# SSL – Secure Socket Layer

## One-way SSL and two-way (mutual) SSL

### One-Way SSL

In one way SSL, only client validates the server to ensure that it receives data from the intended server. For implementing one-way SSL, server shares its public certificate with the clients.

### Two-Way SSL

Contrary to one-way SSL; in case of two-way SSL, both client and server authenticate each other to ensure that both parties involved in the communication are trusted. Both parties share their public certificates to each other and then verification/validation is performed based on that.

## What is CA?

Certificate Authorities is and entity that issues digital certificates and validate public key. CA acts as a trusted third party: Trusted both by the sever(where app hosted) and by client(browser), client validate using certificate, through CA that server is trusted one.

An entity or person who needs a digital certificate can request one from a certificate authority; once the certificate authority verifies the applicant's identity, it generates a digital certificate for the applicant and digitally signs that certificate with the certificate authority's private key. The digital certificate can then be authenticated (for example, by a web browser) using the certificate authority's public key.

The certificate authority's root certificate should never be used directly for signing digital certificates, but rather is used to generate intermediate certificates as needed; different intermediate certificates are generated for different purposes.

Certificate authority activities start with a root certificate, which is used as the ultimate basis for trust in all certificates issued by the authority. The root certificate, along with the private key associated with that certificate, is usually treated with the highest level of security and is usually stored offline in a protected facility and may be stored on a device that is unpowered except when the certificate is needed.

The certificate authority will use that root certificate to create intermediate certificates, which are the certificates used to sign the digital certificates issued by the authority. This allows the public to trust the issued certificates, while at the same time protecting the root in instances where an intermediate certificate expires or is revoked.

### How does a Certificate Authority work?

1. The requester makes a private key and public key pair and submits an “application” called a certificate signing request (CSR) to a trusted certificate authority. The CSR has all the information about the requester that will be shown on the resulting certificate if approved.
2. The certificate authority verifies whether the information on the CSR is true. If so, the certificate authority issues and signs a certificate using its (the certificate authority’s) private key then gives it to the requester to use.
3. The requester can use the signed certificate for the appropriate security protocol:

* HTTPS for web access
* SSH for remote server access

**From certificate how I will know who is CA?**

## What is CSR?

**What is a CSR?** A CSR or Certificate Signing request is a block of encoded text that is given to a Certificate Authority when applying for an SSL Certificate. It is usually generated on the server where the certificate will be installed and contains information that will be included in the certificate such as the organization name, common name (domain name), locality, and country. It also contains the public key that will be included in the certificate. A private key is usually created at the same time that you create the CSR, making a key pair.

### What is contained in a CSR?

|  |  |  |
| --- | --- | --- |
| **Name** | **Explanation** | **Examples** |
| Common Name | The fully qualified domain name (FQDN) of your server. This must match exactly what you type in your web browser or you will receive a name mismatch error. | \*.google.com mail.google.com |
| Organization | The legal name of your organization. This should not be abbreviated and should include suffixes such as Inc, Corp, or LLC. | Google Inc. |
| Organizational Unit | The division of your organization handling the certificate. | Information Technology IT Department |
| City/Locality | The city where your organization is located. | Mountain View |
| State/County/Region | The state/region where your organization is located. This shouldn't be abbreviated. | California |
| Country | The two-letter ISO code for the country where your organization is location. | US GB |
| Email address | An email address used to contact your organization. | webmaster@google.com |
| Public Key | The public key that will go into the certificate. | The public key is created automatically |

### What does a CSR look like?

Most CSRs are created in the Base-64 encoded PEM format. This format includes the "-----BEGIN CERTIFICATE REQUEST-----" and "-----END CERTIFICATE REQUEST-----" lines at the begining and end of the CSR. A PEM format CSR can be opened in a text editor and looks like the following example:

-----BEGIN CERTIFICATE REQUEST-----  
MIIByjCCATMCAQAwgYkxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlh  
MRYwFAYDVQQHEw1Nb3VudGFpbiBWaWV3MRMwEQYDVQQKEwpHb29nbGUgSW5jMR8w  
HQYDVQQLExZJbmZvcm1hdGlvbiBUZWNobm9sb2d5MRcwFQYDVQQDEw53d3cuZ29v  
Z2xlLmNvbTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEApZtYJCHJ4VpVXHfV  
IlstQTlO4qC03hjX+ZkPyvdYd1Q4+qbAeTwXmCUKYHThVRd5aXSqlPzyIBwieMZr  
WFlRQddZ1IzXAlVRDWwAo60KecqeAXnnUK+5fXoTI/UgWshre8tJ+x/TMHaQKR/J  
cIWPhqaQhsJuzZbvAdGA80BLxdMCAwEAAaAAMA0GCSqGSIb3DQEBBQUAA4GBAIhl  
4PvFq+e7ipARgI5ZM+GZx6mpCz44DTo0JkwfRDf+BtrsaC0q68eTf2XhYOsq4fkH  
Q0uA0aVog3f5iJxCa3Hp5gxbJQ6zV6kJ0TEsuaaOhEko9sdpCoPOnRBm2i/XRD2D  
6iNh8f8z0ShGsFqjDgFHyF3o+lUyj+UC6H1QW7bn  
-----END CERTIFICATE REQUEST-----

### How do I generate a CSR and private key?

It is recommended that, generate the CSR from your server.

