



Cryptographic Applications

Dr. Mohamadou Sall

Department of Mathematics and Computer Science / F.S.T.
Université Cheikh Anta Diop de Dakar (UCAD)

msallt12@gmail.com

Summary

- 1 Introduction to SSL/TLS and OpenSSL
- 2 Passwords and Pseudorandom Number Generator
- 3 Symmetric Encryption
- 4 Asymmetric Encryption
- 5 Numeric Signature
- 6 Email Encryption : S/MIME vs PGP
 - Setting Up a Certification Authority
 - Issuing Certificates

SSL protocol

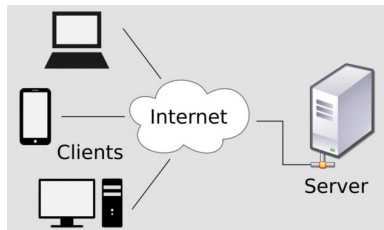
Definition

Secure Socket Layer (SSL) denotes the predominant communication security protocol of the Internet particularly for World Wide Web (WWW) services relating to electronic commerce or home banking.

SSL protocol

Definition

Secure Socket Layer (SSL) denotes the predominant communication security protocol of the Internet particularly for World Wide Web (WWW) services relating to electronic commerce or home banking.

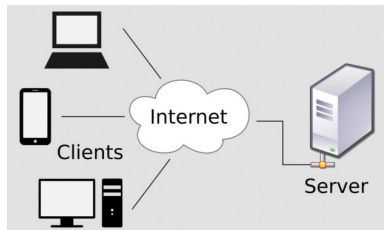


SSL was developed by Netscape Communications Corporation society for securing client/serveur applications.

SSL protocol

Definition

Secure Socket Layer (SSL) denotes the predominant communication security protocol of the Internet particularly for World Wide Web (WWW) services relating to electronic commerce or home banking.



SSL was developed by **Netscape Communications Corporation** society for securing client/serveur applications.

TSL protocol

Definition

The Transport Layer Security (TLS) standard defines the successor SSL.

TSL protocol

Definition

The Transport Layer Security (TLS) standard defines the successor SSL.

- The rough outline of the protocol is very similar to SSLv3.

TSL protocol

Definition

The Transport Layer Security (TLS) standard defines the successor SSL.

- The rough outline of the protocol is very similar to SSLv3.
- The technical differences between SSLv3 and TLS version 1.0 are very small ; most important changes relate to the construction of message authentication codes (HMAC) and the key expansion method.

TSL protocol

Definition

The Transport Layer Security (TLS) standard defines the successor SSL.

- The rough outline of the protocol is very similar to SSLv3.
- The technical differences between SSLv3 and TLS version 1.0 are very small; most important changes relate to the construction of message authentication codes (HMAC) and the key expansion method.
- Descriptions given in Secure Socket Layer (SSL) apply to TLS as well.

TSL protocol

Definition

The Transport Layer Security (TLS) standard defines the successor SSL.

- The rough outline of the protocol is very similar to SSLv3.
- The technical differences between SSLv3 and TLS version 1.0 are very small; most important changes relate to the construction of message authentication codes (HMAC) and the key expansion method.
- Descriptions given in Secure Socket Layer (SSL) apply to TLS as well.

Almost all cryptographically protected World Wide Web (WWW) communication relies on protocols of the TLS/SSL suite.

TSL protocol

Definition

The Transport Layer Security (TLS) standard defines the successor SSL.

- The rough outline of the protocol is very similar to SSLv3.
- The technical differences between SSLv3 and TLS version 1.0 are very small; most important changes relate to the construction of message authentication codes (HMAC) and the key expansion method.
- Descriptions given in Secure Socket Layer (SSL) apply to TLS as well.

Almost all cryptographically protected World Wide Web (WWW) communication relies on protocols of the TLS/SSL suite.

OpenSSL toolkit

Definition

OpenSSL is a cryptographic toolkit implementing SSL and TLS protocols.

