

## SSL/TLS Programming

---

### sslClient.c

```
/*
  A simple SSL client.
  It connects and then forwards data from/to the terminal to/from the server
*/

#define CA_LIST      "root.pem"
#define ServerHOST  "deneb"
#define RANDOM      "random.pem"
#define PORT        10101
#define ClientKEYFILE "client.pem"
#define ClientPASSWORD "oducsc"

int main (argc,argv)
    int argc; char **argv
{

    SSL_CTX *ctx;
    SSL *ssl;
    BIO *sbio;
    int sock;

    /* Build our SSL context*/
    ctx = initialize_ctx (ClientKEYFILE, ClientPASSWORD);

    /* Connect the TCP socket*/
    sock = tcp_connect ();

    /* Connect the SSL socket */
    ssl = SSL_new (ctx);
    sbio = BIO_new_socket (sock, BIO_NOCLOSE);
    SSL_set_bio (ssl, sbio, sbio);
    SSL_connect (ssl);
    check_cert_chain (ssl, ServerHOST);
}
```

```

        /* read and write */
        read_write (ssl, sock);
    }

int tcp_connect ()
{
    struct hostent *hp;
    struct sockaddr_in addr;
    int sock;

    hp = gethostbyname (ServerHOST);

    memset (&addr,0,sizeof(addr));
    addr.sin_addr = *(struct in_addr*)hp->h_addr_list[0];
    addr.sin_family = AF_INET;
    addr.sin_port = htons(ServerPORT);

    sock = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
    connect (sock, (struct sockaddr *)&addr, sizeof(addr));
    return sock;
}

/*
    Read from the keyboard and write to the server
    Read from the server and write to the keyboard
*/
void read_write (ssl, sock)
    SSL *ssl;
{
    int r, c2sl=0;

    int shutdown_wait=0;
    char c2s[BUFSIZZ], s2c[BUFSIZZ];

    while (1) {

        /* Check for input on the console*/
        c2sl = read (0, c2s, BUFSIZZ);
        if (c2sl == 0 ) goto end;
    }
}

```

```

    /* If we've got data to write then try to write it*/
    SSL_write (ssl, c2s, c2sl);

    /* Now check if there's data to read */
    do {
        r = SSL_read (ssl, s2c, BUFSIZZ);

        switch (SSL_get_error (ssl, r) ) {

            case SSL_ERROR_NONE:
                fwrite (s2c, 1, r, stdout);
                break;
            case SSL_ERROR_ZERO_RETURN:
                /* End of data */
                goto end;
                break;
            default:
                berr_exit("SSL read problem");
        }
    } while ( SSL_pending (ssl) );

    /* end of while (1) */
end:
    SSL_shutdown (ssl);
    SSL_free (ssl);
    close (sock);
    return;
}

```

---

### common.c

```

SSL_CTX *initialize_ctx (keyfile, password)
char *keyfile;
char *password;
{

    SSL_METHOD *meth;
    SSL_CTX *ctx;

    if (!bio_err) {

```

```

        /* Global system initialization*/
        SSL_library_init ();
        SSL_load_error_strings ();

        /* An error write context */
        bio_err = BIO_new_fp (stderr, BIO_NOCLOSE)

    }

    /* Create our context*/
    meth = SSLv3_method ();
    ctx = SSL_CTX_new (meth);

    /* Load our keys and certificates*/
    SSL_CTX_use_certificate_file ( ctx, keyfile, SSL_FILETYPE_PEM);

    pass = password;
    SSL_CTX_set_default_passwd_cb (ctx, password_cb);
    SSL_CTX_use_PrivateKey_file (ctx, keyfile, SSL_FILETYPE_PEM);

    /* Load the CAs we trust*/
    SSL_CTX_load_verify_locations (ctx, CA_LIST,0);
    SSL_CTX_set_verify_depth (ctx,1);
    SSL_CTX_set_verify (ctx,
        SSL_VERIFY_PEER|SSL_VERIFY_FAIL_IF_NO_PEER_CERT,
        verify_callback);

    /* Load randomness */
    RAND_load_file (RANDOM,1024*1024);

    return ctx;

}

int verify_callback (int ok, X509_STORE_CTX *store)
{
    char data[256];

    /* if (ok) /* to debug */
    if (!ok)
    {
        X509 *cert = X509_STORE_CTX_get_current_cert(store);
        int depth = X509_STORE_CTX_get_error_depth(store);
        int err = X509_STORE_CTX_get_error(store);

        fprintf(stderr, "-Error with certificate at depth: %i\n", depth);
        X509_NAME_oneline(X509_get_issuer_name(cert), data, 256);
    }
}

```

```

    fprintf(stderr, " issuer = %s\n", data);
    X509_NAME_oneline(X509_get_subject_name(cert), data, 256);
    fprintf(stderr, " subject = %s\n", data);
    fprintf(stderr, " err %i:%s\n", err, X509_verify_cert_error_string(err) );
}

return ok;
}

