Certificator®

Certificator® will create a trio of certificates for DOMAIN intranet networks

Rationale

Create CA Root + Subordinate (wrongly called Intermediate) and SAN/wildcard server certificates:



As always, the CA Root is self-signed

Contents

[Presentation 4](#_Toc187667726)

[Project Location 4](#_Toc187667727)

[Features 4](#_Toc187667728)

[Requires 4](#_Toc187667729)

[How to 5](#_Toc187667730)

[Deployement 5](#_Toc187667731)

[Output 6](#_Toc187667732)

[Files 6](#_Toc187667733)

[PFX content 6](#_Toc187667734)

[Result 7](#_Toc187667735)

[Defaults 7](#_Toc187667736)

[ADDENDUMS 9](#_Toc187667737)

[Sources 9](#_Toc187667738)

[permitted\_intermediate\_section 9](#_Toc187667739)

[Best Practices 10](#_Toc187667740)

[Use 2048-Bit Private Keys 10](#_Toc187667741)

[Protect Private Keys 10](#_Toc187667742)

[Ensure Sufficient Hostname Coverage 11](#_Toc187667743)

[Use Strong Certificate Signature Algorithms 11](#_Toc187667744)

[Use Complete Certificate Chains 11](#_Toc187667745)

[Use Secure Protocols 11](#_Toc187667746)

[Use Secure Cipher Suites 12](#_Toc187667747)

[Encrypt Everything 13](#_Toc187667748)

[Eliminate Mixed Content 13](#_Toc187667749)

[Obtain Certificates from a Reliable CA 13](#_Toc187667750)

[Validate Expiration CRL and OCSP 14](#_Toc187667751)

[Lingo 15](#_Toc187667752)

[RSA vs ECC 15](#_Toc187667753)

[File extensions 15](#_Toc187667754)

[PEM 15](#_Toc187667755)

[KEY 16](#_Toc187667756)

[CSR 16](#_Toc187667757)

[CRT 16](#_Toc187667758)

[PKCS 16](#_Toc187667759)

[PFX 16](#_Toc187667760)

[CRL 16](#_Toc187667761)

[P7B 16](#_Toc187667762)

[JKS 17](#_Toc187667763)

[sha256 17](#_Toc187667764)

[WHAT IS RSA? 17](#_Toc187667765)

[WHAT IS DSA? 18](#_Toc187667766)

[WHAT IS ECC? 18](#_Toc187667767)

# Presentation

## Features

1. Optional ECC/ECDH: Elliptic Curves for all! (incompatible with java, cannot be used with Java)
2. Portable!
3. Fast and foolproof
4. AIA/CDP/OCSP “ready”!
5. Subordinate can limit IP ranges for Servers

A diagram of a root ca

Description automatically generated

This project will not generate an actual Intermediate CA but rather, a [**Subordinate**](https://www.globalsign.com/en/blog/what-is-an-intermediate-or-subordinate-certificate-authority).

## Requires

* **Openssl 1.1.1s** (included) <https://slproweb.com/products/Win32OpenSSL.html>
* **generator.cmd**
* profile templates (included)
  + openssl.TEMPLATE.root.cmd
  + openssl.TEMPLATE.intermediate.cmd
  + openssl.TEMPLATE.server.cmd
* Openssl config templates (included)
  + openssl.TEMPLATE.root.cfg
  + openssl.TEMPLATE.intermediate.cfg
  + openssl.TEMPLATE.server.cfg

## How to

1. Run it and enter Organization Root, ORG Intermediate, and DOMAIN   
   A computer screen shot of a program

   Description automatically generated
2. Edit the 3 profile templates.cmd created under .\ORG\_Root
3. Edit *permitted\_intermediate\_section* in Intermediate.cfg and Server cfg
4. Run CAgenerator again and re-enter same Organization name, Intermediate and Domain
5. You can create multiple Intermediates, each having multiple server certificates; in this case, go back to (2) to update the new profile files created and re-execute (3)

