# PROTOCOALE DE COMUNICAȚIE : LABORATOR 12

## Protocoale de securitate in comunicatii

Responsabil: Alecsandru PATRASCU

# Cuprins



# Obiective

In urma parcurgerii acestui laborator studentul va fi capabil sa:

- Inţeleagă noţiunile de securitate folosite în dezvoltarea aplicaţiilor.
- Deprindă modul de folosire a pachetului OpenSSL.

# Criptarea simetrica

Criptarea simetrica implica folosirea unei singure chei atat pentru criptarea cat si pentru decriptarea datelor. Aceasta metoda este, de altfel, denumita si "cifrarea cu cheie secreta" (expeditorul datelor si destinatarul lor detin un secret – cheia) - Figura 1.

Pentru asigurarea confidentialitatii datelor s-a adoptat urmatoarea procedura: utilizatorul A cripteaza datele folosind cheia secreta (SECRET), apoi datele sunt trimise utilizatorului B. Acesta din urma decripteaza datele folosind aceeasi cheie.Deoarece si transmitatorul si receptorul folosesc aceeasi cheie, masurile ce se iau

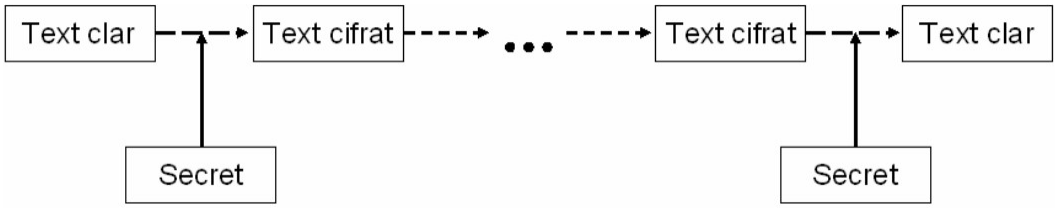


Figura 1: Mecanismul de criptare simetrica

urmaresc sa impiedice un al treilea intrus sa obtina cheia. Pentru a transfera acest tip de chei dintr-o parte intr-alta este necesar un mecanism complex de securitate a distributiei cheilor.

Cifrarea simetrica este adeseori folosita pentru transmiterea confidentiala a datelor prin retea.Exemple de algoritmi de criptare simetrica: AES, DES, IDEA, RC5, Blowfish, Twofish, etc.

## Criptare asimetrica

Principalul dezavantaj al criptarii simetrice este necesitatea unui mecanism de securitate pentru distribuirea cheile secrete. Acest dezavantaj dispare in mai noul sistem de criptare asimetrica, propus de catre Diffie si Hellman in 1976. Acesta presupune existenta a doua chei separate, una pentru criptare (E) si una pentru decriptare (D), care trebuie sa satisfaca trei cerinte:

1. D(E(P)) = P
2. Este mai mult decat dificil sa se deduca D din E
3. E nu poate fi spart printr-un atac cu text clar ales

O persoana care doreste sa primeasca sau sa trimita mesaje secrete criptate asimetric isi stabileste o pereche de chei E si D (de obicei generate de catre un progam special), dintre care una (E) este facuta publica si cealalta (D) este privata, fiind foarte important ca nimeni altcineva sa nu o cunoasca. Pentru a trimite un mesaj secret persoanei respective, se cripteaza mesajul cu cheia publica E, destinatarul fiind singurul capabil de a decripta mesajul cu ajutorul cheii sale private D.

Un algoritm de criptare asimetrica larg utilizat este RSA, dezvoltat la MIT si aparut in 1977 (http://www.di-mgt.com.au/rsa_alg.html).

## Rezumate de mesaje si semnaturi digitale

Rezumatul unui mesaj este un sir de biti de lungime fixa, generat cu ajutorul unei functii de dispersie neinversabile aplicata mesajului. Functia de dispersie (notata MD) trebuie sa aiba 4 proprietati:

1. Dandu-se P, este usor de calculat MD(P)
2. Dandu-se MD(P), este imposibil de calculat P
3. Dandu-se P, nu poate fi gasit P' astfel incat MD(P) = MD(P')
4. O schimbare mica in mesaj (chiar si de 1 bit) produce un rezumat foarte diferit

Rezumatele pot fi utilizate pentru a verifica rapid transmisia corecta a unui mesaj (rezumatul este transmis impreuna cu mesajul si destinatarul verifica daca rezumatul primit coincide cu rezumatul recalculat de catre el) si pentru realizarea semnaturilor digitale.

