

# No-Nonsense Guide to SSL

---

A guide for Application Administrators and  
Developers

**Karun Subramanian**



[www.karunsubramanian.com](http://www.karunsubramanian.com)

## Contents

Section 1: Understanding SSL communication .....	4
What is SSL? .....	4
How does SSL work? .....	6
Where do SSL certificates come into play?.....	7
To summarize:.....	7
Taking a deeper look at a SSL certificate .....	7
What about self signed certificates? .....	13
Section 2: Troubleshooting SSL communication.....	14
Step 0: Determine if the communication really involves SSL. ....	14
Step 1: Determine the client and server .....	14
Step 3: Identify the Error .....	15
Server not reachable .....	15
The Server SSL certificate is NOT trusted by the client ← <i>this is the most common error with SSL handshakes.</i> .....	15
Name of the site and the name on the SSL Certificate do not match .....	16
Certificate has expired .....	17
Cannot negotiate cipher spec .....	17
Cannot handle 2048 bit public key and SHA-256 signature algorithm .....	18
Server requires client auth but the client does NOT have a valid certificate to send to.....	18
Step 4: Fix the error .....	18
Fixing Trust issues .....	18
Fixing Cipher Spec related issues .....	20
Section 3: Using Tools and Commands to manage SSL keys. ....	21
Keytool .....	21
Create a new keystore .....	21
List items from a keystore.....	22
Export a certificate from keystore .....	22
Print a certificate from a certificate file.....	23
Change password of a keystore .....	23
Create a certificate signing request: .....	23
Receive Certificate from CA .....	24

Import a keystore into a destination keystore .....	24
Import a trusted cert.....	25
Create a secret key.....	25
OpenSSL .....	25
Connect to a remote SSL Server to retrieve it's public certificate ( and to make sure you are able to connect ) .....	25
Show protocol messages (sort of verbose logging) .....	26
Create a CSR to be submitted to a CA.....	26
Create CSR.....	27
Create a self signed certificate.....	28
Converting formats .....	28
Further Reading .....	28
OpenSSL Documentation .....	28
Oracle JSEE Documentation.....	28
About the Author .....	29

# No-nonsense Guide to SSL

## A guide for Application Administrators and Developers

### What can you get out of this guide?

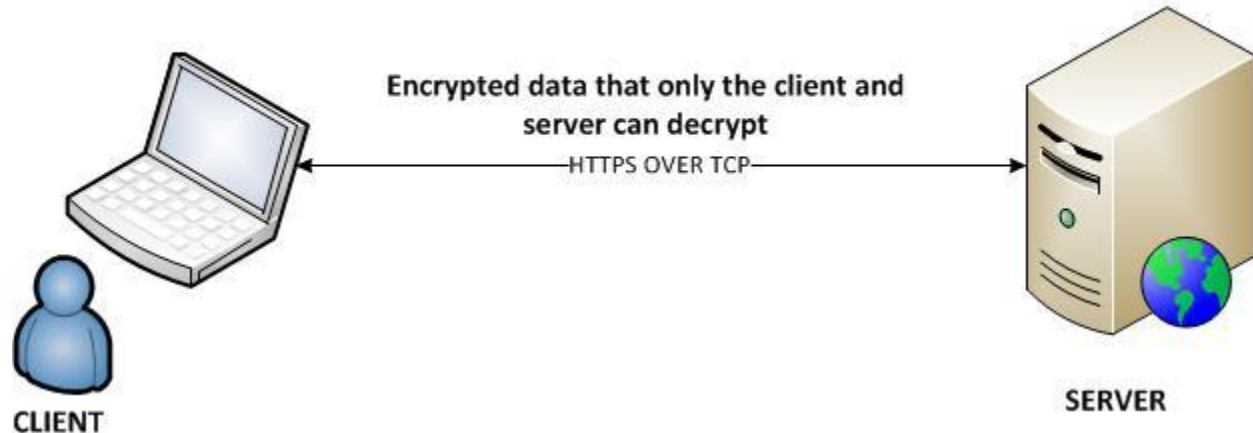
1. A clear understanding of SSL in the capacity of an Application Administrator, Middleware Administrator, Systems Administrator or a developer in charge of support
2. Step by Step instruction on how to troubleshoot SSL handshake issues
3. How to use commands and tools to easily manage SSL keys and certificates

## Section 1: Understanding SSL communication

### What is SSL?

SSL stands for Secure Sockets Layer. It is a technology using which the data exchange between two systems are encrypted so that no one except the systems involved can interpret the data.

### SSL ENABLED COMMUNICATION BETWEEN CLIENT AND SERVER



As you can see in the picture above, SSL communication involves a client and a server. Client is most probably a browser from a user's PC, even though it can be a server application communicating with another server application. The server is most probably a Web Server, even though it can be any ssl enabled server application such as a Database Server or a Message queueing Server.

When you access a website using your browser, if the site is SSL enabled, you can see the security information in the browser. For example, see image below.



**Bank of America**

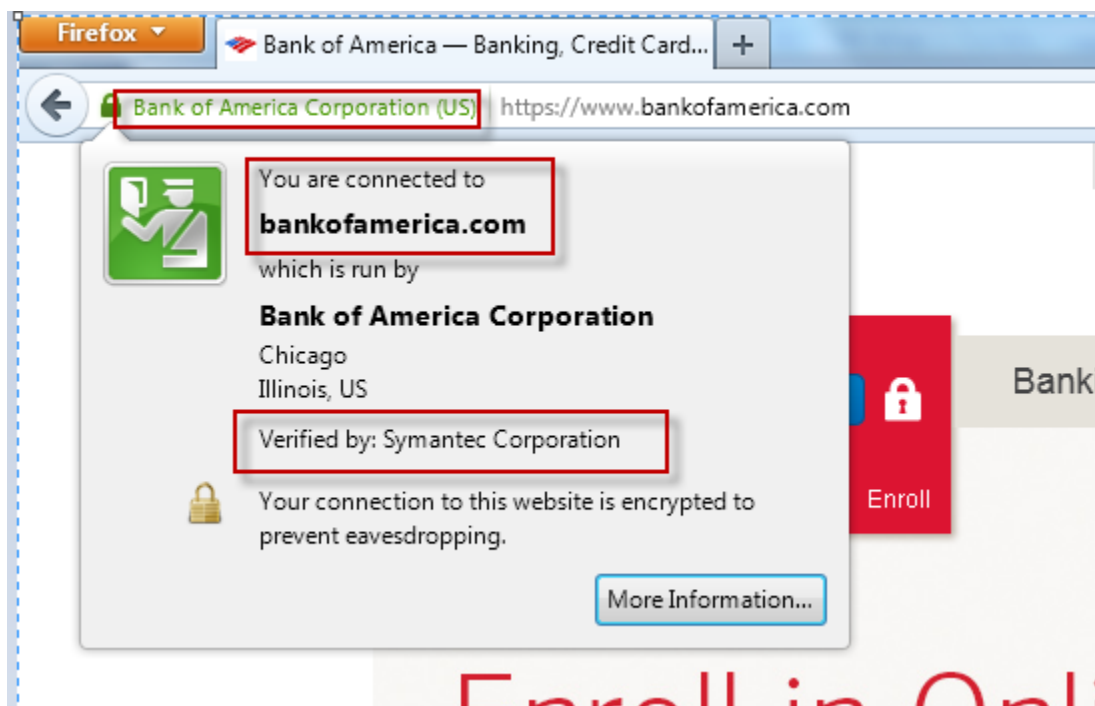
Enter your Online ID

[Sign In](#)

☐ Save this Online ID

[Help/options](#) [Enroll](#)

If you click on the green 'Bank of America Corporation (US)', you can see additional details



Note that SSL is a protocol, meaning various software products and frameworks implement SSL a little differently. However they must all adhere to the SSL specification. From Administration and Support perspective, what this means is you will have to do a bit of a reading to support SSL with various products. For example, while it is true that you need to add the CA certificate of the remote system you are trying to connect to your trust store, the steps you must follow for a WebLogic Application server will be different from a Jboss Application Server. I will explain all about CA certs in a sec.

TLS, which stands for Transport Layer Security is the new generation SSL (After SSL version 3.0, it is TLS). As you can guess, TLS is the preferred protocol as it is newer (TLS 1.2 is the latest as of June 2015).

In the OSI Network model, SSL/TLS will fit in somewhere between Transport and Application layer.

SSL provides for two key security aspects in a client-server communication.

