA-1 values as valid keys, thus re-using the same canonical ordering process of Secure DNS.

We first verify that this solution still works. We do this by describing how a client builds trust that its query for **maria.josefsson.org** does not exist. Assume this key hashes to 142. As illustrated in figure 5.4, the server can respond to the

[4]For brevity, the table uses small values. In practice, the values would have been an encoded form of a 160 bit SHA-1 value.

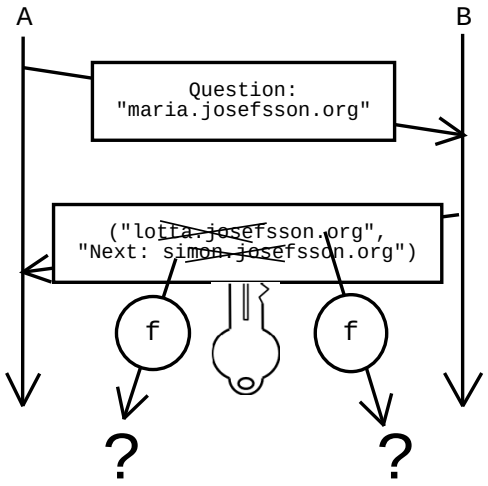


Figure 5.3. Minimum information disclosure and data non-existence.

Key	**Data**
kalle.josefsson.org 127.josefsson.org	[Kalle Josefsson's Certificate] Next key: 156.josefsson.org
lotta.josefsson.org 59.josefsson.org	[Lotta Josefsson's Certificate] Next key: 127.josefsson.org
simon.josefsson.org 156.josefsson.org	[Simon Josefsson's Certificate] Next key: 59.josefsson.org

Table 5.3. Example of non-existence proof data for data in table 5.1.

query with a tuple **(127.josefsson.org, Next key: 156.josefsson.org)**. A client now makes sure a hash calculation of its query hashes to a value between the two returned hash values. If this is the case, it can be certain that the query did not ave any corresponding data. (Of course, it must also verify the corresponding cryptographic signature.)

We return to our problem with NXT records, that a client learns additional information from the non-existence proof. This is now replaced by only learning the hash value of two existing keys in the data collection. This can be abused to chain through hash values of all existing names in the zone. Assuming the hash function is not invertible, this is not a problem. However, it is possible to use this to determine the number of names in the zone[5].

[5]Good estimates on the number of names in a zone can be calculated using only a few non-existence proofs though (assuming hash values are equally distributed.) A rough estimate can be done using only one non-existence proof, assuming a Poisson distribution.

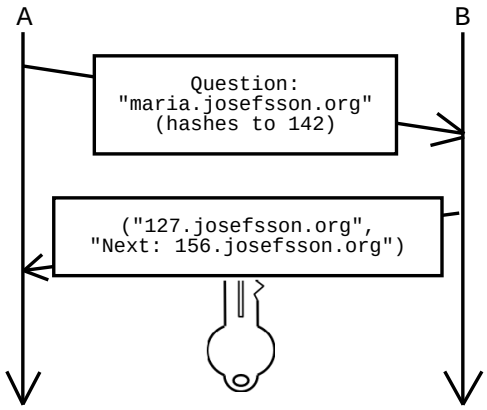


Figure 5.4. Final example of how minimum information disclosure and data non-existence would work using NO records.

5.3 Implementing the Idea in DNS

The previous idea has been described in a standards document [50, work in progress]. It introduces two further technicalities that are of interest here:

- **Hash truncation**

  To reduce size of NO records, hash values can be truncated.

- **Record compression**

  To reduce number of records, hash values from several consecutive NO records can be merged into one, larger, record.

Besides mentioning these, going more into the details of the technical description is outside the scope of this chapter. The technical document, with a complete description, can be found in Appendix A.

Chapter 6

Conclusions

We believe DNS would make a good distribution point of application keys and certificates for large scale systems. The main reason is that DNS is a unique and ubiquitous provider of bindings between commonly used names (i.e., email addresses and hostnames) to pieces of data. We have also seen that DNS is generally more efficient than LDAP.

With regard to the recent Secure DNS standardization process, our results from chapter 5 suggest that Secure DNS should not be used in zones where privacy sensitive information is stored. Applications that require or are able to make use of Secure DNS are recommended to use approaches such as the NO record outlined.

One area that warrents further work is authenticating updates in DNS. As our section 4.3, “Updating Certificates in a Directory”, shows, only shared symmetric keys are in use today.

Bibliography

1. D. Eastlake 3rd. *DNS Request and Transaction Signatures (SIG(0)s)*, September 2000. RFC 2931.
2. *America Online Instant Messenger*. World Wide Web, `http://www.aol.com/aim/`, Last visited 9 November 2001.
3. D. Atkins, W. Stallings, and P. Zimmermann. *PGP Message Exchange Formats*, August 1996. RFC 1991.
4. A. Back. *PGP Timeline and Brief History*. World Wide Web, `http://www.cypherspace.org/~adam/timeline/`, Last visited 9 November 2001.
5. D. Balenson. *Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers*, February 1993. RFC 1423.
6. S. M. Bellovin. Using the Domain Name System for System Break-Ins. In *Proceedings of the Fifth Usenix UNIX Security Symposium*, June 1995.
7. T. Berners-Lee, R. Fielding, and H. Frystyk. *Hypertext Transfer Protocol—HTTP/1.0*, May 1996. RFC 1945.
8. *Berkeley Internet Naming Daemon*. World Wide Web, `http://www.isc.org/bind/`, Last visited 9 November 2001. version 9.0.0.
9. M. Branchaud. *A Survey of Public-Key Infrastructures*. Master's thesis, McGill University, Montreal, March 1997.
10. CCITT. *Recommendation X.500: The Directory: Overview of Concepts, Models and Services*. Technical report, ISO/IEC 9594, 1988. Also published as ISO/IEC 9594.
11. CCITT. *Recommendation X.509: The Directory—Authentication Framework*. Technical report, 1988. Also published as ISO/IEC 9594-8.
12. M. Crispin. *Internet Message Access Protocol—Version 4rev1*, December 1996. RFC 2060.
13. D. Crocker. *Standard for the format of ARPA Internet text messages*, August 1982. RFC 822.
14. S. Crocker, N. Freed, J. Galvin, and S. Murphy. *MIME Object Security Services*, October 1995. RFC 1848.
15. T. Dierks and C. Allen. *The TLS Protocol Version 1.0*, January 1999. RFC 2246.
16. W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, vol IT-22(no. 6):644–654, 1976.

17. C. Dinkel. *Secure Data Network System (SDNS) Network, Transport, and Message Security Protocols*, volume U.S. Department of Commerce, National Institute of Standards and Technology, report NISTIR 90-4250. 1990.

18. *IETF DNS Extensions Workgroup*. World Wide Web, `http://www.ietf.org/html.charters/dnsext-charter.html`, Last visited 9 November 2001. Workgroup chaired by Olafur Gudmundsson and Randy Bush.

19. D. Eastlake. *Secure Domain Name System Dynamic Update*, April 1997. RFC 2137.

20. D. Eastlake. *Domain Name System Security Extensions*, March 1999. RFC 2535.

21. D. Eastlake and O. Gudmundsson. *Storing Certificates in the Domain Name System (DNS)*, March 1999. RFC 2538.

22. M. Elkins. *MIME Security with Pretty Good Privacy (PGP)*, October 1996. RFC 2015.

23. E. A. Young et al. *OpenSSL*. World Wide Web, `http://www.openssl.org/`, Last visited 9 November 2001.

24. G. Combs et al. *Ethereal*. World Wide Web, `http://ethereal.zing.org/`, Last visited 9 November 2001.

25. Menezes et al. *Handbook of applied Cryptography*. CRC Press, 1996. World Wide Web, `http://www.cacr.math.uwaterloo.ca/hac/`, Last visited 9 November 2001.

26. J. Feghhi, J. Feghhi, and P. Williams. *Digital Certificates*. Addison Wesley, 1999.

27. *FidoNet Policy Document*. World Wide Web, `http://www.fidonet.org/policy4.txt`, Last visited 9 November 2001, June 1989.

28. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol—HTTP/1.1*, June 1999. RFC 2616.

29. W. Ford and M. S. Baum. *Secure Electronic Commerce*. Prentice Hall, 1997.

30. N. Freed and N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples*, November 1996. RFC 2049.

31. N. Freed and N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, November 1996. RFC 2045.

32. N. Freed and N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, November 1996. RFC 2046.

33. N. Freed, J. Klensin, and J. Postel. *Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures*, November 1996. RFC 2048.

34. A. Frier, P. Karlton, and P. Kocher. *The SSL 3.0 Protocol*, November 1996. Netscape Communications Corporation.

35. J. Galvin, S. Murphy, S. Crocker, and N. Freed. *Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted*, October 1995. RFC 1847.

36. R. E. Gantenbein. *The Ethics of Information: Protecting Privacy in the Computer Age*. World Wide Web, `http://www.cs.uwyo.edu/~rex/privacy.html`, Last visited 9 November 2001, February 1998.

37. O. Goldreich. *Foundations of Cryptography Vol 1*. Cambridge University Press, 2001.

38. A. Gulbrandsen and P. Vixie. *A DNS RR for specifying the location of services (DNS SRV)*, October 1996. RFC 2052.

39. M. Hamilton and R. Wright. *Use of DNS Aliases for Network Services*, October 1997. RFC 2219.

40. M. Hauben and R. Hauben. *Netizens—On the History and Impact of the Net*. World Wide Web, `http://www.columbia.edu/~hauben/netbook/`, Last visited 9 November 2001, April 1995.

41. R. Hedberg and L. Johansson. *Connection-less Lightweight Directory Access Protocol*, May 2000. Work in Progress, draft-ietf-ldapext-cldap-00.txt.

42. R. Hedberg and R. Moats. *A Taxonomoy of Methods for LDAP Clients Finding Servers*, September 2000. Work in Progress, draft-ietf-ldapext-ldap-taxonomy-03.txt.

43. K.E.B. Hickman. *The SSL Protocol*. World Wide Web, `http://home.netscape.com/eng/security/SSL_2.html`, Last visited 9 November 2001, February 1995. Netscape Communications Corporation.