OpenSSL toolkit

Definition

OpenSSL is a cryptographic toolkit implementing SSL and TLS protocols.

This toolkit offers :

OpenSSL toolkit

Definition

OpenSSL is a cryptographic toolkit implementing SSL and TLS protocols.

This toolkit offers :

- ◆ a C programming library for performing secure client/server applications based on SSL/TLS.

OpenSSL toolkit

Definition

OpenSSL is a cryptographic toolkit implementing SSL and TLS protocols.

This toolkit offers :

- ◆ a C programming library for performing secure client/server applications based on SSL/TLS.
- ◆ an online command (`openssl`) allowing :

OpenSSL toolkit

Definition

OpenSSL is a cryptographic toolkit implementing SSL and TLS protocols.

This toolkit offers :

- ◆ a C programming library for performing secure client/server applications based on SSL/TLS.
- ◆ an online command (openssl) allowing :
 - ▶ symmetric encryption and decryption (AES, DES, IDEA, RC2, RC4, ...)

OpenSSL toolkit

Definition

OpenSSL is a cryptographic toolkit implementing SSL and TLS protocols.

This toolkit offers :

- ◆ a C programming library for performing secure client/server applications based on SSL/TLS.
- ◆ an online command (openssl) allowing :
 - ▶ symmetric encryption and decryption (AES, DES, IDEA, RC2, RC4, ...)
 - ▶ asymmetric communication
 - RSA key generation and encryption
 - DSA signature

OpenSSL toolkit

Definition

OpenSSL is a cryptographic toolkit implementing SSL and TLS protocols.

This toolkit offers :

- ◆ a C programming library for performing secure client/server applications based on SSL/TLS.
- ◆ an online command (openssl) allowing :
 - ▶ symmetric encryption and decryption (AES, DES, IDEA, RC2, RC4, ...)
 - ▶ asymmetric communication
 - RSA key generation and encryption
 - DSA signature
 - ▶ certificates creation X509

OpenSSL toolkit

Definition

OpenSSL is a cryptographic toolkit implementing SSL and TLS protocols.

This toolkit offers :

- ◆ a C programming library for performing secure client/server applications based on SSL/TLS.
- ◆ an online command (openssl) allowing :
 - ▶ symmetric encryption and decryption (AES, DES, IDEA, RC2, RC4, ...)
 - ▶ asymmetric communication
 - RSA key generation and encryption
 - DSA signature
 - ▶ certificates creation X509
 - ▶ hash functions (MD5, SHA, RIPEMD160, ...)

OpenSSL toolkit

Definition

OpenSSL is a cryptographic toolkit implementing SSL and TLS protocols.

This toolkit offers :

- ◆ a C programming library for performing secure client/server applications based on SSL/TLS.
- ◆ an online command (openssl) allowing :
 - ▶ symmetric encryption and decryption (AES, DES, IDEA, RC2, RC4, ...)
 - ▶ asymmetric communication
 - RSA key generation and encryption
 - DSA signature
 - ▶ certificates creation X509
 - ▶ hash functions (MD5, SHA, RIPEMD160, ...)

OpenSSL Installation

The official OpenSSL website offers Linux sources

`https://www.openssl.org/`

OpenSSL Installation

The official OpenSSL website offers Linux sources

`https://www.openssl.org/`

Furthermore many Linux distributions come with pre-compiled OpenSSL packages. For the Windows operating system you can download OpenSSL from

OpenSSL Installation

The official OpenSSL website offers Linux sources

`https://www.openssl.org/`

Furthermore many Linux distributions come with pre-compiled OpenSSL packages. For the Windows operating system you can download OpenSSL from

`https://slproweb.com/products/Win32OpenSSL.html`

OpenSSL Installation

The official OpenSSL website offers Linux sources

`https://www.openssl.org/`

Furthermore many Linux distributions come with pre-compiled OpenSSL packages. For the Windows operating system you can download OpenSSL from

`https://slproweb.com/products/Win32openssl.html`

The latest stable version is the 1.1.1 series. Users are recommended to update to this version.