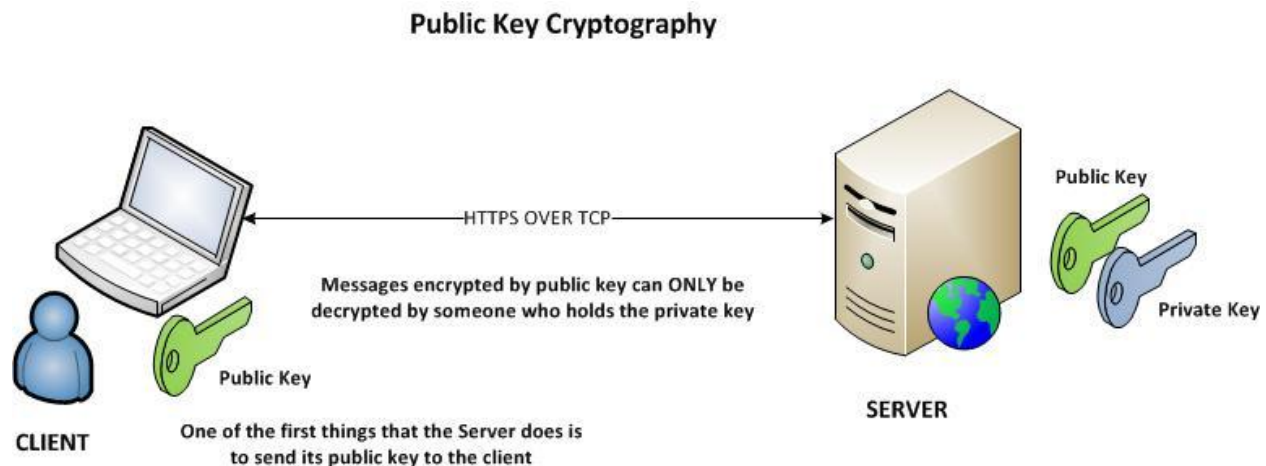
**Authentication:** You are guaranteed to communicate with the system that you are trying to communicate with (unless you are victim of *Man in the Middle attack*)

**Encryption:** The data being exchanged is guaranteed to be decrypted ONLY by you and the system you are communicating with (*unless someone gets access to your private key*)

## How does SSL work?

SSL is based on **Public Key Cryptography**. Let us take a simple case to illustrate how it works.

A Web Server that wants to use SSL for its communication with the clients will first obtain a *Private Key* and a corresponding *Public key* (this is called *Key Pair* as they are mathematically related). When a client accesses the Web Server (for example, using the URL <https://yourserver.com>), the server sends the public key to the client. The client encrypts the data using the public key (or a derivative of the public key) which can be decrypted ONLY by someone who has the private key; in this case it is your server.



Public keys come in various strengths, depending on their size. A strong key size is 2048 bits. A 2048 bits RSA Public key looks like below:

```
30 82 01 0a 02 82 01 01 00 b9 4b 64 43 5c 68 dc 87 6b 29 6f 5d 2b 2d 50 80 e1 38 1e 07 f4
e2 34 b1 cf 73 37 8a 30 9c c8 63 da 8a 50 64 1f 71 80 93 c1 f5 9e d8 4d 71 05 ea ff 8b 43
97 21 47 1a fb f0 6b dc 74 ca da a9 0d a1 cf e7 b8 59 09 27 77 a3 d7 b0 3d 2a fd 44 62 1b
0d fb 68 07 a1 b2 84 cb eb 0c 4c 9c 86 98 55 68 f7 c9 ee 57 d9 a6 88 45 8c ec 18 b4 63 a6
11 55 80 8e 7b 2e 0a db 25 30 48 ad 4c 3a 07 35 d1 1c db c0 3b 24 f8 fd a6 bd 3c ef 3e 2f
10 67 37 78 23 7e bf c6 51 45 86 4d fd 54 a6 3d e0 fa 2b 24 2c c5 ff aa a2 7e 63 67 22 3c
4c 92 b9 92 28 92 8a 41 a6 6e f2 f8 1b b2 1a a5 8f 7f 93 91 e0 ad 22 37 51 ef b3 9c 66 e4
4e 30 57 67 2a ca 7c 24 a0 2a f5 bc d7 98 39 30 94 39 94 cf 10 33 81 89 80 be a0 b0 42 f3
56 b4 9a e6 d1 27 a2 12 43 6c a0 ae 45 c5 a7 6b 7f 0e 8e 3b b0 42 19 78 8d 02 03 01 00 01
```

Note: Larger keys come with some performance hit but it is more difficult to crack. You must use at least 2048 bits.

## Where do SSL certificates come into play?

**A SSL certificate is nothing but a file containing Signed Public Key.** It is a secure mechanism to send the Public key. More importantly, since the public key is signed, it proves the validity of the certificate (Authentication).

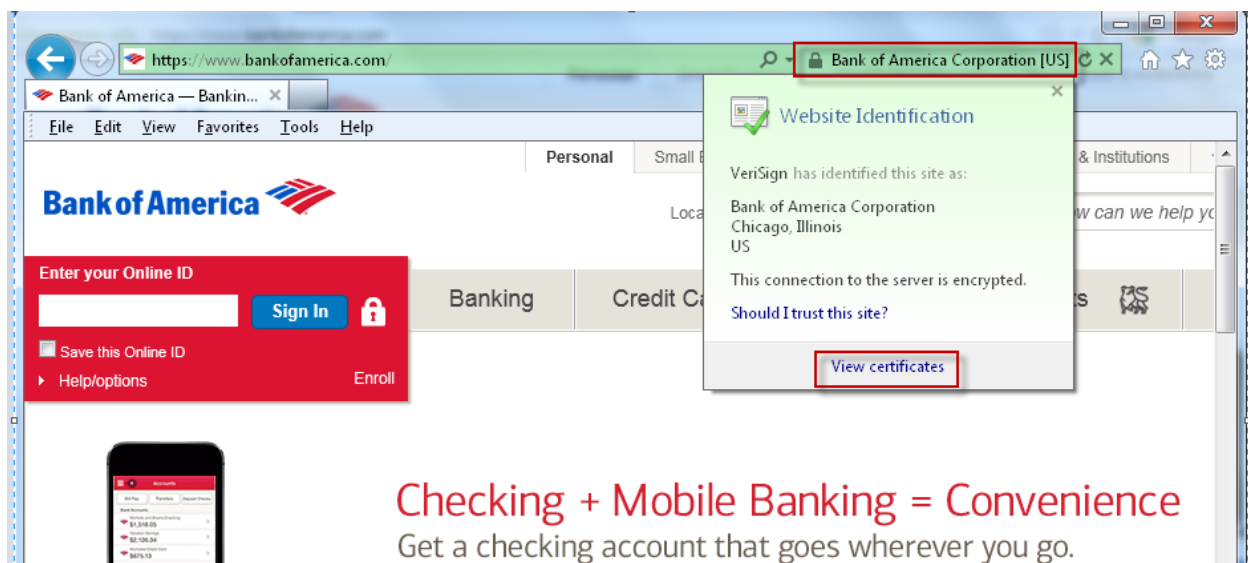
Here is how it works:

When a Server wants to enable SSL, it generates a key pair (the method of creating the key pair depends on the technology/product it is using. But typically java based applications can use the application 'keytool' that comes with Java). It then applies for a signed public key. It does so by creating a CSR (Certificate signing request) and sending it to a CA (Certificate authority). Upon reviewing the application, CA generates a Certificate chain which includes the Server certificate (signed public key) and a certificate (or certificates) of the CA itself. This certificate chain will need to be installed in the Server application (again, installing the server certificate depends on the product/technology the server is using).

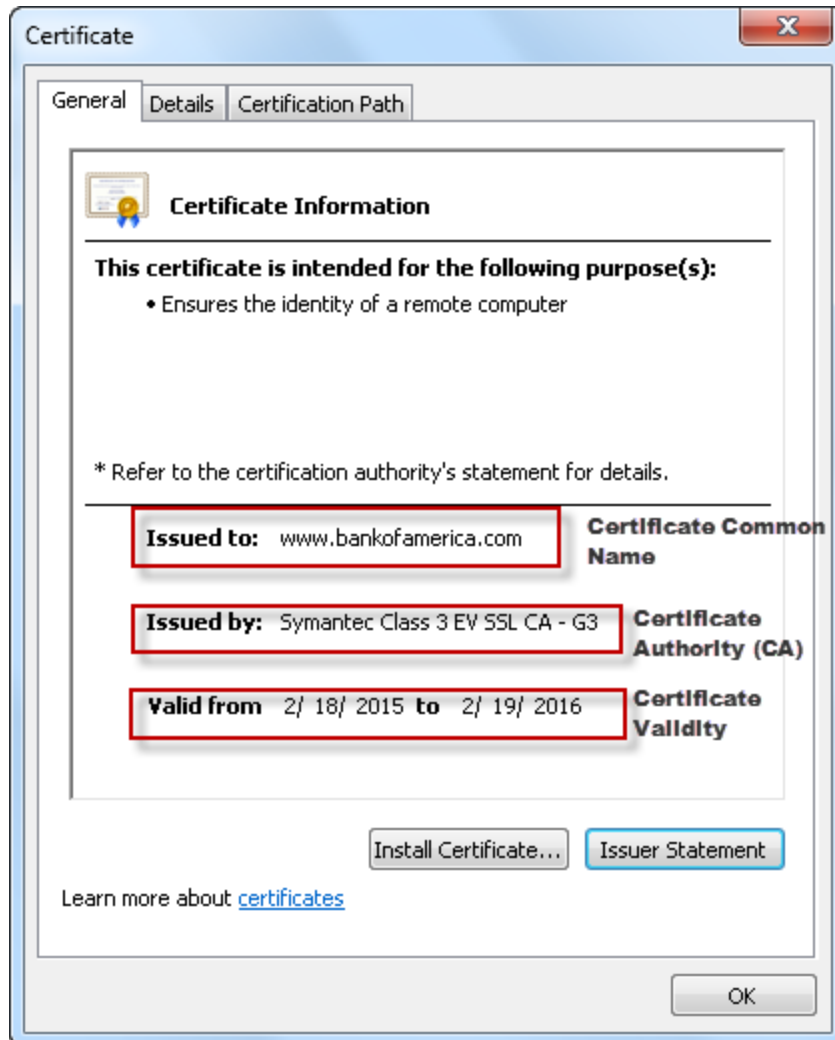
### To summarize:

1. Server generates a private key and a public key
2. Server generates and submits a CSR (Certificate signing request) to a CA (Certificate authority such as Symantec)
3. CA generates a Certificate chain containing the Server certificate (signed public key) and a certificate (or certificates) of the CA itself.
4. This certificate chain will need to be installed in the Server application

## Taking a deeper look at a SSL certificate

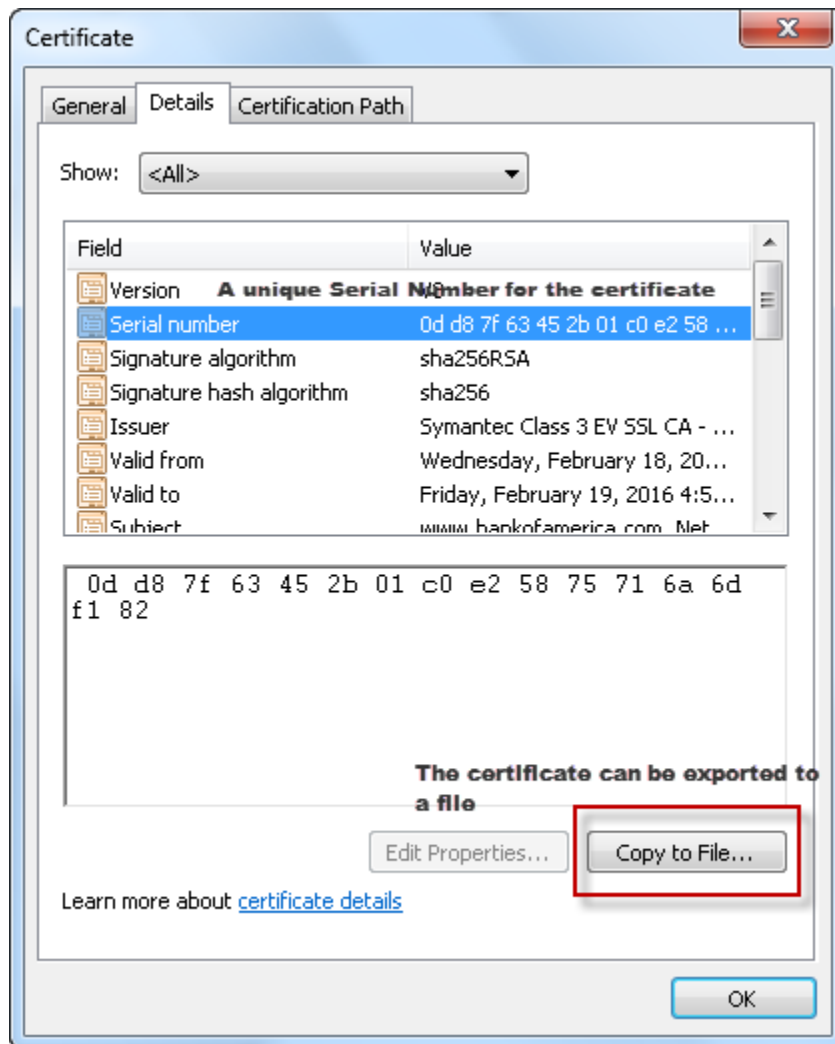


Note: Click on the 'lock' icon that appears on the address bar (Internet explorer) to bring up the website identification window and then click on 'view certificates'



Click on the 'details' tab

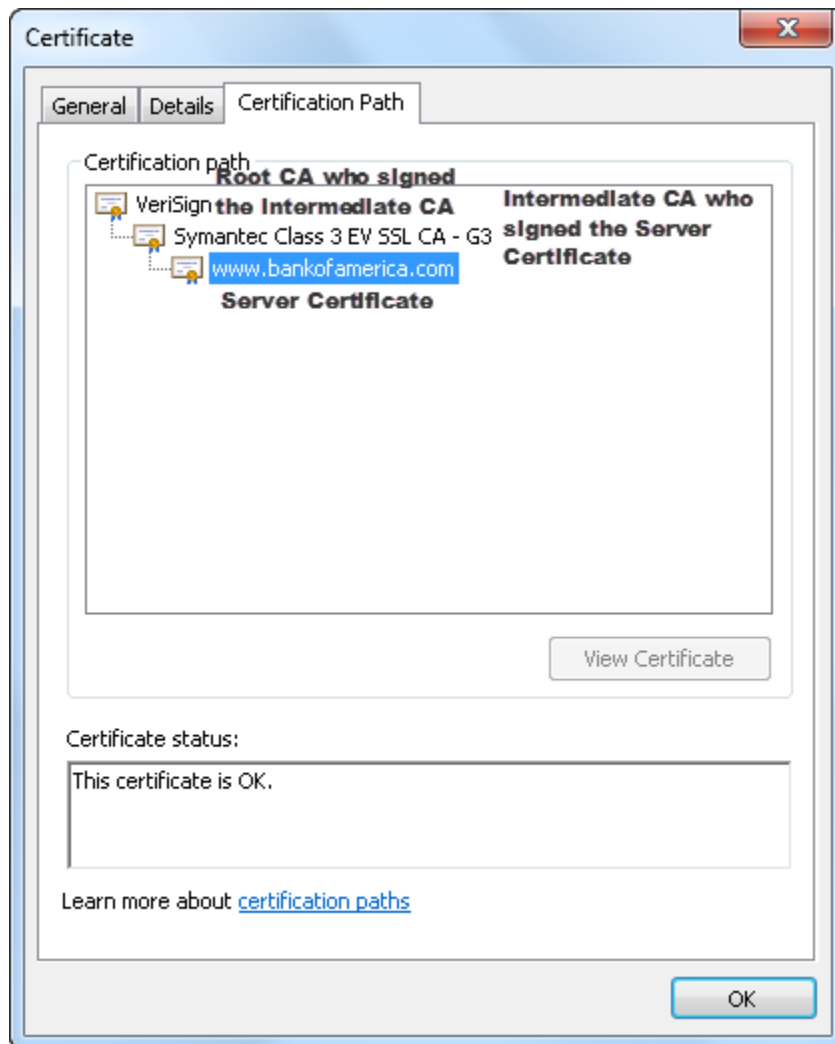




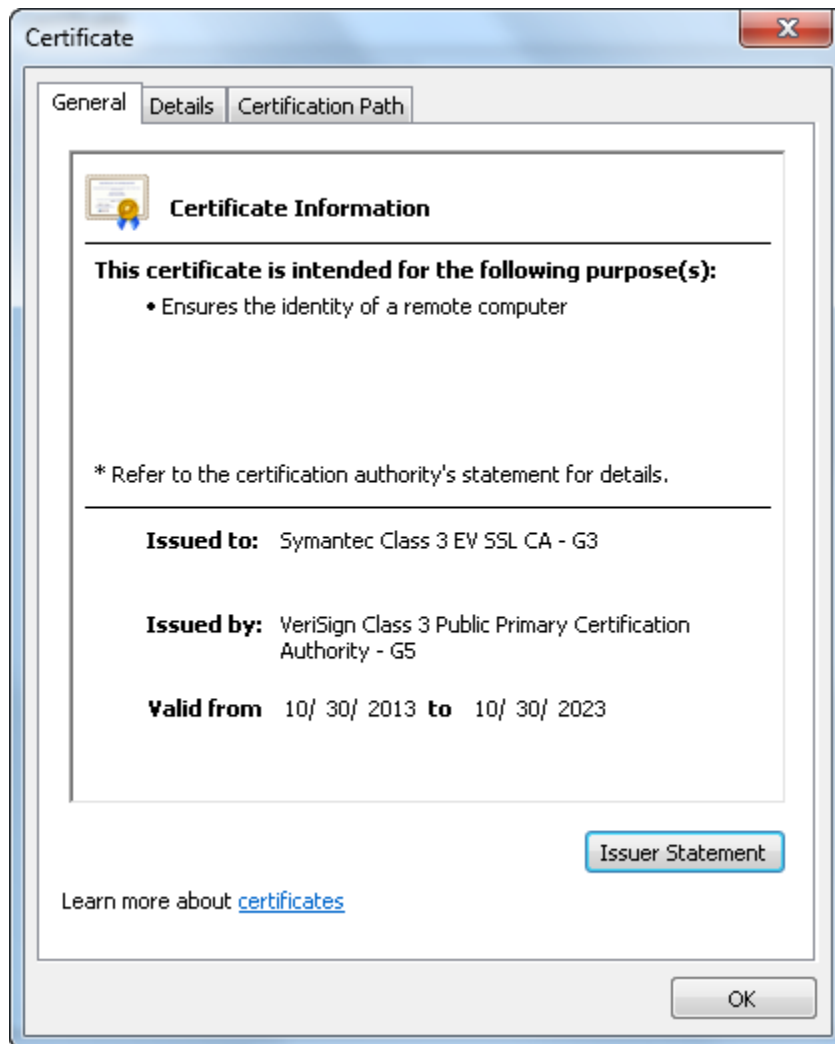
Every certificate has a unique serial number.

