SSL–Secure Socket Layer

Basic SSL:
Encrypt the data that is transmitted over the network.
Operates on top of a normal socket.
Make a normal connection.
Use SSL over that connection.

1) The client requests that the ssl layer be used on an existing connection.

2) The server sends it's public key with it's certificate.

3) The client, if it chooses, may check the certificate to ensure the identity of the server.

4) The client generates a random symmetric encryption key, encrypts it using the server's public key and sends it to the server.
This will be the session key

5) The server decrypts the symmetric encryption key.
Both client and server now have the session key.

6) All future data transmitted over the net is encrypt with the session key
(and decrypted by the receiver).

Certificates

The certificate (if used):
Confirm the identify of the other machine.
You need to know other machines certificate.
Or the other machines certificate needs to be signed (cryptographically) by a machine you trust.
That machine signs using it's certificate.
A trusted central certificate authority (fee) is used.

The client is not required to use the certificate.

These go out of date when you reload. Note:
On the web, most people click through the out-of-date/unknown certificate panels.

SSL overview

SSL needs to be initialized.
The process takes several steps.

0) Get a connected socket.
1) Initialize the libraries.
2) Set up a context.
3) Set up an SSL environment from the context
4) Use the SSL environment to initiate an ssl connection over the socket.
5) Use the SSL environment to read/write over the socket.

Initializing the libraries

```
SSL_library_init()
```

Must precede all other SSL calls.

Registers the ciphers and digests that are available in the library

Setting up the Context

A context consists of:
allowable methods of encryption
allowable methods of generating/exchanging a session keys.
Contexts are used to establish SSL environments.
SSL allows serveral contexts to be in use at the same time (on different sockets).

```
SSL_METHOD* SSLv23_method()
```

This call returns a pointer to an SSL method.
Flavors: version 2 only, version 3 only, version 2 or 3.
As shown: a context created with this method will understand both SSLv2 and SSLv3 protocols
A series of calls is required to set up a context.

```
SSL_CTX* SSL_CTX_new(SSL_METHOD *)
```

Creates a context.
Pass the result of the method call.
The allocator is used, you will a free.

Setting up the Context

```
SSL_CTX_use_certificate_chain_file(SSL_CTX*,
            char* file)
```

Specifies the file in which the certificate (public key) to be used with this context is found.
SSL gives out the public key to the other machine so the other machine can use it to encrypt what it sends to us.
Usually only the session key (and some related data) is encrypted with this.

```
SSL_CTX_set_default_passwd_cb(SSL_CTX*,passwd_cb);
```

You don't want the private key sitting in a file on your disk, someone could steal it.
A password should be used to encrypt/decrypt the private keys.
The `passwd_cb` function wil return the password.
The public key is usually not encrypted.

If you do not set up callback you will be prompted for the password.
This isn't good if your program runs in the background.

Setting up the Context

The function passed as the callback must have the following profile.

```
int passwd_cb(char *buf, int len, int rwflag,
              void* userdata)
```

The password will be returned in `buf`.
When called it should be passed an array whose length is at least `len`.
This function is called automatically when the SSL environment needs to decrypt a private key.

```
SSL_CTX_use_PrivateKey_file(SSL_CTX*,
          char* file,SSL_FILETYPE_PEM);
```

Specifies the file in which the private key is found.
If there are multiple keys in the file, the first key found is decrypted and added to the context.
This is the private key corresponding to the public key we issue to the other machine.
Since a password will be needed to access the key, make sure the callback must already have been set up.
There are a couple different formats the key file can take, the most common is `pem`.

Using an SSL environment

```
SSL* SSL_new(SSL_CTX*);
```

Use a context up to create an ssl environment (object).
Warning: this copies the context, so any changes to the context made after this will not modify the SSL.

```
SSL_set_fd(SSL* ssl, int sock)
```

Associate a socket with the SSL ssl enviroment.
The socket must be already connected.
When we do SSL stuff it will be done over the connection represented by this socket.

Establishing an SSL session

`int SSL_connect(SSL*)`

Turns an ordinary client socket/connection into an SSL connection by initiating an SSL handshake with a server. A connected socket (established by an ordinary `connect`) must already have been associated with the SSL.

return values:
`1`: The handshake was successful
`0`: controlled error
`<0`: fatal error

`int SSL_accept(SSL *ssl)`

Turns an ordinary server conection into an SSL connection by responding to an SSL hand shake.
Waits for the client to initiate the SSL handshake.
A connected socket (created by an ordinary `accept`) must already have been associated with the SSL.

return values: the same as connect.

The socket value is NOT returned.

Write

```
SSL_write(SSL *ssl, const void *buffer, int num);
```

Behaves like read. `num` bytes will be written.

The `ssl` needs to be attached to a socket.

If a SSL session has not been negotiated (`SSL_connect` or `SSL_accept`), it will negotiated.

Returns:
1) the number of bytes written
2) 0 or negative on error

Action: encrypts and writes to the attached socket.

Read

```
SSL_read(SSL *ssl, void *buffer, int num);
```

Action: decrypts and places up to `num` bytes into the buffer.

Be sure to pass it a real buffer (not a dangling pointer).

The `ssl` needs to be attached to a socket.

If a SSL session has not been negotiated (`SSL_connect` or `SSL_accept`), it will negotiated.

Returns
1) the number of bytes read.
2) 0 on EOF
3) negative on error

Action, reads from the attached socket and decrypts.

SSL clean up

If the program is continuing it should free the memory allocated by the `SSL` and the `SSL_CTX`

```
SSL_free(SSL *ssl)
```

Frees memory associated with the ssl.

```
SSL_CTX_free(SSL_CTX *ctx)
```

Frees memory associated with the context

SSL error information

`int SSL_get_error(SSL *ssl, int errorVal)`

Use to get specific information about an error.

Given the `ssl` and the value returned (by read/write) this will return an integer indicating what error condition occurred.
If handed a positive it returns `SSL_ERROR_NONE` The list of error definitions is found in `ssl.h`.
The `ssl` is examined for extra information on the error value.

Keys and certificate.

You can create your own public/private pairs.

```
openssl req -new -x509 -keyout server.pem -out
server.pem
```

Creates an new private key and self-signed certificate with corresponding public key.
Both are placed in the file `server.pem`

```
openssl x509 -text -in server.pem
```

Examines the contents of the certificate/public key. The private key is encrypted in this file, the certificate is not.