OpenSSL Installation

The official OpenSSL website offers Linux sources

`https://www.openssl.org/`

Furthermore many Linux distributions come with pre-compiled OpenSSL packages. For the Windows operating system you can download OpenSSL from

`https://slproweb.com/products/Win32openssl.html`

The latest stable version is the 1.1.1 series. Users are recommended to update to this version.

OpenSSL Command-Line Interface

OpenSSL is also a tool that provides access to much of its functionality from the command line.

Definition

The [command-line tool executable](#) is aptly named *openssl* on Unix, and *openssl.exe* on Windows. It has two modes of operation : interactive and batch.

- ➊ When operating in interactive mode, a prompt is displayed indicating that it is ready to process your command.
- ➋ Commands entered in interactive mode are handled in the same manner as if you'd entered them from the command line in batch mode; the only difference is that you don't need to type "openssl" before each command.

The first part of a command is its name. It's followed by any options that you wish to specify. Options often require a parameter of their own.

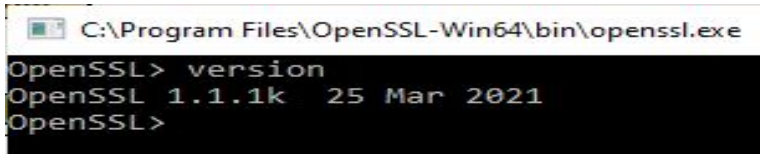
Getting started

The general syntax, in a batch mode, of the openssl command is :

\$ openssl <commande> <options>

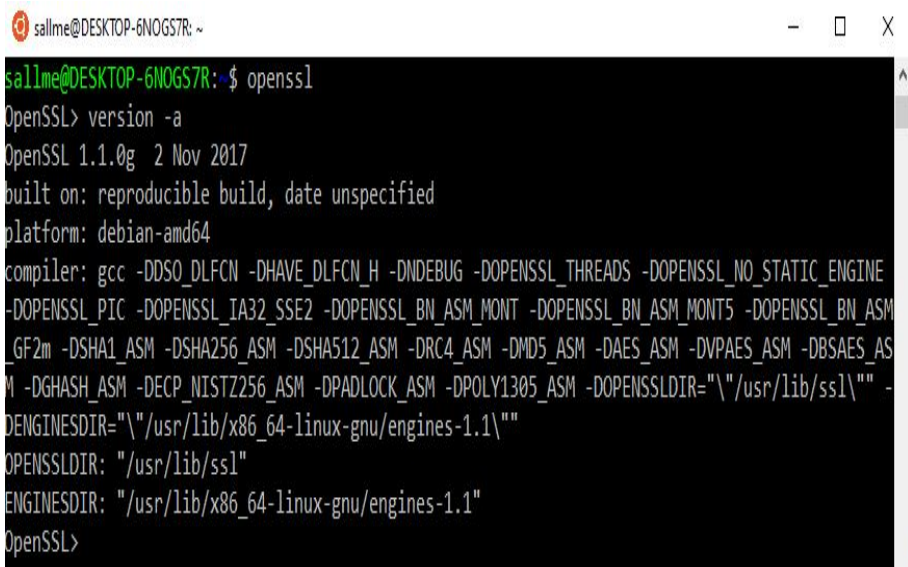
or (in an interactive mode)

><commande> <options>



```
C:\Program Files\OpenSSL-Win64\bin\openssl.exe
OpenSSL> version
OpenSSL 1.1.1k 25 Mar 2021
OpenSSL>
```