You can export the certificate to a file.

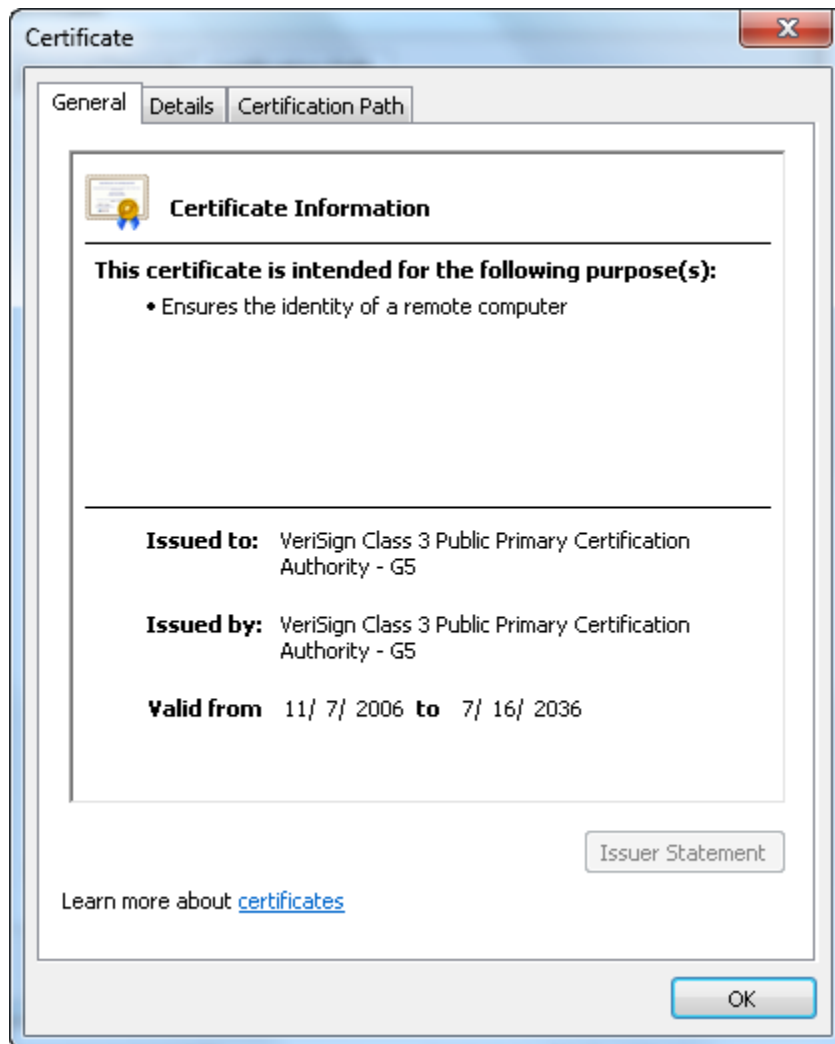
Click on the 'certification path' (This is where the certificate chain is seen)



Double clicking on either Intermediate CA (Symantec Class 3 EV SSL CA –G3) or the root CA (Verisign) will bring up those certificates in its own certificate window. For example double clicking on the intermediate CA:

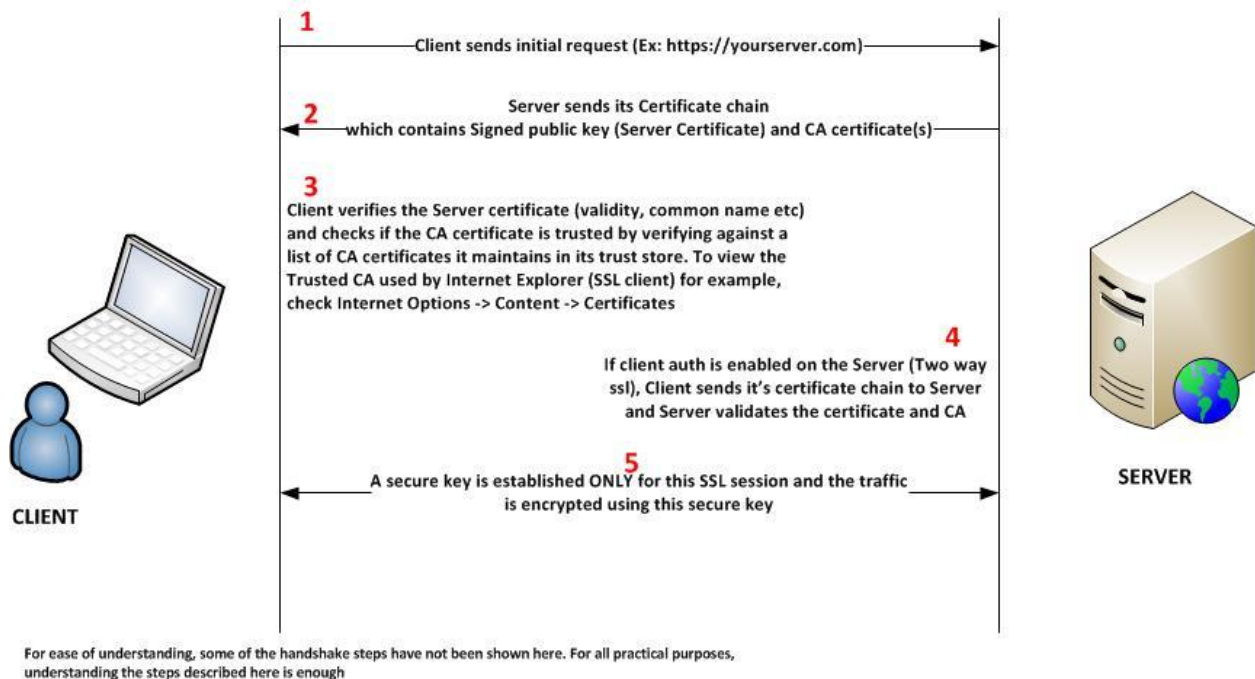


Double clicking on the root CA:



When the client access the ssl enabled Server, the server sends the certificate chain (**note, not the private key**) which contains the public key (in this case a signed public key) to the client. The client checks the Server certificate and the CA certificates. **The client must trust the CA certificate in order to continue the communication.** The client maintains a Trusted Certificate store (again, where this trusted store is located, and what type of file is this trusted store depends on the product/technology the client is using). The client ensures that the CA is trusted (by verifying the CA is in the trusted Certificate store).

## SSL Certificate Exchange



### What about self signed certificates?

If you decide to save money and try to use a self-signed certificate, meaning you sign your own certificate (getting a CA sign your certificate costs), you can do so. **Self signed certificates are NO less secure than CA signed certificate**, but there is one caveat – **maintenance**.

Remember the client needs to trust the CA. It does it by verifying the CA cert against its own list of trusted CAs. So, if you use a self-signed certificate, you must provide your CA cert (in this case, your server certificate) to each of your clients and you must make sure they install it in their trust store. And imagine if you make change (such as renewing your certificate, or go for a higher strength certificate). You must ensure all your clients install the updated certificate in their trust store. It is a management hassle. If you go with a popular CA (such as Symantec, most browsers are likely to have them already in their trust store.

## Section 2: Troubleshooting SSL communication

### Step 0: Determine if the communication really involves SSL.

This means verifying the URL being used is 'https' and not http. For example,

<https://www.yourserver.com>

In Non-Web Applications (that does not involve a webserver or http), this may not be straight forward. You will have to look at the port number and find out if the port is SSL enabled (this depends on the server application)

For example, consider the following JDBC URL of a Database Connection Pool used by a JEE Application Server

```
jdbc:sqlserver://myDBServer;DatabaseName=myDB;encrypt=true;
```

The URL above expects a SSL enabled DB server to connect to.