Semnaturile digitale ofera posibilitatea de a verifica autenticitatea unor docmente/mesaje transmise prin retea, asigurand urmatoarele lucruri: receptorul poate verifica daca transmitatorul are identitatea pe care pretinde ca o are; transmitatorul nu poate nega mai tarziu ca a trimis mesajul; nimeni altcineva in afara de

transmitator nu ar fi putut sa genereze mesajul. Pentru a semna digital un mesaj exista mai multe metode, printre care una bazata pe rezumate si pe criptare cu chei publice, in care se procedeaza astfel:

1. persoana care doreste sa trimita un mesaj semnat calculeaza rezumatul mesajului MD(P)
2. mesajul este trimis in clar, impreuna cu rezumatul criptat cu cheia privata a trimitatorului (D(MD(P)))
3. receptorul decripteaza D(MD(P)) cu ajutorul cheii publice a trimitatorului si verifica daca rezultatul coincide cu rezumatul calculat de el pentru mesaj persoana care doreste sa trimita un mesaj semnat calculeaza rezumatul mesajului

Exemple de algoritmi pentru calculul de rezumate: MD5, SHA-1, SHA-2, SHA-3.

## Gestiunea cheilor publice. Autoritati de certificare

Pentru ca doua entitati sa poata comunica sigur utilizand sistemul de criptare asimetrica, fiecare trebuie sa cunoasca cheia publica a celeilalte si sa nu existe riscul ca un intrus sa transmita uneia dintre entitati o alta cheie in loc de cheia publica a celeilalte. Una dintre solutiile utilizate la ora actuala pentru aceasta problema este certificarea cheilor de catre organizatii speciale numite autoritati de certificare (Certification Authority - CA). O persoana care doreste un certificat trebuie sa se adreseze unei CA, autentificandu-se si furnizand cheia sa publica; autoritatea de certificare poate decide sa acorde persoanei certificatul, care va contine identitatea si cheia publica a solicitatntului, si sa il semneze digital. Formatul utilizat de obicei pentru certificate este X.509 (http://www.ietf.org/rfc/rfc2459.txt).

Autoritatile de certificare sunt organizate ierarhic, existand o serie de CA-uri "radacina" care sunt bine cunoscute, alta serie de CA-uri certificate de CA-urile radacina s.a.m.d. In momentul cand este verificat certificatul unei entitati, se verifica si autoritatea de certificare CA1 care l-a emis, si care are si ea un certificat de la alta autoritate CA2; apoi se verifica CA2 si asa mai departe pana se ajunge la o CA in care se poate avea incredere sau la o CA radacina (astfel se formeaza un "lant de incredere" sau o "cale de certificare"). CA-urile radacina au certificate auto-semnate.

## Biblioteca OpenSSL

Protocolul SSL (Secure Socket Layer - http://wp.netscape.com/eng/ssl3/ssl-toc.html) a aparut ca urmare a necesitatii de a asigura conexiuni sigure in Internet, in stiva de protocoale fiind situat intre nivelul TCP si nivelul aplicatie. Protocolul asigura autentificarea mutuala a celor doua entitati care comunica, secretul comunicarii si protectia integritatii datelor. Una dintre cele mai des intalnite aplicatii ale SSL-ului este HTTPS (HTTP securizat), care consta din protocolul HTTP utilizat peste SSL.

OpenSSL (http://www.openssl.org) este o biblioteca ce implementeaza protocolul SSL si ofera in plus si alte facilitati (rezumate de mesaje, crearea de certificate si semnaturi digitale, generarea de numere aleatoare). Biblioteca este furnizata impreuna cu unele distributii de Linux sau poate fi obtinuta de pe site-ul oficial. Pe langa API-ul in C destinat programatorilor, OpenSSL contine si instrumente ce se pot utiliza din linia de comanda.