Example on Linux-Ubuntu



A terminal window titled 'sallme@DESKTOP-6NOGS7R: ~' with standard window controls. The terminal shows the command 'openssl' being executed, which then prompts 'OpenSSL>'. The user enters 'version -a', and the terminal displays the following output:

```
sallme@DESKTOP-6NOGS7R:~$ openssl
OpenSSL> version -a
OpenSSL 1.1.0g  2 Nov 2017
built on: reproducible build, date unspecified
platform: debian-amd64
compiler: gcc -DDSO_DLFCN -DHAVE_DLFCN_H -DNDEBUG -DOPENSSL_THREADS -DOPENSSL_NO_STATIC_ENGINE
-DOPENSSL_PIC -DOPENSSL_IA32_SSE2 -DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_MONT5 -DOPENSSL_BN_ASM
_GF2m -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM -DRC4_ASM -DMD5_ASM -DAES_ASM -DVPAES_ASM -DBSAES_AS
M -DGHASH_ASM -DECP_NISTZ256_ASM -DPADLOCK_ASM -DPOLY1305_ASM -DOPENSSLDIR="\"/usr/lib/ssl\"
DENGINESDIR="\"/usr/lib/x86_64-linux-gnu/engines-1.1\"
OPENSSLDIR: "/usr/lib/ssl"
ENGINESDIR: "/usr/lib/x86_64-linux-gnu/engines-1.1"
OpenSSL>
```

Configuration Files

The task of managing options is made considerably simpler using configuration files.

- The location of the default configuration file varies greatly.
- An OpenSSL configuration file is organized in sections.
- Each section contains a set of keys, and each key has a value.
- Keys are separated from their associated value by an equals sign

Passwords, Passphrases, and Pseudorandom Number Generator.

Passwords and Passphrases

Many commands require a password or passphrase usually to decrypt a key that is stored securely on a disk.

- `pass :<password>` This method can be used to supply the password or passphrase directly on the command line. We strongly recommend that you do not use this method.
- `env :<variable>` This method obtains the password or passphrase from an environment variable. This method is slightly more secure, but a process's environment is still available to other processes on some operating systems under the right circumstances.
- `file :<filename>` This method obtains the password or passphrase by reading it from the named file. This file should be protected.

Pseudorandom Number Generator

How to seed the OpenSSL PRNG properly from the command line.

- On Windows systems, a variety of sources will be used to seed the PRNG, including the contents of the screen. None of these sources is particularly entropic.
- Unix systems that have a device named */dev/urandom* will use that device to obtain entropy for seeding the PRNG.

In addition to the base entropy sources, the command-line tool will also look for a file to obtain seed data from.

Remark

In particular, any of the commands that generate key pairs always require unpredictable random numbers in order for them to be effective. When the tool is unable to seed the PRNG on its own, the tool provides an option named `rand` that can be used to provide additional entropy sources.

The `rand` option requires a parameter that contains a list of files to be used as entropy sources.

Definition

EGD is an entropy-gathering daemon written in Perl that is intended for use in the absence of `/dev/random` or `/dev/urandom`.

We recommend [EGADS \(Entropy Gathering And Distribution System\)](#), a C-based infrastructure that supports both Unix and Windows. This is a preferable solution even on Unix machines.

- It can be used anywhere an EGD socket is expected.
- EGADS provides an EGD interface and will use `/dev/random` to gather entropy.
- EGADS also contains a cryptographically secure PRNG.

Symmetric encryption using OpenSSL

OpenSSL supports a wide variety of symmetric ciphers.

- Many of the large number of ciphers are variations of a base cipher : Blowfish, CAST5, DES, 3DES, IDEA, RC2, RC4, RC5, and AES.
- They support a variety of different modes : CBC, CFB, ECB, and OFB. The default mode is CBC. You should generally never use ECB.

To see the list of all cipher commands use the command `> ciphers`.