### Step 1: Determine the client and server

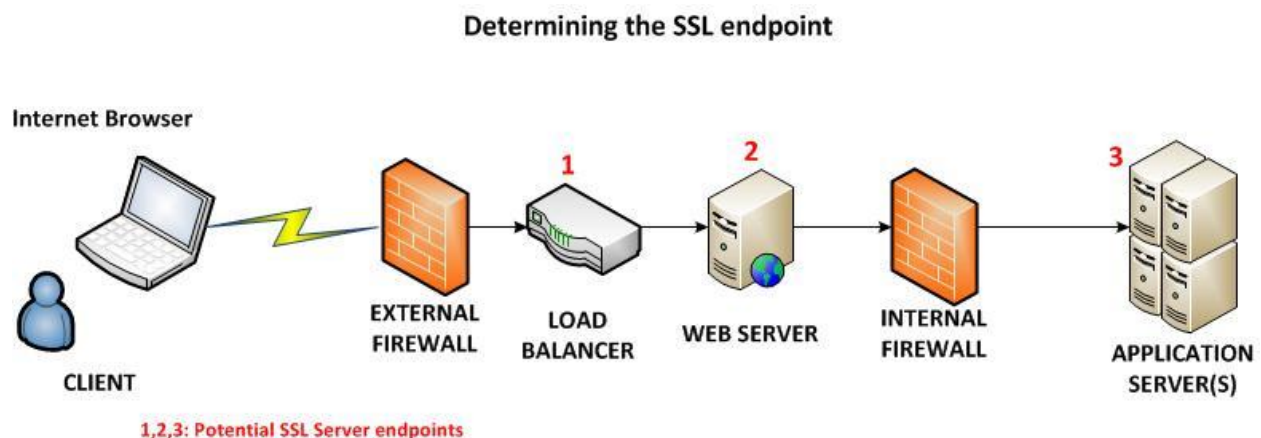
This information is more crucial than you might think. Clearly determine the following

- Where is the SSL call originating from (Client)?** Is it a browser? Is it a Server application talking to a remote Web Service/Application?
- What is the endpoint to which the client is trying to connect to (Server)?** In a typical web application, browser is the client and the Web/Application Server (or load balancer) is the server. Obtain the URL that the client is using to access the web application.

#### Notes on determining the Server

Sometimes the server is not easily identifiable.

Consider the following diagram:



The SSL endpoint could be the Load Balancer, Web Server or the Application Server. You must consult with appropriate support teams to determine the SSL server endpoint as required.

### Step 3: Identify the Error

With ssl issues, when you identify the error exactly, half the battle is already won. But identifying the error can be ironically harder in some situations. I have listed below the most common error conditions.

#### Server not reachable

This issue is not really an SSL issue but it is one of the first things you should check.

Symptoms:

- a. For Web Applications accessed via browser a 'page cannot be displayed' is seen
- b. Connection timeout error messages in the Application log files (on the client side). In a java application, the error could be as shown below:

*java.io.IOException: Connection timed out*

- c. Cannot ping the server (use this symptom only if you know for sure that ping should normally work, as many firewalls don't allow ping)
- d. The Server name is not resolvable (i.e There is no DNS entry for the server you are trying to access using SSL). Do a 'nslookup' to confirm the name resolves to an IP

nslookup <your server name>

#### The Server SSL certificate is NOT trusted by the client ← *this is the most common error with SSL handshakes.*

Recall the SSL Certificate Exchange discussion earlier. When the client initiates the SSL Connection (for example <https://www.yourserver.com>) , the server sends the Certificate chain to the client and the client verifies the CA (Certificate authority) certificate against its own SSL Trusted CAs (this is mostly a keystore). If a match is not found, the connection is rejected. In a Java application using JSSE (Java secure sockets extension), you may see error like the following (on the client side)

**Caused by:**

**sun.security.provider.certpath.SunCertPathBuilderException:  
unable to**

**find valid certification path to requested target**

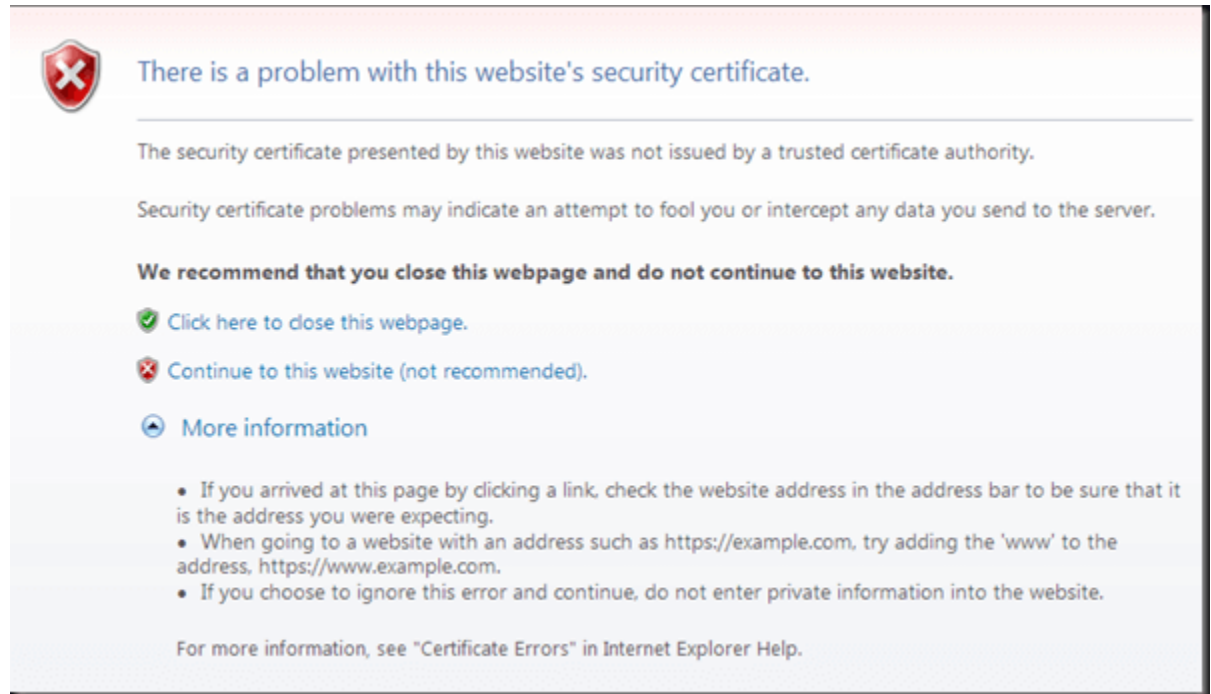
**at**

**sun.security.provider.certpath.SunCertPathBuilder.engineBuild(Unknown**

**Source)**

**at java.security.cert.CertPathBuilder.build(Unknown Source)**

For a Web Application accessed through browser, you see a warning in your browser as shown below

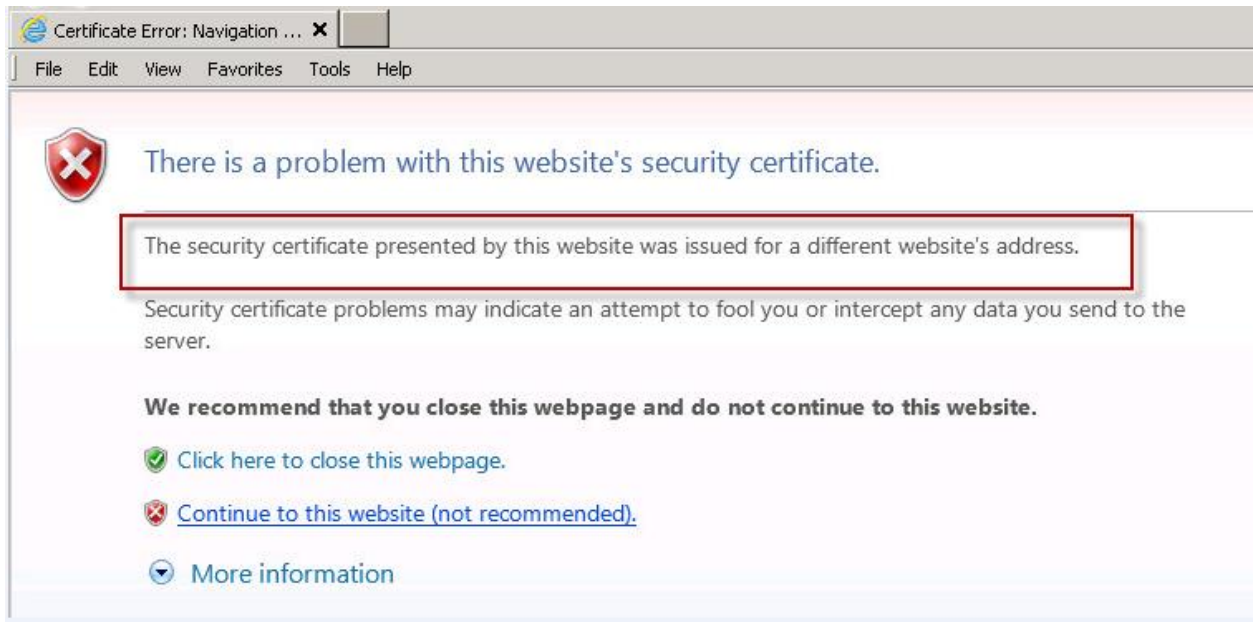


The error above most probably means your browser does not have the CA certificate of the server you are trying to access. We will see how to fix this kind of issues in section 3.

