OpenSSL BASICS

Corso di Laurea Magistrale in Ingegneria Informatica

A.A. 2015/2016

Prof. Simon Pietro Romano

spromano@unina.it

THE OPENSSL PROJECT

- A collaborative effort to develop
  - a toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols
  - a full-strength general purpose cryptography library
- Based on the SSLeay (“Eric Andrew Young”) library, developed until 1998
- “Dual licensed” under the OpenSSL License and the SSLeay license:
  - http://www.gnu.org/licenses/license-list.html
- Major version releases:
  - OpenSSL 1.0.0 was released on March 29, 2010
  - OpenSSL 0.9.8 was released on July 5, 2005
  - …

OPENSSL FEATURES

- Ciphers:
  - AES, Blowfish, CAST-128, DES, IDEA, RC2, RC4, RC5, Triple DES
- Cryptographic hash functions:
  - MD5, MD2, SHA-1, SHA-2, MDC-2
- Public-key cryptography:
  - RSA, DSA, Diffie-Hellman key exchange, Elliptic curve

SSL & CERTIFICATES

- SSL protocols allow to authenticate communication ends, by means of certificates
- Every certificate must be verified by a Certification Authority (CA)
- Usually, a server offers his certificate, optionally (if required by the server) the client does the same

CREATING A CA CERTIFICATE: *"CA.pl"* COMMAND

- CA certificate:
  - "CA.pl –newca"
  - certificate is stored in the "cacert.pem" file and the RSA private key in the "cakey.pem" file
- PEM is just a way of encoding data
  - X.509 certificates are one type of data that is commonly encoded using PEM
- PEM are Base64 encoded ASCII files and contain

  "-----BEGIN CERTIFICATE-----"

  "-----END CERTIFICATE-----"

  statements
- Server certificates, intermediate certificates, and private keys can all be put into the PEM format

