The Open–source PKI Book

A guide to PKIs and Open–source Implementations

Symeon (Simos) Xenitellis
OpenCA Team

The Open–source PKI Book: A guide to PKIs and Open–source Implementations
by Symeon (Simos) Xenitellis

The Open–source PKI Book Version 2.4.6 Edition

This document describes Public Key Infrastructures, the PKIX standards, practical PKI functionality and gives an overview of available open–source PKI implementations. Its aim is foster the creation of viable open–source PKI implementatations.

The latest version of this document can be found at the OSPKI Book WWW site at *http://ospkibook.sourceforge.net/*.

Revision History

Revision 0.6 18 Dec 1999 Revised by: S.Xenitellis@rhbnc.ac.uk
First public distribution, version in LinuxDoc
Revision 0.7 15 Jan 2000 Revised by: S.Xenitellis@rhbnc.ac.uk
Initial conversion to DocBook
Revision 0.8 20 Jan 2000 Revised by: S.Xenitellis@rhbnc.ac.uk
Major additions (still not published)
Revision 0.9 05 Feb 2000 Revised by: S.Xenitellis@rhbnc.ac.uk
Additions in appendices, installation details(still not published)
Revision 0.91 07 Feb 2000 Revised by: S.Xenitellis@rhbnc.ac.uk
Further polishing (still not published)
Revision 1.0 16 Feb 2000 Revised by: S.Xenitellis@rhbnc.ac.uk
First public release
Revision 1.1 2 May 2000 Revised by: S.Xenitellis@rhbnc.ac.uk
Use of the FDL license, cosmetic changes
Revision 2.0 13 May 2000Revised by: S.Xenitellis@rhbnc.ac.uk
Review by Ted Rolle, added critical discussion/colophon, updated program version numbers
Revision 2.1 13 May 2000Revised by: S.Xenitellis@rhbnc.ac.uk
Added PKIX chapter
Revision 2.3 14 May 2000Revised by: S.Xenitellis@rhbnc.ac.uk
Added chapter on implentations
Revision 2.4.516 June 2000Revised by: S.Xenitellis@rhbnc.ac.uk

Populating the implementations chapter
Revision 2.4.626 June 2000Revised by: S.Xenitellis@rhbnc.ac.uk
Fixed HTML production, better Makefile, minor doc changes
Revision 2.4.723 July 2000 Revised by: S.Xenitellis@rhbnc.ac.uk
Updated CDSA, added SSO/PAM/FPKI/APKI

Table of Contents

1. Purpose of this document **11**

2. Introduction to Cryptography **13**

- 2.1. Cryptographic Algorithms 13
- 2.2. Message Digests 14
- 2.3. Digital Signatures 14
- 2.4. Certificates 15
- 2.5. Certification Authority 15

3. Basic functionality of a Public Key Infrastructure[TODO] **17**

- 3.1. Creation of the key–pair and the certificate request 17
- 3.2. Signing of the certificate request by the Certification Authority 17
- 3.3. Certification Authority chains 17
- 3.4. Typical uses of public key cryptography 18

4. General implementation overview **19**

- 4.1. Prerequisites 19
  - 4.1.1. Useful open–source software 19
- 4.2. Initialisation of the Certification Authority 19
  - 4.2.1. Generate the RSA key–pair for the CA 20
  - 4.2.2. Create a self–signed CA Certificate 20
- 4.3. User/Server key generation and signing 22
  - 4.3.1. Generate the RSA key–pair for a user/server 22
  - 4.3.2. Generate a certificate request 23
  - 4.3.3. Ask the CA to sign the certificate request 24

5. PKI standards and specifications **27**

- 5.1. Internet X.509 Public Key Infrastructure (PKIX) 27
- 5.2. Architecture for Public-Key Infrastructure (APKI) 27
- 5.3. The NIST Public Key Infrastructure Program 27

6. Internet X.509 Public Key Infrastructure (PKIX) **29**

- 6.1. Abbreviations 29
- 6.2. Concepts 30
  - 6.2.1. Certificate–using Systems and PKIs 30
  - 6.2.2. Certificate–using Systems and PMIs 31
- 6.3. Overview of the PKIX approach 32
  - 6.3.1. PKIX standardisation areas 32
  - 6.3.2. Public–key infrastructure functionality 34
  - 6.3.3. Public–Key Infrastructure (PKI) 34
  - 6.3.4. Privilege Management Infrastructure (PMI) 36

7. Open-Source Implementations **39**

- 7.1. The pyCA Certification Authority 39
- 7.2. The OpenCA Project[TODO] 39
  - 7.2.1. OpenCA Layout 39

7.2.2. OpenCA Abbreviations 40
7.2.3. Software packages 41
7.2.4. Functionality of the CA Server *(CAServer)* 42
7.2.4.1. Initialisation / CA Management 42
7.2.4.2. Requests 44
7.2.4.3. Certificates 44
7.2.4.4. Certificate Revocation List CRL 45
7.2.5. Functionality of the RA Server *(RAServer)* 45
7.2.5.1. Requests 45
7.2.5.2. Certificates 46
7.2.5.3. Certificate Revocation List CRL 46
7.2.5.4. Miscellaneous Utilities 47
7.2.6. Functionality of the RA Operators *(RAOperators)* 47
7.2.6.1. Get Root CA Certificate 48
7.2.6.2. Certificate Revocation Lists 48
7.2.6.3. Request a Certificate 48
7.2.6.4. Get Requested Certificate 48
7.2.6.5. Issued Certificates List 49
7.2.7. Status of the OpenCA Project 49
7.2.8. Future OpenCA work 49
7.3. The Oscar Public Key Infrastructure Project 50
7.4. Jonah: Freeware PKIX reference implementation 50
7.5. Mozilla Open Source PKI projects 51
7.5.1. Personal Security Manager (PSM) 51
7.5.2. Network Security Services (NSS) 52
7.5.3. JavaScript API for Client Certificate Management 52
7.6. MISPC Reference Implementation 53
8. How to get software support 55
9. Supported Crypto hardware and Software architectures 57
9.1. TrustWay Crypto PCI 2000 57
9.2. PowerCrypt Encryption Accelerator 57
9.3. CryptoSwift eCommerce Accelerator 57
9.4. Movement for the Use of Smart Cards in a Linux Environment (MUSCLE) 58
9.5. Linux Smart Card Starter's Kit from Schlumberger 58
9.6. The gpkcs11 PKCS#11 open–source implementation 58
9.7. Common Data Security Architecture (CDSA) 58
9.8. Single Sign–on 59
9.9. The KeyMan PKI Management Tool 60
9.10. Distributed Audit Service (XDAS) 60
9.11. Generic Security Service API (GSS-API) 60
9.12. Simple Network Time Protocol (SNTP) 61
9.13. Lightweight Directory Access Protocol (LDAP) 61
9.14. S/MIME CMS [TODO] 61

10. Critical discussion[TODO] **63**
11. Benefits of an Open–Source PKI implementation[TODO] **65**
12. Trademarks **67**
13. Contributions **69**
A. Perl modules **71**
A.1. Locating Perl modules 71
A.2. Installing Perl modules 71
B. Sample Certificate Documents **73**
B.1. Sample Encrypted Private Key in PEM format (2048 bits) 73
B.2. Sample Private Key in PEM format (2048 bits) 74
B.3. Sample Private Key in TXT format (2048 bits) 74
B.4. Sample CA Certificate in PEM format 78
B.5. Sample CA Certificate in TXT format 78
B.6. Sample certificate request in PEM format 80
B.7. Sample certificate request in TXT format 81
C. Description of Public Key Algorithms **83**
C.1. How does RSA work? 83
C.1.1. Description 83
C.1.2. Practical example 84
C.2. How does El Gamal work? 84
C.2.1. Description 84
C.2.2. Example 84
D. OpenCA Installation details **85**
D.1. Software installation sequence 85
D.1.1. Installation of Perl modules 85
D.1.2. Installation of OpenCA–specific modules 86
D.1.3. Installation of OpenCA 86
D.1.3.1. CAServer Installation 87
D.1.3.2. RAServer Installation 88
D.1.3.3. RAOperator Installation 89
D.1.4. WWW Server installation 90
D.1.5. LDAP installation 90
D.2. `openssl.cnf` configuration for OpenCA 90
E. License **93**
E.1. GNU Free Documentation License 93
E.1.1. PREAMBLE 93
E.1.2. APPLICABILITY AND DEFINITIONS 93
E.1.3. VERBATIM COPYING 94
E.1.4. COPYING IN QUANTITY 94
E.1.5. MODIFICATIONS 95
E.1.6. COMBINING DOCUMENTS 97
E.1.7. COLLECTIONS OF DOCUMENTS 97

E.1.8. AGGREGATION WITH INDEPENDENT WORKS 97
E.1.9. TRANSLATION 98
E.1.10. TERMINATION 98
E.1.11. FUTURE REVISIONS OF THIS LICENSE 98
Colophon **99**
Glossary **101**
Bibliography **111**

List of Tables

6-1. PKIX Terms 29
6-2. Table of RFCs for PKIX documents 33
6-3. PKI functionality 34
6-4. PKI components 34
6-5. PMI components 36
7-1. OpenCA Abbreviations 40
7-2. Current Versions of OpenCA prerequisite software 41
8-1. WWW Support Locations 55
D-1. Software installation matrix 85
D-2. CAServer installation parameters 87
D-3. RAServer WWW Server installation parameters 87
D-4. RAServer installation parameters 88
D-5. RAServer WWW Server installation parameters 89
D-6. RAOperator WWW Server installation parameters 89

List of Figures

6-1. PKI Entities 35
6-2. Attribute Certificate Exchanges 36
7-1. Current OpenCA Layout 40

Chapter 1. Purpose of this document

This document tries to serve as a source of information on Public Key Infrastructures (PKIs) and focuses on both of the theoretic and practical description of PKIs.

With relation to specific standards, the work of the PKIX Working Group[1] is presented. There is an emphasis on these standards and there is an attempt to classify implementations according to the degree of compliance.

This document starts with an introduction on public–key cryptography. Then, it describes several publicly available implementations providing a feature by feature comparison.

There is a further discussion on security issues with accordance with PKIs.

The implementations are presented as an educational instrument to test the protocols, to provide a source of feedback and to enable the individual to learn more about the wonderfull world of PKIs.

Note: The latest version of this document can be found at the OSPKI Book WWW site [2].

Notes

1. http://www.ietf.org/html.charters/pkix-charter.html
2. http://ospkibook.sourceforge.net/

Chapter 2. Introduction to Cryptography

Communication is an essential part of life. We can say that it marks the progress of human beings. Traditional media for communication are the sending of letters through the Post Office, talking over the phone through the Telecommunications company, or – more commonly – to speak directly with the other person. These traditional media have existed for a long period of time and special provisions have been made so that people can communicate in a secure way, either for personal or for business communication. For face–to–face communication, people can recognise each other's physical characteristics or they can compare hand–written signatures with that of official documents like an ID card. Mimicking all of the physical characteristics of a person is difficult. People can accept with a high level of certainty the identity of their colleague. Signature forging is difficult and there are laws that define forging as a crime. The bottom line is that for each communication medium, there is a transitional period when specific laws and technologies are set in order for people to communicate securely and transparently.

The Internet, as a network that interconnects networks of computers around the world, is a new communication medium that is substantially different from existing ones. For example, on the Internet, the communicating parties do not have physical contact. It is rather more difficult for one to disguise oneself to someone else, immitate the voice and other aspects behaviour and get information on prior common experiences. On–line transactions do not impose such barriers for illegitimate transactions. Additionally, on the Internet, one can automate the same type of fraud bringing higher gains and a bigger incentive. The law and the technologies to let transparent and secure communication have not been fully defined or set yet.

Cryptography has provided us with digital signatures that resemble in functionality the hand–written signatures and digital certificates that relate to an ID card or some other official document. However, in order to use these technologies, we need to make the necessary provisions so that their usage is equally transparent and secure.

The Public Key Infrastructures along with the Priviledge Management Infrastructure are candidates to aid this transparency and security of applications of the Internet. Both of these concepts are described in Chapter 6.

Big parts of the following introduction to cryptography has been taken from the SSLeay Certificate Cookbook[1], written by Frederick J. Hirsch[2].

2.1. Cryptographic Algorithms

Cryptography has several differences from pure mathematics. One of these is that cryptography is more descriptive in its textbooks. While a mathematician may use A and B to explain an algorithm, a cryptographer may use the fictious names Alice and Bob. Thus, in the next sections, the names Alice and Bob are not randomly chosen; they can be found in almost all cryptography textbooks.

Suppose Alice wants to send a message to her bank to transfer money. Alice would like the message to be private, since it includes information such as her account number and transfer amount. One solution is to use a cryptographic algorithm, a technique that would transform her message into an encrypted form, unreadable except by those for whom it is intended. When encrypted, the message can only be interpreted through the use of the corresponding secret key. Without the key the message is useless: good cryptographic algorithms make it so difficult for intruders to decode the original text that it isn't worth their effort.

There are two categories of cryptographic algorithms: conventional and public key.

Conventional cryptography, also known as symmetric cryptography, requires that the sender and receiver share a key: a secret piece of information that is used to encrypt or decrypt a message. If this key is secret, then nobody other than the sender or receiver can read the message. If Alice and the bank each has a secret key, then they may send each other private messages. The task of privately choosing a key before communicating, however, can be problematic.

Public key cryptography, also known as asymmetric cryptography, solves the key exchange problem by defining an algorithm which uses two keys, each of which can be used to encrypt a message. If one key is used to encrypt a message, then the other must be used to decrypt it. This makes it possible to receive secure messages by simply publishing one key (the public key) and keeping the other secret (the private key).

Anyone may encrypt a message using the public key, but only the owner of the private key is able to read it. In this way, Alice may send private messages to the owner of a key–pair (the bank) by encrypting it using their public key. Only the bank can decrypt it.

Examples of public–key algorithms can be found at Appendix C.

2.2. Message Digests

Although Alice may encrypt her message to make it private, there is still a concern that someone might modify her original message message or substitute it with a different one in order to transfer the money to themselves, for instance. One way of guaranteeing the integrity of Alice's message is to create a concise summary of her message and send this to the bank as well. Upon receipt of the message, the bank creates its own summary and compares it with the one Alice sent. If they agree then the message was received intact.

A summary such as this is called a message digest, one–way function, or hash function. Message digests create short, fixed–length representations of longer, variable–length messages. Digest algorithms are designed to produce unique digests for different messages. Message digests make it difficult to determine the message from the digest, and difficult to find two different messages which create the same digest — eliminating the possibility of substituting one message for another while maintaining the same digest.

Another challenge that Alice faces is finding a way to send the digest to the bank securely; when this is achieved, the integrity of the associated message is assured. One way to to this is to include the digest in a digital signature.

2.3. Digital Signatures

When Alice sends a message to the bank, the bank needs to ensure that the message is really from her, and not an intruder requesting a transaction involving her account. A digital signature, created by Alice and included with the message, serves this purpose.

Digital signatures are created by encrypting a digest of the message, and other information (such as a sequence number) with the sender's private key. Though anyone may decrypt the signature using the public key, only the signer knows the private key. This ensures that only the signator signed it. Including the digest in the signature means the signature is only good for that message; it also ensures the integrity of the message since no one can change the digest and still sign it.

To guard against interception and reuse of the signature by an intruder at a later date, the signature contains an unique sequence number. This protects the bank from a fraudulent claim from Alice that she did not send the message — only she could have signed it (non–repudiation).

2.4. Certificates

Although Alice could have sent a private message to the bank, signed it, and ensured the integrity of the message, she still needs to be sure that she is really communicating with the bank. This means that she needs to be sure that the public key she is using corresponds to the bank's private key. Similarly, the bank also needs to verify that the message signature really corresponds to Alice's signature.

If each party has a certificate which validates the other's identity, confirms the public key, and is signed by a trusted agency, then they both are assured that they are communicating with whom they think they are. Each party uses the public key of the trusted agency to verify the certificate of the other party and subsequently to ensure the authenticity of the users' public key.

2.5. Certification Authority

The trusted agency that signs Certificates with its private key and lets others verify Certificates by the usage of the corresponding public key is called a Certification Authority, or CA. This Certification Authority is also known as a Trusted Third Party (TTP), since it is regarded that, in order to be trusted, it should not have common interests with any of the two parties.

> **Note:** It is believed that a bank should not be a Certification Authority and also be the party that you make economic transactions with. They reason is that, as a CA, it can favor itself, as a party one does business with.

In this document we concentrate on the technical aspects of the Certification Authority.

Notes

1. http://www.ultranet.com/~fhirsch/Papers/cook/ssl_cook.html
2. http://www.ultranet.com/~fhirsch/Papers/cook/ssl_intro.html

Chapter 3. Basic functionality of a Public Key Infrastructure[TODO]

Alice wants to communicate securely with Bob. In essence, this means that Alice does not want someone else to listen to the conversation, wants the information sent to Bob not to be altered on their way to him and finally she would possibly like a mechanism to prove that she had this conversation, in case, for some reason, claim he did not. We shall describe all the steps necessary to establish communication using the Certification Authority.

3.1. Creation of the key–pair and the certificate request

Alice creates a public/private key pair using a public key algorithm like RSA. Then, she creates a certificate request, which is the Certificate just prior to signing by the Certification Authority. First, the certificate request contains information about the identity of the user, such as the name, address, telephone number and e–mail address. Second, it contains her public key. Certificates can be used to authenticate not only people but also entities in general, such as a WWW server or a network device. In the latter case, the information in the Certificate would be the URL of the WWW server, the WWW Administrator details, and so on.

3.2. Signing of the certificate request by the Certification Authority

Alice sends her certificate request to the Registration Authority for its signature. Any action of approval or disapproval takes place at the Registration Authority. Then, the RA sends the request to the CA for policy approval and to be signed. The result of the signing — the Certificate — is sent back to Alice through the Registration Authority. They also are often stored on a Directory Server.