List of ciphers

```
OpenSSL> ciphers
```

```
TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:
ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-
POLY1305:DHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES128-
GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES128-
SHA256:ECDHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-
AES256-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES128-SHA:DHE-RSA-AES128-SHA:RSA-PK-AES256-GCM-SHA384:DHE-
PSK-AES256-GCM-SHA384:RSA-PSK-CHACHA20-POLY1305:DHE-PSK-CHACHA20-POLY1305:ECDHE-PSK-CHACHA20-POLY1305:
AES256-GCM-SHA384:PSK-AES256-GCM-SHA384:PSK-CHACHA20-POLY1305:RSA-PSK-AES128-GCM-SHA256:DHE-PSK-AES128-
GCM-SHA256:PSK-AES128-GCM-SHA256:AES256-SHA256:AES128-SHA256:ECDHE-PSK-AES256-CBC-SHA384:ECDHE-PSK-AES256-
CBC-SHA:SRP-RSA-AES-256-CBC-SHA:SRP-AES-256-CBC-SHA:RSA-PSK-AES256-CBC-SHA384:DHE-PSK-AES256-CBC-SHA384:
RSA-PSK-AES256-CBC-SHA:DHE-PSK-AES256-CBC-SHA:AES256-SHA:PSK-AES256-CBC-SHA384:PSK-AES256-CBC-SHA:
ECDHE-PSK-AES128-CBC-SHA256:ECDHE-PSK-AES128-CBC-SHA:SRP-RSA-AES-128-CBC-SHA:SRP-AES-128-CBC-SHA:RSA-PSK-
AES128-CBC-SHA256:DHE-PSK-AES128-CBC-SHA256:RSA-PSK-AES128-CBC-SHA:DHE-PSK-AES128-CBC-SHA256:PSK-AES128-
CBC-SHA
```

With a pass word \Rightarrow **PBE** : **P**assword **B**ased **E**ncryption

The `enc` command is the main command for accessing symmetric ciphers.

With a pass word \Rightarrow **PBE** : **P**assword **B**ased **E**ncryption

The `enc` command is the main command for accessing symmetric ciphers.

Encryption syntax

```
enc <-algo> -in <input-file> -out <output-file>
```

The same command, with different options, is used to decrypt a message.

Decryption syntax

```
enc <-algo> -d -in <input-file> -out <output-file>
```

With a pass word \Rightarrow **PBE** : **P**assword **B**ased **E**ncryption

The `enc` command is the main command for accessing symmetric ciphers.

Encryption syntax

```
enc <-algo> -in <input-file> -out <output-file>
```

The same command, with different options, is used to decrypt a message.

Decryption syntax

```
enc <-algo> -d -in <input-file> -out <output-file>
```

Example of symmetric encryption and decryption

```
sallme@DESKTOP-6NOGS7R:~$ openssl enc -des3 -salt -in plaintext.txt -out ciphertext.bin  
enter des-ede3-cbc encryption password:  
Verifying - enter des-ede3-cbc encryption password:
```

Encrypts the contents of the file plaintext.txt using DES3 in CBC mode

```
$ openssl enc -des3 -d -in ciphertext.bin -out plaintext.doc -pass pass:passer
```

Decrypts the contents of the file ciphertext.bin.

```
OpenSSL> base64 -in ciphertext.bin -out base64.txt
```

Encodes the contents of "ciphertext.bin" in base64 and writes the result to "base64.txt".

With a key generator \Rightarrow **CSPRNG** : Cryptographically Secure Pseudo-Random Number Generator

Generation syntax

Generates a random number which is used as the encryption key.

```
rand -out <key-file> <key-size>
```

Encryption syntax

```
enc <-algo> -in <input-file> -out <output-file> -e -k <key-file>
```

Decryption syntax

```
enc <-algo> -in <input-file> -out <output-file> -d -k <key-file>
```

With a key generator \Rightarrow **CSPRNG** : Cryptographically Secure Pseudo-Random Number Generator

Generation syntax

Generates a random number which is used as the encryption key.

```
rand -out <key-file> <key-size>
```

Encryption syntax

```
enc <-algo> -in <input-file> -out <output-file> -e -k <key-file>
```

Decryption syntax

```
enc <-algo> -in <input-file> -out <output-file> -d -k <key-file>
```

Example with a PRNG

```
sallme@DESKTOP-6NOGS7R:~$ openssl rand -out keyfile.txt 2048
```

```
$ openssl enc -des3 -in plain_rand.txt -out cipher_rand2.txt -e -k keyfile.txt
```

```
$ openssl enc -des3 -in cipher_rand2.txt -out plain_rand.doc -d -k keyfile.txt
```

Asymmetric encryption using OpenSSL

In order for a server to employ the SSL protocol, it requires a private key and a certificate.