SAMPLE “cacert.pem” FILE

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            94:be:15:75:67:d4:9f:c8
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=IT, ST=Napoli, O=unina, OU=comics, CN=spromano/emailAddress=spromano@unina.it
        Validity
            Not Before: Nov 12 13:55:53 2015 GMT
            Not After : Nov 11 13:55:53 2018 GMT
        Subject: C=IT, ST=Napoli, O=unina, OU=comics, CN=spromano/emailAddress=spromano@unina.it
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:e2:2f:fc:d4:dd:f6:65:56:33:b9:32:1f:82:06:
                    c2:92:e2:77:e0:fa:c7:23:78:cf:a7:f5:4d:86:08:
                    61:c4:a2:16:68:9f:74:2b:01:e7:34:0a:11:40:4f:
                    a8:b9:fb:88:3f:30:9d:3a:40:b7:8b:52:36:66:a5:
                    8f:ad:16:34:04:7e:75:a8:b6:f6:99:44:b0:a7:d9:
                    5a:44:04:ca:4b:d5:1f:02:52:b4:c1:31:5c:8b:9a:
                    ee:90:fc:d4:78:39:dd:ec:71:4a:af:14:4a:f7:5a:
                    4d:8f:d8:cd:54:5f:6c:39:84:ef:8a:ac:93:b0:5d:
                    4e:6d:80:f9:4f:e1:a6:f9:f7
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                ED:BB:94:B6:81:41:23:1E:1E:6E:E9:18:7B:A6:3C:BF:D1:2B:2B:74
            X509v3 Authority Key Identifier:
                keyid:ED:BB:94:B6:81:41:23:1E:1E:6E:E9:18:7B:A6:3C:BF:D1:2B:2B:74
                DirName:/C=IT/ST=Napoli/O=unina/OU=comics/CN=spromano/emailAddress=spromano@unina.it
                serial:94:BE:15:75:67:D4:9F:C8
            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: sha1WithRSAEncryption
        64:15:71:3a:a8:26:1c:72:43:6f:37:37:03:1a:f3:8f:52:c1:
        b6:30:bf:00:81:b2:eb:79:6d:0d:51:1c:bf:93:35:12:af:a0:
        f1:77:49:8e:8e:c4:08:70:32:06:3c:39:80:0e:e8:e8:f9:4e:
        f5:12:5a:fc:82:d4:68:ee:4f:f1:48:22:01:64:36:cd:44:20:
        b9:c5:65:d5:43:0b:db:ee:b5:cd:fd:38:03:be:63:71:7b:61:
        43:2d:58:62:d9:f5:d2:ad:c8:6f:24:11:97:6d:43:b6:03:27:
        5f:b8:23:e3:e6:b6:e9:87:d5:80:88:4c:69:97:39:bb:f5:ca:
        60:cc
-----BEGIN CERTIFICATE-----
MIIDQDCCAqmgAwIBAgIJAJS+FXVn1J/IMA0GCSqGSIb3DQEBBQUAMHQxCzAJBgNV
BAYTAklUMQ8wDQYDVQQIEwZOYXBvbGkxDjAMBgNVBAoTBXVuaW5hMQ8wDQYDVQQL
EwZjb21pY3MxETAPBgNVBAMTCHNwcm9tYW5vMSAwHgYJKoZIhvcNAQkBFhFzcHJv
bWFub0B1bmluYS5pdDAeFw0xNTExMTIxMzU1NTNaFw0xODExMTExMzU1NTNaMHQx
CzAJBgNVBAYTAklUMQ8wDQYDVQQIEwZOYXBvbGkxDjAMBgNVBAoTBXVuaW5hMQ8w
DQYDVQQLEwZjb21pY3MxETAPBgNVBAMTCHNwcm9tYW5vMSAwHgYJKoZIhvcNAQkB
FhFzcHJvbWFub0B1bmluYS5pdDCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA
4i/81N32ZVYzuTIfggbCkuJ34PrHI3jPp/VNhghhxKIWaJ90KwHnNAoRQE+oufuI
PzCdOkC3i1I2ZqWPrRY0BH51qLb2mUSwp9laRATKS9UfAlK0wTFci5rukPzUeDnd
7HFKrxRK91pNj9jNVF9s0YTviqyTsF1ObYD5T+Gm+fcCAwEAAaOB2TCB1jAdBgNV
HQ4EFgQU7buUtoFBIx4ebukYe6Y8v9ErK3QwgaYGA1UdIwSBnjCBm4AU7buUtoFB
Ix4ebukYe6Y8v9ErK3SheKR2MHQxCzAJBgNVBAYTAklUMQ8wDQYDVQQIEwZOYXBv
bGkxDjAMBgNVBAoTBXVuaW5hMQ8wDQYDVQQLEwZjb21pY3MxETAPBgNVBAMTCHNw
cm9tYW5vMSAwHgYJKoZIhvcNAQkBFhFzcHJvbWFub0B1bmluYS5pdIIJAJS+FXVn
1J/IMAwGA1UdEwQFMAMBAf8wDQYJKoZIhvcNAQEFBQADgYEAZBVxOqgmHHJDbzc3
Axrzj1LBtjC/AIGy63ltDVEcv5M1Eq+g8XdJjo7ECHAyBjw5gA7o6Pl09RJa/ILU
aO5P8UgiAWQ2zUQgucVl1UML2+61zf04A75jcXthQy1YYtn10q3IbyQRl21DtgMn
X7gj4+a26YfVgIhMaZc5u/XKYMw=
-----END CERTIFICATE-----
```

USER CERTIFICATES

- "CA.pl –newreq"
  - generates:
    - "newreq.pem"
      - request certificate
    - "newkey.pem"
      - the private key (generated from a password)
- User certificate must be signed by the CA:
  - "CA.pl –sign"
    - the CA password is required to sign it
    - a "newcert.pem" file containing the certificate and the public key will be created
    - Note well: you SHOULD modify "newkey.pem" files rights (read-only for the owner: "r-- --- ---", i.e., '400')
      - it contains the private key in clear

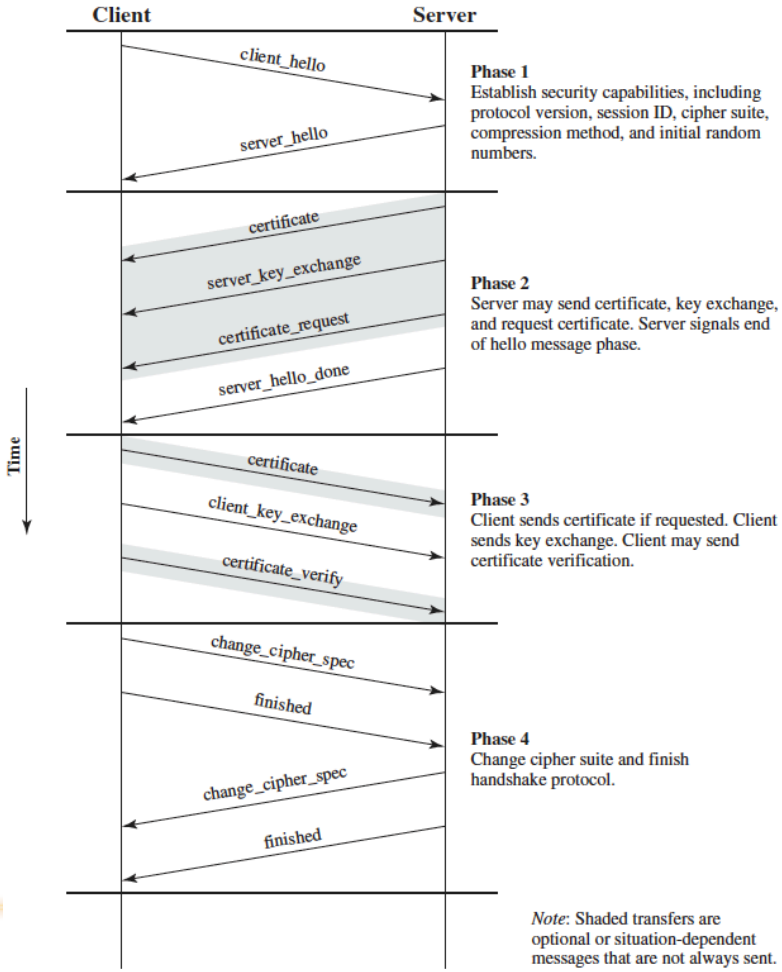
CREATING A CERTIFICATE: "*openssl*" COMMAND

"openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout mycert.pem -out mycert.pem"