44. R. Housley. *Cryptographic Message Syntax*, June 1999. RFC 2630.

45. R. Housley and P. Hoffman. *Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP*, May 1999. RFC 2585.

46. *What is ICQ?* World Wide Web, `http://www.icq.com/products/whatisicq.html`, Last visited 9 November 2001.

47. American National Standards Institute. *American National Standard for Financial Institution Message Authentication*, 1986. ANSI X9.9.

48. *An Introduction to Internet Relay Chat (IRC)*. World Wide Web, `http://www.newircusers.com/ircchat.html`, Last visited 9 November 2001.

49. ITU-T. *ITU-T recommendation X.680-X.683*. Technical report, 1997. Also published as ISO/IEC 8824-1:1998 “Information Technology—Abstract Syntax Notation One (ASN.1): Specification of Basic Notation”, ISO/IEC 8824-2:1998 “Information Technology—Abstract Syntax Notation One (ASN.1): Information Object Specification”, ISO/IEC 8824-3:1998 “Information Technology—Abstract Syntax Notation One (ASN.1): Constraint Specification”, ISO/IEC 8824-4:1998 “Information Technology—Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 Specifications”.

50. S. Josefsson. *Authenticating denial of existence in DNS with minimum disclosure (or; An alternative to DNSSEC NXT records)*, August 2000. Work in Progress, draft-ietf-dnsext-not-existing-rr-00.txt.

51. S. Josefsson. *Base 64, 32 and 16 Encodings*, August 2000. Work in Progress, draft-josefsson-base-encoding-00.txt.

52. S. Josefsson. *Ethereal Network Dumps, raw data*. World Wide Web, `http://josefsson.org/exjobb/`, Last visited 9 November 2001, September 2000.

53. J. Jung, E. Sit, H. Balakrishnan, and R. Morris. *DNS Performance and the Effectiveness of Caching*. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop, 2001*, 2001.

54. B. Kaliski. *A Layman's Guide to a Subset of ASN.1, BER, and DER*. Technical report, RSA Security, November 1993.

55. B. Kaliski. *Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services*, February 1993. RFC 1424.

56. S. Kent. *Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management*, February 1993. RFC 1422.

57. J. Klensin, R. Catoe, and P. Krumviede. *IMAP/POP AUTHorize Extension for Simple Challenge/Response*, September 1997. RFC 2195.

58. L.M. Kohnfelder. *Towards a Practical Public-Key Cryptosystem*. Master's thesis, MIT, May 1978.

59. H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*, February 1997. RFC 2104.

60. RSA Laboratories. *PKCS 7: Cryptographic Message Syntax Standard*. Technical report, RSA Security, November 1993.

61. L. Lamport. *LaTeXUser's Guide and Reference Manual*. Addison-Wesley, 2nd edition, 5th printing edition, 1996.

62. A. Larsson. *Dia*. World Wide Web, `http://www.lysator.liu.se/~alla/dia/`, Last visited 9 November 2001.

63. P. Leach and C. Newman. *Using Digest Authentication as a SASL Mechanism*, May 2000. RFC 2831.

64. J. Linn. *Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures*, February 1993. RFC 1421.

65. J. Linn. *Generic Security Service Application Program Interface Version 2, Update 1*, January 2000. RFC 2743.

66. M. Nystrom and J. Brainard. *An X.509-Compatible Syntax for Compact Certificates.*, In Proceedings of Secure Networking, CQRE, LNCS 1740, 1999.

67. D. Massey, T. Lehman, and E. Lewis. *DNSSEC Implementation in the CAIRN Testbed*. Technical report, CAIRN, October 1999. Work in progress, Internet-Draft draft-ietf-dnsop-dnsseccairn-00.txt.

68. R.C. Merkle. *Secure Communication Over Insecure Channels*. *Communications of the ACM*, vol 21(no. 4):pp. 294–299, 1978.

69. *OSI X.400 Message Handling System Model*, 1984. section 2.2.1.

70. S.P. Miller, B.C. Neuman, J.I. Schiller, and J.H. Saltzer. *Kerberos Authentication and Authorization System*. MIT Project Athena Documentation Section E.2.1, December 1987.

71. P.V. Mockapetris. *Domain names—implementation and specification*, November 1987. RFC 1035.

72. K. Moore. *MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text*, November 1996. RFC 2047.

73. J. Myers. *Simple Authentication and Security Layer (SASL)*, October 1997. RFC 2222.

74. J. Myers and M. Rose. *Post Office Protocol—Version 3*, May 1996. RFC 1939.

75. *Netscape NetCenter*. World Wide Web, `http://home.netscape.com/netcenter/whitepages.html`, Last visited 9 November 2001.

76. NIC-SE. *Reports on DNSSEC*. World Wide Web, `http://www.nic-se.se/dnssec/`, Last visited 9 November 2001.

77. NIST. *Secure Hash Standard*. Technical Report FIPS PUB 180-1, April 1995.

78. M. Nystrom. *The SecurID(r) SASL Mechanism*, April 2000. RFC 2808.

79. *OpenLDAP*. World Wide Web, `http://www.openldap.org/`, Last visited 9 November 2001. version 1.2.9.

80. J. Palme. *History of the KOM computer conferencing system*. World Wide Web, `http://www.dsv.su.se/jpalme/s1/history-of-KOM.html`, Last visited 9 November 2001, October 1997.

81. J. Postel. *User Datagram Protocol*, August 1980. RFC 768.

82. J. Postel. *Internet Protocol*, September 1981. RFC 791.

83. J. Postel. *Transmission Control Protocol*, September 1981. RFC 793.

84. J. Postel. *Simple Mail Transfer Protocol*, August 1982. RFC 821.

85. B. Ramsdell and Ed. *S/MIME Version 3 Certificate Handling*, June 1999. RFC 2632.

86. B. Ramsdell and Ed. *S/MIME Version 3 Message Specification*, June 1999. RFC 2633.

87. R. Rivest. Can We Eliminate Certificate Revocation Lists. *Financial Cryptography, Rafael Hirschfeld, Ed., Anguilla, British West Indies*, vol. 1465:pp. 178–183, February 1998. Springer.

88. R.L. Rivest, A. Shamir, and L. Adleman. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. *Communications of the ACM*, 21(2):120–126, February 1978.

89. B. Schneier. *Applied Cryptography*. Wiley, second edition edition, 1996.

90. M. UMEDA and L.M. Ingebrigtsen et al. *Gnus*. World Wide Web, `http://www.gnus.org/`, Last visited 9 November 2001.

91. U.S. Department of Commerce. *Data Encryption Standard*. Federal Information Processing Standards Publication FIPS PUB 46, 1977. Republished as FIPS PUB 46-2 in 1994.

92. *VeriSign Directory*. World Wide Web, `http://digitalid.verisign.com/services/client/`, Last visited 9 November 2001.

93. P. Vixie, Ed., S. Thomson, Y. Rekhter, and J. Bound. *Dynamic Updates in the Domain Name System (DNS UPDATE)*, April 1997. RFC 2136.

94. P. Vixie, O. Gudmundsson, D. Eastlake, and B. Wellington. *Secret Key Transaction Authentication for DNS (TSIG)*, May 2000. RFC 2845.

95. M. Wahl, H. Alvestrand, J. Hodges, and R. Morgan. *Authentication Methods for LDAP*, May 2000. RFC 2829.

96. M. Wahl, T. Howes, and S. Kille. *Lightweight Directory Access Protocol (v3)*, December 1997. RFC 2251.

97. *Whitepages*. World Wide Web, `http://www.whitepages.com`, Last visited 9 November 2001.

98. P. R. Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.

Index

Abstract Syntax Notation, *see* ASN.1
administrative requirements, 27
America on line AIM, 14
ASN.1, 17, 36
asymmetric cipher, 3
asymmetric ciphers, 4
authentication, 4
authentication mode, 6

CA, *see* certificate authority
certificate
    generate requests, 24
    sign, 24
certificate authority, 11, 16
    verification process, 11
certificates, 7
    credential based, 27
    identity based, 27
    locating, 26
    samples, 81
    size, 40
    updating, 31
cipher, 3
ciphertext, 3
Client Configuration, 30
CMS, *see* Cryptographic Message Syntax
complexity, 7
confidentiality, 4
Cryptographic Message Syntax, 17
cryptography, 3

data integrity, 27
data mining, 45
data non-existence
    chain problem, 49
    in DNS, 47
    naive implementation, 47
    NO implementation, 50
    NXT implementation, 48
    proofs, 48
data origin authentication, 27
    in DNS, 46
decryption, 3
DES, 4
digital signature, 5
distinguished name, 16, 28
domain name system, 9, 13
    ASCII representation, 24
    caching, 34
    internal, 30
    locating directory, 30
    protocol, 35
    publishing certificates, 24
    secure DNS, 45
        design goals, 46

eavesdropper, 3
electronic mail, 9, 14
encryption, 3
ethics of information, 45

Fermat's small theorem, 5
Fidonet, 14

graphein, 3
GSS alogorithm for TSIG, 32

hashed message authentication codes, 4, 32
HTTP, 9, 15
HyperText Transfer Protocol, *see* HTTP

ICQ, 14
IETF, 8, 16, 17
IMAP, 15
instant messaging, 14
integrity, 4, 5
integrity check values, 4
internet, 9
Internet Engineering Task Force, *see* IETF
Internet Mail Message Format, *see* RFC 822
internet protocol
  version 4, 40
  version 6, 41
internet relay chat (IRC), 14
IP, 9
ISO X.500, 10

kerberos, 31
key distribution
  in DNS, 46
key-space, 3
keyed hash function, 4
Keys
  RSA, 24
keys, 3
KOM, 14
kryptos, 3

Latency, *see* Round Trips
LDAP, 36
  add request, 36
  add response, 36
  bind request, 36
  bind response, 36
  locating directory, 29
  modify request, 36
  modify response, 36
  protocol, 36
  publishing certificates, 24
  SASL credentials, 36
  search request, 36
  search result done, 36
  search result entry, 36
LDAP referrals, 29
LDIF, 24
Lightweight Directory Access Protocol, *see* LDAP

mailer, 14
man in the middle, 7
MEMO, 14
message authentication codes, 4
message integrity check, 4
message transfer agents (MTA), 14
messages, 3
MIME, 14, 15
  meta language, 22
  PGP, 17
  secure, 17
  secure multi-parts, 17
MOSS, 14, 17
Multipurpose Internet Mail Extensions, *see* MIME