## Deployement

1. Deploy ca.YOURDOMAIN.chain.pfx via GPO to all your internal client desktops
2. Deploy USERDNSDOMAIN.chain.pfx on every servers needing SSL

# Output

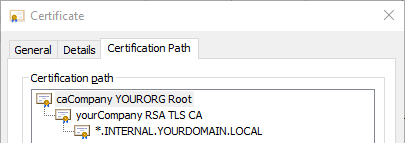
## Files

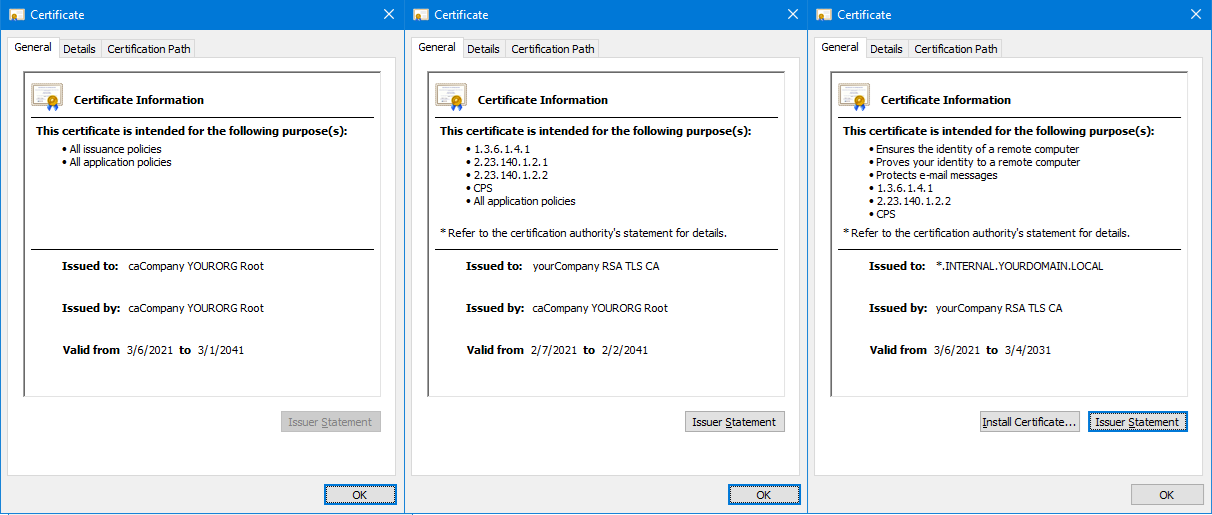


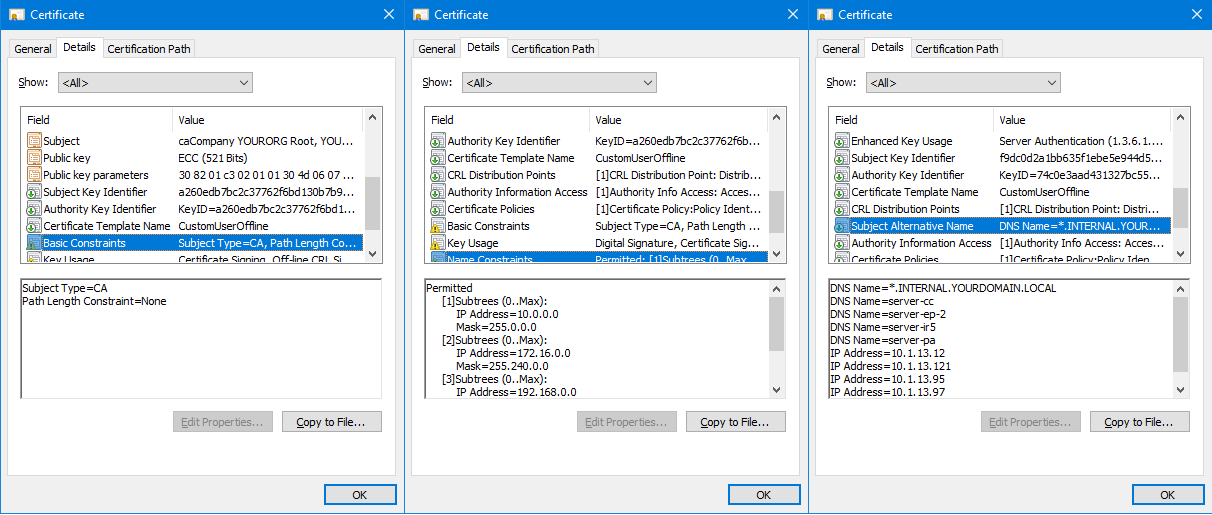
## PFX content

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| level | years | Certificate | chain | |
| Root | 20 | ca.YOURORG.crt | ca.YOURDOMAIN.chain.pfx | USERDNSDOMAIN.chain.pfx |
| Intermediate | 20 | ca.YOURDOMAIN.crt |
| Server | 10 | USERDNSDOMAIN.crt |  |
|  |  |  | Both are password protected | |

## Result







## Defaults

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Root | Intermediate | Server |
| Days | 7300 | 7300 | 3650 |
| Encryption | RSA | RSA | RSA |
| Bits | 4096 | 2048 | 2048 |
| Collision | Sha256 | sha256 | sha256 |
| Key password | root\_key\_pass | intermediate\_key\_pass | server\_key\_pass |
| PFX password | n/a | blank | 1234567890 |

# ADDENDUMS

## Sources

This document won’t cover all the sections in all 3 profile cfg files, because all the related details and sources are included in those files. However, below are the critical literatures needed to explain the various concepts used through the certificate profiles:

1. CFG FILES ARE BASED OFF <https://github.com/openssl/openssl/blob/master/apps/openssl.cnf>

The cfg files used by OpenSSL need some adaptation for Windows, starting with the extension: **cfg** instead of **cnf**

1. THIS IS YOUR BIBLE: <https://cabforum.org/extended-validation/>