```
MacBookPro-spromano:SSL spromano$ more mycert.pem
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQCxqm60+MkC1jDhaDFkxJZjxReKtgei8u5lrUt1bFU9GV9JwlA7
IKegdLmOmYCOa1EN5gOn3geP6oajcaNWZpeIrygHOuQMdwmzZsqBTqZV+FSj8Qt8
n2EkOPkOYJ+cOEGmflUeCMjCBvTXlyh2/4H151sK7ARgRxYue61aFD+VWQIDAQAB
AoGAJ41nxAY2ydjhXO0TQfm5+VAhY5IOtSocBKrFx8hsWYPSgNsvfVpRtxkFTkfC
JPKjQtLYMYDucg2mF7Duygdg6H/1A5xsB/TFrCwHASVaOnh2ZRlhye806VvKFdLn
+aNWDHRJvkKMYHrilFnV9kJYK8mM6LI0xGr/gD4LglRiz+ECQQDfwXr6Wt5NZHg3
Gojl7SFyMO/IHWaZxh6fHFDq2A2SGTmGhTcjLpjcECmqYJOBT2mnMp8LkdO2NQRQ
7IXhOED1AkEAy0Sn8zK9pFXe0/hPgKHoJohAZL9I2n9A9xzlYMM7NgrznLVcxH+u
2RLx21WXMkjmRJE9CzMXQXdxuogc4Ld0VQJAaWAlGmJQ7wGx28GTVzUl2zFQu12d
RC4ZnbUoJnE5cuSOQtvZ66e2NW0nMiXpUaykhSlB3aD/o3+0WKgLJNXpQQJBAJ7J
DBWrzn7gYYfeUmSelZX8G6lXz9Z3T8158ikreUaXOYRyHHyXbYO/C9WVv92VGuP3
PONRztniL6JSD5TvWl0CQQCtfijDCRT8ahkaAXa9QWuynUpHsb1V65A3bRhw/c/0
4mWMt5AyVx4lQwq51KErAGPxRJwCHa25ORhZVmYFWoKY
-----END RSA PRIVATE KEY-----
```

