OpenSSL

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 used from any other programming language only the C API is provided by default (i.e. header files for includes).

In Windows 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. We include <openssl/crypto.h> and other header 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. by symmetric encryption).

The example processes 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,"Alice");

// 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 name hash collision).

Assignments

1. Create a simple program that encrypts/decrypts a file with a fixed key using AES-128. [Enclose the source code] {4}
2. Create a program (using openssl library) that generates a 2048 bit RSA key. [Enclose the source code] {2} (use function RSA_generate_key)
3. Create a program (using openssl library) that connects to a POP3s server and outputs the server headers (certificate does not have to be verified). [Enclose the source code] {4} (use SSL_new, SSL_connect, SSL_read)