Organized in 2005, the CA/Browser Forum is a voluntary group of certification authorities (CAs), vendors of Internet browser software, and suppliers of other applications that use X.509 v.3 digital certificates for SSL/TLS and code signing. They discuss and edict rules and best practices for SSL/TLS, just like the W3C for the Internet.

Other trustful resources used:

* <https://letsencrypt.org/documents/isrg-cp-v2.5/#7.1.2-certificate-extensions>
* <https://www.openssl.org/docs/manmaster/man5/x509v3_config.html>
* <https://superuser.com/questions/738612/openssl-ca-keyusage-extension#1248085>

1. THIS IS YOUR GUIDE: <https://letsencrypt.org/documents/isrg-cp-v2.5/>

Internet Security Research Group (ISRG): Certificate Policy Version 2.5 Updated October 27, 2020 Approved by the ISRG Policy Management Authority.

ISRG: Our mission is to reduce financial, technological, and educational barriers to secure communication over the Internet. ISRG was founded in May of 2013 to serve as a home for public-benefit digital infrastructure projects, the first of which was the Let's Encrypt certificate authority. ISRG's founding directors were Josh Aas and Eric Rescorla. The group's founding sponsors and partners were Mozilla, the Electronic Frontier Foundation, the University of Michigan, Cisco, and Akamai.

We will follow ISRG mostly, but not 100% as the certificates generated are used only on LAN not WEB.

1. THIS IS YOUR TESTER: <https://tls-observatory.services.mozilla.com/static/ev-checker.html>

This box provides a PEM validation for your certificate OIDs.