```
-----BEGIN CERTIFICATE-----
MIIDhDCCAu2gAwIBAgIJAIiq3tt7hV2cMA0GCSqGSIb3DQEBBQUAMIGJMQswCQYD
VQQGEwJJVDETMBEGA1UECBMKU29tZS1TdGF0ZTEPMA0GA1UEBxMGTmFwb2xpMQ4w
DAYDVQQKEwV1bmluYTEPMA0GA1UECxMGY29taWNzMREwDwYDVQQDEwhzcHJvbWFu
bzEgMB4GCSqGSIb3DQEJARYRc3Byb21hbm9AdW5pbmEuaXQwHhcNMTUxMTAxMDIz
ODA1WhcNMTYxMDMxMDIzODA1WjCBiTELMAkGA1UEBhMCSVQxEzARBgNVBAgTClNv
bWUtU3RhdGUxDzANBgNVBAcTBk5hcG9saTEOMAwGA1UEChMFdW5pbmExDzANBgNV
BAsTBmNvbWljczERMA8GA1UEAxMIc3Byb21hbm8xIDAeBgkqhkiG9w0BCQEWEXNw
cm9tYW5vQHVuaW5hLml0MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCxqm60
+MkC1jDhaDFkxJZjxReKtgei8u5lrUt1bFU9GV9JwlA7IKegdLmOmYCOa1EN5gOn
3geP6oajcaNWZpeIrygHOuQMdwmzZsqBTqZV+FSj8Qt8n2EkOPkOYJ+cOEGmflUe
CMjCBvTXlyh2/4H151sK7ARgRxYue61aFD+VWQIDAQABo4HxMIHuMB0GA1UdDgQW
BBS3vPA+Z9AmhJbIdTTfkZQPLLzlZjCBvgYDVR0jBIG2MIGzgBS3vPA+Z9AmhJbI
dTTfkZQPLLzlZqGBj6SBjDCBiTELMAkGA1UEBhMCSVQxEzARBgNVBAgTClNvbWUt
U3RhdGUxDzANBgNVBAcTBk5hcG9saTEOMAwGA1UEChMFdW5pbmExDzANBgNVBAsT
BmNvbWljczERMA8GA1UEAxMIc3Byb21hbm8xIDAeBgkqhkiG9w0BCQEWEXNwcm9t
YW5vQHVuaW5hLml0ggkAiKre23uFXZwwDAYDVR0TBAUwAwEB/zANBgkqhkiG9w0B
AQUFAAOBgQASdjr3b1cdYEhF2JUWelMZFNEVZwNFxsuA6hwZP12LBJLrV4rVBDt5
eAfeXEya2S2TM1Y6V5ZrD02n4rwEjIJY8IqUHTibH0U1id3nS8sa+Gr9dpOHRUSy
Bcm2H69aIH/74Y7qEktc3iz8W/fXDy7TGJ1KkUB+VrPgM8uvnMeVcA==
-----END CERTIFICATE-----
```

SSL HANDSHAKE…



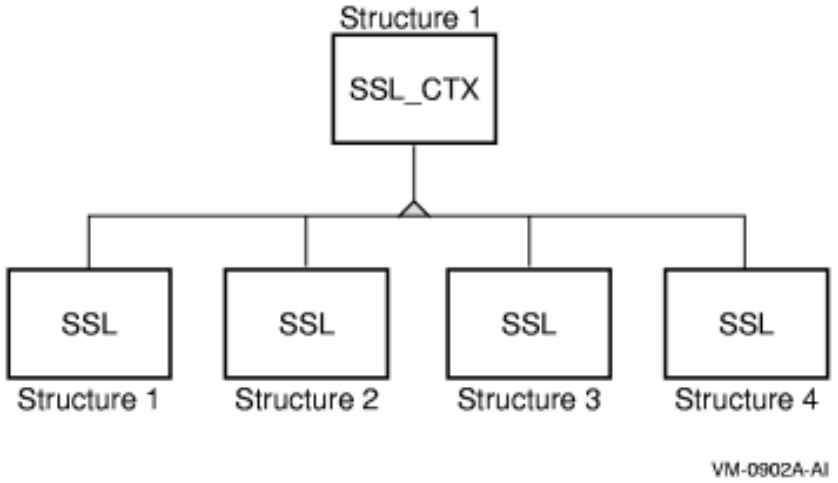
OPENSSL PREPARATORY CALLS

- The “SSL_library_init()” function registers the available ciphers and message digests
  - It must be called before any other action takes place
- “ERR_load_crypto_strings()” registers the error strings for all libcrypto* functions
- “SSL_load_error_strings()” does the same, but also registers libssl error strings
- One of these functions should be called before generating textual error messages

*libcrypto consists of a number of sub-libraries that implement the individual algorithms

OPENSSL STRUCTURES: SSL_CTX

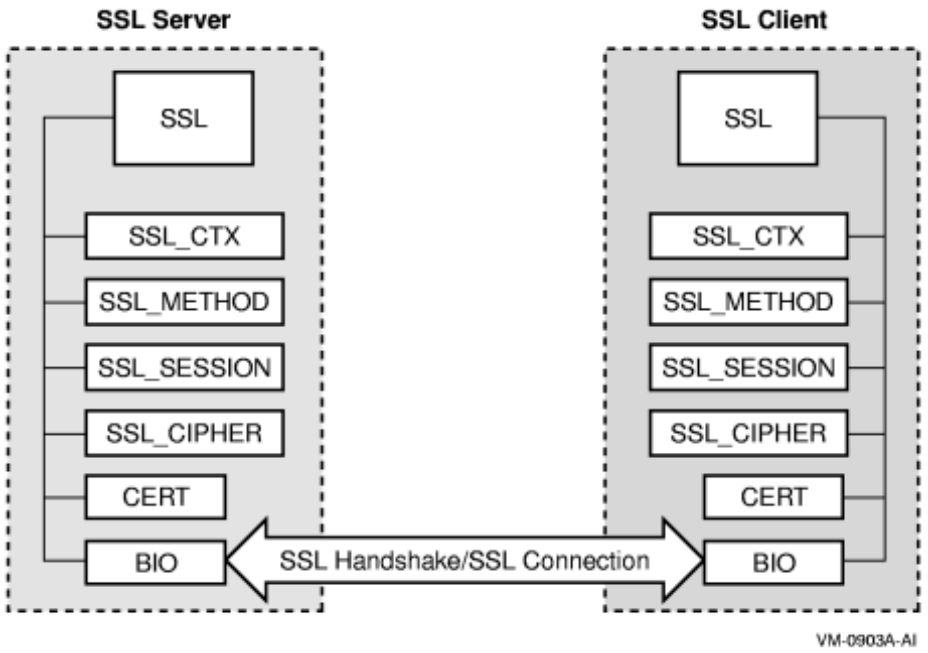
- The context structure (SSL_CTX) contains:
  - Default cipher list
  - Session identifier cache
  - Certificate cache
  - Default session identifier time-out period
  - Certificate verification mode and callback
  - Information callback for state transition logging
  - Default certificate/private key pair
  - Default read ahead mode
  - Session identifier cache callback



- An SSL_CTX structure stores
  - default values for SSL structures (the SSL structures inherit its configuration)
  - information about SSL connections and sessions
    - the number of new SSL connections, renegotiations, session resumptions, etc.
- A CA certificate loaded in the SSL_CTX structure is also loaded into an SSL structure when that SSL structure is created

OPENSSL STRUCTURES: SSL

- An SSL structure is created every time a new SSL connection is created
- It inherits configuration information from the SSL_CTX structure
- It saves the addresses of data structures that store information about SSL connections and sessions
  - The SSL_CTX structure from which the SSL structure is created
  - SSL_METHOD
  - SSL_SESSION
  - SSL_CIPHER
  - CERT: certificate information
  - BIO: an abstraction for generic I/O operations



OPENSSL STRUCTURES (1/2)

- SSL_METHOD
  - the internal ssl library methods/functions which implement the various protocol versions (SSLv1, v2, v3 and TLSv1)
  - needed to create an SSL_CTX
- SSL_CIPHER
  - holds the algorithm information for a particular cipher chosen
  - the available ciphers are configured on a SSL_CTX basis and the actually used ones are then part of the SSL_SESSION
- SSL_SESSION
  - a structure containing the current TLS/SSL session details for a connection
    - SSL_CIPHERs, client and server certificates, keys, etc.

OPENSSL STRUCTURES (2/2)

- X.509 certificate
  - stored as an X509 structure
  - SSL_CTX (or SSL) contains a generic reference to a CERT structure, but the programmer has not to know its details
- BIO structure
  - an I/O abstraction
  - it encapsulates an underlying I/O secured by SSL
  - all communications between the client and server are conducted through this structure

SSL METHODS

- Different roles:
  - combined client/server, dedicated server, client:

SSLv2: SSLv2_method(), SSLv2_server_ method(), SSLv2_client_ method()

SSLv3: SSLv3_method(), SSLv3_server_ method(), SSLv3_client_method()

TLSv1: TLSv1_method(), TLSv1_server_ method(), TLSv1_client_ method()

Example code:

```
//Context structure pointer
SSL_CTX *ctx;
//Method structure pointer
SSL_METHOD *meth;
//define the method structure pointer
meth = SSLv3_client_method();
//define a new context with the specified method
ctx = SSL_CTX_new (meth);
```

SSL NEW CONTEXT CREATION

- #include <openssl/ssl.h>

  SSL_CTX *SSL_CTX_new(SSL_METHOD *meth)

  The SSL_CTX_new function creates a new context (CTX) structure for use by one or more Secure Sockets Layer (SSL) sessions that are not shared. Use the **SSL_CTX_new_shared()** function to create a CTX structure for shared SSL sessions.

- The list of protocols available can later be limited using the SSL_OP_NO_SSLv2, SSL_OP_NO_SSLv3, SSL_OP_NO_TLSv1 options of the **SSL_CTX_set_options()** or **SSL_set_options()** functions. Using these options it is possible to choose e.g. SSLv23_server_method() and be able to negotiate with all possible clients, but to only allow newer protocols like SSLv3 or TLSv1
- SSL_CTX_new() initializes the list of ciphers, the session cache setting, the callbacks, the keys and certificates, and the options to its default values

SET CONTEXT KEYS AND CERTIFICATES (1/3)

- int SSL_CTX_use_certificate_file(SSL_CTX *ctx,const char *file,int type)
  - loads the certificate *(max 255 characters)* for use with Secure Sockets Layer (SSL) sessions using a specific context (CTX) structure
  - “type” indicates the certificate format:
    - SSL_FILETYPE_ASN1
      - The file is in abstract syntax notation 1 (ASN.1) format
    - SSL_FILETYPE_PEM
      - The file is in base64 privacy enhanced mail (PEM) format.

Example code:

```
#define CERTF "Client_cert.pem"
 if (SSL_CTX_use_certificate_file(ctx, CERTF, SSL_FILETYPE_PEM) <= 0) {
        ERR_print_errors_fp(stderr);
 }
```

SET CONTEXT KEYS AND CERTIFICATES (2/3)

- int SSL_CTX_use_PrivateKey_file(SSL_CTX *ctx, const char *file, int type)

  File type, which must be the following: SSL_FILETYPE_PEM

  Example Code:

```
#define KEYF    "Client_key.pem"
if (SSL_CTX_use_PrivateKey_file(ctx, KEYF, SSL_FILETYPE_PEM) <= 0) {
        ERR_print_errors_fp(stderr);
}
```

SET CONTEXT KEYS AND CERTIFICATES (3/3)

- int SSL_CTX_check_private_key(SSL_CTX *ctx)

  The SSL_CTX_check_private_key function verifies that the private key agrees with the corresponding public key in the certificate associated with a specific context (CTX) structure.

  Example code:

  ```
  if (!SSL_CTX_check_private_key(ctx)) {
  fprintf(stderr,“\nPrivate key does not match the certificate public key");
  }
  ```

NEW SESSIONS

- SSL *SSL_new(SSL_CTX *ctx)

  Create a new SSL structure inheriting from ctx

- int SSL_set_fd(SSL *ssl,int fd)

  fd is the socket filedescriptor

- int SSL_CTX_load_verify_locations(SSL_CTX *ctx, const char *CAfile,
  const char *CApath)

  The SSL_CTX_load_verify_locations function loads the certificates of the certificate authorities (CAs) that are trusted by this application and that will be used to verify certificates that are received from remote applications. Certificate revocation lists (CRLs) are also loaded if any exist.

  One of the CAfile or CApath can be NULL, but not both

CONNECTION CALLS (1/2)

- int SSL_accept(SSL *)

  The SSL_accept function accepts a Secure Sockets Layer (SSL) session connection request from a remote client application

- int SSL_connect(SSL *)

  The SSL_connect function starts a Secure Sockets Layer (SSL) session with a remote server application

- int SSL_shutdown(SSL *)

  The SSL_shutdown function shuts down data flow for a Secure Sockets Layer (SSL) session.

  The SSL_shutdown function is the normal way to shut down an SSL session. It is a good idea that you shut down an SSL session before the socket is shut down and closed

- SSL_free(SSL*), SSL_CTX_free(SSL_CTX *)

  The SSL_free function returns to the system the Secure Sockets Layer (SSL) structure associated with an SSL session

CONNECTION CALLS (2/2)

- In a server:

```
nd = accept(socketid, (struct sockaddr*)&client_addr, (socklen_t*)(&addr_length) );
if ( fork()==0 )
{
        ssl = SSL_new(ctx);
        if ( SSL_set_fd(ssl,nd) == 0 ) printf("\nError in SSL_set_fd");
        int err = SSL_accept(ssl);
}
…
```

- In a client:

```
…
connect(socketid, (struct sockaddr *)&server_address, sizeof(server_address))
…
SSL_connect(ssl);
```

ENCRYPTING DATA TRANSFER

- Standard sockets calls sendto(), recvfrom(), read() and write() can be used, but they DO NOT encrypt the payload
- Encrypt the payloads with SSL_write() and SSL_read():

Sending:

int SSL_write(SSL *ssl, const char *buf, int num)

Receiving:

int SSL_read(SSL *ssl, char *buf, int num)

SAMPLE SSL SERVER CODE

