## **DataDemics**

## **Understanding of SSL security**

## A Brief History of SSL and TLS

SSL and TLS are both cryptographic protocols that provide authentication and data encryption between servers, machines and applications operating over a network (e.g. a client connecting to a web server). SSL is the predecessor to TLS. Over the years, new versions of the protocols have been released to address vulnerabilities and support stronger, more secure cipher suites and algorithms.

SSL was originally developed by Netscape and first came onto the scene way back in 1995 with SSL 2.0 (1.0 was never released to the public). Version 2.0 was quickly replaced by SSL 3.0 in 1996 after a number of vulnerabilities were found. *Note: Versions 2.0 and 3.0 are sometimes written as SSLv2 and SSLv3.*

[TLS was introduced](https://tools.ietf.org/html/rfc2246) in 1999 as a new version of SSL and was based on SSL 3.0:

### Domain validation

It will only validate that the person who applies for a certificate is the owner of the domain name (or at least has some sort of access to). This kind of validation usually takes only a few minutes up to a few hours.

### Organization validation

The Certification Authority (CA) not only validates the domain’s ownership but also the owner’s identity. This means that an owner might be asked to provide personal identification documents which prove their identity. It may take several days for the validation to be completed and the certificate to be issued.

### Extended validation (EV)

This is the highest level of validation and it includes validation of domain ownership, owner identity as well as a business’s legal registration proof.

## Certificate Generation

For a Certificate Authority to issue a certificate, it first needs to have our server’s CSR, which stands for Certificate Signing Request. We first create a private key which will be used to decrypt our certificate and then we generate a CSR.

While generating the CSR we will be asked to specify the domain name as well as details about our organization like name, country and email. The following example shows a CSR.

The CSR is then submitted to the CA in order to create our certificate. When the certificate is ready it will be sent to our email in \*.crt (From CA) format and it must then be installed on the server.

What is CA (Certificate Authority ):

In [cryptography](https://en.wikipedia.org/wiki/Cryptography), a certificate authority or certification authority (CA) is an entity that issues [digital certificates](https://en.wikipedia.org/wiki/Public_key_certificate). A digital certificate certifies the ownership of a public key by the named subject of the certificate. This allows others (relying parties) to rely upon [signatures](https://en.wikipedia.org/wiki/Digital_signature) or on assertions made about the private key that corresponds to the certified public key. A CA acts as a [trusted third party](https://en.wikipedia.org/wiki/Trusted_third_party)—trusted both by the subject (owner) of the certificate and by the party relying upon the certificate. The format of these certificates is specified by the [X.509](https://en.wikipedia.org/wiki/X.509) standard.

What CA does:

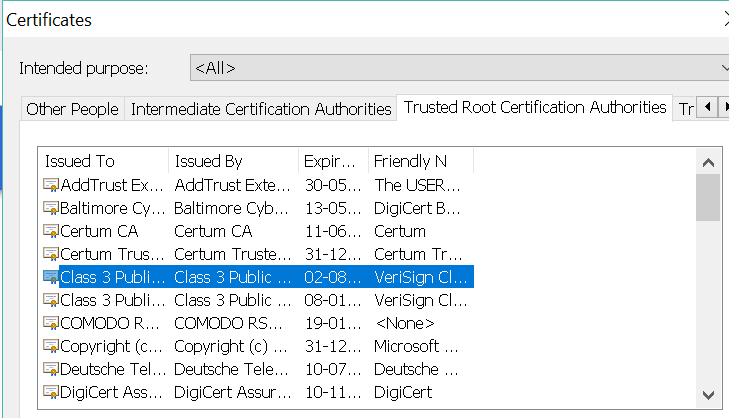
The CA receives certificate requests, validates the applications, issues the certificates, and publishes the ongoing validity status of issued certificates so anyone relying on the certificate has a good idea that the certificate is still valid.

CAs should not issue Digital Certificates directly from the root distributed to the carriers, but instead via one or more of their ICAs (Intermediate CA). This is because a CA should follow [best security practices](https://www.globalsign.com/en/ssl-information-center/ca-network-security-practices/) by minimizing the potential exposure of a Root CA to attackers.

Private CA:Any org CA

Public CA:GoDaddy,VeriSign and GlobalSign etc)

Check in browser chrome:settings-> manage certificates



PKI:

A **public key infrastructure** (**PKI**) is a set of roles, policies, and procedures needed to create, manage, distribute, use, store, and revoke [digital certificates](https://en.wikipedia.org/wiki/Digital_certificates) and manage public-key encryption. The purpose of a PKI is to facilitate the secure electronic transfer of information for a range of network activities such as e-commerce, internet banking and confidential email. It is required for activities where simple passwords are an inadequate authentication method and more rigorous proof is required to confirm the identity of the parties involved in the communication and to validate the information being transferred.

In [cryptography](https://en.wikipedia.org/wiki/Cryptography), a PKI is an arrangement that *binds* [public keys](https://en.wikipedia.org/wiki/Public_key) with respective identities of entities (like people and organizations) like my card is my id proof. The binding is established through a process of registration and issuance of certificates at and by a [certificate authority](https://en.wikipedia.org/wiki/Certificate_authority) (CA). Depending on the assurance level of the binding, this may be carried out by an automated process or under human supervision.

The PKI role that assures valid and correct registration is called a *registration authority* (RA). An RA is responsible for accepting requests for digital certificates and authenticating the entity making the request.[[2]](https://en.wikipedia.org/wiki/Public_key_infrastructure#cite_note-techotopia-2) In a [Microsoft](https://en.wikipedia.org/wiki/Microsoft)PKI, a registration authority is usually called a subordinate CA.[[3]](https://en.wikipedia.org/wiki/Public_key_infrastructure#cite_note-msdn-3)

Truststore:Truststore is used to is used to store certificates from trusted Certificate authorities(CA) which are used to verify certificate presented by Server in SSL Connection.

Keystore:

while keyStore is used to store private key and own identity certificate which program should present to other parties (Server or client) to verify its identity.

One example is setting up SSL for tomcat is server side of SSL while setting up [JDBC over SSL](http://javarevisited.blogspot.sg/2012/08/top-10-jdbc-best-practices-for-java.html) is client side of SSL connection. If you are implementing SSL on Server side you need a KeyStore to store your server certificate and private key.

Anytime a client will connect to the server, server will present its certificate stored in KeyStore and client will verify that certificate by comparing with certificates stored on its trustStore.

Diff between Keystore and Trustore:

1) Keystore is used to store your credential (server or client) while truststore is used to store others credential (Certificates from CA).

2) Keystore is needed when you are setting up server side on SSL, it is used to store server's identity certificate, which server will present to a client on the connection while trust store setup on client side must contain to make the connection work. If you browser to connect to any website over SSL it verifies certificate presented by server against its truststore.

3)You might be thinking that both keystore and truststore on client and server side if the client also needs to authenticate itself on the server. In this case, client will store its private key and identify certificate on keystore and server will authenticate the client against certificate stored on server's trust store.

4)When you install [JDK or JRE](http://javarevisited.blogspot.sg/2011/12/jre-jvm-jdk-jit-in-java-programming.html) on your machine, Java comes with its own truststore (collection of certificate from well known CA like Verisign, goDaddy, thwarte etc. you can find this file inside

JAVA\_HOME/JRE/Security/cacerts where [JAVA\_HOME](http://javarevisited.blogspot.sg/2012/02/how-to-set-javahome-environment-in.html) is your JDK Installation directory.

**Final point to remember**: keystore is used to store server's own certificate while truststore is used to store the certificate of other parties issued by CA like Verisign or goDaday or even self-signed certificates.