NO resource record, 52, 65
nonrepudiation, 4
NXT chaining, 46

one-way hash value, 6
OpenPGP, 14
originator, 14

packet size, 40
password based authentication, 31
PEM, 14
PGP, 14
PGP/MIME, 17
phone book, 9
PKCS #7, 17
pki, *see* public key infrastructure
PKI hosting, 30
plaintext, 3
policy registration authority (PCA), 16
pretty good privacy (PGP), 17
privacy abuse, 45
privacy enhanced mail (PEM), 16
public chat groups, 14

public discussion forums, 14
public key, 4
public key cryptography, 4
public key infrastructure, 8, 11
public-key certificates, 7

RA, *see* registration authority
receivers, 3
recipients, 14
registration authority, 11
request authentication
    in DNS, 46
RFC 822, 15
round trips, 37
    comparison between
        DNS and LDAP, 40
    in DNS, 38
    in LDAP, 39
    in TCP, 37
    in UDP, 37
RSA, 5

S/MIME, 14, 17
secret writing, 3
Secure MIME, *see* MIME
Security Multiparts for MIME, 14
senders, 3
SGML, 22
shared secret key, 3
sign, 6
signature, 6
signature generation, 6
simple distributed security
        infrastructure (SDSI), 27
simple mail transfer protocol, 9
simple public key infrastructure, 27
SMS, 14
SRV records, 29
symmetric cipher, 3

TCP, 9
    round trips in, 37
transaction authentication
    in DNS, 46
Transmission Control
        Protocol, *see* TCP
trusted third party, 7

UDP
    round trips in, 37
up-to-date information, 27
user agents, 14
User Datagram Protocol, *see* UDP
users, 14

verification procedure, 6
verify, 6
VeriSign, 40

well known DNS aliases, 29
white-pages, 27

X.400, 14
X.500, 28
    attributes, 28
    DAP, 29
    DIT, 28
    entries, 28
XML, 22

Appendix A

NO Resource Records

The following is the technical specification of NO Resource Records [50] as discussed in chapter 6. It was submitted as a independent submission to the IETF DNS Extensions Work Group [18]. The Work Group adopted it is an official work item. This is the second version (not a final version) of the document.

Network Working Group S.A. Josefsson
Internet-Draft RSA Security
Expires: February 22, 2001 August 24, 2000

Authenticating denial of existence in DNS with minimum disclosure
(or; An alternative to DNSSEC NXT records)
draft-ietf-dnsext-not-existing-rr-00

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on February 22, 2001.

Abstract

This draft present an alternative to NXT records, used to achieve authenticated denial of existence of a domain name, class and type. Problems with NXT records, as specified in RFC 2535, are identified. One solution, the NO record, is presented.

The NO record differ from the NXT record by using a cryptographic hash value instead of the domain name. This prevent an adversery from collecting information by "chaining" through a zone. It also remove delegation point concerns in NXT records. The document also describe hash truncation and record merging that reduces storage/network load.

Table of Contents

1. Introduction 3
2. Context 3
3. The NO Resource Record 4
3.1 Idea 4
3.2 NO RDATA Format 4
3.3 NO RDATA on-the-wire format example 6
3.4 Owner Names 6
3.5 Additional Complexity Due To Wildcards 7
3.6 No Considerations at Delegation Points 7
3.7 Hash Truncation and Dynamic Updates 8
3.8 Reducing Number of Records 9
3.9 Hash Collisions 9
3.10 Authenticating Denial of NO Records 9
3.11 Case Considerations 10
3.12 Presentation Format 10
3.13 Examples 10
3.13.1 Adding NO Records to a Zone 10
3.13.2 Simple NO creating entity 11
3.13.3 Advanced NO creating entity 11
3.13.4 Resolver Query for Non-existing Domain 11
3.13.5 Resolver Query for Non-existing Type At Existing Domain . 12
4. Security Considerations 13
5. IANA Considerations 13
References 14
Author's Address 14
Full Copyright Statement 15

1. Introduction

"Domain Name Security Extensions", RFC 2535 [1], specifies several extensions to DNS that provides data integrity and authentication. Among them is the NXT record, used to achieve authenticated denial of existence of domains, and authenticated denial of existence on certain class/types on existing domains.

As a consequence of NXT records it is possible to "chain" through a zone secured by DNS security extensions, collecting all names and/or records in the zone. This is the main problem that motivated this draft.

2. Context

There have been arguments that the "chain" problem of NXT records is a non-issue. Often used is the argument that information in DNS is public, and if you wish to keep information private, you shouldn't publish it in DNS. This might be true, but nonetheless major service providers and companies are restricting access to zone transfers. Also, people have debated whether NXT records should be part of DNSSEC at all because of this problem [5].

Another aspect exist. When DNS is used to store certificates, as described in RFC 2538 [2], certificates can identify individuals and/or carry authentication information for special purposes. This context has been the motivation for developing this draft.

The delegation considerations for NXT records (different RRsets in the parent and child for the same domain) has also been regarded as a flaw of the NXT records.

3. The NO Resource Record

This section describe the NO Resource Record.

3.1 Idea

A straight-forward extension to the NXT record that minimize disclosure of information is to store a one-way function value instead of the actual domain name. This is similar to NXT records; where NXT records secure a interval where no existing domain names are to be found, NO records secure a interval where no one-way value of existing domain names are to be found.

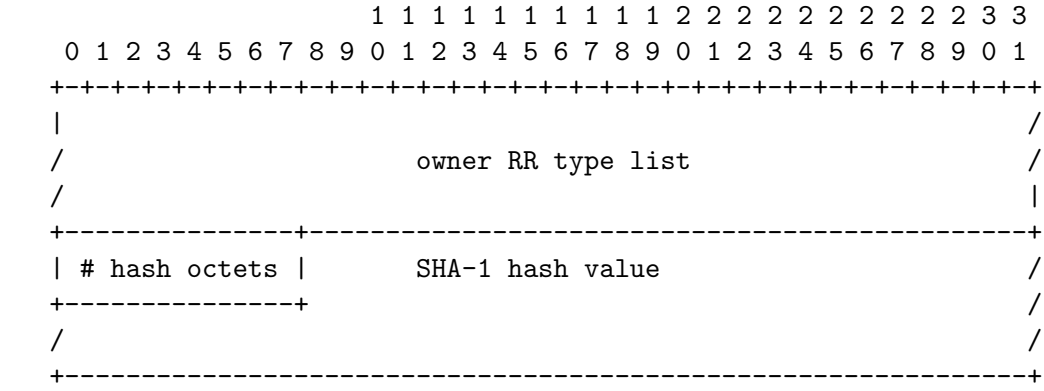
The benefit, of course, is that an adversary does not yield any useful information from the record. Specifically, "chaining" through a zone to collect all records is no longer possible.

This idea has been extended in this document into allowing (but not requiring) one record to deny existence of several records, and truncating the hash value to the shortest unique prefix to preserve

space.

3.2 NO RDATA Format

The RDATA for a NO RR consists of one or more fields with the following structure. The structure have the following elements: a zero-terminated list of RR types, a hash length specifier, and a hash value, as shown below. Both the "RR type" list and the "next hash value" fields are of variable width.



Replacing the NXT RR "type bit map" field is a variable length list of RR types. Each RR type is 16 bits. As the list is of variable length, a end-of-list indicator is require. End of the list is signalled by a all-zero RR type. Each element is a 2 byte RR type. The list MUST be sorted in RR type order. The change from NXT's bitmap field removes the limit of authenticating only the first 127 RR types.

The RR type list indicates what types exist at the previous hash value -- i.e. the first RR type list in the RRdata of a NO record indicate what RR types exist at the domain name that hashes to the owner name of that NO record. The second RR type list, if any, in the RRdata of a NO record indicate what RR types exist at the domain name that hashes the first SHA-1 value in the RRdata. And so on. See below for a complete example of the on-the-wire-format of a NO record with hash truncation and record merging and its interpretation.

Length of the hash value field is denoted by the # hash octets fields, it is a unsigned integer ranging from 0 to 255. The meaning of a zero length integer is reserved.

The SHA-1 hash value field is a variable length field containing the actual hash value.

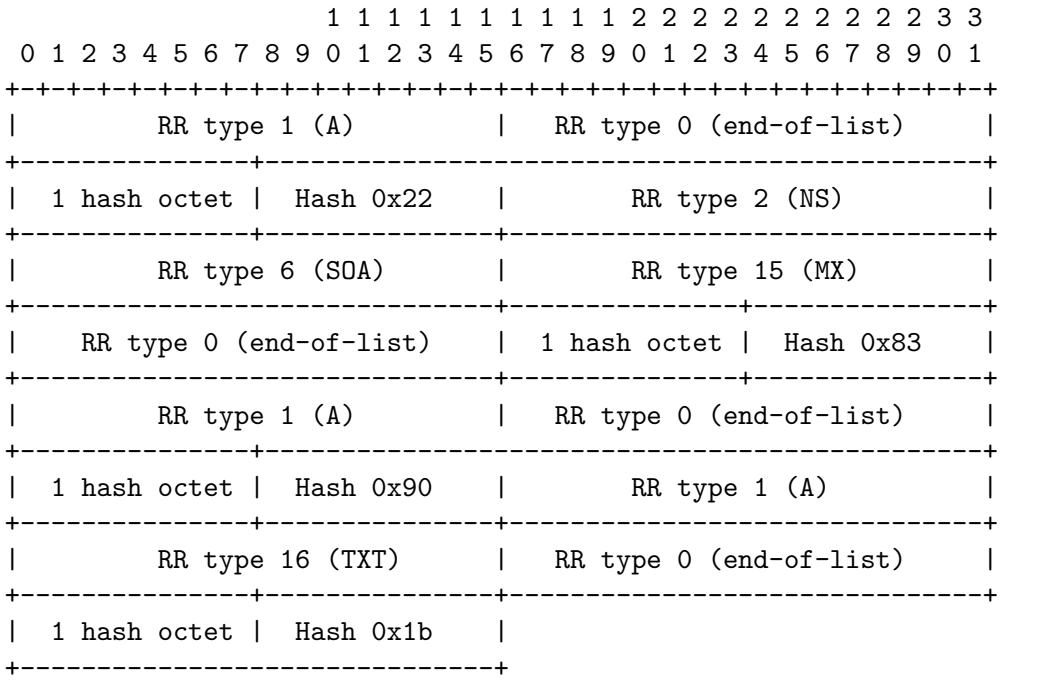
The NO RRs for a zone SHOULD be automatically calculated and added to the zone when SIGs are added. The NO RR's TTL SHOULD NOT exceed the zone minimum TTL.