- The certificate contains the public key that matches the server's private key.
- These keys must be created as part of the process for configuring the server to use SSL.

Remark

SSL isn't the only protocol that makes use of public key cryptography. Most modern software that supports encrypted communications uses it, too. Some of the more popular examples include SSH, PGP (Pretty Good Privacy), and S/MIME.

Diffie-Hellman Key exchange

Diffie-Hellman is used for key agreement(exchange of information). The command-line tool provides a command for generating Diffie-Hellman parameters

```
sallme@DESKTOP-6NOGS7R:~$ openssl dhparam -out dhparam.pem -2 1024
Generating DH parameters, 1024 bit long safe prime, generator 2
This is going to take a long time
.....++*++*++*
```

Generates a new set of Diffie-Hellman parameters and writes the parameters in PEM format to the file *dhparam.pem*.

RSA Cryptosystem

RSA

RSA provides secrecy, authentication, and encryption all in one neat little package. RSA does not require parameters to be generated before keys can be generated.

- The `genrsa` command is used to generate a new RSA private key.
- The `rsa` command is used to manipulate and examine RSA keys. It is capable of producing an RSA public key from a private key.
- The `rsautl` command provides encryption and signatures.
- In general, we do not recommend that you use this command at all for encrypting data. You should use the `enc` command instead.

Creating the RSA key pair file

Generate a pair of keys

```
genrsa -out <key-file> <key size>
```

Visualization of the key file

```
rsa -in <key-file> -text -noout
```

Key file protection

```
rsa -in <key-file> <-algo> -out <key-file>
```

Exporting the public key

```
rsa -in <key-file> -pubout -out <key-file >
```

RSA Encryption/Decryption of data

Encryption

```
rsautl -encrypt -in <input-file> -inkey <key-file> -out < output-file >
```

Note : For encryption, if the keystore contains only the **public key** then add the **-pubin** option in the command line, before or after the key.

Decryption

```
rsautl -decrypt -in <input-file> -inkey <key-file> -out < output-file>
```

Application : RSA key generation

```
sallme@DESKTOP-6NOGS7R:~$ openssl genrsa -out rsaprivatekey.pem -passout pass:passer -des3 2048
Generating RSA private key, 2048 bit long modulus
.....+++
.....+++
e is 65537 (0x010001)
```

Generates and encrypts a RSA private key using 3DES and a password.

```
$ openssl rsa -in rsaprivatekey.pem -passin pass:passer -pubout -out rsapublickey.pem
```

Reads an RSA private key, decrypts it using the password, and writes the corresponding public key to the file *rsapublickey.pem*.

Application : RSA Encrypt-Decrypt-Sign

```
$ openssl rsautl -encrypt -pubin -inkey rsapublickey.pem -in plain.txt -out cipher.txt
```

Encrypts the contents of "plain.txt", and writes it to "cipher.txt".

```
sallme@DESKTOP-6N0GS7R:~$ openssl rsautl -decrypt -inkey rsaprivatekey.pem -in cipher.txt -out plain_rsa.txt  
Enter pass phrase for rsaprivatekey.pem:
```

Decrypts the contents of "cipher.txt", and writes it to "plain_rsa.txt".

```
$ openssl rsautl -sign -inkey rsaprivatekey.pem -in plain.txt -out signature.bin
```

Signs the contents of "plain.txt", and writes it to "signature.bin".

```
$ openssl rsautl -verify -pubin -inkey rsapublickey.pem -in signature.bin -out plain.txt
```

Verifies the signature, and writes out the original unsigned data to *plain.txt*.

Some important notes

Numeric Signature using OpenSSL

Digital Signature Algorithm

DSA is frequently used in combination with Diffie-Hellman. DSA also requires parameters from which keys are generated.