3.3. Certification Authority chains

Using this certificate, Alice can claim that her public key is trustworthy. Bob who wants to communicate with her, asks for her Certificate. Bob, in order to verify her Certificate, finds the public key of the Certification Authority that signed the Alice's public key. He needs to do that securely. If they are both on the same Certification Authority then he has it already. If not, he asks his Certification Authority to contact the other Certification Authority for its public key. For each Certification Authority Bob's Certification Authority asks, he needs the public key of the previous one so that the authenticity of the key is assured. If a chain can be found that leads to the other Certification Authority then communication can be established.

Note: The issue of inter-CA trust is very important since one bad CA can undermine the security of the whole infrastructure. This issue is not covered here (at least in this version).

3.4. Typical uses of public key cryptography

Having the authentic public keys of each other, users can communicate securely. They can encrypt data and make use of digital signatures. For the part of encryption, public key cryptography is too slow to be used for the transfer of large quantity of data. A symmetric cipher is more suited to this purpose. For this reason, the key for the symmetric cipher is transferred encrypted using public–key cryptography.

Chapter 4. General implementation overview

We give a technical overview of the processes of creating a certificate and operating a Certification Authority.

4.1. Prerequisites

We shall discuss here the software needed to create a usable Certification Authority.

4.1.1. Useful open–source software

The following software can provide the collective functionality of a Certification Authority.

- For the Certification Authority Server, any operating system can be used. In case it communicates manually with the Registration Authority (for example, data files are transfered using a floppy disk), it does not even need to have network support. However, it is recommended to use operating systems that provide some sort of assurance of its stability and can have irrelevant system or network services easily removed. We recommend Unix™ or Unix™–like operating systems.
- SSL/TLS software
- WWW server with SSL/TLS support
- LDAP server
- Text/Graphical Interface, possibly in Java/HTML

Note: The PKIX standards do not suggest nor forbid the use of a WWW server for the role of a CA/RA. To remove the need to create standalone network applications for both the CA and RA, it is possible to use individual WWW servers operated by designated Operators.

4.2. Initialisation of the Certification Authority

Here we describe the initialisation phase of the CA. This takes place once. Special care is needed for the protection of the CA's private key.

Note: The following examples require the OpenSSL software installed on your workstation. Also, it is recommended to have the directory that the openssl application resides, in your PATH environment variable. Possible locations for the openssl application are `/usr/local/ssl/bin/` or `/usr/bin/`.

4.2.1. Generate the RSA key–pair for the CA

Use this command to generate the RSA key–pair:

```
CA_Admin% openssl genrsa -des3 -out ca.key 2048
```

Parameters

genrsa

the openssl component to generate an RSA key–pair,

-des3

the symmetric algorithm to encrypt the key–pair,

-out `ca.key`

the filename to store the key–pair,

2048

size of RSA modulus in bits.

Executing the above command, the user is presented with the following information

```
1112 semi-random bytes loaded
Generating RSA private key, 2048 bit long modulus
.+++++
.....................................................+++++
e is 65537 (0x10001)
Enter PEM pass phrase: enter the pass-phrase here
Verifying password - Enter PEM pass phrase: re-enter
the pass-phrase here
```

This creates an RSA key pair which is stored in the file `ca.key`. This key pair is encrypted with 3DES using a password supplied by the user during key generation. The N in RSA (the product of the two prime numbers) is 2048 bits long. For brevity, we say that we use `2048-bit` RSA.

A sample key–pair, encrypted with a pass–phrase, can be found at Section B.1. This same key–pair without the pass–phrase encryption is at Section B.2. The decoded version of the same key can be found at Section B.3.

4.2.2. Create a self–signed CA Certificate

In order to get a self–signed CA Certificate, we need to sign the CA's certificate request with the corresponding private key. The resulting Certificate has the X.509 structure.

```
CA_Admin% openssl req -new -x509 -days 365 -key ca.key -out ca.crt
```

Parameters

req

the openssl component to generate a certificate request,

-new

this is a new certificate,

-x509

generate an X.509 certificate,

-days 365

the time in days that the certificate will be valid, counting from now,

-key `ca.key`

the key–pair file to be used,

-out `ca.crt`

the filename that the new certificate will be written onto

Executing the above command presents this dialogue:

```
Using configuration from /usr/local/ssl/openssl.cnf
Enter PEM pass phrase:  enter the pass-phrase here
You are about to be asked to enter information that will be
incorporated into your certificate request.
What you are about to enter is what is called a Distinguished
Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
---
Country Name (2 letter code) [AU]:GB
State or Province Name (full name) [Some-State]:Surrey
Locality Name (eg, city) []:.
Organization Name (eg, company) [Internet Widgits Pty Ltd]: Best CA Ltd
```

```
Organizational Unit Name (eg, section) []:Class 1 Public Primary Certifica-
tion Authority
Common Name (eg, YOUR name) []:Best CA Ltd
Email Address []:.
CA_Admin%
```

This creates a self–signed certificate, called `ca.crt`. It is valid for 365 days from the date of generation. In this step, the CA Administrator has to enter the X.509 details of the CA Root Certificate.

A sample CA Certificate, in PEM format, can be found at Section B.4. The TXT or human–readable of the same Certificate can be found at Section B.5.

4.3. User/Server key generation and signing

The user generates a key pair for a certificate to be used by that user or any entity that needs to be authenticated by the CA. We also show the signing procedure.

4.3.1. Generate the RSA key–pair for a user/server

Use this command to generate the RSA key pair

```
User% openssl genrsa -des3 -out user.key 2048
```

Parameters

genrsa

the openssl component to generate an RSA key–pair,

-des3

the symmetric algorithm to encrypt the key–pair,

-out `user.key`

the filename to store the key–pair,

2048

size of RSA modulus in bits.

Execution of the above command presents the user with the following dialogue:

```
1112 semi-random bytes loaded
Generating RSA private key, 2048 bit long modulus
```

```
.+++++
........................................................++++++++++++
e is 65537 (0x10001)
Enter PEM pass phrase: enter the pass-phrase here
Verifying password - Enter PEM pass phrase: re-enter pass-phrase here
```

This creates an RSA key pair stored in the file `user.key`. The key pair is encrypted with 3DES with a password supplied by the user during key generation. The N in RSA is 2048 bits long.

The reader should note that this is the same procedure as the generation of the CA key–pair. For sample key–pairs, please see the appendices listed in Section 4.2.1.

4.3.2. Generate a certificate request

The user generates a certificate request with this command. The CSR is sent to the CA for signing. The CA returns the the signed certificate.

```
User% openssl req -new -key user.key -out user.csr
```

Parameters

req

the openssl component to generate a certificate request,

-new

this is a new certificate,

-key `user.key`

the key–pair file to be used,

-out `user.csr`

the filename that the new certificate request will be written onto

By executing the above command, we are presented with the following dialogue:

```
Using configuration from /usr/local/ssl/openssl.cnf
Enter PEM pass phrase:  type the pass-phrase here
You are about to be asked to enter information that will
be incorporated into your certificate request.
What you are about to enter is what is called a
Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
```

```
For some fields there will be a default value,
If you enter '.', the field will be left blank.
---
Country Name (2 letter code) [AU]:GB
State or Province Name (full name) [Some-State]:Surrey
Locality Name (eg, city) []:Egham
Organization Name (eg, company) [MyCo Ltd]:Arts Building Ltd
Organizational Unit Name (eg, section) []:Dept. History
Common Name (eg, YOUR name) []:Simos Xenitellis
Email Address []:S.Xenitellis@rhbnc.ac.uk

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:.
An optional company name []:.
User%
```

This command creates a certificate request stored in the file `user.csr`. In this phase, the user enters the values of the fields for the X.509 Certificate as shown. For a certificate request in PEM format, please see Section B.6. For a TXT or human–readable version, please check Section B.7.

4.3.3. Ask the CA to sign the certificate request

The CA receives the certificate request, and depending on the policy used, will decide whether to sign the CSR. If it trusts the user, it signs the CSR as follows:

```
CA_Admin% ./sign.sh user.csr
CA signing: user.csr -> user.crt:
Using configuration from ca.config
Enter PEM pass phrase: enter the pass-phrase
Check that the request matches the signature
Signature ok
The Subjects Distinguished Name is as follows
countryName           :PRINTABLE:'GB'
stateOrProvinceName   :PRINTABLE:'Surrey'
localityName          :PRINTABLE:'Egham'
organizationName      :PRINTABLE:'Arts Building Ltd'
organizationalUnitName:PRINTABLE:'Dept. History'
commonName            :PRINTABLE:'Simos Xenitellis'
emailAddress          :IA5STRING:'S.Xenitellis@rhbnc.ac.uk'
Certificate is to be certified until Feb  6 13:30:41 2001 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