### Utilizarea in linie de comanda

Sintaxa pentru utilizarea in linia de comanda este urmatoarea:

```
1 openssl comanda [optiuni] [argumente]
```

Printre comenzi se numara:

- ca - utilizata pentru managementul unei autoritati de certificare (se pot genera certificate, care sunt stocate apoi intr-o baza de date)
- dgst - pentru calculul de rezumate de mesaje
- genrsa - pentru generarea de chei RSA
- req - pentru crearea si procesarea de cereri de certificate; se poate utiliza si pentru generarea de certificate auto-semnate
- verify - pentru verificarea de certificate X.509

Un ghid pentru programarea cu biblioteca OpenSSL gasiti aici: http://www.ibm.com/developerworks/linux/library/l-openssl/index.html

Pentru început, emiteţi următoarea comandă pentru a verifica versiunea de OpenSSL pe care o folosiţi:

```
openssl version
```

Generarea cereri de certificat se realizează astfel. Exemplul următor produce un fişier mycert.pem ce conţine atât cheia privată, cât şi cea publică. Certificatul va fi valid timp de 365 de zile iar cheia este neencriptată (opţiunea –nodes).

```
openssl req \
-new -newkey rsa:1024 -nodes \
-keyout mykey.pem -out myreq.pem
```

După apelul comenzii veţi fi puşi să răspundeţi unei serii de întrebări legate de: Ţară, Stat, Oraş, etc. Rezultatul va consta în crearea a două fişiere: mykey.pem va conţine cheia privată, iar myreq.pem va conţine o cerere de certificat. Cererea de certificat este trimisă (pe canale sigure) unei autorităţi de semnare (de exemplu VeriSign). Puteţi verifica conţinutul informaţiilor conţinute în cererea de certificat folosind:

```
# verificarea semnaturii
openssl req -in myreq.pem -noout -verify -key mykey.pem
# verificarea informatiilor
openssl req -in myreq.pem -noout -text
```

După cum aţi putut observa anterior, metoda de generare a cheii private folosită a fost RSA. Metoda folosita insa realiza si o cerere de certificat. Puteti obtine dacă doriţi generarea doar a cheii private RSA emitând o comandă precum cea din exemplul următor:

```
# o cheie implicita pe 512-biti, afisata la iesirea stdout
openssl genrsa
# o cheie pe 1024-biti, salvata in fisierul mykey.pem
openssl genrsa -out mykey.pem 1024
# ca in exemplul anterior, dar cheia este protejata de o parola
openssl genrsa -des3 -out mykey.pem 1024
```

Pornind de la cheia privată puteţi mai departe să generaţi şi o cheie publică corespunzătoare astfel:

```
openssl rsa -in mykey.pem -pubout
```

Generarea digest-urilor folosind opţiunea dgst reprezintă un exemplu de capabilitate oferită de OpenSSL.

```
# MD5 digest
openssl dgst -md5 filename

# SHA1 digest
openssl dgst -sha1 filename
```

Digesturile MD5 sunt similare celor create cu comanda md5sum, deşi formatele de ieşire diferă.

```
$ openssl dgst -md5 foo-2.23.tar.gz
MD5(foo-2.23.tar.gz)= 81eda7985e99d28acd6d286aa0e13e07

$ md5sum foo-2.23.tar.gz
81eda7985e99d28acd6d286aa0e13e07 foo-2.23.tar.gz
```

Digesturile pot fi chiar semnate pentru a vă asigura că ele nu pot fi modificate fără permisiunea explicită a proprietarului.

```
# digestul semnat va fi foo-1.23.tar.gz.sha1
openssl dgst -sha1 -sign mykey.pem -out foo-1.23.tar.gz.sha1 foo-1.23.tar.gz
```

Ulterior, digestul poate fi verificat. Pentru aceasta aveti nevoie de fişierul din care digestul a fost generat, de digest şi de cheia publică a semnatarului.

```
# pentru verificarea arhivei foo-1.23.tar.gz folosind foo-1.23.tar.gz.sha1 si cheia
    publica pubkey.pem
openssl dgst -sha1 -verify pubkey.pem -signature foo-1.23.tar.gz.sha1 foo-1.23.tar.gz
```

Un alt exemplu de folosire a OpenSSL este şi cel legat de criptarea/decriptarea unor documente. Astfel, pentru criptare puteţi emite o comandă precum:

```
# cripteaza file.txt la file.enc folosind 256-bit AES in modul CBC
openssl enc -aes-256-cbc -salt -in file.txt -out file.enc

# acelasi lucru, dar iesirea este de data aceasta codata base64. de exemplu pentru e-
    mail
openssl enc -aes-256-cbc -a -salt -in file.txt -out file.enc
```