- The `dsaparam` command is used to generate and examine DSA parameters.
- The `gensa` command is used for generating private keys from a set of DSA parameters.
- The `dsa` command provides the means by which a public key can be generated from a private key.

```
sallme@DESKTOP-6NOGS7R:~$ openssl dsaparam -out dsaparam.pem 1024
Generating DSA parameters, 1024 bit long prime
This could take some time
```

```
sallme@DESKTOP-6NOGS7R:~$ openssl gendsa -out dsaprivatekey.pem -des3 dsaparam.pem
Generating DSA key, 1024 bits
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

Generates a DSA private key using the parameters from *dsaparam.pem*, encrypts the newly key with the 3DES cipher.

```
sallme@DESKTOP-6NOGS7R:~$ openssl dsa -in dsaprivatekey.pem -pubout -out dsapublickey.pem
read DSA key
Enter pass phrase for dsaprivatekey.pem:
writing DSA key
```

Computes the public key that corresponds to the private key.

Hash functions and Digital Signature

Creating a hash : hashing data

```
dgst <-algo> -out <hash> <file>
```

Digital Signature

```
rsautl -sign -in <hash> -inkey <key-file> -out <signature>
```

Signature verification

```
rsautl -verify -in <signature> -inkey <key-file> -out <hash>
```

Note : For verification, if the keystore contains only the **public key** then add the option **-pubin** in the command line, before or after the key.

Email Encryption using S/MIME

S/MIME vs PGP

Definition

S/MIME (Secure Multipurpose Internet Mail Exchange) is a competing standard to PGP (Pretty Good Privacy) for the secure exchange of email.

S/MIME uses a *public key infrastructure* to establish trust, whereas PGP does not. Because PGP does not rely on a public key infrastructure to establish trust, it is easy to set up and use.

- PGP works for small groups of people, but it does not scale well.
- S/MIME is capable of scaling to support large groups of people.

Public keys are exchanged in the form of X.509 certificates, which require a Certification Authority.

- The body of the message is encrypted using a symmetric cipher.
- The key for the symmetric cipher is encrypted using the recipient's public key.

Certificates

At the heart of PKI is something called a certificate. A certificate binds a public key with a distinguished name. A certificate contains

- information about the subject and the third party that issued it.
- safeguards intended to allow the authenticity of the certificate to be verified, and aid in the detection of forgery or tampering.
- the period for which it is valid. Once it has expired, a new certificate must be issued.

If the issuer is trusted, the certificates that it issues can also be trusted.

Creation

Certificates are also created with a serial number embedded in them. This number is often used to identify a certificate quickly.

Certification Authorities

A Certification Authority (CA) is an organization or company that issues certificates. There are two basic types of CAs.

- ① A private CA has the responsibility of issuing certificates only for members of its own organization.
- ② A public CA, such as *VeriSign* or *Thawte*, has the responsibility of issuing certificates for any member of the public.

A CA must be trusted, so its certificate containing its public key must be widely distributed.

Obtaining a Certificate

You first need to determine what purpose the certificate will serve.

- **Personal Certificates** S/MIME email relies on personal certificates. It is limited to email security only. ??? The first step in obtaining a personal certificate is to visit VeriSign's web site. ??? VeriSign will send an automated email to the address you provided with instructions on how to "pick up" the certificate.
- **Code-Signing Certificates** The purpose is to sign code that users download off the Internet. ??? You must be sure to get the proper certificate for the code that you wish to sign. Depending on VeriSign's workload, it may take several days for the certificate to be issued.
- **Web Site Certificates** for use in securing a web site. ??? Once your request has been submitted, VeriSign takes it under review. If everything is in order, a secure server certificate is issued and the certificate is emailed.

We'll find out how to get a personal certificate for S/MIME.

There are a number of free and commercial CA packages available. OpenSSL provides all of the functionality required to set up a minimal CA.

Environment for Your Certification Authority

- choose a location for all of our CA's files to reside.
- The private key should be stored at least on a machine that is never put on a network.
- Besides key generation, we will create three files that our CA infrastructure will need :
 - ① for keeping track of the last serial number,
 - ② for keeping track of the certificates that have been issued,
 - ③

OpenSSL Configuration File It is used to obtain information about how to issue certificates. We could skip creating this file and use the OpenSSL defaults system-wide configuration file *openssl.cnf*.

Building an OpenSSL Configuration File

We do not want to use the system-wide configuration file. There are two ways to tell OpenSSL where to find our configuration file :

- 1 using the environment variable `OPENSSL_CONF`