```

*/\* Check that the common name matches the host name\*/*

**void check\_cert\_chain (ssl, host)**

```

SSL *ssl;
char *host;
{

    X509 *peer;
    char peer_CN[256];

    if ( SSL_get_verify_result (ssl) != X509_V_OK )
        berr_exit ("Certificate doesn't verify");

    /*Check the common name*/
    peer = SSL_get_peer_certificate (ssl);
    X509_NAME_get_text_by_NID (
        X509_get_subject_name (peer), NID_commonName, peer_CN,
256);
    if (strncasecmp (peer_CN, host))
        err_exit ("Common name doesn't match host name");

}

```

---

### **sslServer.c**

*/\* A multiprocess SSL server \*/*

```

#define ServerKEYFILE    "server.pem"
#define ClientPASSWORD  "oducsc"
#define DHFILE          "dh1024.pem"

```

```
#define CA_LIST      "root.pem"
#define ClientHOST  "vega"
#define RANDOM      "random.pem"
#define ServerPORT  10101
```

```
int main (argc,argv)
```

```
int argc;
char **argv;
{
```

```
    int sock,s;
    BIO *sbio;
    SSL_CTX *ctx;
    SSL *ssl;
    int r;
    pid_t pid;
```

```
    /* Build our SSL context*/
    ctx = initialize_ctx (ServerKEYFILE, ServerPASSWORD);
    load_dh_params (ctx, DHFILE);
    generate_eph_rsa_key (ctx);
```

```
    sock = tcp_listen ();
```

```
    while (1) {
        s = accept (sock, 0, 0);
```

```
        sbio = BIO_new_socket (s, BIO_NOCLOSE);
        ssl = SSL_new(ctx);
        SSL_set_bio (ssl, sbio, sbio);
```

```
        SSL_accept (ssl);
        check_cert_chain (ssl, ClientHOST);
        echo (ssl);
    }
```

```
}
```

```
}
```

```
int tcp_listen ()
```

```
{
```

```
    int sock;
    struct sockaddr_in sin;
```

```
sock = socket (AF_INET,SOCK_STREAM,0);
memset (&sin,0,sizeof(sin));
sin.sin_addr.s_addr = INADDR_ANY;
sin.sin_family = AF_INET;
sin.sin_port = htons(ServerPORT);

if ( bind (sock, (struct sockaddr *)&sin, sizeof(sin))<0)
    berr_exit("Couldn't bind");
listen (sock,5);

return (sock);
```

```
}
```

```
void load_dh_params (ctx, file)
```

```
SSL_CTX *ctx;
```

```
char *file;
```

```
{
```

```
    DH *ret=0;
```

```
    BIO *bio;
```

```
    bio = BIO_new_file (file, "r");
```

```
    ret = PEM_read_bio_DHparams (bio,NULL,NULL,NULL);
```

```
    BIO_free (bio);
```

```
    SSL_CTX_set_tmp_dh (ctx,ret);
```

```
}
```

```
void generate_eph_rsa_key (ctx)
```

```
SSL_CTX *ctx;
```

```
{
```

```
    RSA *rsa;
```

```
    rsa=RSA_generate_key(512,RSA_F4,NULL,NULL);
```

```
    SSL_CTX_set_tmp_rsa(ctx,rsa);
```

```
    RSA_free(rsa);
```

```
}
```

```
void echo (ssl)
```

```
SSL *ssl;
```

```
{
```

```

char buf[BUFSIZZ];
int r,len,offset;

while (1) {

    /* First read data */
    r=SSL_read (ssl, buf, BUFSIZZ);
    switch ( SSL_get_error (ssl,r) ){
        case SSL_ERROR_NONE:
            len=r;
            break;
        case SSL_ERROR_ZERO_RETURN:
            goto end;
        default:
            berr_exit ("SSL read problem");
    }

    /* Now keep writing until we've written everything*/
    offset=0;

    while (len) {

        r = SSL_write (ssl, buf+offset, len);
        switch (SSL_get_error (ssl,r)) {
            case SSL_ERROR_NONE:
                len-=r; offset+=r;
                break;
            default:
                berr_exit("SSL write problem");
        }
    }

} /* while (1) */
end:
    SSL_shutdown(ssl);
    SSL_free(ssl)
}

```

---

### ***Generating the required .pem files:***

```

#define DHFILE          "dh1024.pem"
#define RANDOM          "random.pem"

```

```
#define CA_LIST          "root.pem"  
#define ServerKEYFILE  "server.pem"  
#define ClientKEYFILE  "client.pem"
```

- To create the **dh512.pem** or **dh1024.pem**:  
% openssl dhparam -check -text -5 512 -out dh512.pem  
% openssl dhparam -check -text -5 1024 -out dh1024.pem
- To create the **random.pem**:  
% cp ~/cs772/random.pem .
- To create the **root.pem**:  
% cat ca\_key.pem ca\_cert.pem > root.pem
- To create the **server.pem** & **client.pem**:
  - First generate a cert request, make sure to specify: "localhost" or the actual host name, e.g., "cash.cs.odu.edu" as the CN name.
  - Sign the certificates from the CA and then do the following:  
  
% cat server\_privatekey.pem server\_cert.pem ca\_cert.pem >  
**server.pem**  
% cat cleint\_privatekey.pem client\_cert.pem ca\_cert.pem >  
**client.pem**