În cadrul exemplului a fost folosit unul dintre algoritmii de criptografie suportaţi de OpenSSL. În practică însă puteţi alege oricare dintre cei suportaţi. Pentru a vedea care sunt algoritmii incluşi în distribuţia OpenSSL pe care o folosiţi puteţi emite:

```
openssl list-cipher-commands
```

Decriptarea fişierului rezultat anterior poate fi realizată astfel:

```
# decriptarea fisierului binar file.enc
openssl enc -d -aes-256-cbc -in file.enc

# decriptarea versiunii base64
openssl enc -d -aes-256-cbc -a -in file.enc
```

## Utilizarea programatica

SSL si conexiunile securizate pot fi folosite pentru orice tip de protocol pe Internet, HTTP, POP3 sau FTP. SSL poate fi folosit si pentru a securiza sesiuni de Telnet. Orice tip de conexiune poate fi securizat in teorie folosind SSL, dar ar trebui sa fie folosit numai pentru conexiuni care vor folosi date ce trebuie protejate.

OpenSSL ofera mai mult decit SSL fiind o biblioteca vasta de functii ce permite lucrul cu message digests, encriptari si decriptari de fisiere, certificate digitale, semnaturi digitale. Deasemenea OpenSSL este nu numai un API, ci si un utilitar ce poate fi folosit din linia de comanda care prezinta aceleasi functionalitati expuse de API permitind in plus si testarea serverelor si a clientilor SSL.

Pentru a instala si uneltele de dezvoltare (inclusiv librariile openssl).

```
sudo apt-get install libssl-dev
```

Urmatoarele fisere header *bio.h*, *ssl.h*, *err.h* vor fi folosite pe parcurs. Daca libssl-dev nu e instalat, importul fisierelor header de mai jos va da erori la compilare.

```
/* OpenSSL headers */
#include "openssl/bio.h"
#include "openssl/ssl.h"
#include "openssl/err.h"
```

Deasemenea sunt necesare urmatoarele linii de cod pentru a initializa libraria OpenSSL:

```
/* Initializare OpenSSL */
SSL_library_init ();
SSL_load_error_strings();
ERR_load_BIO_strings();
OpenSSL_add_all_algorithms();
```

OpenSSL foloseste o librarie numita BIO pentru a asigura comunicarea atit securizata cat si nesecurizata. Pentru a crea o conexiune fie ea securizata sau nu, un pointer pentru un obiect de tip BIO trebuie creat similar cu crearea unui pointer la FILE (in C). Crearea unei noi conexiuni se face cu *BIO_new_connect*, specificind hostname si portul. Daca a aparut vreo eroare la crearea obiectului BIO, functia va intoarce NULL. Totodata se va incerca sa se deschida conexiunea. Pentru a verifica daca acest lucru s-a realizat cu succes se apeleaza *BIO_do_connect* care returneaza 0 (OK) sau -1 (eroare).

```
BIO * bio;
bio = BIO_new_connect("hostname:port");

if(bio == NULL)
{
```

```
  /* trateaza eroarea */
}
if(BIO_do_connect(bio) <= 0)
{
  /* trateaza conexiune esuata*/
}
```

Citirea/scrierea de la/la un obiect BIO se face cu functiile *BIO_read()* si respectiv *BIO_write()*.

```
int BIO_read(BIO *b, void *buf, int len);
```

incearca sa citeasca len octeti din b si sa plaseze rezultatul in bufferul buf. Pentru o conexiune blocanta 0 inseamna ca conexiunea a fost inchisa in timp ce -1 indica ca a aparut o eroare. Pentru o conexiune neblocanta o valoare 0 inseamna ca nu s-a citit nimic iar -1 ca a aparut o eroare. In acest caz se poate reincerca operatia care a dat eroare, cu *BIO_should_retry*.

```
int x = BIO_read(bio, buf, len);
if(x == 0)
{
  /* trateaza conexiune inchisa*/
}
else if(x < 0)
{
  if(! BIO_should_retry(bio))
  {
    /* trateaza esec operatie read */
  }
  /* trateaza operatie reusita de retry */
}
```

```
int BIO_write(BIO *b, const void *buf, int len);
```

incearca sa scrie len octeti din buf la b.In ambele variante o valoare returnata de -2 inseamna ca functia nu e implementata pentru tipul specific BIO folosit.

```
if(BIO_write(bio, buf, len) <= 0)
{
  if(! BIO_should_retry(bio))
  {
    /* trateaza esec operatie write */
  }
  /* trateaza retry reusit */
}
```