```
# OPENSSL_CONF=/opt/exampleca/openssl.cnf
```

- 2 specifying the filename with the `config` option on the command line.

Creating a Self-Signed Root Certificate

Our CA needs a certificate of its own with which to sign the certificates that it issues. This certificate will also be used to sign any CRLs (Certificate Revocation Lists).

- ➊ Add some more information to our configuration file.
- ➋ Make sure you've set the *OPENSSL_CONF* environment variable so that OpenSSL can find your configuration file!

The options required on the command line are minimal because we've specified most of the options that we want in the configuration file.

Creating a Self-Signed Root Certificate

```
sallme@DESKTOP-6NOGS7R:/opt/exampleca$ sudo openssl req -x509 -newkey rsa -out cacert.pem -outform PEM
Generating a 2048 bit RSA private key
.....+++
...+++
writing new private key to 'privkey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
```

To see information about the certificate use the following command

```
openssl x509 -in cacert.pem -text -noout
```

Certificates from a CA

First we need a certificate request. To create it, start with a clean shell without the `OPENSSL_CONF` environment variable set so that the default configuration file is used... We use the `req` command.

```
sallme@DESKTOP-6NOGS7R:~$ openssl req -newkey rsa:1024 -keyout testkey.pem -keyform PEM -out testreq.pem
Generating a 1024 bit RSA private key
.....+++++
.....+++++
writing new private key to 'testkey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
```

Creates two files : *testreq.pem* and *testkey.pem* containing respectively the certificate request and the private key that matches the public key embedded in the certificate request.

Issuing a certificate from a certificate request

To view the resulting certificate request use the following command

```
openssl req -in testreq.pem -text -noout
```

We can now use our CA to issue a certificate. Make sure that the OPENSSL_CONF variable is set to the CA's configuration file.

```
root@DESKTOP-6NOGS7R:/opt/exampleca# openssl ca -in testreq.pem
Using configuration from /opt/exampleca/openssl.cnf
Enter pass phrase for /opt/exampleca/private/cakey.pem:
```

The resulting certificate, *01.pem*, is written to the directory that we specified in our configuration file with the *new_certs_dir* key. To view it do

```
openssl x509 -in 01.pem -text -noout
```

Using the *out* option, the name of the file can be specified.

Creating a digital certificate

Certificate request

```
req -config openssl.cfg -new -key <key-file> -out <request-file>
```

Requested file

```
req -config openssl.cfg -in <request-file> -text -noout
```

Generation of the certificate

```
x509 -days <number-days> -CAserial <number-serial> -CA <cert-file-CA>  
-CAkey <key-file-CA> -in <request-file> -req -out <certificate>
```

Visualization of the certificate file

```
x509 -in <certificate> -text -noout
```

Creating a self-signed digital certificate

Creating a key pair

```
genrsa -out <key-file> <size>
```

Creating a CSR file

With the private key, create of a Certificate Signing Request (CSR) file.

```
req -new -key <key-file> -out <request-file>
```

Self-signing a certificate

```
x509 -req -days <number-days> -in <request-file>  
-signkey <key-file> -out <certificate>
```

Email

The `smime` command supports encryption, decryption, signing, and verifying. The following examples illustrate the use of the S/MIME commands

```
$ openssl smime -encrypt -in mail.txt -des3 -out mail.enc cert.pem
```

- Obtains a public key from the X.509 certificate in the file cert.pem.
- Encrypts the contents of the file mail.txt using that key and 3DES.
- The resulting encrypted message is written to the file mail.enc.

End

Thank you