OpenSSL

What is it?

OpenSSL is an application that is able to provide (by setting parameters on the command line) a set of cryptographic functions (hash functions, symmetric and asymmetric encryption, CA); at the same time OpenSSL can be used as a cryptographic library in our own applications. As the name suggests, originally the library offered primarily the SSL protocol support. This functionality has been offered up to now, but the library offers much wider portfolio of cryptographic functions nowadays.

Address, version, license

OpenSSL is downloadable in the form of source code at the URL http://www.openssl.org. The binary form is part of many operating systems or can be downloaded from other source (see Google). OpenSSL compiles under many operating systems including MS Windows and UNIX systems. The authors are Eric A. Young and Tim J. Hudson. The license is better (more free) than GPL, the OpenSSL license is close to the Apache license and enables the noncommercial and commercial use AND MODIFICATION. Currently the latest version is 0.9.8d, some time ago there used to be two the versions of the library – the „engine" version (which included support of cryptographic HW devices) and the non-engine version, today all the versions include the HW device support automatically.

Documentation

Documentation of the OpenSSL *program* is fairly good and is available in the form of man pages (directory openssl/doc/apps), documentation of the API of the library is worse, some functions are sufficiently documented (openssl/doc/crypto, for SSL API it's openssl/doc/ssl), but many functions remain undocumented and for a correct use the inspiration in the source codes of particular modules of the *program* (openssl/apps) or header files (openssl/include/openssl) is necessary.

Advantages/Disadvantages

The advantages of OpenSSL include good licensing terms (commercial use of modified code is permitted), availability of source codes, platform independence and wide functionality. On the other hand the disadvantages include poor documentation, not very well tested Windows platform support and poor code quality ("hacks") in some parts of the library.

Program

The functionality offered by the OpenSSL program (command line) is divided into following modules (further info on parameters „openssl *module* help"):

- asn1parse - ASN.1 parsing tool
- ca - sample minimal CA application
- crl2pkcs7 - Create a PKCS#7 structure from a CRL and certificates.
- crl - CRL utility
- dgst, md5, md4, md2, sha1, sha, mdc2, ripemd160 - message digests
- dhparam - DH parameter manipulation and generation
- dsa - DSA key processing

- dsaparam - DSA parameter manipulation and generation
- ec - EC key processing
- ecparam - EC parameter manipulation and generation
- enc - symmetric cipher routines
- gendsa - generate a DSA private key from a set of parameters
- genrsa - generate an RSA private key
- ocsp - Online Certificate Status Protocol utility
- passwd - compute password hashes
- pkcs7 - PKCS#7 utility
- pkcs8 - PKCS#8 format private key conversion tool
- pkcs12 - PKCS#12 file utility
- rand - generate pseudo-random bytes
- req - PKCS#10 certificate request and certificate generating utility.
- rsa - RSA key processing tool
- rsautl - RSA utility
- s_client - SSL/TLS client program
- s_server - SSL/TLS server program
- sess_id - SSL/TLS session handling utility
- smime - S/MIME utility
- speed - test library performance
- spkac - SPKAC printing and generating utility
- verify - Utility to verify certificates.
- x509 - Certificate display and signing utility
- x509v3_config - X509 V3 certificate extension configuration format

OpenSSL supports algorithms RC4, RC2, DES3, DES, CAST, BLOWFISH, AES, (to get the full list of modes use „openssl list-cipher-commands“), MD2, MD4, MD5, RMD160, SHA, SHA1, RSA, DSS, DH (see openssl ciphers -v).

Usage of the program

Let’s look at one of the typical uses of the OpenSSL program i.e. at the certificate requesting, signing and revoking (CRL generation).

1. Generate the RSA key of the CA
    - **`openssl genrsa -out ca.key 1024`**
2. Creation of the self-signed certificate of the CA
    - **`openssl req -new -x509 -days 365 -key ca.key -out ca.crt`**
3. Generate RSA key of a user (e.g. server)
    - **`openssl genrsa -out server.key 1024`**
4. Generate the certificate request
    - **`openssl req -new -key server.key -out server.csr`**
5. Certificate signature
    - **`openssl ca -cert ca.crt -in server.csr -keyfile ca.key -days 365 –out server.crt`**
    - this command requires a specific directory structure, therefore we better use sign.sh script from the mod-ssl package (directory pkg.contrib) and we comment the deletion of the ca.config file at the end of the script.
    - **`./sign.sh server.csr`**
6. Prepare the PKCS#12 file (certificate plus the private key)
    - **`openssl pkcs12 -export -out server.p12 -in server.crt -inkey server.key`**
7. Issue the CRL (it’s empty now)

    - **openssl ca -gencrl -config ca.config -keyfile ca.key -cert ca.crt -out ca1.crl**
8. Revoke the user's certificate
    - **openssl ca -config ca.config -revoke server.crt -keyfile ca.key –cert ca.crt**
9. Issue a new CRL
    - **openssl ca -gencrl -config ca.config -keyfile ca.key -cert ca.crt -out ca2.crl**
10. Let's examine the CRLs
    - **openssl crl -in ca1.crl -noout -text**
    - **openssl crl -in ca2.crl -noout -text**

As a next example we will focus on the symmetric encryption:

1. file encryption
    - **openssl enc -aes-256-cbc -salt -in file.txt -out file.enc**
2. file decryption
    - **openssl enc -d -aes-256-cbc -in file.enc**

Library

The OpenSSL library consists of two parts: libeay and ssleay. The library can be linked statically or dynamically. The library is written in C and although can be theoretically used from any other programming language by default only the C API is provided (i.e. header files for includes).