### **Name of the site and the name on the SSL Certificate do not match**

As part of SSL protocol, the client must check if the url you tried to access matches with the SSL Certificate common name (the name to which the SSL certificate was issued). This is one of the primary ways SSL ensures that you are talking to the legitimate Server application and not a bogus one. If you are using a browser as the client, you will get prompted about the problem. If you want to find out what is the hostname being used in the SSL certificate, you will have to 'continue' after the warning and pull up the certificate.





Continue to the website (it is not recommended but if you need to see the certificate common name, it is the fastest way)

### Certificate has expired

This means the validity on the SSL certificate installed on the Server has expired. SSL certificates are issued for certain time frame (1,2 or 3 years typically).



### Cannot negotiate cipher spec

During the SSL handshake both Client and Server negotiate on the Cipher Spec.

What is Cipher Spec? It is the algorithm used to encrypt the data. There are various algorithms supported by SSL. The important thing is there has to be a common cipher spec that both client and server understand. The list of supported Algorithms depends on the platform or product

(and their various versions). The errors related to cipher suites may not be apparent and verbose logs (or tcpdump) will need to be analyzed to get better insight.

### Cannot handle 2048 bit public key and SHA-256 signature algorithm

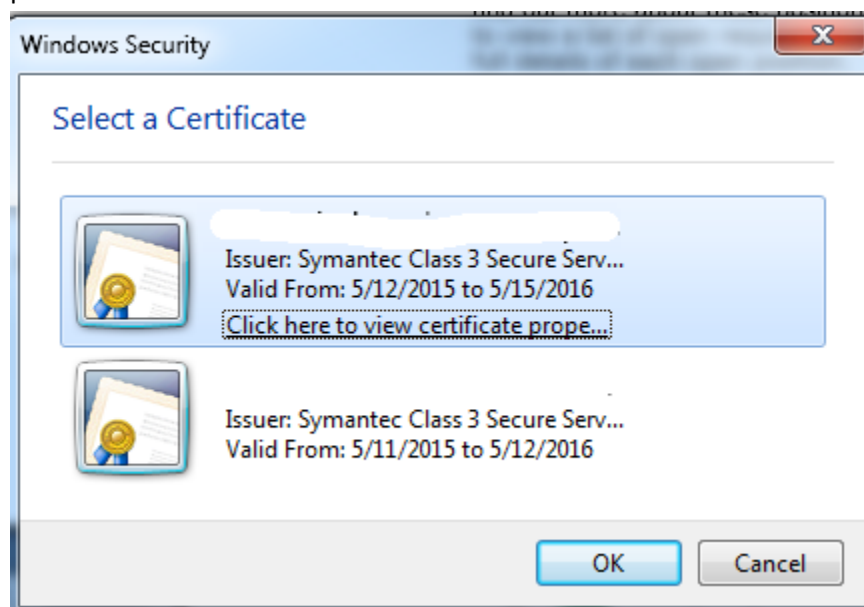
2048 bit public key and SHA256 signature algorithm are the widely accepted strongest standard as of 2015. However some old browsers and frameworks may not support this. If that is the case the SSL client will not be able to successfully connect to the server.

Like cipher spec issues, this error may not be apparent either. So, verbose logs or tcpdump will need to be obtained to determine the issue.

### Server requires client auth but the client does NOT have a valid certificate to send to

When you access a secure site that is enabling with 'Client authentication', the client (browser or the application that is initiating the connection) is required to present a valid SSL certificate that the server trusts. If the client does not present a certificate, the server will reject the connection.

When you access a client-auth enabled web site from your browser, you may be prompted to present a valid certificate as follows:



## Step 4: Fix the error

### Fixing Trust issues

The most common SSL Handshake issue is client not trusting the server certificate because the list of trusted CA that the client maintains does not include the CA that signed the certificate.

**By far the most important step in fixing this error is identifying the location of the list of trusted CA that the client maintains.** When the client is a browser it relatively easy .As a matter of fact, when you access a website who CA is not trusted by the browser, the browser prompts you if you want to trust the Server certificate. If you proceed, the CA of the Server certificate will be permanently added to the trust store. You never really have to know the location of the trust store in case of web browser client.

But what about SSL clients that are NOT browsers (i.e other Software application such as J2EE application) ? Now this need some work. The location of the list of trusted CAs depends on the framework/product being used. Further the application code can specific its own CA trust store at code level. So, some application specific knowledge is required. The following pointers will be helpful.

For a Java application, the default way of maintaining the CA certs is through a Java Key Store (JKS) file named **cacerts** that is found in jdk home\jre\lib\security. This location can be overridden by the following java command line parameter.

**-Djavax.net.ssl.trustStore=<path to trust store file> -  
Djavax.net.ssl.trustStorePassword=<password>**

The default password for the cacerts file is 'changeit'.

For a commercial J2EE Application server (such as Weblogic or WebSphere), the location of the trusted store file is specific to the implementation.

Once you have identified the trust store, it is a matter of obtaining and adding the CA certificates (of the Server you are trying to connect) to the trusted store. You could use varieties of tools to manage the trust store. Popular ones are Java 'keytool', opensource 'portecle' and IBM ikeyman.

After updating the trust store, generally you will have to recycle the client application.

To summarize:

- a. Identify the trust store used by the application
- b. Obtain and add the CA certificates of the remote Server into the trusted store
- c. Restart the client application

### Fixing Cipher Spec related issues

Both SSL client and Server need to negotiate a common Cipher Spec (algorithm used for encryption). In the light of recent attacks such as Poodle, there is an increased pressure on the IT departments worldwide to use ONLY strong ciphers.

When the negotiation of a common cipher spec fails, either server or client will issue a SSL Alert and close the connection. The only way to fix this error is to ensure there is at least one common Cipher suite configured between the client and the server.

The list of supported cipher suites depends on the platform/product.

Note: For java application, you can easily enable verbose SSL logs by the following command line parameter

**-Djavax.net.debug**

## Section 3: Using Tools and Commands to manage SSL keys.

SSL Keys (private and public) and certificates (signed public key) and CA certificates (Certificate authority certificates) are generally managed in files called 'keystores'.

There are various types of keystores. The most popular ones are

JKS – Java Key store

P12 or PKCS12 or PKCS#12 – OpenSSL style (not java specific)

BKS – Bouncy castle provider (primarily used in android)

JCEKS – Keystore provided by JCE (Java Cryptography Extension)

The two tools that are widely used in managing SSL keys and certificates are,

- a. Keytool that comes with Java
- b. Openssl

There are few open source GUI based tools (such as Portecle) as well. But the power of the above two tools cannot be undermined. Let us look at the most common and useful usage of these two tools

### Keytool

**Keytool is located in <jdk home>/bin**

#### Create a new keystore

Creating a new keystore involves creating at least one keypair (private and public key)

```
keytool -genkeypair -v -alias myserver -keyalg RSA -keysize  
2048 -sigalg SHA256WITHRSA -dname "CN=myserver OU=myorg O=myorg  
C=US" -keystore mykeystore
```

```
Enter keystore password:
```

```
Re-enter new password:
```

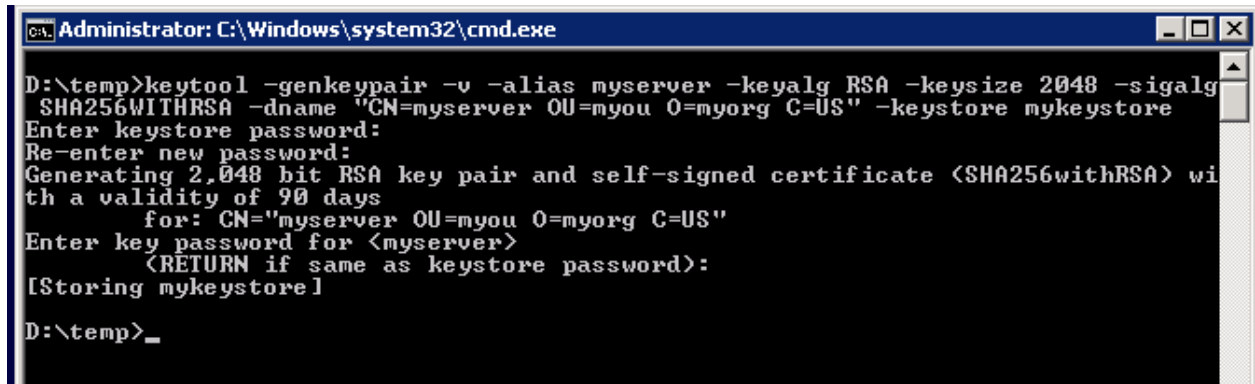
```
Generating 2,048 bit RSA key pair and self-signed certificate  
(SHA256withRSA) with a validity of 90 days
```

```
for: CN="myserver OU=myorg O=myorg C=US"
```

```
Enter key password for <myserver>
```

```
(RETURN if same as keystore password):
```

```
[Storing mykeystore]
```



```
Administrator: C:\Windows\system32\cmd.exe
D:\temp>keytool -genkeypair -v -alias myserver -keyalg RSA -keysize 2048 -sigalg
SHA256WITHRSA -dname "CN=myserver OU=myou O=myorg C=US" -keystore mykeystore
Enter keystore password:
Re-enter new password:
Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) wi
th a validity of 90 days
    for: CN="myserver OU=myou O=myorg C=US"
Enter key password for <myserver>
    <RETURN if same as keystore password>:
[Storing mykeystore]
D:\temp>_
```

