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;
}
```