The type number for the NO RR is TBD.

NO RR's MUST only be signed by zone level keys [7].

3.3 NO RDATA on-the-wire format example

The following is a example of the on-the-wire-format of a complete NO resource record were hash truncation and record merging is used. It is the on-the-wire format of the NO record in section 3.13.1.2.



The intepretation here is that the domain that corresponds with the NO owner name ("1b.example.org.", not shown above) have a A record, that the domain that hash to 0x22 (truncated to one octet) have a NS, SOA and MX record, that the domain that hash to 0x83 (truncated to 1 octet) have a A record etc. Note that the last hash value of NO RRdata does not have a RR type list in the same record.

3.4 Owner Names

As the previous NO RR format describe, the "next domain name" of NXT records is replaced by its hash value. This removes the possibility of chaining both backwards and forwards through a zone.

But without also modifying the owner names of NO records it is still not difficult to chain through a zone. Consider querying a server for (say) "m.example.org", the reply could contain a NO record for "g.example.org", now an adversary can lookup records for "g.example.org", and then issue a query for a domain that would sort (in "the canonical order" described in RFC 2535) just before "g.example.org". Applying the technique over and over again, all records in the zone can still be collected.

To prevent this, the owner names of NO records is replaced by its hash value. As DNS places limits on what characters can be used in owner names, the owner name uses a base 16 "hex" encoding [6].

In order to remove the (very small) chance of generated NO record names colliding with existing, "real", domains, all NO records MUST be stored directly in the "_no" domain of the zone. I.e., a zone "example.org." store its NO records as, say, "35a4d1._no.example.org.".

3.5 Additional Complexity Due To Wildcards

Proving that a non-existent name response is correct, or that a wildcard expansion response is correct, makes things a little more complex.

In particular, when a non-existent name response is returned, an NO must be returned showing that the exact name did not exist and, in general, one or more additional NO need to be returned to also prove that there wasn't a wildcard whose expansion should have been returned. (There is no need to return multiple copies of the same NO.) These NOs, if any, are returned in the authority section of the response.

Furthermore, if a wildcard expansion is returned in a response, in general one or more NOs needs to also be returned in the authority section to prove that no more specific name (including possibly more specific wildcards in the zone) existed on which the response should have been based.

3.6 No Considerations at Delegation Points

When NXT records are used to deny existance, there exists a special case at delegation points. Namely, that two distinct RRsets exist for the same owner name, one in the parent zone and one in the child zone.

```
$ORIGIN parent.example.org.
@       SOA
        NS
        NXT     child   SOA NS SIG NXT
child   NS      foo
        NXT     next    NS SIG NXT
next    A       127.0.0.2

$ORIGIN child.parent.example.org.
@       SOA
        NS
        NXT     name    SOA NS SIG NXT
name    A       127.0.2.1
        NXT     child.parent.example.org.
```

With NO records, the parent would deny existance of domains in "_no.parent.example" and the child in "_no.child.parent.example.org". Thus no NO RRset collision occur.

3.7 Hash Truncation and Dynamic Updates

Entities that create NO records MAY truncate the hash field. It MUST NOT truncate hash fields so they are no longer unique throughout a zone. Hash truncations MUST only be done to octet offsets. Truncation MUST be such that less significant bits are truncated, i.e. higher-order bits are preserved (see the examples).

Zones that are dynamically updated will have to calculate and add NO records on-the-fly. If hash truncation is used, adding a new name to the zone will require updating (and signing) NO records for the conflicting name(s).

As this recalculation might be quite inefficient, the use of "shortest unique prefix" truncation in dynamically updated zones is not recomended. However, a truncation to, say, 64 bits might be possible if the administrators are willing to have their software

perform costly operations once every ~2^32 update (on average).

Since truncation (and also "compression" described in the next section) make it impossible to predict the corresponding NO record given a domain name, resolvers should not ask for a hashed NO record (aaaa._no.example.org. IN NO) for a known domain name if they want to find out what types exist at the domain. Instead, a resolver might ask for NO records on the original name (www.example.org. IN NO). Such records will never exist, and the correct NO record will be returned by the server.

To summarize, the behaviour of hash truncation should be configurable in the entity that creates NO records, to accomodate different usage-patterns. If the zone is not intended to be dynamicly updated, the use of hash truncation reduces size and is recomended.

3.8 Reducing Number of Records

Entitities that create NO records MAY deny existence for several records per NO record. Entities that create NO records should take care so that each resulting NO record is not "too large". [Comments on this? Should there be a specific limit? It might be left as a DNS Operational consideration]

Merging several NO records into one record also place more work on the resolver. Instead of parsing two hash values for each NO record to determine if it's applicaple, a resolver will have to parse several hash values and compare each.

The NO RR record consist of one or more RR type list / hash values, described above, and a resolver need to parse the entire record to collect each individual field. I.e., a NO parse algorithm could looke like: read RRs, stop when you read a zero RR type, read hash length indicator, read hash size, if the entire record is read stop, otherwise read RRs, stop when you read a zero RR type, etc..

3.9 Hash Collisions

Hash value collisions are expected not to occur. For SHA-1, the probability that this should happen is 1 out of 2^80 on average.

However, collisions are actually only a problem if the domain names producing the same hash value have different sets of existing types.

```
Consider the following records

; SHA-1(one.example.org) = SHA-1(two.example.org)

one.example.org. IN A 1.2.3.4
two.example.org. IN A 5.6.7.8

Given that no other RR types exist for neither domain, both
"one.example.org" and "two.example.org" would be authenticated not
to exist by the same NO record.

3.10 Authenticating Denial of NO Records

NO records (together with SIG records) authenticate denial for other
types in a zone.  Unlike NXT records that re-use the namespace in
the zone, NO records are not intended to authenticate denial of
other NO records.

3.11 Case Considerations

Before calculating SHA-1 hash values, domain names MUST be converted
into canonical format as described in RFC 2535. This is to make hash
comparisons possible.

3.12 Presentation Format

NO RRs do not appear in original unsigned zone master files since
they should be derived from the zone as it is being signed.

If a signed file with NO records is printed or NO records are
printed by debugging code, they appear as a list of unsigned
integers or RR mnemonics, and the hash value in base 16 hex
representation [6] with "0x" prepended (to distinguish it from
integer RR codes).  The zero RR that terminate the list of RR types,
and the hash value length indicator, does not appear.

See the next section for examples of printed NO RRs.

3.13 Examples

This section contain examples of NO records, using the reserved
domain exmaple.org [3].
```

3.13.1 Adding NO Records to a Zone

Consider the following zone file.

```
$ORIGIN example.org.
@ IN SOA ...
@ IN NS ns
@ IN MX 0 server
ns IN A ...
www IN A ...
sERVEr IN A ...
sERVEr IN TXT "text"

; SHA1(example.org.)          = 0x222c7a74bc40e818aa53b3eb0b15cd2350fbb3a1
; SHA1(ns.example.org.)       = 0x1b7838c69a66eb50cc215f66ee4554d0c4c940a5
; SHA1(www.example.org.)      = 0x839ebd4386c0b26d81f147421b5b7036e61438cf
; SHA1(server.example.org.)   = 0x906a0ad5e604b1905828499dc8672ecb8de73e2f
```

Note that hash values are calculcated on the canonical form.

The following two sections describe two valid ways of adding NO records to a zone.

3.13.2 Simple NO creating entity

A simple NO creator entity might not implement truncation or provide the possibility to deny more than one records per NO record. In this case, the following would be added to the zone file.

```
$ORIGIN _no.example.org.
1b7838c69a66eb50cc215f66ee4554d0c4c940a5
 IN NO A 0x222c7a74bc40e818aa53b3eb0b15cd2350fbb3a1
222c7a74bc40e818aa53b3eb0b15cd2350fbb3a1
 IN NO NS SOA MX 0x839ebd4386c0b26d81f147421b5b7036e61438cf
839ebd4386c0b26d81f147421b5b7036e61438cf
 IN NO A 0x906a0ad5e604b1905828499dc8672ecb8de73e2f
906a0ad5e604b1905828499dc8672ecb8de73e2f
 IN NO A TXT 0x1b7838c69a66eb50cc215f66ee4554d0c4c940a5
```

3.13.3 Advanced NO creating entity

A more advanced NO creator entity might append the following instead, using both truncation and "compression".

```
$ORIGIN _no.example.org
1b IN NO A 0x22 NS SOA MX 0x83 A 0x90 A TXT 0x1b A
```

Note that this contain 5 hash values while the zone only contain 4 records, the last value in the line above is in fact the first hash value in the zone, closing the circular NO chain.

3.13.4 Resolver Query for Non-existing Domain

Consider a client looking up the non-existant domain name "baz.example.org.", using the zone file from the previous section. First, we note the following calculations.

```
SHA-1(baz.example.org.) = 0xd5d0f98783eec6e9943750f35904304bd1a4090e
SHA-1(*.example.org.)   = 0x7ab3776e3b529eb42467cc5d279c88ec951cf021
```

A server would reply with an RCODE of NXDOMAIN and the authority section data including the following:

```
; backwards compatibility
example.org. IN SOA

; prove no baz.example.org
906a0ad5e604b1905828499dc8672ecb8de73e2f.example.org.
 IN NO A TXT 0x1b7838c69a66eb50cc215f66ee4554d0c4c940a5
906a0ad5e604b1905828499dc8672ecb8de73e2f.example.org. IN SIG NO

; prove no *.example.org:
222c7a74bc40e818aa53b3eb0b15cd2350fbb3a1.example.org.
 IN NO NS SOA MX 0x839ebd4386c0b26d81f147421b5b7036e61438cf
222c7a74bc40e818aa53b3eb0b15cd2350fbb3a1.example.org. IN SIG NO
```

In order for a client to verify the authenticity of this reply, in addition of verifying the SIG record, it will also need to calculate the one-way hash of "baz.example.org." and verify it is contained inside the interval of any NO record in the authority section. Also, to prove there are no wildcard records for baz.example.org., NO records for possible wildcard expansions are returned. A client should similarily calculate hash values of possible wildcards and compare them to the authority section.