```
CA verifying: user.crt <-> CA cert
user.crt: OK
CA_Admin%
```

This command produces a file called `user.crt`, the Certificate of the user. The **sign.sh** script can be found in the modssl package, described above, at the `/pkg.contrib/` directory. This script uses openssl as a backend. We use the script and not the manual procedure because with the latter we would have to perform rather several steps and this would be out of the scope of this book. In a future version of this document, we shall revisit this issue.

Chapter 5. PKI standards and specifications

We describe available PKI-related standards and specifications and discuss the usability of each of them. Open standards and specifications are important to open–source PKIs, since they remove possible interoperability obstacles of closed–source implementations.

5.1. Internet X.509 Public Key Infrastructure (PKIX)

PKIX is covered in Chapter 6.

5.2. Architecture for Public-Key Infrastructure (APKI)

APKI is a specification for a Public Key Infrastructure, created by the The Open Group[1]. It describes the architecture, the requirements and the components of a PKI. Also, it gives recommendations for implementors on the use of protocols and other specifications. This specification [2] is available in HTML and PDF format and one can download it by registering freely on the WWW site. A hard copy is also available.

In a nutshell, this specification gives a high-level overview of the components of a PKI and recommends other protocols and specifications that should be used in order to get a standard-based functionality. Summing up, it recommends the use of

- The PKIX standards, covered in Chapter 6.
- The CDSA 2.0 Common Data Security Architecture, covered in Section 9.7.
- The XDAS Distributed Audit Service, covered in Section 9.10.
- The GSS-API Generic Security API and its extensions (XGSS-API), covered in Section 9.11.
- The LDAP Lightweight Directory Access Protocol, covered in Section 9.13.
- The IETF S/MIME Cryptographic Message Syntax (CMS), version 3, covered in Section 9.14.
- The IETF (RFC2030) Simple Network Time Protocol (SNTP), covered in Section 9.12.

5.3. The NIST Public Key Infrastructure Program

The National Institute of Standards and Technology (NIST), part of the U.S. Department of Commerce, is developing specifications for Public Key Infrastructures for the internal use of the U.S. government electronic infrastructure. These efforts do not aim to duplicate existing work of PKI vendors, rather than to ease the integration of the use of public-key technology from possibly inoperable implementations.

This work is being developed with the help of industry partners, using agreements called CRADAs (Cooperative Research and Development Agreements) in the sense that companies and the government work together to specify the PKI products to be produced that the latter will buy as a consumer. In this sense, since the U.S. government is a big buyer, one can expect that the work of the NIST somehow specifies the future of the PKI products that will be used worldwide.

Among the publicly available documents is the MISPC specification that provides a basis for interoperation between PKI components from different vendors. Vendor willing to get contracts for U.S. Federal agencies should be able to provide compatible PKI components. Possible open-source PKI implementations would obviously need to comply with those specifications. The MISPC specification is the basis for the NIST reference implementation, also described in Section 7.6. It is available as NIST Special Publication 800-15[3] from the NIST WWW site.

Another interesting document is the Proposed Federal PKI Concept of Operation[4].

Among the highlights of the above document is the clear description of available PKI types. The PKI that the browsers implement is described as the *trust-list* PKI. This is a somehow flat type of PKI in the sense that there is only one level of trust. The other two types are the hierarchical and the network (or mesh) PKIs. The former is the typical X.500 PKI while the latter is the mesh type with no single root. One can find analogies of the hierarchical PKI with the structure of the Domain Name Service. The network PKI is like the interconnection of the routers on the Internet.

Another important issue is the same document, is the use the Bridge Certification Authority concept, a CA that bridges different trust domains. This bridging is established upon agreement of the interested parties and its purpose is to limit the propagation of unnecessary trust.

A pilot program is planned to test the bridge CA concept. From the information provided at the NIST PKI Root CA Testbed[5] page, the Bridge CA will be implemented by the NIST and commercial CAs will be tested by being bridged by this Bridge CA. The plan is to have twelve CAs and 4 X.509 Directory servers operational. Information to be sought from this pilot operation has to do with performance and scalability. Finally, the X.509 certification path building and validation will be tested.

The author of these documents (either main author or in co-operation) is William E. Burr[6].

Notes

1. http://www.opengroup.org
2. http://www.opengroup.org/pubs/catalog/g801.htm
3. http://csrc.nist.gov/pki/documents/mispcv1.ps
4. http://csrc.nist.gov/pki/twg/baseline/pkicon20b.PDF
5. http://csrc.nist.gov/pki/rootca/
6. mailto:william.burr@nist.gov

Chapter 6. Internet X.509 Public Key Infrastructure (PKIX)

In this chapter, we shall provide an informal introduction to the PKIX Internet Standards which are being developed by the PKIX Working Group[1].

6.1. Abbreviations

To avoid confusion regarding the PKIX terminology, we include the list of terms as they are found in the PKIX document `draft-ietf-pkix-roadmap-05`. Their full explanation can be found at the Glossary.

Table 6-1. PKIX Terms

Term	Abbreviation
Attribute Authority	AA
Attribute Certificate	AC
Certificate	
Certification Authority	CA
Certificate Policy	CP
Certification Practice Statement	CPS
End–Entity	EE
Public Key Certificate	PKC
Public Key Infrastructure	PKI
Privilege Management Infrastructure	PMI
Registration Authority	RA
Relying Party	
Root CA	
Subordinate CA	
Subject	
Top CA	

With regard to the term X.509, it comes from the X.500 specification on directory services. The directory services serve as a kind of electronic phonebook, where enabled applications can lookup included entities. Each entity has a identifying record or Certificate and the format of that Certificate follows the recommendation X.509 of the International Telecommunication Union (ITU).

X.500 itself is considered as too difficult to catch on, however, the X.509 format for certificates is used by succesive standards. For more information on X.500, one can read the online book entitiled Understanding X.500 – The Directory[2] by D.W.Chadwick.

6.2. Concepts

We describe important concepts with regard to the PKIX standards. A Public Key Infrastructure does not only need an infrastructure to handle identities, it needs an infrastructure to handle privileges. The distinction between the two will become more evident in the following sections.

6.2.1. Certificate–using Systems and PKIs

At the heart of recent efforts to improve Internet security are a group of security protocols such as Secure Multipurpose Internet Mail Extensions (S/MIME), Transport Layer Security (TLS), and Internet Protocol Security (IPSec). All of these protocols rely on public–key cryptography to provide services such as confidentiality, data integrity, data origin authentication, and non-repudiation. The purpose of a PKI is to provide trusted and efficient key and public key certificate management, thus enabling the use of authentication, non-repudiation, and confidentiality.

Security services

Essential services to ensure the security on the Internet are confidentiality, data integrity, data origin authentication and non–repudiation. These can be achieved with protocols like S/MIME, TLS and IPSec . The protocols need a PKI in order to function effectively.

Users of public key-based systems must be confident that, any time they rely on a public key, the associated private key is owned by the subject with which they are communicating. (This applies whether an encryption or digital signature mechanism is used.) This confidence is obtained through the use of PKCs, which are data structures that bind public key values to subjects. The binding is achieved by having a trusted CA verify the subject's identity and digitally sign each PKC.

A PKC has a limited valid lifetime, which is indicated in its signed contents. Because a PKC's signature and timeliness can be independently checked by a certificate-using client, PKCs can be distributed via untrusted communications and server systems, and can be cached in unsecured storage in certificate-using systems.

PKCs are used in the process of validating signed data. Specifics vary according to which algorithm is used, but the general process works as follows:

Note: There is no specific order in which the checks listed below must be made; implementors are free to implement them in the most efficient way for their systems.

- The recipient of signed data verifies that the claimed identity of the user is in accordance with the identity contained in the PKC;
- The recipient validates that no PKC in the path is revoked (e.g., by retrieving a suitably-current Certificate Revocation List (CRL) or querying an on-line certificate status responder), and that all PKCs are within their validity periods at the time the data was signed;
- The recipient verifies that the data are not claimed to have any values for which the PKC indicates that the signer is not authorized;
- The recipient verifies that the data have not been altered since signing, by using the public key in the PKC.

If all of these checks pass, the recipient can accept that the data was signed by the purported signer. The process for keys used for encryption is similar.

Note: It is of course possible that the data was signed by someone very different from the signer, if for example the purported signer's private key was compromised. Security depends on all parts of the certificate-using system, including but not limited to: physical security of the place the computer resides; personnel security (i.e., the trustworthiness of the people who actually develop, install, run, and maintain the system); the security provided by the operating system on which the private key is used; and the security provided the CA. A failure in any one of these areas can cause the entire system security to fail. PKIX is limited in scope, however, and only directly addresses issues related to the operation of the PKI subsystem. For guidance in many of the other areas, see RFC 2527.

6.2.2. Certificate–using Systems and PMIs

Many systems use the PKC to perform identity based access control decisions (i.e., the identity may be used to support identity-based access control decisions after the client proves that it has access to the private key that corresponds to the public key contained in the PKC). For many systems this is sufficient, but increasingly systems are beginning to find that rule-based, role-based, and rank- based access control is required. These forms of access control decisions require additional information that is normally not included in a PKC, because the lifetime of the information is much shorter than the lifetime of the public-private key pair. To support binding this information to a PKC the Attribute Certificate (AC) was defined in ANSI and later incorporated into ITU–T Recommendation X.509. The AC format allows any additional information to be bound to a PKC by including, in a digitally signed data structure, a reference back to one specific PKC or to multiple PKCs, useful when the subject has the same identity in multiple PKCs. Additionally, the AC can be constructed in such a way that it is only useful at one or more particular targets (e.g., web server, mail host).

Users of a PMI must be confident that the identity purporting to posess an attribute has the right to possess that attribute. This confidence may be obtained through the use of PKCs or it may be configured in the AC-using system. If PKCs are used the party making the access control decision can determine "if the AC issuer is trusted to issue ACs containing this attribute."

6.3. Overview of the PKIX approach

PKIX, in order to describe public–key infrastructures, uses the terms PKI and PMI. One can find similarities between the two. The main difference is that the PKI handles the Public Key Certificates while the PMI handles the Attribute Certificates. A good metaphor to distinguish between the two is to associate the former with the passport of a person and the latter with the visa. The one provides identity and the other permission.

6.3.1. PKIX standardisation areas

PKIX is working on the following five areas.

1. Profiles of X.509 v3 Public Key Certificates and X.509 v2 Certificate Revocation Lists (CRLs).

   It describes the basic certificate fields and the extensions to be supported for the Certificates and the Certificate Revocation Lists. Then, it talks about the basic and extended Certificate Path Validation. Finally, it covers the supported cryptographic algorithms.

2. Management protocols.

   First, it discusses the assumptions and restrictions of the protocols. Then, it provides the data structures used for the PKI management messages and defines the functions that conforming implementations must carry out. Finally, it describes a simple protocol for transporting PKI messages.

3. Operational protocols.

   Currently they describe how LDAPv2, FTP and HTTP can be used as operational protocols.

4. Certificate policies and Certificate Practice Statements.

   The purpose of this document is to establish a clear relationship between certificate policies and CPSs, and to present a framework to assist the writers of certificate policies or CPSs with their tasks. In particular, the framework identifies the elements that may need to be considered in formulating a certificate policy or a CPS. The purpose is not to define particular certificate policies or CPSs, per se.

5. Time–stamping and data–certification/validation services.

   There are no RFCs on these services yet, as the documents are still classified as Internet Drafts.

   The time–stamping services define a trusted third–party that creates time stamp tokens in order to indicate that a datum existed at a particular point in time. The data certification and validation services provide certification of possesion of data and claim of possesion of data, and validation of digitally signed documents and certificates.

The relevant Request For Comments (RFC) documents are depicted in the following table

Table 6-2. Table of RFCs for PKIX documents

Subject	RFC
Profiles of X.509 v3 Public Key Certificates and X.509 v2 Certificate Revocation Lists (CRLs)	RFC 2459
PKIX Certificate Management Protocols	RFC 2510
Operational protocols	RFC 2559, RFC 2585, RFC 2560
Certificate Policy and Certification Practices Framework	RFC 2527
Time–stamping and data–certification services	No RFCs yet, only internet drafts available

The specification of the X.509 Certificates is very general and extensible. To ensure interoperability between different Internet-centric implementations, the PKIX Working Group defined a *profile*, which is a description of the format and semantics of certificates and certificate revocation lists for the Internet PKI.

The operational protocols are the protocols that are required to deliver certificates and CRLs (or status information) to certificate–using client systems. There is an emphasis to have a variety of distribution mechanisms for the certificates and the CRLs, using for example, LDAP, HTTP and FTP. For example, the retrieval of the CRL by a merchant to check whether a certificate is valid, constitutes an operational protocol.

Management protocols are the protocols that are required to support on–line interactions between PKI user and management entities. The possible set of functions that can be supported by management protocols is

- registration of entity, that takes place prior to issuing the certificate
- initialisation, for example generation of key–pair
- certification, the issuance of the certificate
- key–pair recovery, the ability to recover lost keys

- key–pair update, when the certificate expires and a new key–pair and certificate have to be generated
- revocation request, when an authorised person advices the CA to include a specific certificate into the revocation list
- cross-certification, when two CAs exchange information in order to generate a cross–certificate

The Certificate Policies and the Certificate Practice Statements are recommendations of documents that will describe the obligations and other rules with regard the usage of the Certificate.

6.3.2. Public–key infrastructure functionality

This is a functionality or operations of a Public Key Infrastructure.

Table 6-3. PKI functionality

Functionality
Registration
Initialisation
Certification
Key–pair recovery
Key generation
Key update
Key expiry
Key compromise
Cross certification
Revocation
Certificate and Revocation Notice Distribution and Publication

6.3.3. Public–Key Infrastructure (PKI)

A PKI is a set of hardware, software, people, policies and procedures needed to create, manage, store, distribute and revoke PKCs based on public–key cryptography.

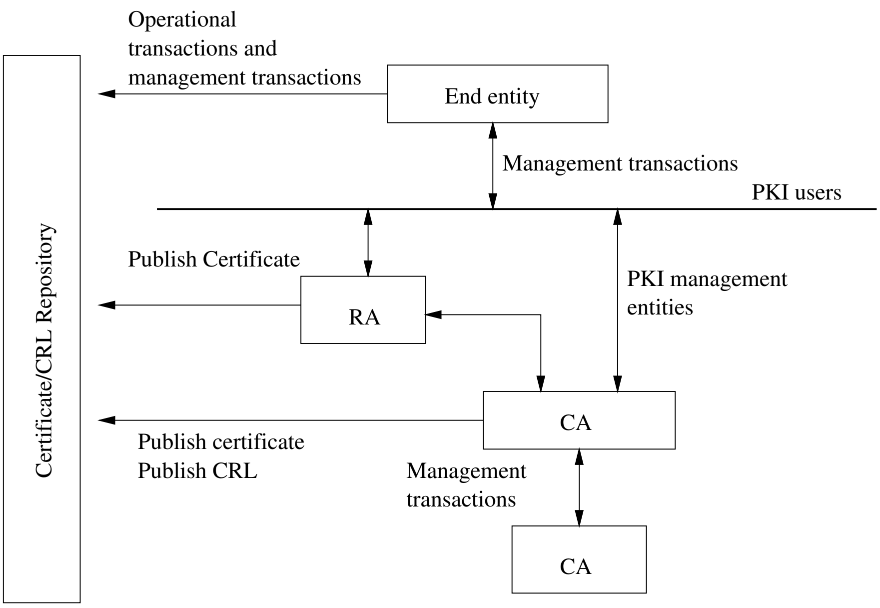
A PKI consists of five types of componets.

Table 6-4. PKI components

Type of component	Description
Certification Authorities (CAs)	to issue and revoke PKCs
Organisational Registration Authorities (ORAs)	to vouch for the binding between public keys and certificate holder identities and other attributes
Certificate holders	to sign and encrypt digital documents
Clients	to validate digital signatures and their certification path from a known public key of a trusted CA
Repositories	to store and make available certificates and Certificate Revocation Lists (CRLs)

In Figure 6-1 there is a simplified view of the architectural model assumed by the PKIX Working Group.

Figure 6-1. PKI Entities



The End–entity, using management transactions, sends its certificate request to the Registration Authority for approval. If it is actually approved, it is forwarded to the Certification Authority for signing. The Certification Authority verifies the certificate request and if it passes the verification, it is signed and the Certificate is produced. To public the Certificate, the CA sends it to Certificate Repository for collection from the End–entity.

The diagram shows that the End–entity can communicate directly with the CA. According to the PKIX recommendations, it is possible to implement the functionality within the CA. Although it is a bit confusing, the diagram shows all possible communications, regardless of the implementation decisions.

Additionally, both the CA and RA are shown to deliver Certificates to the repository. Depending on the implementation, one of the two is chosen.

For the issue of the revocation of the certificates, a similar course with the generation of the Certificates is taken. The End–entity asks the RA to have its Certificate revoked, the RA decides and possibly forwards it to the CA, the CA updates the revocation list and publishes it on the CRL repository.

Finally, the End–entities can check the validity of a specific Certificate using an operational protocol.

6.3.4. Privilege Management Infrastructure (PMI)

PMI is the set of hardware, software, people, policies and procedures needed to create, manage, store, distribute and revoke Attribute Certificates.

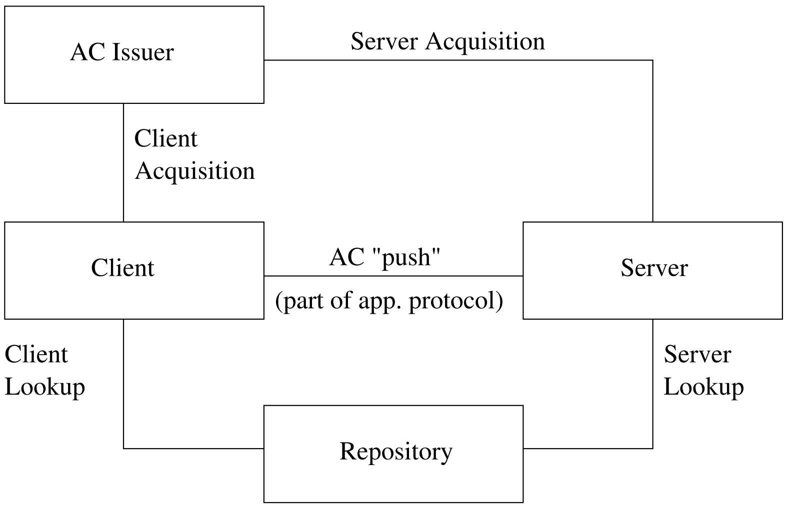
A PMI consists of five types of componets.

Table 6-5. PMI components

Type of component	Description
Attribute Authorities (AAs)	to issue and revoke ACs (also called Attribute Certificate Issuer)
Attribute Certificate Users	to parse or process an AC
Attribute Certificate Verifier	to check the validity of an AC and then make use of the result
Clients	to request an action for which authorisation checks are to be made
Repositories	to store and make available certificates and Certificate Revocation Lists (CRLs)

In Figure 6-2 there is a view of the exchanges that may involve Attribute Certificates

Figure 6-2. Attribute Certificate Exchanges



There are two types of attribute certificate distribution as show in the diagram, *push* and *pull*.

In some environments it is suitable for a client to *push* an AC to a server. This means that no new connections between the client and server are required. It also means that no search burden is imposed on servers, which improves performance.

In other cases, it is more suitable for a client simply to authenticate to the server and for the server to request or *pull* the client's AC from an AC issuer or a repository. A major benefit of the *pull* model is that it can be implemented without changes to the client or to the client–server protocol. It is also more suitable for some inter–domain cases where the client's rights should be assigned within the server's domain, rather than within the client's domain.

Notes

1. http://www.ietf.org/html.charters/pkix-charter.html
2. http://www.salford.ac.uk/its024/X500.htm

Chapter 7. Open-Source Implementations

We are presenting a list of open–source or almost open–source implementations of Public Key Infrastructures. In the following sections, we shall get into more detail for each of the implementations.

1. pyCA, by Michael Stroeder, available at the pyCA WWW site[1].
2. OpenCA, by Massimiliano Pala and the OpenCA Team, available at the OpenCA WWW site[2].
3. Oscar, by DSTC Pty Ltd, Australia, available at the DSTC PKI WWW site[3].
4. Jonah, by IBM (and subsidiaries Lotus, Iris), available at the Jonah WWW site[4].
5. Mozilla Open Source PKI projects, by the Mozilla Organisation, available at the Mozilla WWW site[5].
6. MISPC, by the National Institute of Standards and Technology (NIST) available at the MISPC WWW site[6].

7.1. The pyCA Certification Authority

pyCA is a set of CGI scripts that provide a WWW interface to a Certification Authority. The scripts are written in the programming language Python[7], hence the name pyCA. It uses OpenSSL as the underlying mechanism for the cryptographic infrastructure. pyCA is distributed under the GPL[8] license.

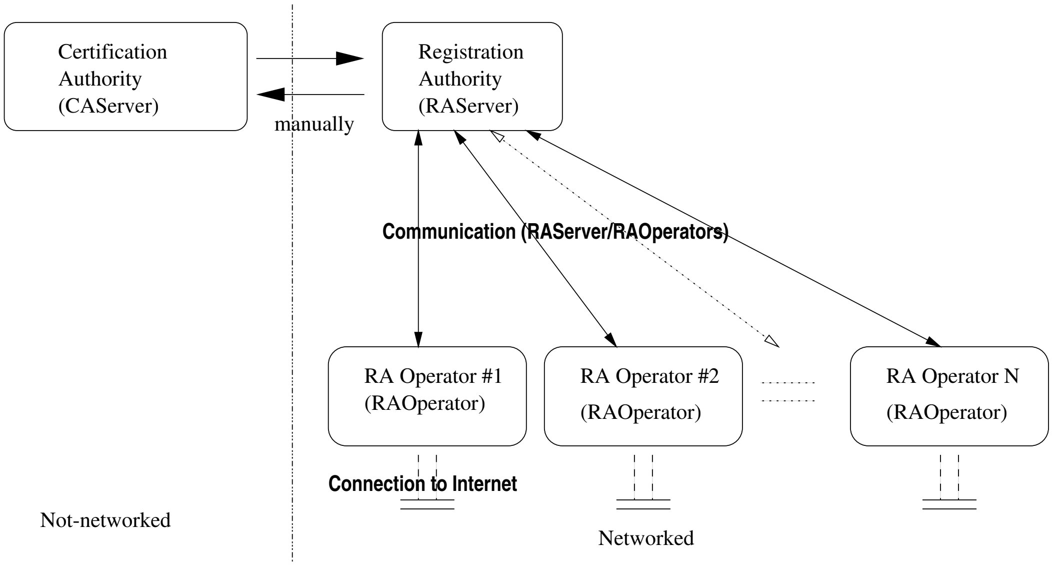
7.2. The OpenCA Project[TODO]

OpenCA is a collaborative effort to create a public–key infrastructure. Programmatically, it resembles pyCA with the exception of using Perl instead of Python for the CGI scripts. It uses OpenSSL for the underlying cryptograpgic infrastructure. OpenCA is distributed with an Apache–style license.

7.2.1. OpenCA Layout

We describe the CA structure as used currently in OpenCA.

Figure 7-1. Current OpenCA Layout



The Certification Authority — for security reasons and in accordance with the current layout — should not be networked. It manually communicates with the Registration Authority, perhaps using removable media.

The Registration Authority should not have direct access to the Internet but be accessed through the RA Operator.

The RA Operator is the interface of OpenCA between users and the Internet.

7.2.2. OpenCA Abbreviations

Throughout the documentation we use the following terminology.

Table 7-1. OpenCA Abbreviations

Term	OpenCA Object Name
Certification Authority	CAServer
Registration Authority	RAServer
RA Operator	RAOperator

Note: The reader should notice the difference between the CA as an organisation or a company and the CA as CAServer. Currently, we do not differentiate explicitly and the user has to identify the correct meaning by the context.

Note: In several parts of the documents we use the terms Certificate Signing Request and requests to describe the same thing. The latter term is common in programming contexts.

7.2.3. Software packages

This is the basic software used to implement OpenCA. Current (May, 2000) versions are depicted in the table following the list.

1. Compatible Operating system like Linux ®, available from http://www.linux.org[9]
2. OpenSSL, available from http://www.openssl.org[10]
3. Apache WWW Server, available from http://www.apache.org[11]
4. mod–ssl Apache module, available from http://www.modssl.org[12]
5. OpenLDAP, available from http://www.openldap.org[13]

We subsequently install the OpenCA software. At the time of writing, the latest version is 0.2.0, available at http://www.openca.org/download.shtml[14]

Table 7-2. Current Versions of OpenCA prerequisite software

Software	Current version
RedHat®Linux	`6.2`
OpenSSL SSL/TLS software	`0.9.5a`
Apache WWW Server	`1.3.12`
modssl SSL/TLS Apache module	`2.6.4`

Software	Current version
OpenLDAP LDAP software	`1.2.10`
Perl interpreter	`5.6.0`

7.2.4. Functionality of the CA Server *(CAServer)*

The functionality of the Certification Authority Server is:

The source code describes the CA server as *CAServer*.

The following sections and subsections are the options presented to the user by the WWW interface to administer the CA. This interface is the recommended method of administration.

Note: The content of this section is subject to change in the future.

7.2.4.1. Initialisation / CA Management

- Generate new CA private key

  This procedure is described in Section 4.3.1. Generating a new secret (private) key for a CA and overwriting the old one is an important procedure, since previously–issued certificates become invalid. The user is warned that the current CA private key will be overwritten.

  Note: In fact, with the current version of OpenCA, the previous current key is not overwritten, as the software saves the current key in a file with an extension of .old. However, previous any previous keys are overwritten.

  The user is prompted with a dialog box for the CA secret key. This *CA secret key* is in fact the *pass–phrase* that protects the CA private key. For example, if we use RSA as the public key algorithm, the key generation procedure generates a set of very large numbers. A part of them constitute the private key. This information must not be compromised. In order to make it more secure, we encrypt this information with a block cipher like DES, Triple–DES or IDEA. Subsequently, when there is a need to have the value of the private key, the application asks us for the pass–phrase, decrypts the encrypted private key, and uses it.

  Note: Safe choices for an encryption algorithm are Triple–DES (3DES or sometimes written DES3) and IDEA. DES is not considered a safe choice, unless key recovery is an issue. `:)` Apart from the joke, the user should be aware that using high–grade encryption does not mean that the system is secure. All components of a system need to be secure in order to have a secure system. In the case of OpenCA, there are a lot of components.

> **Note:** It is common practice that once the private key is used by the application, it should not be kept in the computer's memory any longer. When there is a need for the private key, the application should ask for it. This is more secure but requires human intervention when there is a need for the private key. For example, when we need to restart a server.

Next the user is asked for the size of the CA key in bits. This is the size of N in RSA, the product of the two large prime numbers. This affects the security of the Certification Authority.

A choice of 512 bits is not considered safe while one of 1024 bits is considered relatively safe. 2048 bits are considered to be a secure choice with current (May, 2000) information about factorisation.

> **Note:** Consider that when the key length rises, so does the time needed to generate and do operations with the keys. For instance, on a Pentium® Pro computer, generating a key of size 1024 bits requires approximately 3 seconds; 2048 bits requires around 13 seconds. The tests were carried out using the OpenSSL software and running the Linux® operating system. Key generation takes place once in the lifetime of the key. Other operations, like the signing and verification of digital signatures, take place more frequently and remain to be benchmarked.

> **Note:** With Pentium® II or better computers, the size of 2048 bits is both a fast and secure choice. Once we click OK, the key–generation takes place. This takes several seconds. The user should wait for it to complete.

You can find a sample encrypted private key in PEM format at Section B.1.

- Generate new CA Certificate Signing Request

  This is the procedure described in Section 4.3.2. Essentially, the certificate request is generated to be later self–signed with the public key of the CA, generated with the previous option.

- Export CA Certificate Request

  This option exports the CA certificate request generated above. A file is created in the file system that corresponds with the CSR.

- Generate Self–Signed CA Certificate

  This option uses the generated CSR to create the CA Certificate. It signs it with the public key of the CA.

- Export CA Certificate

  This option exports the generated CA Certificate or as it is sometimes called, the Root CA Certificate. Copies of this Certificate should be given to the public.

7.2.4.2. Requests

- Import requests

  This imports requests (CSRs) for signing to the CA. The RAServer Administrator has used the *Export requests* command to export the Certificate Signing Requests to, possibly, a removable medium. With this command, the CAServer Administrator will retrieve them for signing.

- Pending requests

  This shows the pending requests that reside on the CA. We should note that as *request* we describe the Certificate Signing Request. Pending requests are the requests that have been uploaded to the Certification Authority and wait to be signed.

  > **Note:** The same terminology, *pending requests* is used on the Registration Authority with a different meaning. On the Registration Authority, a pending request is a Certificate Signing Request that remains to be approved by the Registration Authority Administrator and be sent over to the Certification Authority.

- Deleted Requests

  This shows the deleted requests to the CA. A Certificate Signing Request that has been uploaded to the Certification Authority may not be finally granted permission and be signed. With the current layout of the relationship of the CAServer and the RAServer, the RAServer signs each Certificate Signing Request with its own private key. The CAServer checks the signature and if it is verified, it creates the Certificate. Otherwise it deletes it and it is shown here.

- Remove Deleted Requests

  This removes the deleted requests from the CA. It means that the requests are physically removed from the file system of the CAServer.

7.2.4.3. Certificates

- Issued Certificates

  This shows all Certificates ever issued by the Certification Authority.

- Export Certificates

  This exports the Certificates to a removable media in order to be delivered to the RAServer. It is the responsibility of the RAServer to distribute the Certificates to the individual owner.

7.2.4.4. Certificate Revocation List CRL

- Export CRL

  This exports the Certificate Revocation List to the RAServer. The RAServer has the responsibility to make the Certificate Revocation List known and available to the individual users.

7.2.5. Functionality of the RA Server *(RAServer)*

This is the functionality of the Registration Authority (RAServer) Server. The various local Registration Authority Operators communicate with this intermediary on behalf of the users' requests, in order to have access to the CA. No user communicates directly with the RA server. The RA server should be placed at a very high security level to prevent unauthorized access. The RA Server is administered by the Registration Authority Administrator. The actions available are listed next.

While perusing the source code, you will see the principal Registration Authority Server to be described as *RAServer*.

> **Note:** The content of this section is subject to change in the future.

7.2.5.1. Requests

- Export Requests

  Export the approved requests to the CAServer.

- Pending Requests

  Show Certificate Signing Requests waiting for approval by the RAServer Administrator. Approval can be based to Identification Documents or other credentials.

- Approved Requests

  Show Certificate Signing Requests that have already been approved by the RAServer Administrator. These Certificate Signing Requests will be sent to the CAServer using the *Export requests* function.

- Remove Exported Requests

  The approved requests, once they are exported to the CAServer, can be removed with this option.

7.2.5.2. Certificates

- Import CA Certificate

  This imports the Certification Authority Certificate and saves it on the local filesystem. This copy of the Certificate will be published using the adjacent commands to the interested parties.

- Import New Certificates

  This imports the newly signed Certificates from the CAServer. The Certificates are copied to the local file system.

- Export Certificates onto LDAP

  This command exports the Certificates to the specified LDAP server. The users will retrieve their Certificate by accessing the LDAP server, rather then contacting directly the RAServer.

7.2.5.3. Certificate Revocation List CRL

- Import CRL

  This imports the Certificate Signing Request from the Certification Authority so that it can be published.