Conexiunea poate fi inchisa in doua moduri, cu *BIO_reset()* astfel incat poate fi reutilizata ulterior sau cu *BIO_free_all()* caz in care se elibereaza memoria alocata si se inchide socketul asociat.

```
/* reutilizare conexiune*/
BIO_reset(bio);

```

```
/* eliberare memorie */
BIO_free_all(bio);
```

**Stabilirea unei conexiuni securizate**

Conexiunile securizate necesita un *handshake* dupa stabilirea conexiunii. In cursul handshake-ului, serverul trimite clientului un certificat, pe care apoi clientul il verifica folosind un set de certificate (certificate store) pe care le considera de incredere (trusted). Deasemenea verifica daca certificatul nu a expirat. Clientul trimite un certificat catre server numai daca acesta din urma cere unul (autentificarea clientului). Folosind transferul de certificate se face *setup-ul* conexiunii securizate. Clientul sau serverul pot solicita un nou *handshake* la orice moment. RFC 2246 detaliaza aspectele referitoare la protocolul de handshake.

Pentru a stabili o conexiune securizata mai sunt necesari doi pointeri unul de tip SSL_CTX (context object) si unul de tip SSL.

```
SSL_CTX * ctx = SSL_CTX_new(SSLv23_client_method());
SSL * ssl;
```

Urmatorul pas este includerea *certificate store* mentionat anterior. OpenSSL furnizeaza un set de certificate trusted. In arhiva cu codul sursa din distributia OpenSSL se afla un fisier "TrustStore.pem." care include toate certificatele trusted.

Functia *SSL_CTX_load_verify_locations* e folosita pentru a incarca acest fisier. Aceasta functie are trei parametri: pointerul la context calea catre fisierul *.pem si calea catre un director ce contine certificate. Unul din ultimii doi parametri trebuie specificat (ori fisierul *.pem sau, alternativ, directorul ce contine certificatele trusted). Returneaza 1 in caz de succes si 0 daca a aparut o problema.

```
if(! SSL_CTX_load_verify_locations(ctx, "TrustStore.pem", NULL))
{
  /* trateaza esec incarcare trust store */
}
```

Crearea unei noi conexiuni securizate se face cu

```
bio = BIO_new_ssl_connect(ctx);
BIO_get_ssl(bio, & ssl);
SSL_set_mode(ssl, SSL_MODE_AUTO_RETRY); // cu aceasta optiune daca serverul
                                        // doreste sa initieze din nou un handshake la
    un moment oarecare de timp,
                                       // OpenSSL se va ocupa in background de acest
   task
```

Dupa ce acest set-up a fost realizat se poate trece la deschiderea propriu-zisa a conexiunii

```
/* incearca sa se conecteze */
BIO_set_conn_hostname(bio, "hostname:port");

/* verifica conexiunea deschisa si efectueaza handshake */
if(BIO_do_connect(bio) <= 0)
{
```

```
  /* trateaza esec conexiune */
}
```

Pentru a verifica daca validarea certificatului s-a facut cu succes se apeleaza *SSL_get_verify_result()* cu unic parametru structura SSL. Daca validarea s-a efectuat cu succes returneaza X509_V_OK, altfel intoarce un cod de eroare care poate fi documentat daca se foloseste optiunea verify in linia de comanda a utilitarului.

```
if(SSL_get_verify_result(ssl) != X509_V_OK)
{
  /* trateaza esec verificare */
}
```

Stiva de erori poate fi scrisa intr-un fisier cu *ERR_print_errors_fp(FILE *);*

Erorile au urmatorul format:

```
[pid]:error:[error code]:[library name]:[function name]:[reason string]:[file name]:[
    line]:[optional text message]
```

## Aplicatie

Completati scheletul de cod astfel incat sa se efectueze o conexiune prin HTTPS la verisign.com si sa se citeasca local, intr-un fisier pe disc, pagina home, printr-un apel de tip GET.

Portul HTTPS implicit este 443.

Pentru compilare se poate folosi:

```
gcc sablon.c -lssl -lcrypto
```

## Referințe

1. http://www.openssl.org/
2. http://www.linuxjournal.com/article/4822
3. http://www.rfc-editor.org/rfc/rfc2246.txt