Of course, if the zone was generated with the more advanced NO creating entity, only the NO record from the previous section would have to be returned.

3.13.5 Resolver Query for Non-existing Type At Existing Domain

Consider a client looking up TXT records for the existing domain "www.example.org.", again, using the same zone file as previous. A server would reply with an authority section like the following:

```
839ebd4386c0b26d81f147421b5b7036e61438cf.example.org.
 IN NO A 0x906a0ad5e604b1905828499dc8672ecb8de73e2f
839ebd4386c0b26d81f147421b5b7036e61438cf.example.org. IN SIG NO
```

The resolver verifies the signature and make sure SHA-1("bar.example.org.") hashes correctly.

4. Security Considerations

Chaining through all NO records is still technically possible, altough it cannot be used to collect names and/or records in the zone (other than NO records themself).

The security of NO record hash values is dependent on the security of the SHA-1 hash functions used.

It should be pointed out that given this scheme, it is easy to estimate the number of records within a zone, considering hash values are supposed to be equally distributed. This can be foiled by adding any number of bogus records in the zone.

The authentication of NO records is provided by DNS SIG records, as specified in RFC 2535. The security considerations of RFC 2535 is not affected by this document, and should also be considered.

5. IANA Considerations

The NO RR type number should be selected by the IANA from the normal RR type space.

The meaning of a zero hash length value can only be assigned by a standards action.

Acknowledgements

The idea of encrypting names, that later evolved into just hashing them, was originally proposed by Jonas Holmerin in private

discussions about DNS Security. Magnus Nyström proposed truncating the hash values.

Thanks to John Linn and Magnus Nyström for comments on a early version of this draft.

Olafur Gudmundsson pointed out delegation point issues, suggested the use of a "_no" subdomain, and suggested replacing the type bit map field with a sorted list. From the namedroppers mailing list, I'd like to thank Andrew Draper, Andreas Gustafsson, Peter Koch and Brian Wellington for comments and suggestions.

References

[1] Eastlake, D., "Domain Name System Security Extensions", RFC 2535, March 1999.

[2] Eastlake, D. and O. Gudmundsson, "Storing Certificates in the Domain Name System (DNS)", RFC 2538, March 1999.

[3] Eastlake, D. and A. Panitz, "Reserved Top Level DNS Names", RFC 2606, June 1999.

[4] NIST, , "Secure Hash Standard", FIPS PUB 180-1, April 1995.

[5] Massey, D., Lehman, T. and E. Lewis, "DNSSEC Implementation in the CAIRN Testbed.", I.D. draft-ietf-dnsop-dnsseccairn-00.txt, October 1999.

[6] Josefsson, S.A. (editor), "Base 64, 32 and 16 Encodings", I.D. draft-josefsson-base-encoding-00.txt, August 2000.

[7] Wellington, B, "Domain Name System Security (DNSSEC) Signing Authority", I.D. draft-ietf-dnsext-signing-auth-01.txt, May 2000.

Author's Address

Simon Josefsson
RSA Security
Arenavägen 29
121 29 Stockholm
Sweden

Phone: +46 8 7250914
EMail: sjosefsson@rsasecurity.com

Full Copyright Statement

Acknowledgement

Funding for the RFC editor function is currently provided by the Internet Society.

Appendix B

Sample Certificates

This appendix contains text-versions of the Certificates that were used in section 4.4.5. This is intended as a detailed reference when comparing the amount of additional information (names, addresses etc) that was stored in the certificates we used. The Certificates were prepared using Open SSL [23].

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: O=S. Josefsson CA, OU=Class 1 Public Primary Certification Authority, CN=S. Josefsson CA
        Validity
            Not Before: Aug 25 10:46:59 2000 GMT
            Not After : Aug 25 10:46:59 2001 GMT
        Subject: CN=User 0/Email=user0@josefsson.org
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (512 bit)
                Modulus (512 bit):
                    00:ad:68:34:e6:fb:f1:91:fa:06:53:4f:ed:e0:05:
                    4c:58:c8:5b:74:db:19:e0:45:4d:34:41:5d:ee:6a:
                    40:ab:04:75:61:57:84:88:4b:45:62:4b:28:41:76:
                    d9:ba:2e:b8:04:c6:b2:c7:11:d2:8d:31:07:7a:9d:
                    b9:ec:0a:54:75
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                email:user0@josefsson.org
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Authority Key Identifier:
                keyid:0C:C8:A6:BD:22:C2:F5:2C:79:43:95:A2:72:FC:EB:3B:37:0E:9E:66

            X509v3 Extended Key Usage:
                TLS Web Client Authentication, E-mail Protection
    Signature Algorithm: md5WithRSAEncryption
        8f:94:d9:65:34:87:c9:3b:66:31:1a:a4:ee:dd:87:d9:f0:d2:
        51:ac:f1:5b:76:53:41:53:4e:50:6b:a0:2c:8b:43:f1:f4:83:
        a9:91:9b:16:00:a6:f2:10:74:e2:d8:e3:88:6d:dc:bd:d2:2f:
        5c:1c:3b:aa:9b:49:92:d1:39:58
-----BEGIN CERTIFICATE-----
MIICAzCCAa2gAwIBAgIBATANBgkqhkiG9w0BAQQFADBtMRgwFgYDVQQKEw9TLiBK
b3NlZnNzb24gQ0ExNzA1BgNVBAsTLkNsYXNzIDEgUHVibGljIFByaW1hcnkgQ2Vy
dGlmaWNhdGlvbiBBdXRob3JpdHkxGDAWBgNVBAMTD1MuIEpvc2Vmc3NvbiBDQTAe
Fw0wMDA4MjUxMDQ2NTlaFw0wMTA4MjUxMDQ2NTlaMDUxDzANBgNVBAMTBlVzZXIg
MDEiMCAGCSqGSIb3DQEJARYTdXNlcjBAam9zZWZzc29uLm9yZzBcMA0GCSqGSIb3
DQEBAQUAA0sAMEgCQQCtaDFNORDR+gZTT+3gBUxYyFt02xngRU0OQV3uakCrBHVh
V4SISOViSyhBdtm6LrgExrLHEdKNMQd6nbnsClR1AgMBAAGjcDBuMB4GA1UdEQQX
MBWBE3VzZXIwQGpvc2Vmc3Nvbi5vcmcwDAYDVR0TAQH/BAIwADAfBgNVHSMEGDAW
gBQMyKa9IsL1LHlDlaJy/Os7Nw6eZjAdBgNVHSUEFjAUBggrBgEFBQcDAgYIKwYB
BQUHAwQwDQYJKoZIhvcNAQEEBQADQQCPlNllNIfJO2YxGqTu3YfZ8NJRrPFbdlNB
U05Qa6AsiOPx9IOpkZsWAKbyEHTi2OOIbdy90i9cHDuqm0mSOTlY
-----END CERTIFICATE-----
```

Figure B.1. 512 bit RSA certificate.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: O=S. Josefsson CA, OU=Class 1 Public Primary Certification Authority, CN=S. Josefsson CA
        Validity
            Not Before: Aug 25 10:45:37 2000 GMT
            Not After : Aug 25 10:45:37 2001 GMT
        Subject: CN=User 0/Email=user0@josefsson.org
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:9b:48:7c:10:6d:49:bf:96:a1:fa:63:3c:22:21:
                    58:93:a1:f5:9d:d8:d8:5a:a3:f2:bb:d7:fc:18:c8:
                    7a:8f:ce:da:f8:82:eb:ad:c5:1a:ef:66:34:d2:56:
                    e2:4b:3a:82:1e:ca:68:06:95:a7:51:9a:ac:55:66:
                    e7:12:8c:77:cb:eb:eb:89:a0:05:73:a4:c5:df:4b:
                    8b:a0:db:9b:5e:5d:2f:ed:45:be:80:0d:f3:5d:90:
                    2b:b4:81:95:8f:ca:56:ab:41:4d:4c:7d:d5:00:03:
                    71:f7:3e:8b:10:6a:12:d6:3d:08:12:fe:38:c4:6c:
                    8d:b3:1e:85:5e:f3:c3:16:43
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                email:user0@josefsson.org
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Authority Key Identifier:
                keyid:0C:C8:A6:BD:22:C2:F5:2C:79:43:95:A2:72:FC:EB:3B:37:0E:9E:66

            X509v3 Extended Key Usage:
                TLS Web Client Authentication, E-mail Protection
    Signature Algorithm: md5WithRSAEncryption
        5b:f7:8e:7c:0a:30:a5:71:b6:82:e3:a4:4d:24:16:0f:ef:be:
        b9:28:41:a7:95:9e:cd:b3:64:f7:b4:bb:e5:89:f4:7f:fc:15:
        63:b4:f6:bb:ad:42:f8:16:32:98:01:e1:67:48:f6:e9:c2:a1:
        0e:b2:e9:75:d0:e4:0c:0b:d1:e3
-----BEGIN CERTIFICATE-----
MIICRzCCAfGgAwIBAgIBATANBgkqhkiG9w0BAQQFADBtMRgwFgYDVQQKEw9TLiBK
b3NlZnNzb24gQOExNzA1BgNVBAsTLkNsYXNzIDEgUHVibGljIFByaW1hcnkgQ2Vy
dGlmaWNhdGlvbiBBdXRob3JpdHkxGDAWBgNVBAMTD1MuIEpvc2Vmc3NvbiBDQTAe
Fw0wMDA4MjUxMDQ1MzdaFw0wMTA4MjUxMDQ1MzdaMDUxDzANBgNVBAMTBlVzZXIg
MDEiMCAGCSqGSIb3DQEJARYTdXNlcjBAam9zZWZzc29uLm9yZzCBnzANBgkqhkiG
9w0BAQEFAAOFNORDgYkCgYEAm0h8EG1Jv5ah+mM8IiFYk6H1ndjYWqPyu9f8GMh6
j87a+ILrrcUa72YOOlbiSzqCHspoBpWnUZqsVWbnEox3y+vriaAFc6TF3OuLoNub
Xl0v7UW+gA3zXZArtIGVj8pWqOFNTH3VAANx9z6LEGoS1jOIEv44xGyNsx6FXvPD
FkMCAwEAAaNwMG4wHgYDVRORBBcwFYETdXNlcjBAam9zZWZzc29uLm9yZzAMBgNV
HRMBAf8EAjAAMB8GA1UdIwQYMBaAFAzIpr0iwvUseUOVonL86zs3Dp5mMB0GA1Ud
JQQWMBQGCCsGAQUFBwMCBggrBgEFBQcDBDANBgkqhkiG9w0BAQQFAANBAFv3jnwK
MKVxtoLjpEOkFg/vvrkoQaeVns2zZPeOu+WJ9H/8FWOO9rutQvgWMpgB4WdI9unC
oQ6y6XXQ5AwLOeM=
-----END CERTIFICATE-----
```