In Windows OS we add libraries libeay32.lib and ssleay32.lib to our project (Project/Properties; Linker/Input/Additional Dependencies); if dynamic linking is used two dll files (libeay32.dll a ssleay32.dll) must be accessible for the application. Under UNIX systems we link -lssl –lcrypto and include <openssl/crypto.h> and other files (depends on which parts of the library we need – the names of the header files are intuitive).

For I/O operations the OpenSSL provides a general interface (so called "bio"), which facilitates the use of I/O, makes the I/O independent on the storage type (memory buffer, file etc.) and by using filters (which can be chained) it is easily possible to cryptographically process data (e.g. symmetric encryption).

As an example I present processing of a memory buffer by signing it digitally and encrypting it symmetrically. The result is stored in a file, the file is subsequently read, the data is decrypted and the digital signature is verified.

```
// header file for OpenSSL library
#include <openssl/crypto.h>
#include <openssl/x509.h>
#include <openssl/pkcs12.h>
#include <openssl/pkcs7.h>

// can be undefined in older versions of OpenSSL (otherwise in pkcs7.h)
#define PKCS7_NOCRL             0x2000

// reads a certificate and a private key from PKCS#12 file
// (in this case the file cannot be protected by password)
void load_pkcs12(BIO *in, EVP_PKEY **pkey, X509 **cert, STACK_OF(X509)
**ca)
{
```

```
  PKCS12 *p12;

  p12 = d2i_PKCS12_bio(in, NULL);
  if (p12 == NULL) …
  if (!PKCS12_verify_mac(p12, "", 0) && !PKCS12_verify_mac(p12, NULD,0))…
  int ret = PKCS12_parse(p12, "", pkey, cert, ca);
  if(p12)PKCS12_free(p12);
  if(!ret)…
}


int main()
{
// initialize library
CRYPTO_malloc_init();
ERR_load_crypto_strings();
OpenSSL_add_all_algorithms();
ERR_load_OBJ_strings();


// read the private key and corresponding certificate
X509 *signer=NULL;
EVP_PKEY *key = NULL;

BIO *pkcs12file = BIO_new_file("klic.p12", "rb");
load_pkcs12(pkcs12file,&key,&signer, NULL);
BIO_free(pkcs12file);

// fill in the data
struct data
{
char name[100], surname[100];
unsigned char minutiae[512];
} record1;

strcpy(record1.name,"Magda");

// sign and encrypt
BIO *in_record=BIO_new_mem_buf((void *)&record1, sizeof(struct data));
BIO *out_signed = BIO_new_file("signed_and_encrypted_file.txt", "wb");
PKCS7 *p7 = PKCS7_sign(signer, key, NULL, in_record, PKCS7_BINARY);

BIO *encipher = BIO_new(BIO_f_cipher());
BIO *out_encrypted = BIO_push(encipher,out_signed);
BIO_set_cipher(out_encrypted,EVP_aes_128_cbc(),(unsigned
char*)"12345678ABCDEFGH",(unsigned char*)"00000000",1);

i2d_PKCS7_bio(out_encrypted, p7);
BIO_flush(out_encrypted);

BIO_free(in_record);
BIO_free_all(out_encrypted);

// decrypt and verify signature
X509_STORE *store = X509_STORE_new();
X509_LOOKUP *lookup=X509_STORE_add_lookup(store,X509_LOOKUP_hash_dir());
X509_LOOKUP_add_dir(lookup,"root",X509_FILETYPE_PEM);
STACK_OF(X509) *othercerts = sk_X509_new_null();
sk_X509_push(othercerts, signer);

BIO *p7_in = BIO_new_file("signed_and_encrypted_file.txt", "rb");
```

```
BIO *decipher = BIO_new(BIO_f_cipher());
BIO_set_cipher(decipher,EVP_aes_128_cbc(),(unsigned
char*)"12345678ABCDEFGH",(unsigned char*)"00000000",0);
p7_in = BIO_push(decipher,p7_in);

BIO *data_out = BIO_new(BIO_s_mem());

PKCS7 *read_p7 = d2i_PKCS7_bio(p7_in, NULL);

if(PKCS7_verify(read_p7,othercerts,store,NULL,data_out,
PKCS7_NOCRL|PKCS7_BINARY))
printf("Signature verified\n");
else
printf("Signature verification failed\n");

BIO_free_all(p7_in);

struct data *record2;
int length = BIO_get_mem_data(data_out, &record2);
if(length!=sizeof(struct data))
 printf("Size of data does not match\n");

printf("The data is '%s'\n",record2->name);

BIO_free(data_out);
return 0;
}
```

For digital signatures we need to have a PKCS#12 file with the private key and the certificate, the result is stored in the file “signed_and_encrypted_file.txt”. The symmetric encryption is done by a fixed encryption key and initialization vector. To verify the signature successfully we need a subdirectory “root” with a structure of trusted root certificates (in this case PEM format is required) with names matching the hash of certificate subjects (openssl x509 -hash) and extension .0 or higher numbers in the case of hash collision.

Assignments

1. How do you obtain the SHA-1 hash of a file by using the openssl program? [describe the command line parameters] {1}
2. Create a program (in C) that generates a 1024 bit RSA key. [Enclose the source code] {3} (use function RSA_generate_key)
3. Create a program (in C) that connects to a POP3s server and outputs the server headers (certificate does not have to be verified). [Enclose the source code] {3} (use SSL_new, SSL_connect, SSL_read)