# SSL

## A:Getting your certificate

## Step 1: Generate a Private Key

The **openssl** toolkit is used to generate an **RSA Private Key** and **CSR (Certificate Signing Request)**. It can also be used to generate self-signed certificates which can be used for testing purposes or internal usage.

The first step is to create your RSA Private Key. This key is a 1024 bit RSA key which is encrypted using Triple-DES and stored in a PEM format so that it is readable as ASCII text.

```
openssl genrsa -des3 -out server.key 1024

Generating RSA private key, 1024 bit long modulus
.....................................................++++++
........++++++
e is 65537 (0x10001)
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
```

## Step 2: Generate a CSR (Certificate Signing Request)

Once the private key is generated a Certificate Signing Request can be generated. The CSR is then used in one of two ways. Ideally, the CSR will be sent to a Certificate Authority who will verify the identity of the requestor and issue a signed certificate.

During the generation of the CSR, you will be prompted for several pieces of information. These are the X.509 attributes of the certificate. One of the prompts will be for "Common Name (e.g., YOUR name)". It is important that this field be filled in with the fully qualified domain name of the server to be protected by SSL. If the website to be protected will be https://public.akadia.com, then enter public.akadia.com at this prompt. The command to generate the CSR is as follows:

```
openssl req -new -key server.key -out server.csr

Country Name (2 letter code) [GB]:CH
State or Province Name (full name) [Berkshire]:Bern
Locality Name (eg, city) [Newbury]:Oberdiessbach
Organization Name (eg, company) [My Company Ltd]:Akadia AG
Organizational Unit Name (eg, section) []:Information Technology
Common Name (eg, your name or your server's hostname) []:public.akadia.com
Email Address []:martin dot zahn at akadia dot ch
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

## Step 3: Remove Passphrase from Key

One unfortunate side-effect of the pass-phrased private key is **that Apache will ask for the pass-phrase each time the web server is started**. Obviously this is not necessarily convenient as someone will not always be around to type in the pass-phrase,

such as after a reboot or crash. mod_ssl includes the ability to use an external program in place of the built-in pass-phrase dialog, however, this is not necessarily the most secure option either. **It is possible to remove the Triple-DES encryption from the key**, thereby no longer needing to type in a pass-phrase. If the private key is no longer encrypted, it is critical that this file only be readable by the root user! If your system is ever compromised and a third party obtains your unencrypted private key, the corresponding certificate will need to be revoked. With that being said, use the following command to remove the pass-phrase from the key:

```
cp server.key server.key.org
openssl rsa -in server.key.org -out server.key
```

The newly created server.key file has no more passphrase in it.

```
-rw-r--r-- 1 root root 745 Jun 29 12:19 server.csr
-rw-r--r-- 1 root root 891 Jun 29 13:22 server.key
-rw-r--r-- 1 root root 963 Jun 29 13:22 server.key.org
```

## Step 4: Generating a (Self-Signed) Certificate

At this point you will need to generate a self-signed certificate or *request* certificate signed by a CA by forwading your CSR.

Example of generating self-signed certificate is shown below:

This temporary certificate will generate an error in the client browser to the effect that the signing certificate authority is unknown and not trusted.

To generate a temporary certificate which is good for 365 days, issue the following command:

```
openssl x509 -req -days 365 -in server.csr -signkey server.key -out
server.crt
Signature ok
subject=/C=CH/ST=Bern/L=Oberdiessbach/O=Akadia AG/OU=Information
Technology/CN=public.akadia.com/Email=martin dot zahn at akadia dot ch
Getting Private key
```

## B:Using your certificate

We will use an example of configuring apache server for ssl.

## Step 1: Installing the Private Key and Certificate

When Apache with mod_ssl is installed, it creates several directories in the Apache config directory. The location of this directory will differ depending on how Apache was compiled.