Figure B.2. 1024 bit RSA certificate.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: O=S. Josefsson CA, OU=Class 1 Public Primary Certification Authority, CN=S. Josefsson CA
        Validity
            Not Before: Aug 25 10:47:54 2000 GMT
            Not After : Aug 25 10:47:54 2001 GMT
        Subject: CN=User 0/Email=user0@josefsson.org
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:eb:7b:bc:4c:5d:48:2c:80:ac:39:2e:ac:1e:9f:
                    88:5c:27:22:e7:3d:0a:b4:56:ca:de:90:05:2c:aa:
                    d7:c9:87:30:b6:8b:cb:67:07:1f:c6:51:0d:05:b0:
                    20:fb:0a:02:73:63:21:88:56:a8:9c:fa:f7:09:10:
                    c4:ea:c0:eb:49:f6:66:2b:e6:b0:cd:d7:93:b4:62:
                    a9:e8:5d:48:62:1e:99:ff:f2:a9:60:45:8a:02:ab:
                    16:50:7c:8a:ab:c7:5f:09:d8:c2:f2:02:24:90:bd:
                    57:2d:2c:99:be:11:69:85:d0:09:1f:98:cf:bd:a6:
                    bb:84:83:bc:cb:1e:55:ae:0c:29:39:1e:51:41:18:
                    ab:fb:4f:ff:02:b8:7a:f2:17:e0:72:61:36:28:69:
                    dc:e8:54:2d:b3:af:b9:65:9e:b3:25:59:17:37:66:
                    d5:d8:ec:ee:13:1a:6a:11:84:4b:dd:05:2b:f4:b9:
                    70:10:ab:ab:a3:12:2d:b7:bf:df:f3:0d:1f:cc:fe:
                    a9:6e:53:db:d0:e7:7a:a1:45:ff:79:c9:2e:9b:74:
                    0d:5a:43:2f:0b:a5:69:b9:5c:80:63:7c:04:67:bd:
                    26:a3:10:b2:b7:4a:07:d1:32:0b:40:fd:47:3f:61:
                    c4:70:45:69:ed:7f:12:d2:c8:34:76:62:1a:a2:07:
                    5b:cf
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                email:user0@josefsson.org
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Authority Key Identifier:
                keyid:0C:C8:A6:BD:22:C2:F5:2C:79:43:95:A2:72:FC:EB:3B:37:0E:9E:66

            X509v3 Extended Key Usage:
                TLS Web Client Authentication, E-mail Protection
    Signature Algorithm: md5WithRSAEncryption
        89:96:56:ba:71:ee:97:3c:ee:28:a4:f8:9e:ff:eb:a1:15:01:
        08:86:69:e5:b0:95:b5:fd:b6:ae:c0:b6:db:76:fd:85:e0:6e:
        55:03:76:04:ac:39:7e:66:d9:c3:9c:a6:3a:76:74:9b:d6:8c:
        61:5a:22:0d:f4:2f:aa:0a:52:c3
-----BEGIN CERTIFICATE-----
MIICyzCCAnWgAwIBAgIBATANBgkqhkiG9w0BAQQFADBtMRgwFgYDVQQKEw9TLiBK
b3NlZnNzb24gQ0ExNzA1BgNVBAsTLkNsYXNzIDEgUHVibGljIFByaW1hcnkgQ2Vy
dGlmaWNhdGlvbiBBdXRob3JpdHkxGDAWBgNVBAMTD1MuIEpvc2Vmc3NvbiBDQTAe
Fw0wMDA4MjUxMDQ3NTRaFw0wMTA4MjUxMDQ3NTRaMDUxDzANBgNVBAMTBlVzZXIg
MDEiMCAGCSqGSIb3DQEJARYTdXNlcjBAam9zZWZzc29uLm9yZzCCASIwDQYJKoZI
hvcNAQEBBQADggEPADCCAQoCggEBAOt7vExdSCyArDkurB6fiFwnIuc9CrRWyt6Q
BSyq18mHMLaLy2cHH8ZRDQWwIPsKAnNjIYhWqJz69wkQxOrA6On2ZivmsM3Xk7Ri
qehdSGIemf/yqWBFigKrFlB8iqvHXwnYwvICJJC9Vy0smb4RaYXQCR+Yz72mu4SD
vMseVa4MKTkeUUEYq/tP/wK4evIX4HJhNihp3OhULbOvuWWesyVZFzdm1djs7hMa
ahGES9OFK/S5cBCrq6MSLbe/3/MNH8z+qW5T29DneqFF/3nJLpt0DVpDLwulablc
gGN8BGe9JqMQsrdKB9EyCOD9Rz9hxHBFae1FNORDNHZiGqIHW88CAwEAAaNwMG4w
HgYDVR0RBBcwFYETdXNlcjBAam9zZWZzc29uLm9yZzAMBgNVHRMBAf8EAjAAMB8G
A1UdIwQYMBaAFAzIpr0iwvUseUOVonL86zs3Dp5mMB0GA1UdJQQWMBQGCCsGAQUF
BwMCBggrBgEFBQcDBDANBgkqhkiG9w0BAQQFAANBAImWVrpx7pc87iik+J7/66EV
AQiGaeWwlbX9tq7Attt2/YXgblUDdgSsOX5m2cOcpjp2dJvWjGFaIg3OL6oKUsM=
-----END CERTIFICATE-----
```

Figure B.3. 2048 bit RSA certificate.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: O=S. Josefsson CA, OU=Class 1 Public Primary Certification Authority, CN=S. Josefsson CA
        Validity
            Not Before: Aug 25 10:43:17 2000 GMT
            Not After : Aug 25 10:43:17 2001 GMT
        Subject: CN=User 0/Email=user0@josefsson.org
        Subject Public Key Info:
            Public Key Algorithm: dsaEncryption
            DSA Public Key:
                pub:
                    48:62:07:ea:59:e7:8f:72:a6:af:96:8b:8a:ba:36:
                    76:79:16:10:c9:3b:3c:cc:be:1b:d3:bc:19:61:f0:
                    f1:e5:6e:f8:e4:27:57:19:36:cb:48:3f:00:9d:fc:
                    c0:21:ce:33:bf:7e:05:08:c2:df:c0:be:76:d1:3d:
                    e8:c0:1a:c5
                P:
                    00:c4:f0:7d:21:b0:21:3f:3d:ba:36:d0:42:92:51:
                    0a:68:7f:a2:63:5f:34:a6:1b:62:46:22:e8:6d:47:
                    23:18:c5:9c:eb:0f:4a:ba:81:dc:dc:66:30:d9:d3:
                    83:ea:e7:3b:78:7c:00:6b:6a:5b:91:9c:0a:14:2f:
                    b6:0c:0e:97:dd
                Q:
                    00:c2:50:de:0b:7d:44:f0:e9:c6:59:fa:c3:d5:ce:
                    fe:c0:c8:e6:ca:0b
                G:
                    4e:aa:29:bf:ba:35:20:a8:0b:c4:72:25:b0:f6:0e:
                    2f:68:10:e6:e6:7b:67:4e:d0:96:bb:43:95:82:9f:
                    e1:ce:7b:3f:5c:b8:26:4a:c9:bb:f7:45:05:7e:c9:
                    f8:38:2c:50:73:64:ca:14:d7:22:3c:17:b1:38:04:
                    73:fa:12:d0
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                email:user0@josefsson.org
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Authority Key Identifier:
                keyid:0C:C8:A6:BD:22:C2:F5:2C:79:43:95:A2:72:FC:EB:3B:37:0E:9E:66

            X509v3 Extended Key Usage:
                TLS Web Client Authentication, E-mail Protection
    Signature Algorithm: md5WithRSAEncryption
        47:ea:97:3c:5d:32:8e:f0:79:72:a8:df:34:54:a7:2c:b1:a9:
        ea:16:71:94:84:38:6c:65:43:af:4e:aa:7b:af:5b:cf:5a:87:
        05:42:0c:4e:21:1d:8a:12:55:5b:c6:0e:e6:57:80:81:24:1c:
        33:80:8e:c9:99:08:60:7a:f1:1b
-----BEGIN CERTIFICATE-----
MIICmDCCAkKgAwIBAgIBATANBgkqhkiG9w0BAQQFADBtMRgwFgYDVQQKEw9TLiBK
b3NlZnNzb24gQOExNzA1BgNVBAsTLkNsYXNzIDEgUHVibGljIFByaW1hcnkgQ2Vy
dGlmaWNhdGlvbiBBdXRob3JpdHkxGDAWBgNVBAMTD1MuIEpvc2Vmc3NvbiBDQTAe
Fw0wMDA4MjUxMDQzMTdaFw0wMTA4MjUxMDQzMTdaMDUxDzANBgNVBAMTBlVzZXIg
MDEiMCAGCSqGSIb3DQEJARYTdXNlcjBAam9zZWZzc29uLm9yZzCB8DCBqAYHKoZI
zjgEATCBnAJBAMTwfSGwIT89ujbQQpJRCmh/omNfNKYbYkYi6G1HIxjFnOsPSrqB
3NxmMNnTg+rnO3h8AGtqW5GcChQvtgwOl9OCFQDCUN4LfUTw6cZZ+sPVzv7AyObK
CwJATqopv7o1IKgLxHIlsPYOL2gQ5uZ7ZO7QlrtDlYKf4c57P1y4JkrJu/dFBX7J
+DgsUHNkyFNORDwXsTgEc/oSOANDAAJASGIH6lnnj3Kmr5aLiro2dnkWEMk7PMy+
G9O8GWHw8eVu+OQnVxk2yOg/AJ38wCHOM79+BQjC38C+dtE96MAaxaNwMG4wHgYD
VR0RBBcwFYETdXNlcjBAam9zZWZzc29uLm9yZzAMBgNVHRMBAf8EAjAAMB8GA1Ud
IwQYMBaAFAzIpr0iwvUseUOVonL86zs3Dp5mMB0GA1UdJQQWMBQGCCsGAQUFBwMC
BggrBgEFBQcDBDANBgkqhkiG9w0BAQQFAANBAEfqlzxdMo7weXKo3zRUpyyxqeoW
cZSEOGxlQ69OqnuvW89ahwVCDE4hHYoSVVvGDuZXgIEkHDOAjsmZCGB68Rs=
-----END CERTIFICATE-----
```