- Export Certificate Revocation Requests

  This command exports approved Revocation Requests to the CAServer. Then, the CAServer revokes these Certificates.

7.2.5.4. Miscellaneous Utilities

- Send e–mail to users for newly–issued certificates

  This informs the users that the Certificate has been prepared and that they should follow the indicated procedure to collect it.

- Delete Temp files (After importing certificates).

  This is a clean–up command. With the current implementation of OpenCA, when the users are being sent a notification, temporary files are created to indicate the e–mail to be sent. If these files are not deleted, then, on the next batch mailing, users who have already received a notification are notified again.

7.2.6. Functionality of the RA Operators *(RAOperators)*

The Public Servers, — the servers that the users actually have access to — are securely–configured servers that ask for Certificates, deliver them, and so on. This is the only entry point to the CA infrastructure from the Internet.

The source code describes the local Secure RA servers as *RAOperators*.

Note: The content of this section is subject to change in the future.

7.2.6.1. Get Root CA Certificate

This allows the user to import the root Certificate of the Certification Authority into the browser. This is a basic and important procedure. It takes place once in the life–time of the Certification Authority Certificate. Other documentation describes this Certificate as the *Root Certificate*. It is the starting point to enable the client to communicate securely with the Certification Authority.

7.2.6.2. Certificate Revocation Lists

This brings up the Certificate Revocation List page. Here the Certificate Revocation List, produced by the Certification Authority is imported into the browser or other application.

- OpenCA's Certificate Revocation List (DER format)

  With this option, a browser–importable Certificate Revocation List is generated to be automaticaly included in the CRL list of the browser. The CRL is in the DER format.

- OpenCA's Certificate Revocation List (PEM format)

  With this option, the Certificate Revocation List is generated into the PEM format. Similar to above.

- OpenCA's Certificate Revocation List (TXT format)

  With this option, the Certificate Revocation List is generated into text format. The file generated by this command can be very big.

7.2.6.3. Request a Certificate

Initiate the procedure to request a certificate.

7.2.6.4. Get Requested Certificate

This allows the user to retrieve the issued certificate and subsequently import it to the application. The user has received the notification e–mail from the Registration Authority and is prompted with intructions to retrieve the Certificate. In the e–mail, there is a serial number of the Certificate that has to be presented to the RAOperator in order to retrieve the Certificate. The serial number serves as an identification as to which Certificate will be retrieved. It is not used for authentication purposes.

7.2.6.5. Issued Certificates List

This option presents a list of the issued certificates of this Certification Authority.

7.2.7. Status of the OpenCA Project

The OpenCA Project is evolving quickly. The current version at the time of writing (May, 2000) is 0.2.0. Latest release information can be found at the OpenCA Status[15] page.

7.2.8. Future OpenCA work

This section describes the future work needed for OpenCA.

- The current layout of OpenCA (see Figure 7-1) is not yet scalable to support multiple CAServers or RAServers. Currently this is not a high–priority issue as it is more important to come up with a simple, secure, and clean implementation of a CA.
- Do more work on the LDAP support.
- Also, there are scalability issues with high usage of OpenCA. The current implementation uses Perl CGI scripts. These scripts invoke the openssl application. The overhead of invoking these two big executables (perl and openssl) is considerable. Depending on the hardware configuration, there is a limit where the physical memory becomes exhausted. The system starts swapping heavily and the load goes high.

  Possible solutions here would be to make use of mod-perl for the Apache WWW Server. This adds a new component that needs to be included in a future security review.

  Calling the OpenSSL library would be much more efficient than invoking the openssl application. Both Perl and C support library function invocation.

- In the current OpenCA layout (see Figure 7-1) the CAServer is shown to not be networked. It communicates with the RAServer using removable media. There could be a solution that allows a networked configuration and maintains a high degree of security.

- A test–suite is needed to test the installation for correctness and provide an estimation of thoughput capabilities. For the current implementation of OpenCA applications like cURL could be used to write a test–suite. cURL supports SSL/TLS connections. It is an open–source command–line application. It is found at cURL - Client to fetch URLs [16] link.
- OpenCA software and its components require a security review.
- Smart cards could be used in The OpenCA Project. Linux® supports smart cards. Information is at MUSCLE Smartcard Home Page[17]. MUSCLE supports PC/SC and OCF (through JNI). The PC/SC support is more complete and could provide the necessary performance needed. Also, it can be accessed through Perl and C.
- OpenCA could be implemented in various other languages. The decision for this should be the weighing of the benefits and the source–code fork problem.
- Internationalisation of OpenCA. This could be accomplished with the gettext support that perl has. However, this should wait until the software has been stabilised.

7.3. The Oscar Public Key Infrastructure Project

Oscar is a project of the Distributed Systems Technology Center of the Queensland University of Technology to create a public–key infrastructure. It implements the cryptographic functions using directly the GMP library, unlike other implementations that use OpenSSL. The programming language chosen is C++.

The licensing of Oscar allows the non-commercial usage of the software. To use it commercially, one needs to obtain a license. This is not compatible to either the GNU or the Apache style license.

7.4. Jonah: Freeware PKIX reference implementation

Jonah is a freeware (term used by IBM, it is not mentioned as *open–source*) implementation of the available PKIX standards for a public–key infrastructure. It is one of the implementations that was designed to follow the PKIX standards from the beginning and it generated important feedback to the standardisation process. The core of Jonah is written in C++ and the server components are accessed using Java applets. The C++ code is compatible with Windows NT, Solaris and AIX. We must emphasise that Jonah was written basicaly for the internal purposes of IBM and its subsidiaries Lotus and Iris as an interoperability tool between different implementations and to be part of future IBM products.

There is no Linux port yet, and this can be attributed to several issues. The development environment chosen initially was making use of the MKS Toolkit, which provides a Unix-like environment to Windows NT. In the next version, the ODE environment was chosen. Both are incompatible with

Linux and require some work to make them work together. Additionally, for the cryptographic support one would need to use the Cylink crypto libraries, which at the moment were export restricted. In the next version, there the BSafe Toolkit from RSA Security was chosen. Cryptographically-wise, BSafe is much more mature and popular. However, it is a commercial product and it remains to have investigated its licensing issues.

Jonah provides transparency to the choice of cryptographic support by making use of the Common Data Security Architecture (CDSA). The Common Data Security Architecture (CDSA) is a set of layered security services and cryptographic framework that provide the infrastructure for creating cross-platform, interoperable, security-enabled applications for client-server environments. The CDSA solutions cover all the essential components of security capability, to secure electronic commerce and other business applications with services that provide facilities for cryptography, certificate management, trust policy management, and key recovery. CDSA was developed by Intel and is being standardised by OpenGroup.

Jonah is not distributed with the full functionality due to the crypto issues. To be precise, Jonah is not available at the moment due to a mysterious licensing issue with the CDSA support. It is claimed that Intel, by releasing version 2 of CDSA, has made the version of CDSA used by Jonah illegal to distribute. The last free distribution of Jonah was using version 1.2 of CDSA.

Jonah is distributed under this[18] license.

7.5. Mozilla Open Source PKI projects

Currently, the software application that makes most use of PKI technology is the WWW browser. This importance was realised by Netscape and has lead to the creation of two libraries to aid the unified support of cryptography and security for both the browser and server software. These libraries are the Network Security Services (NSS) and the Personal Security Manager (PSM) and provide the functionality of a PKI.

These libraries are also plagued with the export–control regulations and currently it is under consideration to receive an export license. However, in this case, there is an additional problem with the patents and the licenses that covers parts of the cryptographic software that makes the release of the source code even more difficult. Currently, the source code distributed does not contain the full functionality and thus, cannot be compiled. The result of this procedure remains to be seen. On the other hand, binary version of these libraries are both available and exportable from the US. For more information on the licensing and crypto issues, there is an appropriate Mozilla Crypto FAQ[19].

These libraries (with the exclusion of code on crypto and patented components by third–parties) are covered by the Mozilla Public License[20] and the GNU General Public License[21]. The use is free to choose under which of the two licenses to use the source code, either the MPL terms or the GPL terms.

7.5.1. Personal Security Manager (PSM)

Personal Security Manager (PSM) is a client-independent desktop security module. It performs PKI operations on behalf of desktop client applications, including certificate and key management, SSL, S/MIME, cryptographic token support, and centralized administration.

More information can be found at the Personal Security Manager (PSM)[22] WWW page.

7.5.2. Network Security Services (NSS)

Network Security Services (NSS) is a set of libraries designed to support cross-platform development of security-enabled server applications. Applications built with NSS can support SSL v2 and v3, TLS, PKCS #5, PKCS #7, PKCS #11, PKCS #12, S/MIME, X.509 v3 certificates, and other security standards.

More information can be found at the Network Security Services (NSS)[23] WWW page.

7.5.3. JavaScript API for Client Certificate Management

As part of the Personal Security Manager, a new JavaScript API for Client Certificate Management has been implemented. The effect of this is that specific client PKI functionality can move to the browser, allowing implementations like Section 7.2 and Section 7.1 to fully take advantage of it.

This is the functionality supported by PSM version 1.0.

1. User fill out enrollment form.
2. User action initiates script (for example, pressing submit).
3. The script calls the key–generation method.
4. Encryption and Signing key–pairs are generated.
5. The Encryption Private Key is wrapped with the the public key of the Key Recovery Authority (KRA). The public key of the KRA is passed in in the form of a certificate as part of the script and is checked against a pre–installed certificate copy in the local certificate database.
6. Both the Encryption and Signing Public keys, the wrapped encryption public key and a text string from the script are signed by the user's Signing Private Key. The text string can contain naming or enrollment information.
7. The signed information is returned to the script (from the PSM).
8. The script submits the signed information and other necessary information to the CA/RA.
9. The CA/RA verify the signature of the signed information.
10. The CA/RA validate the identity of the user.
11. The CA/RA sends the wrapped Encryption Private Key to the KRA.

12. The KRA sends escrow verification information back to the CA.
13. The CA creates and signs the certificates.
14. The CA sends the created certificates back to the PSM–capable browser.
15. The certificates are stored.

More information can be found at the JavaScript API for Client Certificate Management[24] WWW page.

7.6. MISPC Reference Implementation

The National Institute of Standards and Technology of the U.S. government (NIST) developed the Minimum Interoperability Specifications for PKI Components (MISPC), Version 1[25] with the assistance of ten partners: AT&T, BBN, Certicom, Cylink, DynCorp, IRE, Motorola, Nortel (Entrust), Spyrus, and Verisign. The specification includes a certificate and CRL profile, message formats and basic transactions for a PKI issuing signature certificates. Also, it includes support for multiple signature algorithms and transactions to support a broad range of security policies.

Along with the specification, there is a reference implementation that is available only by ordering a CDROM from NIST. The CDROM does not contain support for the cryptographic module. This implementation appears to run only on Windows. The source code is in C++.

More information on the reference implementation can be found at the MISPC Reference Implementation[26] WWW page.

Notes

1. http://sites.inka.de/ms/python/pyca/
2. http://www.openca.org
3. http://oscar.dstc.qut.edu.au/
4. http://www.foobar.com/jonah/
5. http://www.mozilla.org/projects/security/pki/
6. http://csrc.nist.gov/pki/mispc/welcome.html
7. http://www.python.org
8. http://www.gnu.org/copyleft/gpl.html
9. http://www.linux.org
10. http://www.openssl.org
11. http://www.apache.org
12. http://www.modssl.org

13. http://www.openldap.org
14. http://www.openca.org/download.shtml
15. http://www.openca.org/docs/releases/
16. http://curl.haxx.nu/
17. http://www.linuxnet.com/
18. http://www.foobar.com/jonah/Jonah-License.html
19. http://www.mozilla.org/crypto-faq.html
20. http://www.mozilla.org/MPL/
21. http://www.gnu.org/copyleft/gpl.html
22. http://www.mozilla.org/projects/security/pki/psm/
23. http://www.mozilla.org/projects/security/pki/nss/
24. http://docs.iplanet.com/docs/manuals/psm/11/cmcjavascriptapi.html
25. http://csrc.nist.gov/pki/mispc/welcome.html
26. http://csrc.nist.gov/pki/mispc/refimp/referenc.htm

Chapter 8. How to get software support

Open–source PKI implementations use other open–source applications and libraries as shown in Section 4.1.1. We give general pointers to the different components that a typical implementation may use.

Table 8-1. WWW Support Locations

Software	Support page
RedHat™Linux	RedHat™Linux Support Center [1]
SuSE Linux	SuSE Support Page[1]
Caldera™Linux	Caldera™Support Programs[1]
Debian Linux	Debian Support Page[1]
Corel™Linux	Corel™ Support[1]
OpenSSL SSL/TLS software	OpenSSL Support Page[1]
Apache WWW Server	Apache Documentation[1]
modssl SSL/TLS Apache module	Support Forums[1]
OpenLDAP LDAP software	Technical Support[1]
Perl programming language	Enterprise Support[1]

Chapter 9. Supported Crypto hardware and Software architectures

We provide a short list of compatible with Linux hardware that can be used for the implementation of a Public Key Infrastructure. Currently we focus on crypto hardware acceleration expansion cards and smart cards.

For the succesfull use of smart cards, a programming interface to the smart card needs to be implemented. Currently, there are two such standards that describe interfaces to access a smart card from a computer. The first is PC/SC[1] which was originally developed for the Windows platform and the latter is the OpenCard Framework (OCF)[2] which is a cross–platform solution, since it uses Java.

9.1. TrustWay Crypto PCI 2000

Bull[3] manufactures a cryptographic accelerator called *TrustWay Crypto PCI 2000*. Cryptographic accelerators and other hardware devices can be accessed from the applications using the Common Data Security Architecture (CDSA). Bull has created a Linux implementation of CDSA and is selling a VPN product that bundles both the implementation and the hardware device. For more on CDSA, please read Section 9.7. For the part of the cryptographic accelerator, it can be sold seperately for about 2500 Euros.

9.2. PowerCrypt Encryption Accelerator

Global Technologies Group, Inc.[4] sells the PowerCrypt Encryption Accelerator, a hardware crypto accelerator that is compatible with OpenBSD, FreeBSD and Linux. It is based on the HiFn 7751[5] crypto acceleration chip that is compatible with the IPsec standards. Currently it only supports symmetric cryptography while assymetric cryptography is planned for the future.

9.3. CryptoSwift eCommerce Accelerator

Rainbow Technologies[6] manufacture a set of cryptographic hardware products and they offer good support for open–source operating systems. One of their products is the CryptoSwift eCommerce Accelerator[7], which is a hardware crypto accelerator that supports, among others, RSA, DSS, random number generation and secure storage of private keys. For Linux, a binary kernel module is included.

At Rainbow's ISG Labs[8], one can find performance data and other information relating to benchmarking of the product in common test cases. For open–source applications, a comprehensive study On the performance of Stronghold/Apache+SSL secure web servers[9], shows significant gains using hardware acceleration.

Interested parties can apply to qualify for a demo card[10]. They need to describe a project that they will be undertaking.

Finally, there is the CryptoSwift Software Development Kit[11] to aid the development of hardware crypto accelerated software.

9.4. Movement for the Use of Smart Cards in a Linux Environment (MUSCLE)

MUSCLE[12] provides support for smart cards for the Linux operating system. Among the project goals are the implementation of the PC/SC standard for accessing smart cards, better S/MIME integration with the Netscape browser, drivers for cards and card readers and PAM support smart card authentication. Additionally, this PC/SC implementation can be used as an abstraction layer to enable OCF applications to operate.

9.5. Linux Smart Card Starter's Kit from Schlumberger

Schlumberger[13] produces the Cyberflex[14] range of smart cards. The flagship Cyberflex™ card currently is the Cyberflex™Access smart card and it supports strong cryptography. Schlumberger offers the Cyberflex for Linux Starter's Kit 2.1[15] which is a collection of hardware and software needed to program the Schlumberger Cyberflex Access cards on Linux. The source code of the software is included in the Kit. The product ships with two Cyberflex Access Cards, Class 00, Augmented Crypto and may be purchased with or without the Reflex 64 Serial Port Smart Card Reader with a PIN Pad.

9.6. The gpkcs11 PKCS#11 open–source implementation

gpkcs11[16] is an implementation of the *PKCS #11: Cryptographic Token Interface Standard*, available under the LGPL distribution license.

PKCS#11 defines an interface for the communication of arbitrary applications with systems that perform cryptogrphic operations, like encryption and decryption, signing and verifying. These systems, called *tokens*, may be smart cards (with appropriate reader), discrete hardware systems or pure software implementations.

The gpkcs11 software is intended to be used by software developers to integrate PKCS#11 support to their applications. It is currently under development and at the moment of writing this document, the latest version is 0.6.1.

9.7. Common Data Security Architecture (CDSA)

CDSA[17] eases the process of adding security to software products. By writing to one common API, a software developer can add authentication services (such as smart card readers), encryption services (such as DES) and the ability to manage security processes (key recovery, export restrictions, prevention of attacks on the internal software pieces).

CDSA is a specification developed by Intel and the current version, version 2 has been adopted by the The Open Group[18] as an Open Group Technical Standard in 1997. The CDSA standard is available in hard–copy and electronic form (HTML and PDF) from the Common Security: CDSA and CSSM, Version 2 (with corrigenda)[19] page at the The Open Group website.

Currenly, the source code of CDSA is available for the Windows platform. Intel, along with Caldera Systems[20] and the Bull TrustWise[21] organisation are developing a Linux port of CDSA and it is expected that it will be available in September 2000.

In order for CDSA to be usable in Linux, it needs software cryptographic support for symmetric and asymmetric cryptographic algorithms. For the previous version of CDSA, version 1.2, there was no publicly available cryptographic support or Cryptographic Service Provider (CSP) as it is called. CSPs can come in two flavours, hardware implementation on an expansion card or a software version. For development purposes, it is important to have at least a software version.

CDSA has adaptation layers to use existing cryptography software for CSPs and it is possible, in the case there is no native CDSA CSP for Linux, to use one that has OpenSSL as the backend. Such a CSP based on OpenSSL was announced on the Jonah mailling list, however the correspondance e–mail to the free e–mail account does not seem to be active. However, with the newer revision 1.3 of CDSA 2.0, there is official support for use of OpenSSL as a plug–in for a CSP. This is very positive news for the soon to come Linux port.

Among the future plans for the implementation of CDSA 2.0[22] is the Linux support for the Itanium™ processor.

The implementation of CDSA 2.0 that is provided by Intel is distributed under the Intel Open Source License[23] which is the BSD license[24] with an additional export notice. This license has been reviewed and approved by the Open Source Initiative (OSI)[25], so this implementation of CDSA is OSI Certified Open Source software.

9.8. Single Sign–on

Single Sign–On (SSO) is a mechanism whereby a single initial action of user authentication and authorisation can permit the user to access all computer resources where she has access persmission without the need to authenticate/authorise subsequent times.

Among the benefits of SSO is the transparency of usage of a computer system where full access control takes place but the user is not encumbered by repeated authentications and authorisations. SSO in a nutshell requires applications to use a common security mechanism and make use of the user credentials for all the session access control requirements.

The The Open Group[26] has standardised the Pluggable Authentication Mechanism (PAM) and the corresponding standard is available at the X/Open Single Sign-On Service (XSSO)[27] page. A general description of the SSO standard can be found at the Single Sign-On[28] page.

9.9. The KeyMan PKI Management Tool

KeyMan[29] is a management tool for the client side of the Public Key Infrastructure.

KeyMan is a management tool for the client side of the public key infrastructure (PKI). KeyMan manages keys, certificates, certificate revocation lists (CRLs), and the respective repositories to store and retrieve these items. The full life cycle of certificates is supported and processes involved in handling user certificates.

KeyMan features at a glance:

- Full support of user certificate life cycle
- Management of various key/certificate repositories
- Supports cryptographic tokens via PKCS#11 interface
- Ready-to-go support for IBM Smart Card for e-Business (IBM JavaCard)
- X.509/PKIX (certificates V3, CRLs V2) supported
- Compliant with PKCS standards (#7,#10,#11,#12)
- Supports Netscape certificate requests (SPKAC)
- Integration with VeriSign and other CAs
- 100% Java, runs on JDK 1.1/1.2
- Easy to use GUI

KeyMan was developed by Thomas Eirich [30] of the IBM Zurich Research Laboratory.

9.10. Distributed Audit Service (XDAS)

The purpose of security audit services is to provide support for the principle of accountability and detection of security-policy violations in distributed systems. The XDAS specification defines a set of generic events of relevance at a global distributed system level, and a common portable audit record format to facilitate the merging and analysis of audit information from multiple components at the distributed system level. Four groups of APIs are provided to accomplish this. Source XDAS Open Group Preliminary Specification page[31].

The HTML and PDF version of the above specification is available from the XDAS Open Group Preliminary Specification page[32]. A hard copy is also available for purchase from the same page.

9.11. Generic Security Service API (GSS-API)

GSS-API is a application programming interface that provides security services to software. The definition of the API allows different software to be written and work on all compliant GSS-API library implementations. GSS-API is defined in RFC 1508 and RFC 1509.

9.12. Simple Network Time Protocol (SNTP)

SNTP is an adaptation of the Network Time Protocol (NTP) used to synchronise computer clocks in the Internet. SNTP is described in RFC 1769.

9.13. Lightweight Directory Access Protocol (LDAP)

LDAP is a protocol used to access directory services. It was originally designed to be used with the X.500 directory. However, it is currently used as a generic directory access protocol. The core of LDAP is described in RFC 1777.

9.14. S/MIME CMS [TODO]

S/MIME defines a framework within which security services may be applied to MIME body parts. S/MIME is described in RFC 1847.

Notes

1. http://www.pcscworkgroup.com/
2. http://www.opencard.org
3. http://www.bull.com
4. http://www.powercrypt.com/
5. http://www.hifn.com/
6. http://www.rainbow.com
7. http://isg.rainbow.com/products/cryptoswift.html
8. http://isglabs.rainbow.com/
9. http://isglabs.rainbow.com/isglabs/shperformance/SHPerformance.html
10. http://isg.rainbow.com/qrf_free.html
11. http://isg.rainbow.com/products/cs_3.html

12. http://www.linuxnet.com/
13. http://www.slb.com/
14. http://www.cyberflex.slb.com/
15. http://www.cyberflex.slb.com/Support/support-linux.html
16. http://www.trustcenter.de/html/Produkte/TC_PKCS11/1494.htm
17. http://developer.intel.com/ial/security/
18. http://www.opengroup.org
19. http://www.opengroup.org/publications/catalog/c914.htm
20. http://www.caldera.com/
21. http://www.servers.bull.com/trustway/
22. http://developer.intel.com/ial/security/plans.htm
23. http://developer.intel.com/ial/security/viewlicense.htm
24. http://www.opensource.org/licenses/bsd-license.html
25. http://www.opensource.org/
26. http://www.opengroup.org
27. http://www.opengroup.org/publications/catalog/p702.htm
28. http://www.opengroup.org/security/sso/index.htm
29. http://www.alphaworks.ibm.com/tech/keyman
30. mailto:eir@zurich.ibm.com
31. http://www.opengroup.org/pubs/catalog/p441.htm
32. http://www.opengroup.org/pubs/catalog/p441.htm

Chapter 10. Critical discussion[TODO]

Include info from "10 risks on PKIs", ellison/schneier. [TODO]

Security of a system is equal to the "security" of it's weakest link. People don't usually see all the links. People don't count both network/human factors. From each discipline, they stress the factor they have familiarity.

Should we have CA AND RA? Network security says it's safer, layered security, hierarchy, etc. Theoretic ppl says no much difference, or it is worse to have two different. Standards (PKIX) propose to use an RA, although do not oblige.

Watch the interactions of your system to secure it.

Human factor is greatly ignored. CS disciplines ignore the study as too law-bound, non-CS disciplines don't have the whole picture. Is it important to study this one? Can traditional methods solve the problem?

Who has the private key? It's stored in a security module, right? If it fails, what happens? Have a backup? To store in different locations (geographically)? There was a recent relevant discussion on those two MS keys.

We cannot draw the whole picture at once. We need to do it step by step. Open-Source reference implementations, widely/wildly used can show the way. Need to test and analyse feedback.

We need SSO software, openproject has PAM draft and it looks nice. There is a "killer" applicance from Samba developers that does SSO?

CDSA version 2 is very nice and standardised. openproject tambien. Bull.fr has the responsibility for the Linux port or implementation, along with Intel. Results promised in September 2000.

In the Department of Defense Appropriations Bill of the US for the year 2001 there is a description of the budget allocations. The document mentions the budget for the usage of PKIs and the recommendation is for $18.6m US dollars. It is important to notice that the description of the expense is *Information Assurance*. The document is available from the House Reports Online via GPO Access[1] link as report number 106-644.

Notes

1. wais://wais.access.gpo.gov

Chapter 11. Benefits of an Open–Source PKI implementation[TODO]

We are providing a list of the benefits that an open–source implementation of a PKI would have. We are investigating whether such an implementation will make an impact to the standardisation process, to the adoption to PKIs, to the issue of interoperability among other implementations and the course of e–commerce.

A paper presented by the development team behind Jonah (TODO Add to bibliography) showed several areas of the standards that needed correction or clarification.

An easily accesible and unencumbered PKI will let Internet users evaluate the benefits of transparent security.

The Jonah case proved valuable to the testing of interoperability between PKI implementations.

E-commerce and PKIs?

Chapter 12. Trademarks

Linux® is a registered trademark of Linus Torvalds.

RedHat™ is a trademark or registered trademark of RedHat, Inc. in the United States and other countries.

Pentium® is a registered trademark of Intel.

The modssl FAQ is copyrighted by Ralf S. Engelschall[1].

All other trademarks and registered trademarks in this document are owned by their respective companies.

Notes

1. http://www.engelschall.com/

Chapter 13. Contributions

The chapter Chapter 2 has been taken from the SSLeay Certificate Cookbook[1] written by Frederick J. Hirsch.

The chapter Chapter 6 is based on the document `draft-ietf-pkix-roadmap` and other documents produced by the PKIX Working Group.

The author would like to thank *Massimiliano Pala and the OpenCA Group* for writing the OpenCA software.

Miguel Armas[2], contributed in Appendix D and in Section 7.2.

The author would like to thank Chris Mitchell[3] for reviewing an earlier version of this document. The author would like to thank Ted Rolle[4] for proof–reading the initial DocBook version of this document.

Readers are encouraged to send feedback and contributions to this document. Users may either use the ospkibook mailling list[5] or send e-mail directly to the author[6].

Notes

1. http://www.ultranet.com/~fhirsch/Papers/cook/ssl_cook.html
2. mailto:kuko@openca.org
3. mailto:C.Mitchell@rhbnc.ac.uk
4. mailto:ted@acacia.datacomm.com
5. http://sourceforge.net/mail/?group_id=6356
6. mailto:simos@hellug.gr

Appendix A. Perl modules

A.1. Locating Perl modules

Perl modules can be found at the Comprehensive Perl Archive Network (CPAN). For more information on CPAN, please check the CPAN FAQ[1]. We list the current versions at the time of writing. For the latest version, you may check the Modules on CPAN alphabetically[2].

You may also use one of the several mirror CPAN sites available around the world for faster access. For a list of mirror sites, please check the MIRRORED.BY list[3]. For an easy installation procedure, you may find this quickinstall[4] script interesting. For manual installation of Perl modules, please check Section A.2.

A.2. Installing Perl modules

We assume that you are familiar with locating and retrieving the Perl module of your interest. This is also described in detail in Section D.1.

1. To install a Perl module, you need to *uncompress* and *untar* it.

   ```
   user% tar xvfz a-perl-module-name.tar.gz
   ```

2. Enter the created directory and run

   ```
   user% perl Makefile.PL
   ```

   > **Note:** You may get an error that a dependancy is not met. In this case, you need to meet the requirement and try again. The most common case is that you need to install another Perl module first.

3. Then run the **make** command.

   ```
   user% make
   ```

4. Test the result with

   ```
   user% make test
   ```

5. Complete the installation with

   ```
   user% make install
   ```

Notes

1. http://www.perl.com/CPAN-local/misc/cpan-faq.html
2. http://www.perl.com/CPAN-local/modules/01modules.index.html
3. http://www.perl.com/CPAN-local/MIRRORED.BY
4. http://theoryx5.uwinnipeg.ca/auto/install.html

Appendix B. Sample Certificate Documents

B.1. Sample Encrypted Private Key in PEM format (2048 bits)

This is a sample private key in PEM format, encrypted with a pass phrase for more security.


