# Fonctions communes ou presque

#include <stdio.h>,<stdlib.h>,<string.h>,<netinet/in.h>

,<netdb.h><sys/socket.h>,<openssl/ssl.h>,<openssl/err.h>

// Compilation gcc -lssl -lcrypto

SSL\_CTX\* InitCTX(void)

{ const SSL\_METHOD \*method;

 SSL\_CTX \*ctx;

 OpenSSL\_add\_all\_algorithms(); /\* load & register all cryptos. \*/

 SSL\_load\_error\_strings(); /\* load all error messages \*/

 method = SSLv3\_**{server ou client}**\_method();
 /\* new server/client -method instance \*/

 ctx = SSL\_CTX\_new(method); /\* create new context from method \*/

 if ( ctx == NULL )

 { ERR\_print\_errors\_fp(stderr);

 abort(); }

 return ctx;

}

void ShowCerts(SSL\* ssl)

{ X509 \*cert;

 char \*line;

 cert = SSL\_get\_peer\_certificate(ssl); /\* Get certificates (if available) \*/

 if ( cert != NULL )

 {

 printf("Server certificates:\n");

 line = X509\_NAME\_oneline(X509\_get\_subject\_name(cert), 0, 0);

 printf("Subject: %s\n", line);

 free(line);

 line = X509\_NAME\_oneline(X509\_get\_issuer\_name(cert), 0, 0);

 printf("Issuer: %s\n", line);

 free(line);

 X509\_free(cert);

 }

 else

 printf("Aucun certificat.\n");

}

# Fichier serveur\_ssl.c pour ipv4

int main(int argc, char \*argv[])

{ SSL\_CTX \*ctx;

 SSL \*ssl;

 char service[20],host[100];

 int s\_ecoute,s\_com, debut=1;

 int portnum = 2000;

 struct sockaddr\_in appelant;

 socklen\_t len = sizeof(appelant);

 SSL\_library\_init(); /\* initialisation librairie ssl \*/

 ctx= InitCTX(); /\* initialisation context SSL -> fct \*/

 LoadCertificates(ctx, "server.crt", "server.key");

 /\* load certs \*/

 s\_ecoute = OpenListener(portnum);

 while(debut)

 {

 s\_com=accept(s\_ecoute, (struct sockaddr\*)&appelant,&len);

 ssl = SSL\_new(ctx); /\* get new ssl state with context \*/

 SSL\_set\_fd(ssl, s\_com); /\* set connection socket to SSL state \*/

 Servlet(ssl); /\* coeur du progamme \*/

 }

 close(s\_ecoute);

 SSL\_CTX\_free(ctx);

}

void LoadCertificates(SSL\_CTX\* ctx, char\* CertFile, char\* KeyFile)

{

 /\* set the local certificate from CertFile \*/

if ( SSL\_CTX\_use\_certificate\_file(ctx, CertFile,SSL\_ FILETYPE\_PEM)<=0 )

 { ERR\_print\_errors\_fp(stderr);

 abort(); }

 /\* set the private key from KeyFile (may be the same as CertFile) \*/

if ( SSL\_CTX\_use\_PrivateKey\_file(ctx,KeyFile, SSL\_FILETYPE\_PEM)<=0 )

 { ERR\_print\_errors\_fp(stderr);

 abort(); }

 /\* verify private key \*/

 if ( !SSL\_CTX\_check\_private\_key(ctx) )

 { fprintf(stderr, "Problem key and public certificate\n");

 abort();}

}

int OpenListener(int port)

{ int sd;

 struct sockaddr\_in addr;

 sd = socket(AF\_INET, SOCK\_STREAM, 0);

 bzero(&addr, sizeof(addr));

 addr.sin\_family = AF\_INET;

 addr.sin\_port = htons(port);

 addr.sin\_addr.s\_addr = INADDR\_ANY;

 if ( bind(sd, (struct sockaddr\*)&addr, sizeof(addr)) != 0 )

 {

 perror("Probleme sur le bind\n");

 abort();

 }

 listen(sd, 10);

 return sd;

}

void Servlet(SSL\* ssl)

{ char buf[1024];

 char reply[1024];

 int sd, bytes;

 char \*retour="Merci, mais ici tout va bien";

 if ( SSL\_accept(ssl) == -1 ) /\* do SSL-protocol accept \*/

 ERR\_print\_errors\_fp(stderr);

 else

 {

 ShowCerts(ssl); /\* get any certificates du client\*/

 bytes = SSL\_read(ssl, buf, sizeof(buf)); /\* lecture \*/

 printf("lecture de %d bytes\n",bytes);

 if ( bytes > 0 )

 {

 buf[bytes] = 0;

 printf("Client msg: \"%s\"\n", buf);

 sprintf(reply, retour, buf);

 SSL\_write(ssl, reply, strlen(reply)); /\* envoie \*/

 }

 else

 ERR\_print\_errors\_fp(stderr);

 }

 sd = SSL\_get\_fd(ssl); /\* get socket connection \*/

 SSL\_free(ssl); /\* release SSL state \*/

 close(sd); /\* close connection \*/

}

**Fichier client\_ssl.c pour ipv4**

int main(int argc, char \*argv[])

{

 SSL \*ssl;

 SSL\_CTX \*ctx;

 int s\_com, bytes;

 char buf[1024];

 if (argc !=3) {

 printf("usage : client nom\_serveur port\n");

 exit(1); }

 if (SSL\_library\_init() < 0)

 printf("probleme initialisation Openssl library\n") ;

 ctx= InitCTX();

 s\_com=OpenConnection(argv[1], atoi(argv[2]));

 ssl = SSL\_new(ctx); // create new SSL connection state

 SSL\_set\_fd(ssl, s\_com ); // attach the socket descriptor

 if ( SSL\_connect(ssl) == -1 ) //perform the connection

 ERR\_print\_errors\_fp(stderr);

 else

 { char \*msg = "Hello";

 printf("Connected with %s encryption\n", SSL\_get\_cipher(ssl));

 ShowCerts(ssl); /\* get any certs \*/

 SSL\_write(ssl, msg, strlen(msg)); /\* encrypt & send message \*/

 bytes = SSL\_read(ssl, buf, sizeof(buf)); /\* reponse \*/

 buf[bytes] = 0;

 printf("Received: \"%s\"\n", buf);

 SSL\_free(ssl); /\* release connection state \*/

 }

 close(s\_com ); /\* close socket \*/

 SSL\_CTX\_free(ctx); /\* release context \*/

 return 0;}

int OpenConnection(const char \*hostname, int port)

{ int sd;

 struct hostent \*host;

 struct sockaddr\_in addr;

 if ( (host = gethostbyname(hostname)) == NULL )

 {

 perror(hostname);

 abort();

 }

 sd = socket(AF\_INET, SOCK\_STREAM, 0);

 bzero(&addr, sizeof(addr));

 addr.sin\_family = AF\_INET;

 addr.sin\_port = htons(port);

 addr.sin\_addr.s\_addr = \*(long\*)(host->h\_addr);

 if ( connect(sd, (struct sockaddr\*)&addr, sizeof(addr)) != 0 )

 {

 close(sd);

 perror(hostname);

 abort();

 }

 return sd;

}