## permitted\_intermediate\_section

This is embedded in the Intermediate cfg and is repeated in the Server cfg. We do not offer customization via profile yet; you have to edit those files manually:

1. openssl.YOURDOMAIN.cmd
2. openssl.DOMAIN.COM.cfg

This section will limit the DNS/IP ranges that the server certificate will cover, as defined in section 7.1.5 of the ISRG CP v2.5 <https://letsencrypt.org/documents/isrg-cp-v2.5/#7.1.5-name-constraints>

# cannot permit DNS roots, or you won't be able to sign for short machine names...

# permitted;DNS.0 = .com

# permitted;DNS.1 = .lan

# permitted;DNS.2 = .local

# permitted;DNS.3 = .private

# You can only sign for IP within those ranges:

permitted;IP.0 **=** 10.0.0.0/255.0.0.0

permitted;IP.1 **=** 172.16.0.0/255.240.0.0

permitted;IP.2 **=** 192.168.0.0/255.255.0.0

# cannot use this this certificate to sign emails which domain is not the below:

permitted;email **=** yourcompany.com

# Cannot wildcard IP ranges: Error permitted subtree violation getting chain.

# excluded;IP.0 = 0.0.0.0/0.0.0.0

# excluded;IP.1 = 0:0:0:0:0:0:0:0/0:0:0:0:0:0:0:0

## Best Practices

The CA/BROWSER Forum provides best practices: <https://cabforum.org/resources/tools/>

Updated document: <https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices>

In a nutshell, those recommendations are the most critical and widely agreed on:

### Use 2048-Bit Private Keys

For most web sites, security provided by 2,048-bit RSA keys is sufficient. The RSA public key algorithm is widely supported, which makes keys of this type a safe default choice. At 2,048 bits, such keys provide about 112 bits of security. If you want more security than this, note that RSA keys don't scale very well. To get 128 bits of security, you need 3,072-bit RSA keys, which are noticeably slower.

ECDSA keys provide an alternative that offers better security and better performance. At 256 bits, ECDSA keys provide 128 bits of security. A small number of older clients don't support ECDSA, but modern clients do. It's possible to get the best of both worlds and deploy with RSA and ECDSA keys simultaneously if you don't mind the overhead of managing such a setup.

RSA 2048 bits won’t be at risk until year 2037 AD

ECDSA 384 bits won’t be at risk until year 2177 AD

### Protect Private Keys

Generate private keys on a trusted computer with sufficient entropy. Some CAs offer to generate private keys for you; run away from them.

Password-protect keys from the start to prevent compromise when they are stored in backup systems. Private key passwords don’t help much in production because a knowledgeable attacker can always retrieve the keys from process memory. There are hardware devices (called Hardware Security Modules, or HSMs) that can protect private keys even in the case of server compromise, but they are expensive and thus justifiable only for organizations with strict security requirements.

Passwords should be 10-20 characters long

### Ensure Sufficient Hostname Coverage

Even when you expect to use only one domain name, remember that you cannot control how your users arrive at the site or how others link to it. In most cases, you should ensure that the certificate works with and without the www prefix (e.g., that it works for both example.com and [*www.example.com*](http://www.example.com/)). The rule of thumb is that a secure web server should have a certificate that is valid for every DNS name configured to point to it.

If not using wildcard \*.domain.com, add all the FQDN servers in the range

Also add single machine names without domain extension

Also add their IP addresses

### Use Strong Certificate Signature Algorithms

Certificate security depends (1) on the strength of the private key that was used to sign the certificate and (2) the strength of the hashing function used in the signature. Until recently, most certificates relied on the SHA1 hashing function, which is now considered insecure. As a result, we're currently in transition to SHA256. As of January 2016, you shouldn't be able to get a SHA1 certificate from a public CA. Leaf and intermediate certificates having SHA1 hashing signature are now considered insecure by browser.