```
---BEGIN RSA PRIVATE KEY---
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,19BE1C31E4FD722A

jO1O1v2ftXMsawM90tnXwc6xhOAT1gDBC9S8DKeca..JZNUgYYwNS0dP2UK
tmyN+XqVcAKw4HqVmChXy5b5msu8eIq3uc2NqNVtR..2ksSLukP8pxXcHyb
+sEwvM4uf8qbnHAqwnOnP9+KV9vds6BaH1eRA4CHz..n+NVZlzBsTxTlS16
/Umr7wJzVrMqK5sDiSu4WuaaBdqMGfL5hLsTjcBFD..Da2iyQmSKuVD4lIZ
yPQqjHKT70kEuSz+vdKuAzoIGNCvgQxXyqKSSX7td..1r7GBbjlIT7xgo8B
LvNaqyvLW5qKCMfWSVJr7xnP1xUU3MVoahhUPxOKX..sEvVM+tkeSPh7GxF
U9OQ79lqjt5iZVSJOzRsgxZ66ZsrG5b3xL+FQf6z5..WUM1uVAJ9zVv6sYV
JURDlKbTkS2pm84CXI6TTJUx/msopB0MFJ+QRobLk..TtteSqpOQopTy7/k
WVoiZfbjIx5yzE0gC72E5bqn/kk7looqshvHt5o1T..OeJ06cGJz4o6bhvL
E7djV3lKpKI4xhxo9nLsij87ByU4pZPZwa3ahh02r..VhkUWPmqwElO9mSf
7QQjk4VpzzxuHx9XKPnYMOE9p8EEJiAyMW+Ms6blh..t3P9GPUJ9aRaH7yl
uUwJ2JXIZu1us4oObAi2mAmSWBebKiWQYBzuNDryK..iNAcY/7kndVqcxV2
PCFMM9TwsiJq6r38+CfvdIkol7sQcPf4us1fpVJSc..EB9U7obrrgX6s2PG
yye805Bd/4dIFb0CqYrejbBfl5ZDXpFIMCrpETEQG..AdnFGO3wysU/Eylu
qzIsBzPdGAoSqa/Y+jdpQRIpWK3vc+nVKMrAzDOC+..pp092QQvokWkyHzO
B4H4DDDuZo9lnt0YgUA0zN6BGhPh6VPys9NgoGPCu..XbymSIq0xLdm7Yb6
2lvmO9/MslBMwNphEWc4EkkUNaoPf6V8OZ33B81Ch..D0bIvA5RhgX3ysd1
sk7m2Q7oNdJWLX8IP9Ubz2L3VpQQ20Vd90yx26smE..xuNXLk0JAgVgagBK
7nbB88S60oXjF1lTckLPfZrCLjFW7M1A4m2f/Xbee..CcS0fPTKp7DF4dwn
ifDTV8A4wSCe+MqWuWqOzYcYE8PpsM7WL2xsV3yPe..X3PF2s/Xub84GPD+
cmYQxBoghfTFiFBmyR85ivc5c+jIxY1PF4r2cO5Gv..3PWTmv8/9W7QvL0g
LPp6cKH9b9d+DDueLvuF3GYG4RTdJrYpn8v7cX+jo..cML31exYsqzCHXad
TqFpESeSK0Zrk3pNRDAHf8wh/cKaElJzGrfSUtCTr..+ct8Auw9ZQmJ0+Dv
XHhV92QUxvAgenoTQn0PBz87AEMQ6pM6413yEv6Ab..rLurwA5E1JoZhZLt
a1/eZjUYAxDn07eabeiAvYwuwCqDQD1SQ6BIJ4taN..c8kfiaGpZCbWCic8
---END RSA PRIVATE KEY---
```


Note: Seven collumns have been removed and replaced with dots. This was done for printing purposes, as the full text exceeds the page margins, when generating the DVI document version. If you would like to see the private key, just pass to the next section.

Note: The careful reader will see that the encryption algorithm used is 3DES. 3DES is a US standard (NIST FIPS 46-3). As a sidenote, 3DES is DES used 3 times in ENCRYPT, DECRYPT and finally ENCRYPT mode (EDE). The DES component was used in Cipher Block Chaining (CBC) mode, a common mode of encryption for block ciphers.

B.2. Sample Private Key in PEM format (2048 bits)

This is a sample private key in PEM format.