Figure B.4. 512 bit DSA certificate.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: O=S. Josefsson CA, OU=Class 1 Public Primary Certification Authority, CN=S. Josefsson CA
        Validity
            Not Before: Aug 25 10:35:12 2000 GMT
            Not After : Aug 25 10:35:12 2001 GMT
        Subject: CN=User 0/Email=user0@josefsson.org
        Subject Public Key Info:
            Public Key Algorithm: dsaEncryption
            DSA Public Key:
                pub:
                    3e:f8:dd:27:33:e9:dd:e9:04:4d:25:39:26:4c:78:
                    42:18:88:15:b6:65:8b:3d:22:d4:72:73:fb:0d:5d:
                    6e:fa:d4:d7:6f:02:35:ec:49:65:c4:8e:26:43:7e:
                    07:47:90:a0:5f:04:f1:7e:88:65:7a:e5:5b:f7:c6:
                    40:19:cb:8e:b2:2f:da:a5:96:60:51:2e:2e:55:ff:
                    5d:eb:be:40:ca:d4:1a:31:2e:ea:a2:8a:02:56:33:
                    9e:89:3a:99:5a:5f:01:dc:1d:b2:81:1f:22:ba:1d:
                    c5:2f:39:49:27:d2:ac:7b:68:f0:a1:4e:46:30:e8:
                    2a:54:9b:37:9e:87:93:83
                P:
                    00:d2:93:cf:b3:9d:1a:61:ae:f5:4b:55:39:b3:c8:
                    88:3e:10:28:d2:81:4f:11:a6:c3:32:6b:cf:bc:4a:
                    cd:6f:0a:4c:39:52:4d:7b:f7:b5:36:49:07:ff:64:
                    2b:9d:50:6b:4c:3a:2e:1f:1d:fa:1e:a6:9b:71:40:
                    ef:f9:e5:dd:32:27:c8:b5:6b:52:6f:d9:cf:f3:96:
                    c0:ed:ee:e5:a2:39:99:c5:76:fb:83:cf:3f:ad:cb:
                    7e:a5:6f:a6:34:67:c6:fe:c7:ed:fb:4b:ef:e3:d3:
                    ec:e3:19:15:e0:74:9f:b2:a6:32:43:dc:75:2a:6f:
                    c4:e0:65:e9:6c:45:14:06:1f
                Q:
                    00:c4:fc:6a:88:d3:93:5b:df:16:55:70:54:ca:f7:
                    56:2f:72:2a:fd:87
                G:
                    19:37:a5:2a:2b:23:9b:69:ae:b3:90:56:54:e4:4a:
                    e9:7e:9e:38:e2:83:98:84:1c:46:40:0e:6d:2d:95:
                    4c:0e:38:83:7f:78:4c:29:a3:03:5c:1d:5b:b9:13:
                    1b:57:4b:c8:97:a0:e1:e4:db:a6:bb:5e:60:02:e5:
                    16:f9:76:c1:02:f7:24:fa:4a:ed:ca:b2:f1:14:35:
                    54:0b:53:f8:60:c7:ac:a9:6e:fd:4c:36:3f:5d:8d:
                    d3:3a:7a:63:53:d0:1a:c4:df:2f:3b:46:d1:ff:87:
                    cd:03:ef:f9:3d:e0:fb:12:5f:75:12:f0:2d:ed:e1:
                    55:a0:6c:cf:1d:d5:d9:bf
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                email:user0@josefsson.org
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Authority Key Identifier:
                keyid:0C:C8:A6:BD:22:C2:F5:2C:79:43:95:A2:72:FC:EB:3B:37:0E:9E:66