```
int main(int count, char *strings[])
{   SSL_CTX *ctx;
    int server;
    char *portnum;

    if(!isRoot())
    {
        printf("This program must be run as root/sudo user!!");
        exit(0);
    }
    if ( count != 2 )
    {
        printf("Usage: %s <portnum>\n", strings[0]);
        exit(0);
    }
    SSL_library_init();

    portnum = strings[1];
    ctx = InitServerCTX();        /* initialize SSL */
    LoadCertificates(ctx, "mycert.pem", "mycert.pem"); /* load certs */
    server = OpenListener(atoi(portnum));    /* create server socket */
    while (1)
    {   struct sockaddr_in addr;
        socklen_t len = sizeof(addr);
        SSL *ssl;

        int client = accept(server, (struct sockaddr*)&addr, &len);  /* accept connection as usual */
        printf("Connection: %s:%d\n",inet_ntoa(addr.sin_addr), ntohs(addr.sin_port));
        ssl = SSL_new(ctx);              /* get new SSL state with context */
        SSL_set_fd(ssl, client);      /* set connection socket to SSL state */
        Servlet(ssl);         /* service connection */
    }
    close(server);          /* close server socket */
    SSL_CTX_free(ctx);         /* release context */
}
```

```
SSL_CTX* InitServerCTX(void)
{   SSL_METHOD *method;
    SSL_CTX *ctx;

    OpenSSL_add_all_algorithms();  /* load & register all cryptos, etc. */
    SSL_load_error_strings();   /* load all error messages */
    method = SSLv3_server_method();  /* create new server-method instance */
    ctx = SSL_CTX_new(method);   /* create new context from method */
    if ( ctx == NULL )
    {
        ERR_print_errors_fp(stderr);
        abort();
    }
    return 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, "Private key does not match the public certificate\n");
        abort();
    }
}
```

```
int OpenListener(int port)
{   int sd;
    struct sockaddr_in addr;

    sd = socket(PF_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("can't bind port");
        abort();
    }
    if ( listen(sd, 10) != 0 )
    {
        perror("Can't configure listening port");
        abort();
    }
    return sd;
}
```

```
void Servlet(SSL* ssl) /* Serve the connection -- threadable */
{   char buf[1024];
    char reply[1024];
    int sd, bytes;
    const char* echo="--- %s ---";

    if ( SSL_accept(ssl) == FAIL )     /* do SSL-protocol accept */
        ERR_print_errors_fp(stderr);
    else
    {
       ShowCerts(ssl);        /* get any certificates */
       while(1){
              bytes = SSL_read(ssl, buf, sizeof(buf)); /* get request */
              if ( bytes > 0 )
               {
                  if(strncmp(buf,"BYE",strlen("BYE")) == 0) {
                     printf("Client sent a BYE! Going to close connection…\n");
                     break;
                  }
                  else{
                     buf[bytes] = 0;
                     printf("Client msg: \"%s\"\n", buf);
                     sprintf(reply, echo, buf);   /* construct reply */
                     SSL_write(ssl, reply, strlen(reply)); /* send reply */
                  }
               }
              else
                  ERR_print_errors_fp(stderr);
       }
    }
    sd = SSL_get_fd(ssl);       /* get socket connection */
    SSL_free(ssl);         /* release SSL state */
    close(sd);          /* close connection */
}
```

```
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("No certificates.\n");
}
```

COMPILATION COMMAND

- Source code:
  - “SSL-Server.c”
- Compilation command:

```
gcc -Wall -o ssl-server SSL-Server.c -L/usr/lib -lssl -lcrypto
```

- Output produced (executable server program):
  - “ssl-server”

SAMPLE SSL CLIENT CODE

```
int main(int count, char *strings[])
{   SSL_CTX *ctx;
    int server;
    SSL *ssl;
    char buf[1024];
    int bytes;
    char *hostname, *portnum;

    if ( count != 3 )
    {
        printf("usage: %s <hostname> <portnum>\n", strings[0]);
        exit(0);
    }
    SSL_library_init();
    hostname=strings[1];
    portnum=strings[2];

    ctx = InitCTX();
    server = OpenConnection(hostname, atoi(portnum));
    ssl = SSL_new(ctx);      /* create new SSL connection state */
    SSL_set_fd(ssl, server);    /* attach the socket descriptor */
    if ( SSL_connect(ssl) == FAIL )   /* perform the connection */
        ERR_print_errors_fp(stderr);
    else
    {
       char msg[100]

       printf("Connected with %s encryption\n", SSL_get_cipher(ssl));
       ShowCerts(ssl);        /* get any certs */

       for(;;){
              scanf("%[^\n]%*c",&msg);
              printf("Message to be sent to server: %s\n",msg);
               SSL_write(ssl, msg, strlen(msg));   /* encrypt & send message */
              if(strncmp(msg,"BYE",strlen(msg)) == 0) break;
              bytes = SSL_read(ssl, buf, sizeof(buf)); /* get reply & decrypt */
               buf[bytes] = 0;
              printf("Received: \"%s\"\n", buf);
       }
        SSL_free(ssl);        /* release connection state */
    }
    close(server);         /* close socket */
    SSL_CTX_free(ctx);        /* release context */
    return 0;
}
```