```
---BEGIN RSA PRIVATE KEY---
MIIEpQIBAAKCAQEA3Tz2mr7SZiAMfQyuvBjM9Oi..Z1BjP5CE/Wm/Rr500P
RK+Lh9x5eJPo5CAZ3/ANBE0sTK0ZsDGMak2m1g7..3VHqIxFTz0Ta1d+NAj
wnLe4nOb7/eEJbDPkk05ShhBrJGBKKxb8n104o/..PdzbFMIyNjJzBM2o5y
5A13wiLitEO7nco2WfyYkQzaxCw0AwzlkVHiIyC..71pSzkv6sv+4IDMbT/
XpCo8L6wTarzrywnQsh+etLD6FtTjYbbrvZ8RQM..Hg2qxraAV++HNBYmNW
s0duEdjUbJK+ZarypXI9TtnS4o1Ckj7POfljiQI..IBAFyidxtqRQyv5KrD
kbJ+q+rsJxQlaipn2M4lGuQJEfIxELFDyd3XpxP..Un/82NZNXlPmRIopXs
2T91jiLZEUKQw+n73j26adTbteuEaPGSrTZxBLR..yssO0wWomUyILqVeti
6AkL0NJAuKcucHGqWVgUIa4g1haE0ilcm6dWUDo..fd+PpzdCJf1s4NdUWK
YV2GJcutGQb+jqT5DTUqAgST7N8M28rwjK6nVMI..BUpP0xpPnuYDyPOw6x
4hBt8DZQYyduzIXBXRBKNiNdv8fum68/5klHxp6..4HRkMUL958UVeljUsT
BFQlO9UCgYEA/VqzXVzlz8K36VSTMPEhB5zBATV..PRiXtYK1YpYV4/jSUj
vvT4hP8uoYNC+BlEMi98LtnxZIh0V4rqHDsScAq..VyeSLH0loKMZgpwFEm
bEIDnEOD0nKrfT/9K9sPYgvB43wsLEtUujaYw3W..Liy0WKmB8CgYEA34xn
1QlOOhHBn9Z8qYjoDYhvcj+a89tD9eMPhesfQFw..rsfGcXIonFmWdVygbe
6Doihc+GIYIq/QP4jgMksE1ADvczJSke92ZfE2i..fitBpQERNJO0BlabfP
ALs5NssKNmLkWS2U2BHCbv4DzDXwiQB37KPOL1c..kBHfF2/htIs20d1UVL
+PK+aXKwguI6bxLGZ3of0UH+mGsSl0mkp7kYZCm..OTQtfeRqP8rDSC7DgA
kHc5ajYqh04AzNFaxjRo+M3IGICUaOdKnXd0Fda..QwfoaX4QlRTgLqb7AN
ZTzM9WbmnYoXrx17kZlT3lsCgYEAm757XI3WJVj..WoLj1+v48WyoxZpcai
uv9bT4Cj+lXRS+gdKHK+SH7J3x2CRHVS+WH/SVC..DxuybvebDoT0TkKiCj
BWQaGzCaJqZa+POHK0klvS+9ln0/6k539p95tfX..X4TCzbVG6+gJiX0ysz
Yfehn5MCgYEAkMiKuWHCsVyCab3RUf6XA9gd3qY..fCTIGtS1tR5PgFIV+G
engiVoWc/hkj8SBHZz1n1xLN7KDf8ySU06MDggB..hJ+gXJKy+gf3mF5Kmj
DtkpjGHQzPF6vOe907y5NQLvVFGXUq/FIJZxB8k..fJdHEm2M4=
---END RSA PRIVATE KEY---
```


Note: Seven collumns have been removed and replaced with the dots. This was done for printing purposes, as the full text exceeds the page margins, when generating the DVI document version. If you really would like to see the private key, just pass to the next section.

B.3. Sample Private Key in TXT format (2048 bits)

This is a sample private key in TXT format.

```
Private-Key: (2048 bit)
modulus:
    00:dd:3c:f6:9a:be:d2:66:20:0c:7d:0c:ae:bc:18:
    cc:f4:e8:89:8d:16:b3:5c:16:75:06:33:f9:08:4f:
    d6:9b:f4:6b:e7:4d:0f:44:af:8b:87:dc:79:78:93:
    e8:e4:20:19:df:f0:0d:04:4d:2c:4c:ad:19:b0:31:
    8c:6a:4d:a6:d6:0e:e8:ae:e2:37:75:8d:d5:1e:a2:
    31:15:3c:f4:4d:ad:5d:f8:d0:23:c2:72:de:e2:73:
    9b:ef:f7:84:25:b0:cf:92:4d:39:4a:18:41:ac:91:
    81:28:ac:5b:f2:7d:74:e2:8f:f9:a7:c1:c0:b1:93:
    dd:cd:b1:4c:23:23:63:27:30:4c:da:8e:72:e4:0d:
    77:c2:22:e2:b4:43:bb:9d:ca:36:59:fc:98:91:0c:
    da:c4:2c:34:03:0c:e5:91:51:e2:23:20:ae:68:5e:
    30:8f:9e:f5:a5:2c:e4:bf:ab:2f:fb:82:03:31:b4:
    ff:5e:90:a8:f0:be:b0:4d:aa:f3:af:2c:27:42:c8:
    7e:7a:d2:c3:e8:5b:53:8d:86:db:ae:f6:7c:45:03:
    35:b6:52:9d:a0:c1:e0:da:ac:6b:68:05:7e:f8:73:
    41:62:63:56:b3:47:6e:11:d8:d4:6c:92:be:65:aa:
    f2:a5:72:3d:4e:d9:d2:e2:8d:42:92:3e:cf:39:f9:
    63:89
publicExponent: 65537 (0x10001)
privateExponent:
    5c:a2:77:1b:6a:45:0c:af:e4:aa:c3:91:b2:7e:ab:
    ea:ec:27:14:25:6a:2a:67:d8:ce:25:1a:e4:09:11:
    f2:31:10:b1:43:c9:dd:d7:a7:13:d7:14:21:91:c5:
    15:27:ff:cd:8d:64:d5:e5:3e:64:48:a2:95:ec:d9:
    3f:75:8e:22:d9:11:42:90:c3:e9:fb:de:3d:ba:69:
    d4:db:b5:eb:84:68:f1:92:ad:36:71:04:b4:4a:f6:
    03:2f:5f:6c:ac:b0:ed:30:5a:89:94:c8:82:ea:55:
    eb:62:e8:09:0b:d0:d2:40:b8:a7:2e:70:71:aa:59:
    58:14:21:ae:20:d6:16:84:d2:29:5c:9b:a7:56:50:
    3a:10:0b:c6:70:2b:97:dd:f8:fa:73:74:22:5f:d6:
    ce:0d:75:45:8a:61:5d:86:25:cb:ad:19:06:fe:8e:
    a4:f9:0d:35:2a:02:04:93:ec:df:0c:db:ca:f0:8c:
    ae:a7:54:c2:37:a1:11:7b:9f:40:54:a4:fd:31:a4:
    f9:ee:60:3c:8f:3b:0e:b1:e2:10:6d:f0:36:50:63:
    27:6e:cc:85:c1:5d:10:4a:36:23:5d:bf:c7:ee:9b:
    af:3f:e6:49:47:c6:9e:b8:00:b0:d9:d2:de:07:46:
    43:14:2f:de:7c:51:57:a5:8d:4b:13:04:54:25:3b:
    d5
prime1:
    00:fd:5a:b3:5d:5c:e5:cf:c2:b7:e9:54:93:30:f1:
    21:07:9c:c1:01:35:64:7e:90:93:a7:13:d1:89:7b:
    58:2b:56:29:61:5e:3f:8d:25:23:be:f4:f8:84:ff:
    2e:a1:83:42:f8:19:44:32:2f:7c:2e:d9:f1:64:88:
```

```
        74:57:8a:ea:1c:3b:12:70:0a:be:86:28:3b:4c:d5:
        72:79:22:c7:d2:5a:0a:31:98:29:c0:51:26:6c:42:
        03:9c:43:83:d2:72:ab:7d:3f:fd:2b:db:0f:62:0b:
        c1:e3:7c:2c:2c:4b:54:ba:36:98:c3:75:b1:8f:69:
        4b:5b:62:e2:cb:45:8a:98:1f
    prime2:
        00:df:8c:67:d5:09:4e:3a:11:c1:9f:d6:7c:a9:88:
        e8:0d:88:6f:72:3f:9a:f3:db:43:f5:e3:0f:85:eb:
        1f:40:5c:26:6f:31:49:82:4a:ec:7c:67:17:22:89:
        c5:99:67:55:ca:06:de:e8:3a:22:85:cf:86:21:82:
        2a:fd:03:f8:8e:03:24:b0:4d:40:0e:f7:33:25:29:
        1e:f7:66:5f:13:68:b6:d2:5b:a8:54:17:e2:b4:1a:
        50:11:13:49:3b:40:65:69:b7:cf:00:bb:39:36:cb:
        0a:36:62:e4:59:2d:94:d8:11:c2:6e:fe:03:cc:35:
        f0:89:00:77:ec:a3:ce:2f:57
    exponent1:
        00:c2:f9:01:1d:f1:76:fe:1b:48:b3:6d:1d:d5:45:
        4b:f8:f2:be:69:72:b0:82:e2:3a:6f:12:c6:67:7a:
        1f:d1:41:fe:98:6b:12:97:49:a4:a7:b9:18:64:29:
        89:b6:4c:30:c6:83:93:42:d7:de:46:a3:fc:ac:34:
        82:ec:38:00:90:77:39:6a:36:2a:87:4e:00:cc:d1:
        5a:c6:34:68:f8:cd:c8:18:80:94:68:e7:4a:9d:77:
        74:15:d6:b3:64:ca:50:85:14:30:7e:86:97:e1:09:
        51:4e:02:ea:6f:b0:0d:65:3c:cc:f5:66:e6:9d:8a:
        17:af:1d:7b:91:99:53:de:5b
    exponent2:
        00:9b:be:7b:5c:8d:d6:25:58:d7:98:1f:5b:cc:d5:
        a8:2e:3d:7e:bf:8f:16:ca:8c:59:a5:c6:a2:ba:ff:
        5b:4f:80:a3:fa:55:d1:4b:e8:1d:28:72:be:48:7e:
        c9:df:1d:82:44:75:52:f9:61:ff:49:50:92:b7:67:
        b3:c1:80:f1:bb:26:ef:79:b0:e8:4f:44:e4:2a:20:
        a3:05:64:1a:1b:30:9a:26:a6:5a:f8:f3:87:2b:49:
        25:bd:2f:bd:96:7d:3f:ea:4e:77:f6:9f:79:b5:f5:
        f1:50:80:c7:6c:65:f8:4c:2c:db:54:6e:be:80:98:
        97:d3:2b:33:61:f7:a1:9f:93
    coefficient:
        00:90:c8:8a:b9:61:c2:b1:5c:82:69:bd:d1:51:fe:
        97:03:d8:1d:de:a6:23:be:61:0b:02:d7:c2:4c:81:
        ad:4b:5b:51:e4:f8:05:21:5f:86:7a:78:22:56:85:
        9c:fe:19:23:f1:20:47:67:3d:67:d7:12:cd:ec:a0:
        df:f3:24:94:d3:a3:03:82:00:74:0b:68:1d:5b:88:
        49:fa:05:c9:2b:2f:a0:7f:79:85:e4:a9:a3:0e:d9:
        29:8c:61:d0:cc:f1:7a:bc:e7:bd:d3:bc:b9:35:02:
        ef:54:51:97:52:af:c5:20:96:71:07:c9:17:00:6d:
        ab:7d:27:c9:74:71:26:d8:ce
```

Note: The numbers are in hexadecimal notation where each couple of digits represents 8 bits.

In decimal, the modulus n is:

```
27928727520532098560054510086934803266769027328779773633
51762493251995978285544035350906266382585272722398629867
67263282027760422651274751164233304322779357458680526177
93594651686619933029730312573799176384081348734718092523
53476550057243981913102899068449856388885987417785575633
66522578044678796800808595716146657069948593436088106761
86674067708949755093039975941211253008157978789036441127
01109572656021257137086334620169063315388954284609394192
32250643688514600699603929824545296848370051254650037973
10139479221307918200583851065828489354285517184240655579
54933738674003130224949637988279936009837240188474132980
1
```

If an adversary managed to factorise the modulus, she would come up with the factors p and q, where p is:

```
17791143933509595918127954499653383601218835098160342274
21719349464132778400846891474457120589082133325302604179
82181001327467441044697854896458761089076165690493808885
78606941384914032562858753139200694087767527290102835209
36343115102676302117059691295229400834867089684114302209
27632138221540171427701495839
```

and q is:

```
15698106667513592225651910118661853088086996081175911345
49581990193390503622003253143718326860723480921952218366
69795595987275285870475032000847646645415387334949112223
81409068648841957504994872889663428380162653646162371919
71899699949089072105502530930366392712822832371160724348
51400420434671809603239292759
```

The coefficient and the exponents 1 and 2 are used to increase the performance of those operations of RSA that make use of the private key. That is, they are used by the owner of the key and they are only visible to her.

Note: For information on software that works with natural numbers of arbitrary size, you may find the GMP library[1] quite useful.

B.4. Sample CA Certificate in PEM format

This is a sample Certificate in PEM format.