Explanation of the command options:

- genkeypair: Creates a private and public key pair. The public key will be created as a self-signed certificate
- v: Verbose output as the command executes
- alias: Alias for this new key
- keyalg: Algorithm to use for the key. RSA is preferred.
- keysize: Size of the key. 2048 and above are preferred
- sigalg: Algorithm used for the signature. SHA256WITHRSA preferred
- dname: Distinguished name for the certificate. It should at least have a CN (common name), OU (Organizational Unit), O (Organization), C (country)
- keystore: name of the keystore file (fully qualified path and file name. When no path is given, file is created in the current directory)

When prompted, enter a strong password for the keystore. When prompted for the key password, press enter to accept the same password.

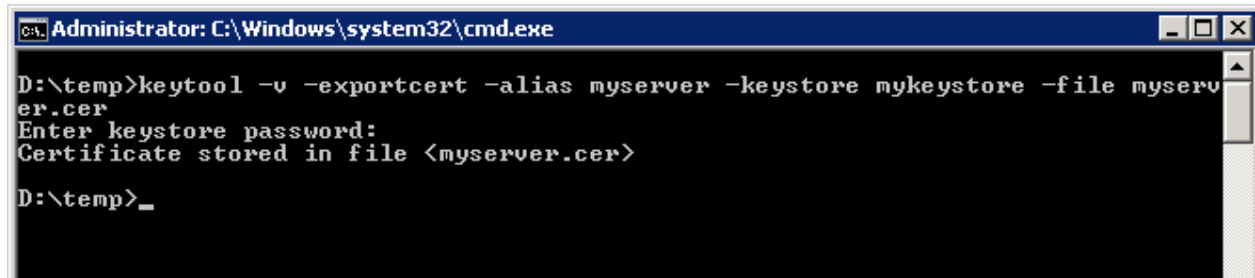
Note: There is nothing stopping you from giving a different password. However, several products/frameworks, expect the keystore password and key password to be the same. Otherwise loading of the key/certificate breaks

### List items from a keystore

```
Keytool -list -v -keystore <keysotre file> -storepass
<keystore password>
```

### Export a certificate from keystore

```
keytool -v -exportcert -alias myserver -keystore mykeystore -
file myserver.cer
Enter keystore password:
Certificate stored in file <myserver.cer>
```



```
Administrator: C:\Windows\system32\cmd.exe

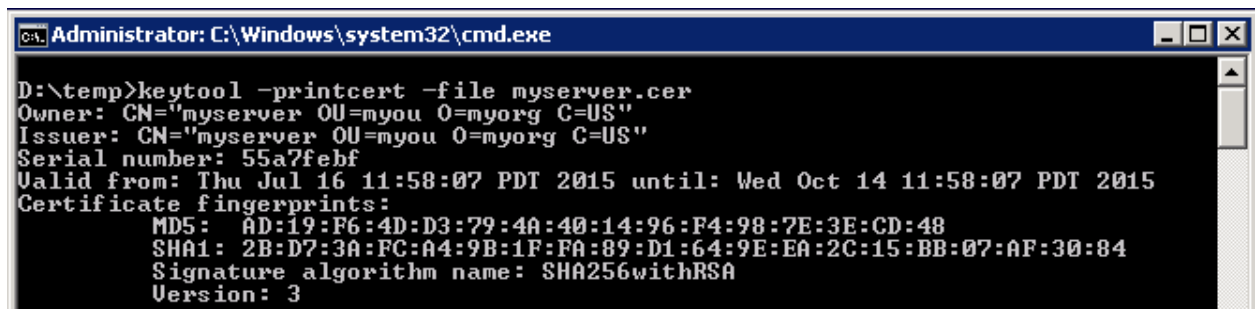
D:\temp>keytool -v -exportcert -alias myserver -keystore mykeystore -file myserver.cer
Enter keystore password:
Certificate stored in file <myserver.cer>

D:\temp>_
```

Note: The exported certificate will be in binary format. If you want the certificate in PEM format, just add the option “-rfc” to the command

### Print a certificate from a certificate file

```
keytool -printcert -file myserver.cer
```



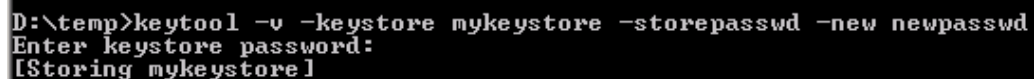
```
Administrator: C:\Windows\system32\cmd.exe

D:\temp>keytool -printcert -file myserver.cer
Owner: CN="myserver OU=myou O=myorg C=US"
Issuer: CN="myserver OU=myou O=myorg C=US"
Serial number: 55a7febf
Valid from: Thu Jul 16 11:58:07 PDT 2015 until: Wed Oct 14 11:58:07 PDT 2015
Certificate fingerprints:
    MD5: AD:19:F6:4D:D3:79:4A:40:14:96:F4:98:7E:3E:CD:48
    SHA1: 2B:D7:3A:FC:A4:9B:1F:FA:89:D1:64:9E:EA:2C:15:BB:07:AF:30:84
Signature algorithm name: SHA256withRSA
Version: 3
```

Note: You can also copy the cert file to a Windows system, rename the file with an extension ‘.cer’ and simply double click the file. The certificate will open in a certificate window.

### Change password of a keystore

```
keytool -v -keystore mykeystore -storepasswd -new newpasswd
```

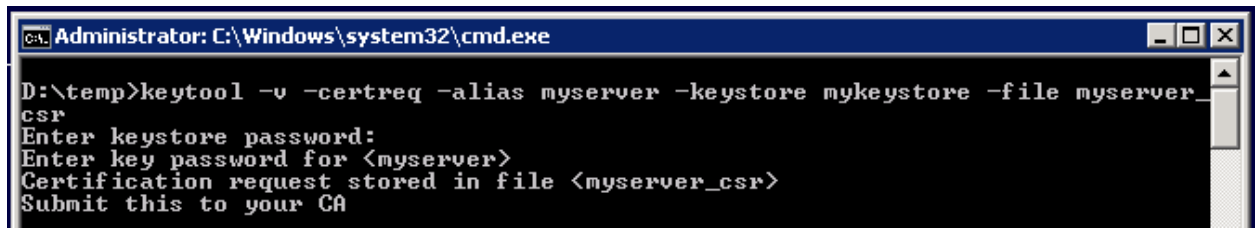


```
D:\temp>keytool -v -keystore mykeystore -storepasswd -new newpasswd
Enter keystore password:
[Storing mykeystore]
```

### Create a certificate signing request:

When you created the key pair, keytool created a self-signed certificate for you. If you prefer to have a CA (Certificate Authority) sign certificate, you need to create a CSR (Certificate Signing Request) and submit it to a CA.

```
keytool -v -certreq -alias myserver -keystore mykeystore -file  
myserver_csr
```



```
Administrator: C:\Windows\system32\cmd.exe  
D:\temp>keytool -v -certreq -alias myserver -keystore mykeystore -file myserver_  
csr  
Enter keystore password:  
Enter key password for <myserver>  
Certification request stored in file <myserver_csr>  
Submit this to your CA
```

The CSR content will look something like below:



```
D:\temp>type myserver_csr  
-----BEGIN NEW CERTIFICATE REQUEST-----  
MIICbTCCAUAUQAwwKDEmMCQGA1UEAxMdbXJ2ZXIgdT1U9bXlvdSBPPW15b3JnIEM9UUMwggEi  
MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCnLmQ7xY5dxggLJ+exkbfaiZa1nAUchsvbU  
8Lp2xXUnYyWZKK+19gOI kXUY39SA+w6+KaedHPs4HXtfWEo/9WkeIplikEBvg4HHjzhXC3K181LI  
/UTWtYb7j5ULy9CR8BNXFego18wtOEPCrkDgz+F2HyDi9wU9+5U8W3uKfuOUfgZS2J04ISF1Y6ny  
0MMmqo4cnJYrF+xfRtPOa+9uWs7UnuNyLDxr5SpOmWfECTNhpw3Zto26Px7+5IGrzIP4yMAxv1q2  
FffYEDX7meukb4WAcg+0sWokgizgv0Ns9MuuEr7CR7qHh11PRy0GP6bQ9bzGqbCzB7z/Xxur8LcT  
AgMBAAAGgADANBgkqhkiG9w0BAQUFAAOCAQEA1174ay0ow+pZNqbPloBpqBgy+stDy/rZ5W+/ZzLw  
GUUXhn3ycqXgtAdwgL5RLNu0xIJFF+1k6qvQq77WnDHAbk8x8mzW7Jz3fxxloJCw4dpqh1Khf4QW  
ywZCHBe12ZDJPHGJrOUVE1Sd2C06Qz48PsMHnL7Qadml41TKUkEmQ4foiqPu8WinljC2gERQiMpV  
IewLg/0N6o7nU1/Pseh6YMjQ031S8wae300pMShrsJBC5eirSYiGtICI7vvzWcYgoCQikn2SQWc  
bx76YeRbrWLsmnMz0/LdBQrM4crZq5FEY07xv0sOdKTAp/hHmoY9zWcWqbgMgX8Ob6LKKjOn4w==  
-----END NEW CERTIFICATE REQUEST-----
```

If the private key password is different from keystore password, you will be prompted for private key password.

