Lecture 12 openSSL source

1. demos/ssl/cli.cpp

meth = TLSv1\_client\_method();

ctx = SSL\_CTX\_new(meth);

sd=socket(......);

err = connect(sd, ....);

ssl = SSL\_new(ctx);

SSL\_set\_fd(ssl, sd);

err=SSL\_connect(ssl);

Where is TLSv1\_client\_method()? Find with grep.

$ grep –nr "TLSv1\_client\_method" \*

2. ssl/t1\_clnt.c

IMPLEMENT\_tls\_meth\_func(TLS1\_VERSION,TLSv1\_client\_method, ssl\_undefined\_function,

ssl3\_connect, tls1\_get\_client\_method)

3. ssl/ssl\_locl.h

#define IMPLEMENT\_tls\_meth\_func(version, func\_name, s\_accept, s\_connect, s\_get\_meth)

SSL\_METHOD \*func\_name(void){

static SSL\_METHOD func\_name##\_data={

version, tls1\_new, tls1\_clear, tls1\_free,

s\_accept, s\_connect, ssl3\_read, ............};

return &func\_name##\_data;

}

The above definition will generate (by the compiler)

SSL\_METHOD \* TLSv1\_client\_method(void){

static SSL\_METHOD TLSv1\_client\_method\_data={

TLS1\_VERSION, tls1\_new, tls1\_clear, tls1\_free,

ssl\_undefined\_function, ssl3\_connect, .......};

return &TLSv1\_client\_method\_data;

}

4. ssl/ssl\_lib.c

int SSL\_connect(SSL \*s){

return (s->method->ssl\_connect(s));

}

5. include/openssl/ossl\_typ.h

typedef struct ssl\_st SSL;

5-1. crypto/ossl\_typ.h

typedef struct ssl\_ctx\_st SSL\_CTX;

5-2.

ssl/ssl.h

struct ssl\_ctx\_st{

const SSL\_METHOD \*method;

STAC\_OF(SSL\_CIPHER) \*cipher\_list;

.............

struct cert\_st \*cert;

........

}

ssl/ssl\_locl.h

typedef struct cert\_st{

CERT\_PKEY \*key;

...

} CERT;

typedef struct cert\_pkey\_st{

X509 \*x509;

EVP\_PKEY \*privatekey;

...

} CERT\_PKEY;

crypto/ossl\_typ.h

typedef struct evp\_pkey\_st EVP\_PKEY;

crypto/evp/evp.h

struct evp\_pkey\_st{

int type;

...

union{

char \*ptr;

#ifndef OPENSSL\_NO\_RSA

struct rsa\_st \*rsa; // RSA case

#endif

#ifndef OPENSSL\_NO\_DSA

struct dsa\_st \*dsa; // DSA case

#endif

..........

} pkey;

...

};

crypto/rsa/rsa.h

struct rsa\_st{

........

BIGNUM \*n, \*e, \*d, \*p, \*q;

............

};

crypto/bn/bn.h

struct bignum\_st{

BN\_ULONG \*d; // array of BN\_BITS bit chunks

.......

};

#define BN\_ULONG unsigned long

6. ssl/ssl.h

struct ssl\_st{

int version;

int type; // SSL\_ST\_CONNECT or SSL\_ST\_ACCEPT

SSL\_METHOD \*method;

BIO \*rbio; // used by SSL\_read

BIO \*wbio; // used by SSL\_write

BIO \*bbio; // used druing session-id reuse

int rwstate;

int in\_handshake; // true when we are in SSL\_accept() or SSL\_connect()

int (\*handshake\_func)(SSL \*);

int server;

int new\_session;

int state;

unsigned char \*packet; // raw packet

unsigned char packet\_length;

struct ssl3\_state\_st \*s3; // SSLv3 variables

X509\_VERIFY\_PARAM \*param;

STACK\_OF(SSL\_CIPHER) \*cipher\_list;

STACK\_OF(SSL\_CIPHER) \*cipher\_list\_by\_id;

struct cert\_st \*cert; // server certificate used

unsigned int sid\_ctx\_length;

unsigned char sid\_ctx[SSL\_MAX\_SID\_CTX\_LENGTH];

SSL\_SESSION \*session;

int verify\_mode; // whether to req client cert, etc.

SSL\_CTX \*ctx;

STACK\_OF(X509\_NAME) \*client\_CA; // list of CA

}

struct ssl3\_state\_st{

........

unsigned char server\_random[SSL3\_RADOM\_SIZE];

unsigned char client\_random[SSL3\_RADOM\_SIZE];

SSL3\_BUFFER rbuf; // read IO goes into here

SSL3\_BUFFER wbuf; // write IO goes into here

SSL3\_RECORD rrec; // each decoded record goes in here

SSL3\_RECORD wrec; // goes out from here

..........

}

typedef struct ssl3\_record\_st { // SSL3\_RECORD

int type; // type of record

unsigned int length; // how many bytes available

unsigned int off; // read/write offset int 'buf'

unsigned char \*data; // pointer to the record data

unsigned char \*input; // where the decode bytes are

unsigned char \*comp; // only used with decompression

unsigned long epoch; // epoch number, needed by DTLS1

unsigned char seq\_num[8]; // sequence number, needed by DTLS1

} SSL3\_RECORD;

7. ssl/s3\_clnt.c/int ssl3\_send\_client\_key\_exchange(SSL \*s){

unsigned char \*p, \*d;

int n;

unsigned long alg\_k;

unsigned char \*q;

EVP\_PKEY \*pkey=NULL;

d=s->init\_buf->data;

p=&(d[4]);

alg\_k=s->s3->tmp.new\_cipher->algorithm\_mkey;

RSA \*rsa;

unsigned char tmp\_buf[SSL\_MAX\_MASTER\_KEY\_LENGTH];

pkey=X509\_get\_pubkey(s->session->sess\_cert->peer\_pkeys[SSL\_PKEY\_RSA\_ENC].x509);

rsa=pkey->pkey.rsa;

tmp\_buf[0]=s->client\_version>>8;

tmp\_buf[1]=s->client\_version&0xff;

RAND\_bytes(&(tmp\_buf[2]), sizeof tmp\_buf-2); // ssl version number in the first 2 bytes

// and random premaster secret in the rest of it.

s->session->master\_key\_length=sizeof tmp\_buf;

q=p;

// fix buf for TLS and beyond

if (s->version > SSL3\_VERSION) p+=2;

n=RSA\_public\_encrypt(sizeof tmp\_buf, tmp\_buf, p, rsa, RSA\_PKCS1\_PADDING);