```
SSL_CTX* InitCTX(void)
{   SSL_METHOD *method;
    SSL_CTX *ctx;

    OpenSSL_add_all_algorithms();  /* Load cryptos, et.al. */
    SSL_load_error_strings();   /* Bring in and register error messages */
    method = SSLv3_client_method();  /* Create new client-method instance */
    ctx = SSL_CTX_new(method);   /* Create new context */
    if ( ctx == NULL )
    {
        ERR_print_errors_fp(stderr);
        abort();
    }
    return ctx;
}
```

```
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(PF_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;
}
```

```
void ShowCerts(SSL* ssl)
{   X509 *cert;
    char *line;

    cert = SSL_get_peer_certificate(ssl); /* get the server's certificate */
    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);       /* free the malloc'ed string */
        line = X509_NAME_oneline(X509_get_issuer_name(cert), 0, 0);
        printf("Issuer: %s\n", line);
        free(line);       /* free the malloc'ed string */
        X509_free(cert);     /* free the malloc'ed certificate copy */
    }
    else
        printf("No certificates.\n");
}
```

THE REAL THING IN ACTION…

```
MacBookPro-spromano:SSL spromano$ sudo ./ssl-server 5000
Connection: 127.0.0.1:55691
No certificates.
Client msg: "Ciao server!"
Client msg: "Tutto OK?"
Client msg: "Ti saluto..."
Client sent a BYE! Going to close connection…
```

```
MacBookPro-spromano:SSL spromano$ sudo ./ssl-client localhost 5000
Connected with AES256-SHA encryption
Server certificates:
Subject: /C=IT/ST=Some-State/L=Napoli/O=unina/OU=comics/CN=spromano/emailAdd
ress=spromano@unina.it
Issuer: /C=IT/ST=Some-State/L=Napoli/O=unina/OU=comics/CN=spromano/emailAddr
ess=spromano@unina.it
Ciao server!
Message to be sent to server: Ciao server!
Received: "--- Ciao server! ---"
Tutto OK?
Message to be sent to server: Tutto OK?
Received: "--- Tutto OK? ---"
Ti saluto...
Message to be sent to server: Ti saluto...
Received: "--- Ti saluto... ---"
BYE
Message to be sent to server: BYE
MacBookPro-spromano:SSL spromano$
```

BEHIND THE SCENES…

Source	Destination	Protocol	Length	Info
127.0.0.1	127.0.0.1	TCP	68	55697→5000 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS=32 TSval=279035949 TSecr=0
127.0.0.1	127.0.0.1	TCP	68	5000→55697 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16344 WS=32 TSval=279035
127.0.0.1	127.0.0.1	TCP	56	55697→5000 [ACK] Seq=1 Ack=1 Win=408288 Len=0 TSval=279035949 TSecr=279035949
127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 5000→55697 [ACK] Seq=1 Ack=1 Win=408288 Len=0 TSval=279035
127.0.0.1	127.0.0.1	SSLv3	150	Client Hello
127.0.0.1	127.0.0.1	TCP	56	5000→55697 [ACK] Seq=1 Ack=95 Win=408192 Len=0 TSval=279035949 TSecr=279035949
127.0.0.1	127.0.0.1	SSLv3	1070	Server Hello, Certificate, Server Hello Done
127.0.0.1	127.0.0.1	TCP	56	55697→5000 [ACK] Seq=95 Ack=1015 Win=407264 Len=0 TSval=279035949 TSecr=279035
127.0.0.1	127.0.0.1	SSLv3	268	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
127.0.0.1	127.0.0.1	TCP	56	5000→55697 [ACK] Seq=1015 Ack=307 Win=407968 Len=0 TSval=279035949 TSecr=27903
127.0.0.1	127.0.0.1	SSLv3	131	Change Cipher Spec, Encrypted Handshake Message
127.0.0.1	127.0.0.1	TCP	56	55697→5000 [ACK] Seq=307 Ack=1090 Win=407200 Len=0 TSval=279035950 TSecr=27903
127.0.0.1	127.0.0.1	SSLv3	130	Application Data, Application Data
127.0.0.1	127.0.0.1	TCP	56	5000→55697 [ACK] Seq=1090 Ack=381 Win=407904 Len=0 TSval=279042593 TSecr=27904
127.0.0.1	127.0.0.1	SSLv3	146	Application Data, Application Data

QUESTIONS?