SHA1 is dead, use sha256 or sha512; these are used to alleviate uuid collisions

### Use Complete Certificate Chains

In most deployments, the server certificate alone is insufficient; two or more certificates are needed to build a complete chain of trust. A common configuration problem occurs when deploying a server with a valid certificate, but without all the necessary intermediate certificates. To avoid this situation, simply use all the certificates provided to you by your CA in the same sequence.

An invalid certificate chain effectively renders the server certificate invalid and results in browser warnings. In practice, this problem is sometimes difficult to diagnose because some browsers can reconstruct incomplete chains and some can’t. All browsers tend to cache and reuse intermediate certificates.

Always install PFX chains where they should be, use the cmd scripts provided for that purpose

### Use Secure Protocols

There are six protocols in the SSL/TLS family: SSL v2, SSL v3, TLS v1.0, TLS v1.1, TLS v1.2, and TLS v1.3:

* SSL v2 is insecure and must not be used. This protocol version is so bad that it can be used to attack RSA keys and sites with the same name even if they are on an entirely different servers (the DROWN attack).
* SSL v3 is insecure when used with HTTP (the SSLv3 POODLE attack) and weak when used with other protocols. It’s also obsolete and shouldn’t be used.
* TLS v1.0 and TLS v1.1 are legacy protocol that shouldn't be used, but it's typically still necessary in practice. Its major weakness (BEAST) has been mitigated in modern browsers, but other problems remain. TLS v1.0 has been deprecated by PCI DSS. Similarly, TLS v1.0 and TLS v1.1 has been deprecated in January 2020 by modern browsers. Check the SSL Labs blog [link](https://blog.qualys.com/ssllabs/2018/11/19/grade-change-for-tls-1-0-and-tls-1-1-protocols)
* TLS v1.2 and v1.3 are both without known security issues.

TLS v1.2 or TLS v1.3 should be your main protocol because these version offers modern authenticated encryption (also known as AEAD). If you don't support TLS v1.2 or TLS v1.3 today, your security is lacking.

In order to support older clients, you may need to continue to support TLS v1.0 and TLS v1.1 for now. However, you should plan to retire TLS v1.0 and TLS v1.1 in the near future. For example, the PCI DSS standard will require all sites that accept credit card payments to remove support for TLS v1.0 by June 2018. Similarly, modern browsers will remove the support for TLS v1.0 and TLS v1.1 by January 2020.

Benefits of using TLS v1.3:

* Improved performance i.e improved latency
* Improved security
* Removed obsolete/insecure features like cipher suites, compression etc.

Max TLS version you can use depends on the **client**: can it handle it?

### Use Secure Cipher Suites

To communicate securely, you must first ascertain that you are communicating directly with the desired party (and not through someone else who will eavesdrop) and exchanging data securely. In SSL and TLS, cipher suites define how secure communication takes place. They are composed from varying building blocks with the idea of achieving security through diversity. If one of the building blocks is found to be weak or insecure, you should be able to switch to another.

#### Notes on ECC

Source: <https://crypto.stackexchange.com/questions/70889/is-curve-p-384-equal-to-secp384r1?newreg=a86ae3c6cbfd427e94e0a8682450c2cf>

In practice, average clients only support two curves, the ones which are designated in so-called NSA Suite B: these are NIST curves P-256 and P-384 (in OpenSSL, they are designated as, respectively, "prime256v1" and "secp384r1").

If you use any other curve, then some widespread Web browsers (e.g. Internet Explorer, Firefox...) will be unable to talk to your server.