            X509v3 Extended Key Usage:
                TLS Web Client Authentication, E-mail Protection
    Signature Algorithm: md5WithRSAEncryption
        69:4c:80:55:c4:61:16:14:72:21:aa:56:2d:d7:da:46:75:84:
        c0:36:5d:b4:dd:ba:d5:3a:cb:34:9c:7b:c4:d8:75:66:ab:d2:
        53:6c:0b:79:76:9d:51:07:30:0f:48:4c:54:77:68:43:df:5b:
        9b:59:db:04:5b:2d:c8:0a:56:04
-----BEGIN CERTIFICATE-----
MIIDXzCCAwmgAwIBAgIBATANBgkqhkiG9w0BAQQFADBtMRgwFgYDVQQKEw9TLiBK
b3NlZnNzb24gQ0ExNzA1BgNVBAsTLkNsYXNzIDEgUHVibGljIFByaW1hcnkgQ2Vy
dGlmaWNhdGlvbiBBdXRob3JpdHkxGDAWBgNVBAMTD1MuIEpvc2Vmc3NvbiBDQTAe
Fw0wMDA4MjUxMDM1MTJaFw0wMTA4MjUxMDM1MTJaMDUxDzANBgNVBAMTBlVzZXIg
MDEiMCAGCSqGSIb3DQEJARYTdXNlcjBAam9zZWZzc29uLm9yZzCCAbYwggErBgcq
hkjOOAQBMIIBHgKBgQDSk8+znRphrvVLVTmzyIg+ECjSgU8RpsMya8+8Ss1vCkw5
Uk1797U2SQf/ZCudUGtMOi4fHfoepptxQO/55d0yJ8i1a1Jv2c/zlsDt7uWiOZnF
dvuDzz+ty36lb6YOZ8b+x+37S+/j0+zjGRXgdJ+ypjJD3HUqb8TgZelsRRQGHwIV
AMT8aojTk1vfFlVwVMr3Vi9yKv2HAoGAGTelKisjm2mus5BWVORK6X6eOOKDmIQc
RkAObS2VTA44g394TCmjA1wdW7kTG1dLyJeg4eTbprteYALlFvl2wQL3JPpK7cqy
8RQ1VAtT+GDHrKlu/Uw2P12NOzp6Y1PQGsTfLzFNORDHzQPv+T3g+xJfdRLwLe3h
VaBszx3V2b8DgYQAAoGAPvjdJzPp3ekETSU5Jkx4QhiIFbZlizOi1HJz+w1dbvrU
128CNexJZcSOJkN+BOeQoF8E8X6IZXrlW/fGQBnLjrIv2qWWYFEuLlX/Xeu+QMrU
GjEu6qKKAlYznok6mVpfAdwdsoEfIrodxS85SSfSrHto8KFORjDoKlSbN56Hk4Oj
cDBuMB4GA1UdEQQXMBWBE3VzZXIwQGpvc2Vmc3Nvbi5vcmcwDAYDVROTAQH/BAIw
ADAfBgNVHSMEGDAWgBQMyKa9IsL1LHlDlaJy/Os7Nw6eZjAdBgNVHSUEFjAUBggr
BgEFBQcDAgYIKwYBBQUHAwQwDQYJKoZIhvcNAQEEBQADQQBpTIBVxGEWFHIhqlYt
19pGdYTANl2O3brVOss0nHvE2HVmq9JTbAt5dp1RBzAPSExUd2hD31ubWdsEWy3I
ClYE
-----END CERTIFICATE-----
```

Figure B.5. 1024 bit DSA certificate.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            2e:10:37:03:df:46:85:9d:7a:55:0d:a6:59:61:85:38
        Signature Algorithm: md5WithRSAEncryption
        Issuer: O=VeriSign, Inc., OU=VeriSign Trust Network, OU=www.verisign.com/repository/RPA Incorp. By Ref.,LIAB.LTD(c)98,
        CN=VeriSign Class 1 CA Individual Subscriber-Persona Not Validated
        Validity
            Not Before: Jun 26 00:00:00 2000 GMT
            Not After : Aug 25 23:59:59 2000 GMT
        Subject: O=VeriSign, Inc., OU=VeriSign Trust Network, OU=www.verisign.com/repository/RPA Incorp. by Ref.,LIAB.LTD(c)98,
         OU=Persona Not Validated, OU=Digital ID Class 1 - Netscape, CN=Simon Josefsson/Email=simon@josefsson.org
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:c9:0c:ce:8a:fe:71:46:9b:ca:1d:e5:90:12:a5:
                    11:0b:c6:2d:c4:33:c6:19:e8:60:59:4e:3f:64:3d:
                    e4:f7:7b:b0:be:f9:10:07:e9:7c:a6:c6:5a:51:33:
                    24:97:7b:a3:e1:08:b4:52:b6:06:10:7d:65:df:6e:
                    52:bd:81:3f:39:ad:b3:ad:17:13:88:22:e7:43:8c:
                    39:b7:c2:c4:ba:4a:8b:54:15:49:55:a4:4d:cc:00:
                    56:7b:c8:63:4e:37:de:fb:79:0f:45:dc:e9:5c:cd:
                    70:f0:64:42:35:84:db:e6:59:a4:cb:4b:fe:0f:47:
                    28:0c:35:11:a9:40:fc:ba:a5
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Certificate Policies:
                Policy: 2.16.840.1.113733.1.7.1.8
                  CPS: https://www.verisign.com/rpa

            Netscape Cert Type:
                SSL Client
            2.16.840.1.113733.1.6.3:
                .vd4652bd63f2047029298763c9d2f275069c7359bed1b059da75bc4bc9701747da5d5e4141beadb2bd2e88317af7bf5d5114997a3bf45f8f3ea450c
            X509v3 CRL Distribution Points:
                URI:http://crl.verisign.com/class1.crl

    Signature Algorithm: md5WithRSAEncryption
        09:38:2f:57:9e:91:a4:d2:42:d9:d7:44:c1:d8:17:14:49:00:
        69:9f:6b:e4:95:93:35:fd:96:76:ff:8b:bf:9e:dd:05:6b:82:
        b2:f3:af:0f:f8:a0:2f:8d:65:08:27:54:d4:8f:47:79:c9:be:
        d9:f9:ce:af:7f:2a:06:17:26:f3:b9:e6:74:ba:b9:35:3e:36:
        56:5d:41:9c:ce:68:fc:db:c5:31:42:09:32:37:e7:b7:2e:a4:
        c5:51:e5:fe:e5:45:59:0c:44:ca:ce:ad:77:24:52:b4:78:5f:
        cc:4f:15:a7:8f:20:81:56:65:08:50:37:75:bc:a2:11:82:72:
        48:76
-----BEGIN CERTIFICATE-----
MIIEhDCCA+2gAwIBAgIQLhA3A99GhZ16VQ2mWWGFODANBgkqhkiG9w0BAQQFADCB
zDEXMBUGA1UEChMOVmVyaVNpZ24sIEluYy4xHzAdBgNVBAsTFlZlcmlTaWduIFRy
dXNOIE5ldHdvcmsxRjBEBgNVBAsTPXd3dy52ZXJpc2lnbi5jb20vcmVwb3NpdG9y
eS9SUEEgSW5jb3JwLiBCeSBSZWYuLExJQUIuTFREKGMpOTgxSDBGBgNVBAMTP1Zl
cmlTaWduIENsYXNzIDEgQOEgSW5kaXZpZHVhbCBTdWJzY3JpYmVyLVBlcnNvbmEg
Tm90IFZhbGlkYXRlZDAeFw0wMDA2MjYwMDAwMDBaFw0wMDA4MjUyMzU5NTlaMIIB
CDEXMBUGA1UEChMOVmVyaVNpZ24sIEluYy4xHzAdBgNVBAsTFlZlcmlTaWduIFRy
dXNOIE5ldHdvcmsxFNORDgNVBAsTPXd3dy52ZXJpc2lnbi5jb20vcmVwb3NpdG9y
eS9SUEEgSW5jb3JwLiBieSBSZWYuLExJQUIuTFREKGMpOTgxHjAcBgNVBAsTFVBl
cnNvbmEgTm90IFZhbGlkYXRlZDEmMCQGA1UECxMdRG1naXRhbCBJRCBDbGFzcyAx
ICOgTmV0c2NhcGUxGDAWBgNVBAMUD1NpbW9uIEpvc2Vmc3NvbjEiMCAGCSqGSIb3
DQEJARYTc2ltb25Aam9zZWZzc29uLm9yZzCBnzANBgkqhkiG9w0BAQEFAAOBjQAw
gYkCgYEAyQzOiv5xRpvKHeWQEqURC8YtxDPGGehgWU4/ZD3k93uwvvkQB+l8psZa
UTMkl3uj4QiOUrYGEH1l325SvYE/Oa2zrRcTiCLnQ4w5t8LEukqLVBVJVaRNzABW
e8hjTjfe+3kPRdzpXM1w8GRCNYTb5lmky0v+D0coDDURqUD8uqUCAwEAAaOCASYw
ggEiMAkGA1UdEwQCMAAwRAYDVROgBD0wOzA5BgtghkgBhvhFAQcBCDAqMCgGCCsG
AQUFBwIBFhxodHRwczovL3d3dy52ZXJpc2lnbi5jb20vcnBhMBEGCWCGSAGG+EIB
AQQEAwIHgDCBhgYKYIZIAYb4RQEGAwR4FnZkNDY1MmJkNjNmMjAONzAyOTI5ODc2
M2M5ZDJmMjc1MDY5YzczNTliZWQxYjA1OWRhNzViYzRiYzk3MDE3NDdkYTVkNWUO
MTQxYmVhZGIyYmQyZTg4MzE3YWY3YmY1ZDUxMTQ5OTdhM2JmNDVmOGYzZWEONTBj
MDMGA1UdHwQsMCowKKAmoCSGImh0dHA6Ly9jcmwudmVyaXNpZ24uY29tL2NsYXNz
MS5jcmwwDQYJKoZIhvcNAQEEBQADgYEACTgvV56RpNJC2ddEwdgXFEkAaZ9r5JWT
Nf2Wdv+Lv57dBWuCsvOvD/igL41lCCdU1I9Hecm+2fnOr38qBhcm87nmdLq5NT42
Vl1BnM5o/NvFMUIJMjfnty6kxVHl/uVFWQxEys6tdyRStHhfzE8Vp48ggVZlCFA3
dbyiEYJySHY=
-----END CERTIFICATE-----
```

Figure B.6. VeriSign 1024 bit RSA certificate.

Appendix C

Benchmarking Tool

This appendix contains source code of the benchmarking tool used in 4.4.6. It is included here for inspection of how the test proceeded, points to note are that only TCP is used in the DNS case and that the TCP connection is closed between every connection.

```
/* DNS/LDAP performance program,
 * by Simon Josefsson <sjosefsson@rsasecurity.com>
 *
 * Compile with
 *
 * cc -g -o bench bench.c dnssec.c res_searchN.c -lresolv -lldap -llber -pg -a
 *
 * If running on linux, you need to run this:
 *
 * sysctl -w net.ipv4.ip_local_port_range="1024     30000"
 *
 * otherwise DNS will flood your local tcp ports.
 *
 * Example:
 *
 * ./bench ldap 172.16.13.119 "cn=User 5, dc=josefsson, dc=org" 5000
 * ./bench dns 172.16.13.119 user5.josefsson.org 5000
 */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>
#include <netdb.h>
#include "dnssec.h"
#include <lber.h>
#include <ldap.h>


#define USAGE "Usage: %s dns <server> <domain name> <iterations>\n" \
              "          ldap <server> <ldap address> <iterations>\n"

int
ldap (int argc, char *argv[])
{
    LDAP *ld;
    LDAPMessage *result, *e;
    BerElement *ber;
    char *a, *dn;
    char **vals;
    int i = 0;
    char *attrs[2];

    for (;;)
    {
        if ((ld = ldap_init(argv[2], LDAP_PORT)) == NULL) {
            perror("ldap_init");
            return 1;
        }


        if (ldap_simple_bind_s(ld, NULL, NULL) != LDAP_SUCCESS) {
            ldap_perror(ld, "ldap_simple_bind_s");
            return 1;
        }

        attrs[0] = "usercertificate;binary";
        attrs[1] = NULL;

        if (ldap_search_s(ld, argv[3], LDAP_SCOPE_SUBTREE,
                          "(objectClass=*)", attrs, 0,
                          &result) != LDAP_SUCCESS) {
            ldap_perror(ld, "ldap_search_s");
            return 1;
        }
```

```
        e = NULL;
        e = ldap_first_entry(ld, result);
        if (!e) {
            fprintf(stdout, "no answer in query\n");
            return 1;
        }

        vals = NULL;
        vals = ldap_get_values(ld, e, "usercertificate;binary");
        if (!vals) {
            fprintf(stdout, "no answer in query\n");
            return 1;
        }

        ldap_msgfree(result);
        ldap_unbind(ld);

        if ((i % 100) == 0) {
            printf("Query ok %d...\r", i);      fflush(stdout);
        }
        i++;
        if (i == atoi(argv[4])) {
            fprintf(stdout, "Ok, done %d iterations\n", atoi(argv[4]));
            return 0;
        }
    }

    return 1;
}

int
dns (int argc, char *argv[])
{
    struct rrinfo *rr, *r, rhint;
    extern int h_errno;
    struct hostent *h;
    int i = 0;

    bzero (&rhint, sizeof (struct rrinfo));

    res_init();
    _res.options |= RES_USEVC; // use TCP
    //_res.options |= RES_STAYOPEN; // do not use!  apples and oranges

    h = gethostbyname(argv[2]);
```

```
    if (!h)
        fprintf(stderr, "Can't find name server '%s'!\n", argv[2]);
    else {
        _res.nsaddr_list[0].sin_addr.s_addr =
            ((struct in_addr*) h->h_addr_list[0])->s_addr;
    }

    for (;;)
    {
        if (getcertinfo (argv[3], NULL, &rr) != 0) {
            fprintf(stderr, "query failed. h_errno = %d\n", h_errno);
            return 1;
        }

        {
            int flag = 0;

            for (r = rr; r; r = r->next)
                if (r->type == T_CERT)
                {
                    flag = 1;
                }

            if (!flag) {
                fprintf(stderr, "No answer in response...\n");
                return 1;
            }
        }
        freerrinfo (rr);

        if ((i % 100) == 0) {
            printf("Query ok %d...\r", i);      fflush(stdout);
        }
        i++;
        if (i == atoi(argv[4])) {
            fprintf(stdout, "Ok, done %d iterations\n", atoi(argv[4]));
            return 0;
        }
    }

    return 1;
}
```

```
int
main (int argc, char *argv[])
{
    int ret = 0;
    int i;

    if (argc < 5) {
        printf(USAGE, argv[0]);
        return 1;
    }

    if (strcmp(argv[1], "dns") == 0) {
        ret = dns(argc, argv);
    } else if (strcmp(argv[1], "ldap") == 0) {
        ret = ldap(argc, argv);
    } else {
        printf("Syntax error\n");
        printf(USAGE, argv[0]);
        ret = 1;
    }

    return ret;
}
```