```
---BEGIN CERTIFICATE---
MIIEczCCA1ugAwIBAgIBADANBgkqhkiG9w0BAQQFAD..AkGA1UEBhMCR0Ix
EzARBgNVBAgTClNvbWUtU3RhdGUxFDASBgNVBAoTC0..0EgTHRkMTcwNQYD
VQQLEy5DbGFzcyAxIFB1YmxpYyBQcmltYXJ5IENlcn..XRpb24gQXV0aG9y
aXR5MRQwEgYDVQQDEwtCZXN0IENBIEx0ZDAeFw0wMD..TUwMTZaFw0wMTAy
MDQxOTUwMTZaMIGHMQswCQYDVQQGEwJHQjETMBEGA1..29tZS1TdGF0ZTEU
MBIGA1UEChMLQmVzdCBDQSBMdGQxNzA1BgNVBAsTLk..DEgUHVibGljIFBy
aW1hcnkgQ2VydGlmaWNhdGlvbiBBdXRob3JpdHkxFD..AMTC0Jlc3QgQ0Eg
THRkMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCg..Tz2mr7SZiAMfQyu
vBjM9OiJjRazXBZ1BjP5CE/Wm/Rr500PRK+Lh9x5eJ../ANBE0sTK0ZsDGM
ak2m1g7oruI3dY3VHqIxFTz0Ta1d+NAjwnLe4nOb7/..k05ShhBrJGBKKxb
8n104o/5p8HAsZPdzbFMIyNjJzBM2o5y5A13wiLitE..fyYkQzaxCw0Awzl
kVHiIyCuaF4wj571pSzkv6sv+4IDMbT/XpCo8L6wTa..sh+etLD6FtTjYbb
rvZ8RQM1tlKdoMHg2qxraAV++HNBYmNWs0duEdjUbJ..XI9TtnS4o1Ckj7P
OfljiQIDAQABo4HnMIHkMB0GA1UdDgQWBBQ8urMCRL..5AkIp9NJHJw5TCB
tAYDVR0jBIGsMIGpgBQ8urMCRLYYMHUKU5AkIp9NJH..aSBijCBhzELMAkG
A1UEBhMCR0IxEzARBgNVBAgTClNvbWUtU3RhdGUxFD..AoTC0Jlc3QgQ0Eg
THRkMTcwNQYDVQQLEy5DbGFzcyAxIFB1YmxpYyBQcm..ENlcnRpZmljYXRp
b24gQXV0aG9yaXR5MRQwEgYDVQQDEwtCZXN0IENBIE..DAMBgNVHRMEBTAD
AQH/MA0GCSqGSIb3DQEBBAUAA4IBAQC1uYBcsSncwA..DCsQer772C2ucpX
xQUE/C0pWWm6gDkwd5D0DSMDJRqV/weoZ4wC6B73f5..bLhGYHaXJeSD6Kr
XcoOwLdSaGmJYslLKZB3ZIDEp0wYTGhgteb6JFiTtn..sf2xdrYfPCiIB7g
BMAV7Gzdc4VspS6ljrAhbiiawdBiQlQmsBeFz9JkF4..b3l8BoGN+qMa56Y
It8una2gY4l2O//on88r5IWJlm1L0oA8e4fR2yrBHX..adsGeFKkyNrwGi/
7vQMfXdGsRrXNGRGnX+vWDZ3/zWI0joDtCkNnqEpVn..HoX
---END CERTIFICATE---
```


Note: This is the CA Certificate, also called the Root CA Certificate. The goal is to make the CA Certificate available to the bigger possible audience. Also, we would ask companies that make WWW browsers to include it in their list of Root CA Certificates.

B.5. Sample CA Certificate in TXT format

This is a sample Certificate in TXT format.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 0 (0x0)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=GB, ST=Surrey, O=Best CA Ltd,
          OU=Class 1 Public Primary Certification Authority,
          CN=Best CA Ltd
        Validity
          Not Before: Feb  5 19:50:16 2000 GMT
          Not After : Feb  4 19:50:16 2001 GMT
        Subject: C=GB, ST=Surrey, O=Best CA Ltd,
          OU=Class 1 Public Primary Certification Authority,
          CN=Best CA Ltd
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
              Modulus (2048 bit):
               00:dd:3c:f6:9a:be:d2:66:20:0c:7d:0c:ae:bc:18:
               cc:f4:e8:89:8d:16:b3:5c:16:75:06:33:f9:08:4f:
               d6:9b:f4:6b:e7:4d:0f:44:af:8b:87:dc:79:78:93:
               e8:e4:20:19:df:f0:0d:04:4d:2c:4c:ad:19:b0:31:
               8c:6a:4d:a6:d6:0e:e8:ae:e2:37:75:8d:d5:1e:a2:
               31:15:3c:f4:4d:ad:5d:f8:d0:23:c2:72:de:e2:73:
               9b:ef:f7:84:25:b0:cf:92:4d:39:4a:18:41:ac:91:
               81:28:ac:5b:f2:7d:74:e2:8f:f9:a7:c1:c0:b1:93:
               dd:cd:b1:4c:23:23:63:27:30:4c:da:8e:72:e4:0d:
               77:c2:22:e2:b4:43:bb:9d:ca:36:59:fc:98:91:0c:
               da:c4:2c:34:03:0c:e5:91:51:e2:23:20:ae:68:5e:
               30:8f:9e:f5:a5:2c:e4:bf:ab:2f:fb:82:03:31:b4:
               ff:5e:90:a8:f0:be:b0:4d:aa:f3:af:2c:27:42:c8:
               7e:7a:d2:c3:e8:5b:53:8d:86:db:ae:f6:7c:45:03:
               35:b6:52:9d:a0:c1:e0:da:ac:6b:68:05:7e:f8:73:
               41:62:63:56:b3:47:6e:11:d8:d4:6c:92:be:65:aa:
               f2:a5:72:3d:4e:d9:d2:e2:8d:42:92:3e:cf:39:f9:
               63:89
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                3C:BA:B3:02:44:B6:18:30:75:0A:53:90:24:22:\
                  9F:4D:24:72:70:E5
            X509v3 Authority Key Identifier:
                keyid:3C:BA:B3:02:44:B6:18:30:75:0A:53:90:\
                  24:22:9F:4D:24:72:70:E5
                DirName:/C=GB/ST=Some-State/O=Best CA Ltd/\
```

```
                OU=Class 1 Public Primary Certification
                Authority/CN=Best CA Ltd
              serial:00

          X509v3 Basic Constraints:
              CA:TRUE
 Signature Algorithm: md5WithRSAEncryption
     b5:b9:80:5c:b1:29:dc:c0:03:db:28:c8:a3:08:30:ac:41:ea:
     fb:ef:60:b6:b9:ca:57:c5:05:04:fc:2d:29:59:69:ba:80:39:
     30:77:90:f4:0d:23:03:25:1a:95:ff:07:a8:67:8c:02:e8:1e:
     f7:7f:96:06:3e:7e:90:99:b2:e1:19:81:da:5c:97:92:0f:a2:
     ab:5d:ca:0e:c0:b7:52:68:69:89:62:c9:4b:29:90:77:64:80:
     c4:a7:4c:18:4c:68:60:b5:e6:fa:24:58:93:b6:72:ef:5c:9b:
     a0:3a:c7:f6:c5:da:d8:7c:f0:a2:20:1e:e0:04:c0:15:ec:6c:
     dd:73:85:6c:a5:2e:a5:8e:b0:21:6e:28:9a:c1:d0:62:42:54:
     26:b0:17:85:cf:d2:64:17:89:c3:99:94:cf:0d:bd:e5:f0:1a:
     06:37:ea:8c:6b:9e:98:22:df:2e:9d:ad:a0:63:89:76:3b:ff:
     e8:9f:cf:2b:e4:85:89:96:6d:4b:d2:80:3c:7b:87:d1:db:2a:
     c1:1d:71:7a:d1:fe:36:59:a7:6c:19:e1:4a:93:23:6b:c0:68:
     bf:ee:f4:0c:7d:77:46:b1:1a:d7:34:64:46:9d:7f:af:58:36:
     77:ff:35:88:d2:3a:03:b4:29:0d:9e:a1:29:56:78:60:fe:00:
     15:98:7a:17
```

Note: This is the CA Certificate, also called the Root CA Certificate. It is in TXT format which is another way to say that it is in a human–readable format.

Note: Notice the modulus. It has 2048 bits and it is the product of two big primes. Each prime has about 1024 bits. The security of the certificate relies on the difficulty to factorise this 2048–bits long (or over 600 decimal digits long) number. Since we generated this key–pair, we already know these two primes. All the mentioned values, in decimal, are in Section B.3.

Note: We have chosen RSA for the public key algorithm. We could have chosen one of the alternatives, like El Gamal or elliptic curves.

B.6. Sample certificate request in PEM format

This is a sample certificate request in PEM format.

```
---BEGIN CERTIFICATE REQUEST---
MIIC5DCCAcwCAQAwgZ4xCzAJBgNVBAYTAkdCMQ8wDQ..wZTdXJyZXkxDjAM
BgNVBAcTBUVnaGFtMRowGAYDVQQKExFBcnRzIEJ1aW..Ex0ZDEWMBQGA1UE
CxMNRGVwdC4gSGlzdG9yeTEZMBcGA1UEAxMQU2ltb3..XRlbGxpczEfMB0G
CSqGSIb3DQEJARYQc2ltb3NAb3BlbmNhLm9yZzCCAS..oZIhvcNAQEBBQAD
ggEPADCCAQoCggEBAN089pq+0mYgDH0MrrwYzPToiY..QYz+QhP1pv0a+dN
D0Svi4fceXiT6OQgGd/wDQRNLEytGbAxjGpNptYO6K..R6iMRU89E2tXfjQ
I8Jy3uJzm+/3hCWwz5JNOUoYQayRgSisW/J9dOKP+a..c2xTCMjYycwTNqO
cuQNd8Ii4rRDu53KNln8mJEM2sQsNAMM5ZFR4iMgrm..aUs5L+rL/uCAzG0
/16QqPC+sE2q868sJ0LIfnrSw+hbU42G2672fEUDNb..Nqsa2gFfvhzQWJj
VrNHbhHY1GySvmWq8qVyPU7Z0uKNQpI+zzn5Y4kCAw..A0GCSqGSIb3DQEB
BAUAA4IBAQC2y+cj6EmXzHunozGDv3fu9rw+T7SLrh..tY0K4L5w/4jOXRS
Q5VHn8o2M1E8JE2iK9tg24Nkh9GvkODxbP2ABYKslT..pZ8KC+wHCDZyXCY
Fgrass8oENyZG2VFFlfgbtRUssdKldJcJKpgnsHyt1..xJ11Y0t0n9ruayu
Oqp9lTEu6e+Lhhcuad4JncXiSR0EdG75AqN9bbI8NG..tgzzOrvfYNtGe9t
EI/wriWPQvl4QLJ5VevzuIC62dQztVQmDR2hPd2J8/..1ArMX5olNCef2XB
Rghkcki7R/ZpuuwaXkT+qDu+eoDwju0P
---END CERTIFICATE REQUEST---
```


Note: This is the certificate request that a Certification Authority needs to sign. Typical CAs could be Verisign, Thawte and of course OpenCA.

B.7. Sample certificate request in TXT format

This is a sample certificate request in TXT format.

```
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=GB, ST=Surrey, L=Egham,
 O=Arts Building Ltd,
 OU=Dept. History,
 CN=Simos Xenitellis/Email=simos@openca.org
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
             Modulus (2048 bit):
               00:dd:3c:f6:9a:be:d2:66:20:0c:7d:0c:ae:bc:18:
               cc:f4:e8:89:8d:16:b3:5c:16:75:06:33:f9:08:4f:
               d6:9b:f4:6b:e7:4d:0f:44:af:8b:87:dc:79:78:93:
               e8:e4:20:19:df:f0:0d:04:4d:2c:4c:ad:19:b0:31:
```

```
                8c:6a:4d:a6:d6:0e:e8:ae:e2:37:75:8d:d5:1e:a2:
                31:15:3c:f4:4d:ad:5d:f8:d0:23:c2:72:de:e2:73:
                9b:ef:f7:84:25:b0:cf:92:4d:39:4a:18:41:ac:91:
                81:28:ac:5b:f2:7d:74:e2:8f:f9:a7:c1:c0:b1:93:
                dd:cd:b1:4c:23:23:63:27:30:4c:da:8e:72:e4:0d:
                77:c2:22:e2:b4:43:bb:9d:ca:36:59:fc:98:91:0c:
                da:c4:2c:34:03:0c:e5:91:51:e2:23:20:ae:68:5e:
                30:8f:9e:f5:a5:2c:e4:bf:ab:2f:fb:82:03:31:b4:
                ff:5e:90:a8:f0:be:b0:4d:aa:f3:af:2c:27:42:c8:
                7e:7a:d2:c3:e8:5b:53:8d:86:db:ae:f6:7c:45:03:
                35:b6:52:9d:a0:c1:e0:da:ac:6b:68:05:7e:f8:73:
                41:62:63:56:b3:47:6e:11:d8:d4:6c:92:be:65:aa:
                f2:a5:72:3d:4e:d9:d2:e2:8d:42:92:3e:cf:39:f9:
                63:89
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: md5WithRSAEncryption
        b6:cb:e7:23:e8:49:97:cc:7b:a7:a3:31:83:bf:77:ee:f6:bc:
        3e:4f:b4:8b:ae:1b:ed:e2:82:89:2a:d6:34:2b:82:f9:c3:fe:
        23:39:74:52:43:95:47:9f:ca:36:33:51:3c:24:4d:a2:2b:db:
        60:db:83:64:87:d1:af:90:e0:f1:6c:fd:80:05:82:ac:95:3c:
        4f:a0:3d:f1:96:96:7c:28:2f:b0:1c:20:d9:c9:70:98:16:0a:
        da:b2:cf:28:10:dc:99:1b:65:45:16:57:e0:6e:d4:54:b2:c7:
        4a:95:d2:5c:24:aa:60:9e:c1:f2:b7:5e:a7:24:fe:6f:6f:12:
        75:d5:8d:2d:d2:7f:6b:b9:ac:ae:3a:aa:7d:95:31:2e:e9:ef:
        8b:86:17:2e:69:de:09:9d:c5:e2:49:1d:04:74:6e:f9:02:a3:
        7d:6d:b2:3c:34:64:8f:ec:33:e3:56:d8:33:cc:ea:ef:7d:83:
        6d:19:ef:6d:10:8f:f0:ae:25:8f:42:f9:78:40:b2:79:55:eb:
        f3:b8:80:ba:d9:d4:33:b5:54:26:0d:1d:a1:3d:dd:89:f3:fb:
        bf:f0:c7:4a:73:50:2b:31:7e:68:94:d0:9e:7f:65:c1:46:08:
        64:72:48:bb:47:f6:69:ba:ec:1a:5e:44:fe:a8:3b:be:7a:80:
        f0:8e:ed:0f
```

Note: This is the expanded version of the certificate request from Section B.6. You can notice the user information and the *n*, *e* from RSA.

Notes

1. http://www.swox.com/gmp/

Appendix C. Description of Public Key Algorithms

C.1. How does RSA work?

We show a high–level though working description of RSA. Then, we give an example with easy to work with numbers.

C.1.1. Description

To initialise RSA, follow the steps

1. Pick two large primes, `p` and `q`.
2. Find `N` = `p` * `q`. `N` is the RSA modulus.
3. Let `e` be a number relatively prime to `(p-1)*(q-1)`.
4. Find `d`, so that `d*e = 1 mod (p-1)*(q-1)` .
5. The set `(e, N)` is the public key. Make it known to every one.

   The set `(d, N)` is the private key. Keep it private and safe.

To encrypt a message `m`,

1. Make sure `m < N`, otherwise chop `m` in suitably small pieces and perform RSA on each individual piece.
2. Compute `c = m ^ e mod N`
3. `c` is the encrypted message

To decrypt a ciphertext `c`,

1. Compute `m = c ^ d mod N`
2. `m` is the original message

To sign message `m`,

1. Compute `s = m ^ d mod N`
2. `s` is the digital signature. Send along with message `m`.

To verify signed message `s`,

1. Compute `m = s ^ e mod N`
2. Check if `m` from above calculation is the same with message sent.

C.1.2. Practical example

TODO

C.2. How does El Gamal work?

TODO

C.2.1. Description

TODO

C.2.2. Example

TODO

Appendix D. OpenCA Installation details

As described in Figure 7-1, OpenCA requires three distinctive servers. However, this makes the software less accesible. We describe how to install all the components on a single computer.

We assume the character of Woody Allen in the movie "Bananas", where, while he was on trial in the court, he was playing both the role of the defendant and the laywer by switching places quickly.

First, we determine the software components to install and the server on which we install them.

Table D-1. Software installation matrix

Software	CAServer	RAServer	RAOperator
Perl Generic modules	✓	✓	✓
OpenCA Perl modules	✓	✓	•
WWW Server	✓	✓	✓
SSL/TSL module	✓	✓	✓
LDAP Server	•	•	✓
OpenSSL	•	•	•

Note: The above table is not yet final and is subject to changes as the project evolves.

Using the above table, you may proceed with the installation, as described in the following chapters. Keep in mind that if you are doing an all–in–one installation — all servers on a single workstation — then you do not need to install the same software component multiple times or in different directories. We will note any special configuration setting to be made in regard to this issue.

D.1. Software installation sequence

It is recommended that the software components be installed in this sequence:

D.1.1. Installation of Perl modules

Information about how to find the latest version of a Perl module can be found at Appendix A.

Note: These Perl modules must be installed in the sequence shown because of dependencies. However, if you make a mistake in the sequence, you receive an informative error that indicates the module was skipped.

1. `Convert::BER` is a perl object class implementation to encode and decode objects as described by ITU-T standard X.209 (ASN.1) using Basic Encoding Rules (BER). The filename is `Convert-BER-1.26.tar.gz`[1]
2. `MIME::Base64` and `MIME::QuotedPrint` provide a base64 encoder/decoder and a quoted-printable encoder/decoder. These encoding methods are specified in RFC 2045 – MIME (Multipurpose Internet Mail Extensions). The filename is `MIME-Base64-2.11.tar.gz`[2]
3. The `URI` perl object class provides functionality regarding the Uniform Resource Identifier, as specified in RFC 2396. The filename is `URI-1.04.tar.gz`[3]
4. The `Digest::*` perl object class provides implementations for the MD5 (RFC 1321), MD2 (RFC 1319) and SHA-1 (FIPS PUB 180-1) hash functions. Also, an implementation of the HMAC (RFC 2104) MAC function is provided. The filename is `Digest-MD5-2.09.tar.gz`[4]
5. perl-ldap provides access to LDAP servers. A requirement to install it is to already have `Convert::BER` installed on your system. The filename is `perl-ldap-0.13.tar.gz`[5]
6. I have the idea that this and the above have overlapping functionality. The filename is `Net-LDAPapi-1.42.tar.gz`[6]

D.1.2. Installation of OpenCA–specific modules

The OpenCA–specific modules can be found at either at CPAN or at the OpenCA WWW site.

The functionality of these perl modules is not entirely OpenCA–specific. In general, they help to parse configuration files.

1. This perl module is used in order to access the configuration files of OpenCA. Currently, the configuration files are

   - `ca.conf`
   - `raserver.conf`
   - `secure.cnf`

   The filename is `OpenCA-Configuration-1.2.tar.gz`[7]
2. This perl module provides access to configuration variables that can have three states. It is used to ease the access to the OpenCA configuration files. The filename is `OpenCA-TRIStateCGI-1.02.tar.gz`[8]

D.1.3. Installation of OpenCA

This is described in three major sections, the installation of the CAServer, the RAServer and the RAOperator(s).

The installation procedure involves setting up the configuration files, copying the HTML pages to the appropriate directories and finally adding the CGI scripts in the corresponding directories.

D.1.3.1. CAServer Installation

This is the installation of the Certification Authority. Please refer to Figure 7-1 for more information.

It is assumed that you have uncompressed and *untarred* the OpenCA software with the following command.

```
root# tar xvfz OpenCA-0.2.0.tar.gz
```

To install the software, enter the directory created (`OpenCA-0.2.0`) and type

```
root# make install-ca
```

Use the following parameters when installing the OpenCA component for the CAServer.

Table D-2. CAServer installation parameters

Parameter	Value
OpenSSL installation directory	`/usr/local/ssl`
Base directory for CAServer	`/usr/local/RAServer`
Webserver user	nobody.nobody
Use found OpenSSL command	**Y**
Continue installation	**yes**
Edit openssl.cnf	Check Section D.2

Subsequently, to install the WWW pages that accompany the CAServer do

```
root# make install-ca-web
```

Use the following parameters when installing the WWW pages of the OpenCA component for the

CAServer.

Table D-3. RAServer WWW Server installation parameters

Parameter	Value
HTML pages directory	`/usr/local/apache/htdocs/ca`
CGI directory	`/usr/local/apache/cgi-bin`
Continue installation	**`yes`**

Finally, follow the instructions from the WWW pages to initialise the CAServer by creating the CA private key and certificate.

D.1.3.2. RAServer Installation

This is the installation of the Registration Authority. Please refer to Figure 7-1 for more information.

> **Note:** The RAServer is supposed to be installed on a separate system than the CAServer. Furthermore, it is assumed that the steps that led to the installation of the CAServer will have to be duplicated to create the RAServer. However, for limited testing purposes, all of them could be installed on the same system.

It is assumed that you have uncompressed and *untarred* the OpenCA software with the following commands.

```
root# tar xvfz OpenCA-0.2.0.tar.gz
```

To install the RAServer software, enter the directory created (`OpenCA-0.2.0`) and type

```
root# make install-raserver
root# make install-raserver-web
```

You can use the following parameters when installing the OpenCA component for the RAServer.