It can be generated by using java keytool of apache openssl.

<https://www.digicert.com/easy-csr/keytool.htm>

Graphical user interface, text, application

Description automatically generated

keytool -genkey -alias server -keyalg RSA -keysize 2048 -keystore www\_example\_com.jks -dname

"CN=www.example.com,OU=BFS, O=cognizant, L=Amstelveen, ST=Amsterdam, C=NL" && keytool -certreq -alias server -file www\_example\_com.csr -keystore www\_example\_com.jks

it will create two file one .csr and one .jks file.

.csr file is the Certificate Signing Request and .jks is the private key.

The three different levels of trusted certificates include:

**Extended Validation (EV)**

**Organization Validated (OV)**

**Domain Validated (DV)**

## What is root certificate? What is intermediate certificate?

## What is SSL certificate?

An SSL certificate is a type of digital certificate that binds the ownership details of a web server/website to public keys.

Anyone can issue SSL certificates, but those certificates would not be trusted automatically by web browsers. Certificates such as these are called self-signed. The CA has the responsibility to validate the entity behind an SSL certificate request and, upon successful validation, the ability to issue publicly trusted SSL certificates that will be accepted by web browsers. Essentially, the browser vendors rely on CAs to validate the entity behind a web site.

## What is truststore? What is keystore?

|  |  |
| --- | --- |
| **Keystore** | **Truststore** |
| Keystore stores your credential. (server or client) | Truststore stores others credentials (CA) |
| Keystore is needed when you are setting up the server side on SSL | Truststore setup is required for the successful connection at the client side |
| Client will store its private key and identify certificate on Keystore | Server will authenticate the client against the certificate stored on the server’s Truststore |
| javax.net.ssl.keyStore is used to specify Keystore | javax.net.ssl.trustStore is used to specify Truststore. |

## What is .crt, .jks, .pem, .csr?

## What is PKI

## What is certificate chain?

## What is X.509

## How SSL handshake happen?

<https://medium.com/@kasunpdh/ssl-handshake-explained-4dabb87cdce>

Step 1: Client Hello

Text

Description automatically generated

In the above log, we can see that the client hello with TLS v1.2. By this, the client notifies the server that it has the support for TLS[1] versions 1.2 and below. List of ciphers[2] that are supported by the client can also be seen from the above log. Out of this list, the server will select a cipher suite that it supports. If the list contains cipher suites that server does not recognize, support, or wish to use, the server will ignore those ciphers. If no supported cipher suites were found the server will send a failure alert and close the connection.

Step 2: Server Hello

Text, letter

Description automatically generated

Sever will select the TLS version according to the lower of that suggested by the client in the Client Hello message and the highest supported by the server. The server will also send back the cipher suite it selected from the list of ciphers presented by the client.

Along with the Server Hello, the server will also send the certificate[3] of the server with the certificate chain. The certificate chain will be validated against the certificates in the client trust store

Text

Description automatically generated

Step 3: **Server Key Exchange Message**

Step 4: Certificate Request

This is the place where one-way SSL defers from two-way SSL. In one-way SSL, the authenticity of the client is not being validated. Hence, this step is omitted in one-way SSL handshake.

During this step, the server will send a certificate request from the client with the certificate type, certificate signature algorithms and certificate authorities [6] supported by the server.

Step 5: Client Certificate

The client presents its certificate chain to the server. The certificate needs to be appropriate for the negotiated cipher suite’s key exchange algorithm, and any negotiated extensions.

Step 6: Client Key Exchange Message

This message needs to be sent by the client following the Client Certificate message. If the client certificate is not being presented (in one-way SSL), the client key exchange message should be sent after the client receives the ServerHelloDone message.

As we all know the data transferred between the server and the client in an HTTPS connection will be encrypted. Symmetric encryption[7] is being used for this purpose as the computational cost is much lower than Asymmetric encryption. In order to use symmetric encryption, there needs to be a common key between the two ends. The purpose of this message is to generate that common key between that client and the server without exposing to an outsider.

## **Is SSL/TLS encryption asymmetric or symmetric?**

**First one, then the other.**

## Asymmetric vs symmetric encryption

The handshake itself uses **asymmetric encryption** – two separate keys are used, one public and one private. Since asymmetric encryption systems have much higher overhead, they are not usable to provide full-time, real-world security. Thus, the public key is used for encryption and the private key for decryption during the handshake only, which allows the two parties to confidentially set up and exchange a newly-created “shared key”. The session itself uses this single shared key to perform **symmetric encryption**, and this is what makes a secure connection feasible in actual practice

## Keystore Explorer