### Receive Certificate from CA

Once you submit your CSR, the CA generally provides a certificate or certificate chain. You use the `importcert` to receive the certificate

```
keytool -v -importcert -alias myserver -keystore mykeystore -  
file <CA reply file>
```

### Import a keystore into a destination keystore

You can import specific aliases or the entire keystore to a destination keystore. If an entry being imported is protected with password, you need to provide the password.

The source keystore could be a PFX file with private key

```
keytool -v -importkeystore -srcstoretype <srcstoretype> -  
deststoretype <deststoretype> -srcstorepass <srcstorepass> -  
deststorepass <deststorepass> -srcalias <srcalias> -destalias  
<destalias>
```



## Import a trusted cert

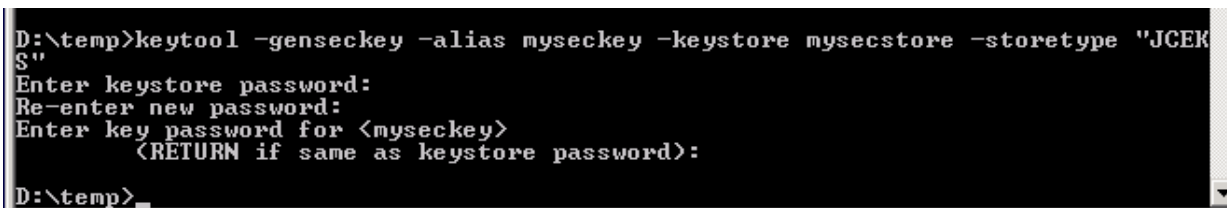
```
keytool -v -importcert -alias myserver -keystore mykeystore -  
file <cert file>
```

As you can see, you use 'importcert' command for both importing a trusted certificate as well as importing a CSR reply. Keytool adds the certificate according to the alias entered.

## Create a secret key

This is not common. But if you use JCEKS, you will come across secret keys. In secret key cryptography both client and server use the same key to encrypt/decrypt data. Note that these are NOT private keys (that are used in key pairs, which is public key cryptography)

```
keytool -genseckey -alias myseckey -keystore mysecstore -  
storetype "JCEKS"
```



```
D:\temp>keytool -genseckey -alias myseckey -keystore mysecstore -storetype "JCEK  
S"  
Enter keystore password:  
Re-enter new password:  
Enter key password for <myseckey>  
    (RETURN if same as keystore password):  
D:\temp>
```

## OpenSSL

Another popular tool to manage SSL is OpenSSL command line tool. There is virtually nothing that cannot be done by openssl.

Connect to a remote SSL Server to retrieve it's public certificate ( and to make sure you are able to connect )

```
Openssl s_client -connect <host:port>
```

Example:

```
openssl s_client -connect wells Fargo.com:443
```

```
Loading 'screen' into random state - done
```

```
CONNECTED(00000160)
```

```
depth=1 /C=US/O=VeriSign, Inc./OU=VeriSign Trust
```

```
Network/OU=Terms of use at http
```

```
s://www.verisign.com/rpa (c)10/CN=VeriSign Class 3 Secure
```

```
Server CA - G3
```

```
verify error:num=20:unable to get local issuer certificate
```

```
verify return:0
```

```
---
```

```
Certificate chain
```

```

0  s:/C=US/ST=California/L=San Francisco/O=Wells Fargo and
Company/OU=DCT-PSG-IS
G/CN=wellsfargo.com
    i:/C=US/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=Terms
of use at https:/
/www.verisign.com/rpa (c)10/CN=VeriSign Class 3 Secure Server
CA - G3
1  s:/C=US/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=Terms
of use at https:/
/www.verisign.com/rpa (c)10/CN=VeriSign Class 3 Secure Server
CA - G3
    i:/C=US/O=VeriSign, Inc./OU=VeriSign Trust Network/OU=(c)
2006 VeriSign, Inc.
- For authorized use only/CN=VeriSign Class 3 Public Primary
Certification Auth
ority - G5
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIFNTCCBB2gAwIBAgIQJ7woY+Hdlo6sFim9JxYf4zANBgkqhkiG9w0BAQUFAD
CB
tELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBjbmMuMR8wHQYDVQ
QL
ExZWZlXJpU2lnbiBUcnVzdCB0ZXR3b3JrMTswOQYDVQQLEzJUZlJtcyBvZiB1c2
Ug
YXQgaHR0cHM6Ly93d3cudmVyaXNpZ24uY29tL3JwYSAoYykyMDEvMC0GA1UEAx
Mm
VmVyaVNpZ24gQ2xhc3MgMyBTZW51cmUgU2VydMvYIENBIC0gRzMwHhcNMTQxMD
Mw
MDAwMDAwWhcNMTUxMDMxMjM1OTU5WjCBizELMAkGA1UEBhMCVVMxFzAVBgNVBA
gT
...
...

```

**Show protocol messages (sort of verbose logging)**

Like a network packet packet sniffer, openssl can show you the protocol messages going back and forth. To enable this, just add `-msg` parameter

```
openssl s_client -connect wellsfargo.com:443 -msg
```

## Create a CSR to be submitted to a CA

Like keytool, you first create the keypair and then create a CSR for it

```
openssl genrsa -aes256 -out mykey.key 2048
```

```
Loading 'screen' into random state - done
Generating RSA private key, 2048 bit long modulus
.....+++
.....+++
e is 65537 (0x10001)
Enter pass phrase for mykey.key:
Verifying - Enter pass phrase for mykey.key:
```

The resulting file is a text file that can be viewed via a text editor. The key is in PEM format (Privacy Enhanced Mail)

A note about passphrase:

A passphrase is a password to access the private key. The problem with this is most of the applications/frameworks will need you to enter the passphrase whenever the application/framework is restarted (for example a Web Server). It is unbelievably inconvenient. In Java world, this is one of the reasons the keystore password and the private key password are kept the same. Note that you are not compromising any security here as you are still protecting the private key with 'a' password (the keystore password)

## Create CSR

```
openssl req -key mykey.key -new -out myserver.csr
```

```
Enter pass phrase for mykey.key:
Loading 'screen' into random state - done
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished
Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:.
Locality Name (eg, city) []:.
Organization Name (eg, company) [Internet Widgits Pty
Ltd]:myorg
Organizational Unit Name (eg, section) []:myou
Common Name (eg, YOUR name) []:myserver
```

Email Address []:.

Please enter the following 'extra' attributes  
to be sent with your certificate request

A challenge password []:

An optional company name []:

Note: Leave the 'challenge password' blank (press enter when prompted)

Check the newly created CSR

```
Openssl req -in myserver.csr -text -noout
```

### Create a self signed certificate

```
openssl req -new -x509 -key mykey.key -out myserver-  
selfsigned.cer
```

### Converting formats

There are various formats in which the keys and certificates are stored. Openssl provides easy ways to convert them among these formats. Several conversions are possible. I'm listing one of them here.

DER to PEM conversion

```
openssl x509 -inform DER -in mycert.der -outform PEM -out  
mycert.pem
```

You can use `openssl pkcs12 -export` command to create a PKCS12 file (aka PFX file) to be deployed to a Microsoft product.

## Further Reading

### OpenSSL Documentation

<https://www.openssl.org/docs/apps/openssl.html>

### Oracle JSEE Documentation

<http://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSERefGuide.html>

## About the Author

### Karun Subramanian

#### Application Support Expert and Productivity Enthusiast

I am always in the lookout for tools and processes to increase the productivity of IT support engineers. In my years of experience as a consultant, I have helped several fortune 100 companies run their mission critical applications at their peak performance, securely.



[www.karunsubramanian.com](http://www.karunsubramanian.com)