Table D-4. RAServer installation parameters

Parameter	Value
OpenSSL installation directory	`/usr/local/ssl`
Base directory for RAServer	`/usr/local/RAServer`
Webserver user	nobody.nobody
Use found OpenSSL command	**Y**
Continue installation	**yes**

Table D-5. RAServer WWW Server installation parameters

Parameter	Value
HTML pages directory	`/usr/local/apache/htdocs/ra`
CGI directory	`/usr/local/apache/cgi-bin`
Continue installation	**yes**

D.1.3.3. RAOperator Installation

This is the installation of the RA Operator. Please refer to Figure 7-1 for more information.

It is assumed that you have uncompressed and *untarred* the OpenCA software with the following commands.

```
root# tar xvfz OpenCA-0.2.0.tar.gz
```

To install the software, enter the directory created (`OpenCA-0.2.0`) and type

```
root# make install-secure
```

Note: Again, the RAOperator is supposed to be installed on a separate system other than the CAServer and the RAServer. Furthermore, it is assumed that the steps that led to the installation of the CAServer and the RAServer will have to be duplicated to create the RAOperator. However, for limited testing purposes, both of them could be installed on the same system. We must say that installing the CAServer, the RAServer and the RAOperators on the same system, will make it rather difficult to use and probably error-prone in the testing.

Table D-6. RAOperator WWW Server installation parameters

Parameter	Value
HTML pages directory	`/usr/local/apache/htdocs/rao`
CGI directory	`/usr/local/apache/cgi-bin`
Continue installation	**`yes`**

D.1.4. WWW Server installation

Installation of the WWW server and the SSL/TLS WWW Server component. This will be a rather lengthly procedure, unless you use RPM files. This software can be found at Section 7.2.3. Support information is at Chapter 8.

D.1.5. LDAP installation

An independent step is the installation of the LDAP software. This is usually installed on RAOperator. Recommended LDAP software is at Section 7.2.3. For support information, please see Chapter 8.

D.2. `openssl.cnf` configuration for OpenCA

These are configuration instructions for the `openssl.cnf` of the CAServer.

We describe the values in this file that require modification. Most of the default values remain the same.

- In the `[ CA_default ]` section, the value of `dir` should be changed to the directory that has the Certification Authority installed. Typically, it is `/usr/local/OpenCA`.
- In the `[ req ]` section, you should modify all the variables that their name ends with *_default* . The default values of these variables serve as an example. These are:

Table D-7. `openssl.cnf` default values

Variable	Sample value
organizationalUnitName_default	OpenCA User
0.organizationName_default	OpenCA

Variable	Sample value
countryName_default	GB
stateOrProvinceName_default	Surrey
1.organizationName_default	Arts Buildings Ltd

> **Note:** The essence of the default values is that when you create new users, you are prompted with these values. If this value applies to the user, you can accept it without having to retype it.

> **Note:** For the country name, you need to specify the ISO 3166 country code[9]. There are two- and three-letter country codes. The current configuration supports two-letter codes.

> **Note:** Notice that in some cases, the ISO 3166 is not the same with the Internet country domain name. For example, for the United Kingdom, the ISO 3166 country code is *GB*.

- In the `[ user_cert ]` section, you may need to modify the `nsCertType` variable. With this variable, you specify the capabilities of the certificate. This area will be tackled in future versions of this document.
- In the `[ user_cert ]` section, you can set the comment that appears in the Certificate Signers' Certificate window. The variable is `nsComment` and you should provide a suitable description for the certificate.
- In the `[ user_cert ]` section, you can specify the revocation URLs for both the Root CA Certificate and the other certificates.

> **Note:** In the same group of variables, care should be taken with the `nsSslServerName` variable as it crashes certain versions of the Netscape® WWW browser, if it is set.

Notes

1. http://www.perl.com/CPAN-local/authors/id/GBARR/Convert-BER-1.26.tar.gz
2. http://www.perl.com/CPAN-local/authors/id/GAAS/MIME-Base64-2.11.tar.gz

3. http://www.perl.com/CPAN-local/authors/id/GAAS/URI-1.04.tar.gz
4. http://www.perl.com/CPAN-local/authors/id/GAAS/Digest-MD5-2.09.tar.gz
5. http://www.perl.com/CPAN-local/authors/id/GBARR/perl-ldap-0.13.tar.gz
6. http://www.perl.com/CPAN-local/authors/id/CDONLEY/Net-LDAPapi-1.42.tar.gz
7. http://www.perl.com/CPAN-local/authors/id/M/MA/MADWOLF/OpenCA-Configuration-1.2.tar.gz
8. http://www.perl.com/CPAN-local/authors/id/M/MA/MADWOLF/OpenCA-TRIStateCGI-1.02.tar.gz
9. ftp://ftp.ripe.net/iso3166-countrycodes

Appendix E. License

This documentation is released under the following license.

E.1. GNU Free Documentation License

Version 1.1, March 2000

E.1.1. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

E.1.2. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

E.1.3. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

E.1.4. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

E.1.5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

1. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
2. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
3. State on the Title page the name of the publisher of the Modified Version, as the publisher.
4. Preserve all the copyright notices of the Document.

5. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
6. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
7. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
8. Include an unaltered copy of this License.
9. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
10. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
11. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
12. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
13. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
14. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

E.1.6. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

E.1.7. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

E.1.8. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document. If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

E.1.9. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

E.1.10. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

E.1.11. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/. Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Colophon

This document was written in DocBook, an SGML DTD. In order to produce HTML, DVI, PostScript® and other file formats, we used the DSSSL engine jade by James Clark. More information about all these can be found at the The DocBook DTD[1] WWW pages. The DSSSL stylesheets were provided by Normal Walsh[2]. A useful book about DocBook can be found at [WalshMuellner99].

The JadeTeX macros[3] were used for the DVI, PS and PDF output. JadeTeX is under development and specific mistakes in the rendering of this document are due to bugs (most commonly with regards with tables). It is expected to be fixed soon. The author of JadeTeX is Sebastian Rantz.

The figures were created with xfig. More information can be found at the XFIG Drawing Program for X Window System[4].

The text editor used was vim by Bram Moolenaar. More information can be found at the VIM (Vi IMproved) Home Page.[5]

This document was produced on a workstation running the Linux® Operating System.

The author of this document is Symeon (Simos) Xenitellis and he can be reached at this e–mail[6].

Notes

1. http://www.oasis-open.org/docbook/
2. http://www.nwalsh.com
3. http://www.tug.org/applications/jadetex/
4. http://www.xfig.org
5. http://www.vim.org
6. mailto:S.Xenitellis@rhbnc.ac.uk

Glossary

CAServer (OpenCA terminology)

The Certification Authority. In this document it is used to describe the CA as described in Figure 7-1

RAServer (OpenCA terminology)

The Registration Authority. In this document it is used to describe the RA as described in Figure 7-1

RAOperator (OpenCA terminology)

The front–end of the Registration Authority that interacts with the users. In this document its functionality is described at Figure 7-1

Entity authentication mechanisms

Entity authentication mechanisms allow the verification, of an entity's claimed identity, by another entity. The authenticity of the entity can be ascertained only for the instance of the authentication exchange.

Peer entity authentication

Peer entity authentication is the corroboration that a peer entity in an association is the one claimed. This service is provided for use at the establishment of, or at times during, the data transfer phase of a connection to confirm the identities of one or more of the entities connected to one or more of the other entities.

Algorithm

An unambiguous formula or set of rules for solving a problem in a finite number of steps. Algorithms for encryption are usually called Ciphers.

Certification Authority (CA)

An entity that attests to the identity of a person or an organisation. A Certificate Authority might be an external company such as VeriSign that offers certificate services or they might be an internal organisation such as a corporate MIS department. The Certificate Authority's chief function is to verify the identity of entities and issue digital certificates attesting to that identity.

The acronym CA can be found in different variations.

- Certification Authority (Used in this document and found in most documents)
- Certifying Authority (Found in the RSA Security Crypto FAQ[1])
- Certificate Authority (Found in various documents)

Certificate Request

An unsigned certificate for submission to a Certification Authority, which signs it with the Private Key. Once the certificate request gets signed, it becomes a Certificate. This term is used in PKIX terminology and it is the same with the Certificate Signing Request. We use both terms to describe the same thing.

Certificate Signing Request (CSR) (OpenCA terminology)

An unsigned certificate for submission to a Certification Authority, which signs it with the Private Key of their CA Certificate. Once the CSR is signed, it becomes a real certificate.

Cipher

An algorithm or system for data encryption. Examples are DES, IDEA, RC4, etc.

Ciphertext

The result of the encryption of ciphertext, using a cipher.

Configuration Directive

A configuration command that controls one or more aspects of a program's behavior. In Apache context these are all the command names in the first column of the configuration files.

Cross–certificate

A cross–certificate is a certificate issued by one CA to another CA which contains a CA signature key used for issuing certificates.

DER format

A binary format to encode certificates.

Digital Signature

A method of signing electronic documents (otherwise digital data) using Public Key Cryptography.

Digital Timestamp

An electronic record that mathematically links a document to a time and date.

Electronic Commerce

The exchange of goods, services and fiduciary information or instruments via distributed computer and communication networks.

Export–Crippled

Diminished in cryptographic strength (and security) in order to comply with the United States' Export Administration Regulations (EAR). Export–crippled cryptographic software is limited to a small key size, resulting in Ciphertext which usually can be decrypted by brute force.

Currently there is draft policy in the United States that provides substantial freedom to the availability of cryptographic software. This policy remains to be finalised and voted in order to become effective. Similar legislation is expected to be voted in the European Parliament soon.

Fully–Qualified Domain–Name (FQDN)

The unique name of a network entity, consisting of a hostname and a domain name that can resolve to an IP address. For example, www is a hostname, whatever.com is a domain name, and www.whatever.com is a fully–qualified domain name.

HyperText Transfer Protocol (HTTP)

The HyperText Transport Protocol is the standard transmission protocol used on the World Wide Web.

HTTPS

The HyperText Transport Protocol (Secure), the standard encrypted communication mechanism on the World Wide Web. This is actually just HTTP over SSL.

Keyholder

The entity (often a person) that controls a private key.

Key recovery

The ability of an individual, organisation or their authorised agents to obtain an extra copy of a key (or other information necessary for decryption) that enables them to decrypt the ciphertext.

Lightweight Directory Access Protocol (LDAP)

LDAP is a specification for a client–server protocol to retrieve and manage directory information.

Message Digest

A hash of a message, which can be used to verify that the contents of the message have not been altered in transit.

OpenLDAP

OpenLDAP is an open–source implementation of LDAP. It provides a stand–alone LDAP server, a stand–alone LDAP replication server, libraries implementing the LDAP protocol, and other relevant software. For more information on OpenLDAP, see http://www.openldap.org/.

OpenSSL

An open–source implementation of the SSL/TLS protocol. It is based on SSLeay. For more about OpenSSL, see http://www.openssl.org/[2].

Pass Phrase

The word or phrase that protects private key files. It prevents unauthorized users from encrypting them.

PEM format

A text (ASCII) format that can be used to encode Certificates. It is essentially the Certificate in DER format that has been encoded with Base64 and had a header and footer added.

Plaintext

The text that will be encrypted. If we decrypt succesfully a ciphertext, the result is the plaintext.

Private Key

The secret key in a Public Key Cryptography system, used to decrypt incoming messages and sign outgoing ones.

Public Key

The publically available key in a Public Key Cryptography system, used to encrypt messages bound for its owner and to verify signatures made by its owner.

Public Key Cryptography

The study and application of asymmetric encryption systems, which use one key for encryption and another for decryption. A corresponding pair of such keys constitutes a key pair. Also called Asymmetric Cryptography.

Public Key Cryptography Standards PKCS

A series of cryptographic standards dealing with public-key issues, published by RSA Laboratories.

S–expressions

Data structures that are suitable for representing arbitrary complex data structures.

Secure Sockets Layer (SSL)

A protocol created by Netscape Communications Corporation for general communication authentication and encryption over TCP/IP networks. The most popular usage is HTTPS, i.e. the HyperText Transfer Protocol (HTTP) over SSL.

Single Sign–On (SSO)

The ability to authenticate once and use several security services based on that authentication.

SSLeay

The original SSL/TLS implementation library developed by Eric A. Young[3]; see http://www.ssleay.org/[4]. Now it has been renamed to OpenSSL; see OpenSSL.

Symmetric Cryptography

The study and application of Ciphers that use a single secret key for both encryption and decryption operations.

Transport Layer Security (TLS)

The successor protocol to SSL, created by the Internet Engineering Task Force (IETF) for general communication authentication and encryption over TCP/IP networks. The current version, TLS version 1, is nearly identical with SSL version 3.

Trusted Third Party (TTP)

Another description for the Certification Authority that stresses that the keeper of the CA private key should be an organisation or an entity that has no interests or ties of any kind with the clients.

Uniform Resource Locator (URL)

The formal identifier to locate various resources on the World Wide Web. The most popular URL scheme is http. SSL uses the scheme HTTPS.

X.500

A CCITT specification for directory services.

X.509

An authentication certificate scheme recommended by the International Telecommunication Union (ITU–T) which is used for SSL/TLS authentication.

Attribute Authority (AA)

An authority trusted by one or more users to create and sign attribute certificates. It is important to note that the Attribute Authority is responsible for the attribute certificates during their whole lifetime, not just for issuing them.

Attribute Certificate (AC)

A data structure containing a set of attributes for an end-entity and some other information, which is digitally signed with the private key of the AA which issued it.

Certificate

Can refer to either an Attribute Certificate or a Public Key Certificate certificate. Where there is no distinction made the context should be assumed to apply to both an AC and a public key certificate.

Certification Authority (CA)

An authority trusted by one or more users to create and assign public key certificates. Optionally the Certification Authority may create the user's keys. It is important to note that the Certification Authority is responsible for the public key certificates during their whole lifetime, not just for issuing them.

Certificate Policy (CP)

A named set of rules that indicates the applicability of a public key certificate to a particular community or class of application with common security requirements. For example, a particular certificate policy might indicate applicability of a type of public key certificate to the authentication of electronic data interchange transactions for the trading of goods within a given price range.

Certification Practice Statement (CPS)

A statement of the practices which a Certification Authority employs in issuing public key certificates.

End–entity (EE)

A subject of a certificate who is not a Certification Authority in the Public Key Infrastructure or an Attribute Authority in the Priviledge Management Infrastructure. (An End–entity from the Public Key Infrastructure can be an Attribute Authority in the Priviledge Management Infrastructure.)

Public Key Certificate (PKC)

A data structure containing the public key of an end-entity and some other information, which is digitally signed with the private key of the Certification Authority which issued it.

Public Key Infrastructure (PKI)

The set of hardware, software, people, policies and procedures needed to create, manage, store, distribute, and revoke PKCs based on public-key cryptography.

Priviledge Management Infrastructure (PMI)

A collection of Attribute Certificates, with their issuing Attribute Authority's, subjects, relying parties, and repositories, is referred to as a Priviledge Management Infrastructure

Registration Authority (RA)

An optional entity given responsibility for performing some of the administrative tasks necessary in the registration of subjects, such as: confirming the subject's identity; validating that the subject is entitled to have the values requested in a Public Key Certificate and verifying that the subject has possession of the private key associated with the public key requested for a Public Key Certificate.

Relying Party

A user or agent (e.g., a client or server) who relies on the data in a certificate in making decisions.

Root CA

A Certification Authority that is directly trusted by an End–entity; that is, securely acquiring the value of a Root CA public key requires some out-of-band step(s). This term is not meant to imply that a Root CA is necessarily at the top of any hierarchy, simply that the CA in question is trusted directly.

Subordinate CA

A *subordinate CA* is one that is not a Root CA for the End–entity in question. Often, a subordinate CA will not be a Root CA for any entity but this is not mandatory.

Subject

A subject is the entity (Attribute Authority, Certification Authority, or End–entity) named in a certificate. Subjects can be human users, computers (as represented by Domain Name Service (DNS) names or Internet Protocol (IP) addresses), or even software agents.

Top CA

A Certification Authority that is at the top of a PKI hierarchy.

Notes

1. http://www.rsasecurity.com/rsalabs/faq/index.html
2. http://www.openssl.org/
3. mailto:eay@aus.rsa.com
4. http://www.ssleay.org/

Bibliography

Books

[FeghhiFeghhiWilliams1999] Jalal Feghhi, Jalil Feghhi, and Peter Williams, 1999, 0-201-30980-7, Addison-Wesley Publishing Company, *Digital Certificates*.

[GarfinkelSpafford1999] Simson Garfinkel and Gene Spafford, 1997, 1-56592-269-7, Edited by Deborah Russell, O'Reilly & Associates, Inc., *Web Security & Commerce*.

[MenezesOorschotVanstone1997] Alfred Menezes, Paul van Oorschot, and Scott Vanstone, 1997, 0-8493-8523-7, CRC Press LLC, *Handbook of applied cryptography*.

[Schneier1995] Bruce Schneier, 1995, 0-4711-1709-9, John Wiley & Sons, *Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd Edition*.

[WalshMuellner1999] Norman Walsh and Leonard Muellner, 1999, 1-56592-580-7, O'Reilly & Associates, Inc., *DocBook: The Definitive Guide*.

Periodicals

[Walsh1997] *XML: Principles, Tools, and Techniques*, O'Reilly & Associates, Inc., 1085-2301, Edited by Dan Connolly, "A Guide to XML", Norman Walsh, 1997, 97-108.

[EllisonSchneier2000] *Ten Risks of PKI: What You're not Being Told about Public Key Infrastructure*, Computer Security Institute, "Ten risks of PKI: What You're not Being Told about Public Key Infrastructure", Carl Ellison, Bruce Schneier, 2000.

[Gollmann2000] *On the Verification of Cryptographic Protocols - A Tale of Two Committees*, Elsevier Science B. V., "Electronic Notes in Theoretical Computer Science", Dieter Gollmann, 2000.