[www.google.com](http://www.google.com) uses secp384r1; that means 2 things:

* if your browser cannot access google, consider upgrading it
* if your browser cannot access google, you can use ECC for your internal domain servers
* if any of your clients/browser uses Java, you cannot use ECC

#### Which EC cypher to use?

* **secp384r1** (ASN1 OID) == **P-384** (NIST CURVE) = NIST/SECG curve over a 384 bit prime field
  + NIST-P: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>
  + SECG : <https://www.secg.org/sec2-v2.pdf>
* prime256v1 = X9.62/SECG curve over a 256 bit prime field
* Curve25519 = contender, without participation of the NSA
  + UMAC is much faster than HMAC for message authentication in TLS. see RFC <http://www.ietf.org/rfc/rfc4418.txt> or <http://fastcrypto.org/umac/>

default\_ecc\_Server curve chosen by CAgenerator®: **secp384r1**

### Encrypt Everything

The fact that encryption is optional is probably one of the biggest security problems today. We see the following problems:

* No TLS on sites that need it
* Sites that have TLS but that do not enforce it
* Sites that mix TLS and non-TLS content, sometimes even within the same page
* Sites with programming errors that subvert TLS

Although many of these problems can be mitigated if you know exactly what you’re doing, the only way to reliably protect web site communication is to enforce encryption throughout—without exception.

It’s an all-or-nothing situation.

### Eliminate Mixed Content

Mixed-content pages are those that are transmitted over TLS but include resources (e.g., JavaScript files, images, CSS files) that are not transmitted over TLS. Such pages are not secure. An active man-in-the-middle (MITM) attacker can piggyback on a single unprotected JavaScript resource, for example, and hijack the entire user session. Even if you follow the advice from the previous section and encrypt your entire web site, you might still end up retrieving some resources unencrypted from third-party web sites.

Mixed content is disabled by default in Chrome since version 78, and relegated to “insecure content” under chrome://flags/ - to scare people

### Obtain Certificates from a Reliable CA

**Services offered** At a minimum, your selected CA should provide support for both Certificate Revocation List (CRL) and Online Certificate Status Protocol (OCSP) revocation methods, with rock-solid network availability and performance. Many sites are happy with domain-validated certificates, but you also should consider if you'll ever require Extended Validation (EV) certificates. In either case, you should have a choice of public key algorithm. Most web sites use RSA today, but ECDSA may become important in the future because of its performance advantages.

CAgenerator® provides CRL lists that you can serve somewhere on your domain, over http.

WIP: we can serve them and verify the OCSP request with certutil, but what happen when you notify the expiration of the current certificate? You better have already installed the new one!

### Validate Expiration CRL and OCSP

You can validate CRL+OCSP for your certificates using this command, after having installed the chain:

certutil -verify -urlfetch INTERNAL.USERDNSDOMAIN.crt

# Lingo

## RSA vs ECC

Never use ECC if any of your client/server is using **Java**.

“Experts” constantly predict the end of 1024bit encryption but, as of 2021 it still has not been breached; using 2048 bits over 1024, your security is improved 2^1024 times!

How Does ECC Compare to RSA and DSA?

| **Symmetric Key Size (bits)** | **RSA Size (bits)** | **Elliptic Curve Key Size (bits)** |
| --- | --- | --- |
| 80 | 1024 | 160 |
| **112** | **2048** | **224** |
| 128 | 3072 | 256 |
| **192** | **7680** | **384** |
| 256 | 15360 | 521 |

<https://sectigo.com/resource-library/rsa-vs-dsa-vs-ecc-encryption>

Considerations for using ECC: it’s faster, better, and safer than RSA. However, your client must be able to use it.

**Oracle Java** is as of 2025, **still unable** **to use ECC**. If **any** of the parent certificates (CA/Int/Sub) in your chain uses it, no java server can load the server certificate and it will fail with an error such as “EC Cypher unrecognized”.

OpenJDK and Azul JDK 11 seem to be compatible, to be tested.