```
cp server.crt /usr/local/apache/conf/ssl.crt
cp server.key /usr/local/apache/conf/ssl.key
```

## Step 2: Configuring SSL Enabled Virtual Hosts

```
SSLEngine on
SSLCertificateFile /usr/local/apache/conf/ssl.crt/server.crt
SSLCertificateKeyFile /usr/local/apache/conf/ssl.key/server.key
SetEnvIf User-Agent ".*MSIE.*" nokeepalive ssl-unclean-shutdown
CustomLog logs/ssl_request_log \
   "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"
```

## Step 3: Restart Apache and Test

```
/etc/init.d/httpd stop
/etc/init.d/httpd stop

https://public.akadia.com
```

## Another example might be to use the cerficate ***programmatically*** to communicate over ssl.

### Step 1:Create a .pfx file from the signed certificate and key

```
openssl pkcs12 -inkey bob_key.pem -in bob_cert.cert -export -out
bob_pfx.pfx
```

Add the certificate(.cert) to your keystore.

### Step 2: create a TrustManager and HostNameVerifier in your code:

```
/** TrustAllCerts accepts all cert

    * Trust All Certificates - Needed for SSL Client

    */

    private static void TrustAllCerts() throws
java.security.NoSuchAlgorithmException,
java.security.KeyManagementException {

        // Create a trust manager that does not validate certificate chains

        TrustManager[] trustAllCerts = new TrustManager[] {

            new X509TrustManager() {

                public X509Certificate[] getAcceptedIssuers() {

                    return null;

                }
```

```
                public void checkClientTrusted(X509Certificate[] certs,
String authType) {

                    // Trust always

                }



                public void checkServerTrusted(X509Certificate[] certs,
String authType) {

                   // Trust always

                    System.out.println("authType is " + authType);

                    System.out.println("cert issuers");

                    for (int i = 0; i < certs.length; i++) {

                            System.out.println("\t" +
certs[i].getIssuerX500Principal().getName());

                            System.out.println("\t" +
certs[i].getIssuerDN().getName());

                    }

                }

            }

        };



        // Install the all-trusting trust manager

        SSLContext sc = SSLContext.getInstance("SSL");



        // Create empty HostnameVerifier

        HostnameVerifier hv = new HostnameVerifier() {

            public boolean verify(String arg0, SSLSession arg1) {

                return true;

            }

        };
```

```
        String keyStorePath = ".";

        String keyStorePass = " bob_pfx";

        String certAlias = "1e-b76a47a5-5e07-487f-b6f9-0f944b4b4d51";

        String certPass = "";

        boolean isKeyStore = true;

        KeyManagerFactory keyManagerFactory =
KeyManagerFactory.getInstance("SunX509");



        try {

            KeyStore certStore = KeyStore.getInstance("PKCS12");



            if (!isKeyStore) {

                File certFile = new File(keyStorePath);

                InputStream keyInput = new FileInputStream(certFile);

                certStore.load(keyInput, certPass.toCharArray());

                keyInput.close();

            }

            else {

                //FileInputStream kfis = new FileInputStream(keyStorePath +
File.separatorChar + "keystore.keystore");

                FileInputStream kfis = new FileInputStream("bob_pfx.pfx");

                KeyStore keyStore = KeyStore.getInstance("PKCS12");

                keyStore.load(kfis, keyStorePass.toCharArray());



                // Get certificate

                Certificate cert = keyStore.getCertificate(certAlias);

                certStore =
KeyStore.getInstance(KeyStore.getDefaultType());

                certStore.load(null, null);
```

```
                certStore.setCertificateEntry(certAlias, cert);



            for (Enumeration<String> e = certStore.aliases();
e.hasMoreElements();)

                  System.out.println(e.nextElement());

            }



            keyManagerFactory.init(certStore, certPass.toCharArray());

        }

        catch (KeyStoreException ke) {

            System.out.println(gt.getDateTime() + "|Error loading client
certificate: " + ke.toString());

        }

        catch (CertificateException ce) {

            System.out.println(gt.getDateTime() + "|Error loading client
certificate: " + ce.toString());

        }

        catch (NoSuchAlgorithmException ne) {

            System.out.println(gt.getDateTime() + "|Error loading client
certificate: " + ne.toString());

        }

        catch (UnrecoverableKeyException ue) {

            System.out.println(gt.getDateTime() + "|Error loading client
certificate: " + ue.toString());

        }

        catch (IOException io) {

            System.out.println(gt.getDateTime() + "|Error loading client
certificate: " + io.toString());

        }

        finally {

            sc.init(keyManagerFactory.getKeyManagers(), trustAllCerts, new
java.security.SecureRandom());
```

```
        }


HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
        HttpsURLConnection.setDefaultHostnameVerifier(hv);
    }
```

**Step 3:Initialize the SSL in your code:**

**Public void SendRequest()**

**{**

```
      TrustAllCerts();
      --Rest of your code
```

**}**

Done!