## File extensions

### PEM

.pem stands for [PEM, Privacy Enhanced Mail](https://tools.ietf.org/html/rfc7468); it simply indicates a base64 encoding with header and footer lines. Mail traditionally only handles text, not binary which most cryptographic data is, so some kind of encoding is required to make the contents part of a mail message itself (rather than an encoded attachment). The contents of the PEM are detailed in the header and footer line - .pem itself doesn't specify a data type - just like .xml and .html do not specify the contents of a file, they just specify a specific encoding;

### KEY

.key can be any kind of key, but usually it is the private key - OpenSSL can wrap private keys for all algorithms (RSA, DSA, EC) in a generic and standard PKCS#8 structure, but it also supports a separate 'legacy' structure for each algorithm, and both are still widely used even though the documentation has marked PKCS#8 as superior for almost 20 years; both can be stored as DER (binary) or PEM encoded, and both PEM and PKCS#8 DER can protect the key with password-based encryption or be left unencrypted;

### CSR

.csr or .req or sometimes .p10 stands for Certificate Signing Request as defined in [PKCS#10](https://tools.ietf.org/html/rfc2986); it contains information such as the public key and common name required by a Certificate Authority to create and sign a certificate for the requester, the encoding could be PEM or DER (which is a binary encoding of an ASN.1 specified structure);

### CRT

.crt or .cer stands simply for certificate, usually an [X509v3](https://tools.ietf.org/html/rfc5280) certificate, again the encoding could be PEM or DER; a certificate contains the public key, but it contains much more information (most importantly the signature by the Certificate Authority over the data and public key, of course).

### PKCS

.p8, .pkcs8 are private keys. PKCS#8 defines a way to encrypt private keys using e.g. a password. However, quite often, only the inner unencrypted PKCS#8 structure is used instead (which just defines the type of key). The inner structure can then e.g. contain a PKCS#1 formatted private key for RSA or a SEC1 one for Elliptic Curves.

[dave\_thompson: X9.62 defined commonly-used formats for EC public key and parameters (curve), but I'm pretty sure not private key although I won't spend my money to verify; for publickey and parameters SEC1 cites X9.62, and rfc5480 cites both SEC1 and X9.62, but for privatekey rfc5915 cites only SEC1 which cites nothing. Also I would say: generally not transported, but still fairly commonly PEM encoded, because many programs that use a separate key file or section, often via the OpenSSL library, require or prefer PEM: Apache httpd, nginx, haproxy, nodejs, python, wget and some curl; also Apache tomcat depending on version and option. But Java natively uses PKCS#8 DER, unencrypted.]

### PFX

.p12 or .pfx is a [PKCS#12](https://tools.ietf.org/html/rfc7292) defined key store, commonly password protected. It can contain trusted certificates, private key(s) and their certificate chain(s), but also other information such as secret keys and (very uncommonly) other personal information; .p12 is usually binary / DER encoded. PKCS#12 has lots of options plus extensions (i.e. attribute OIDs) with varying support, so it is not safe to assume that every P12 file will work in anything that uses (some) P12 files.

### CRL

.crl is a Certificate Revocation List which is defined within the [X.509v3](https://tools.ietf.org/html/rfc5280) certificate specifications, and this is usually DER encoded as well.

### P7B

.p7b or .p7c is a specialized kind of [PKCS#7/CMS](https://tools.ietf.org/html/rfc5652#section-5) message: a SignedData that doesn't contain data and isn't signed, and is used only to as a way to conveniently handle a group of certificates and/or CRLs. In particular it is often used as a way to handle the certificates which make up a 'chain' or 'bundle' as a single, well-defined unit. Other kinds of PKCS#7/CMS messages exist but are less used, and may have extensions like .p7 or .p7m, except that detached signatures, as a special case, are usually .p7s. In addition S/MIME is layered on top of CMS: S/MIME messages are really CMS messages wrapped in MIME format, and as such are usually identified by the MIME-type (aka media-type) in the message not by a file extension.

Beware that not everyone may use the same extensions - there is no official register or anything like that. You're probably better off using the POSIX file command line utility first.

### JKS

jks stands for Java Key Store. It can be used to store private keys with their certificate chains (root CA, intermediate CA's, leaf certificates or just a single self-signed certificate), certificates of other parties (usually but not necessarily CAs) to form a trust store, or both.

The JKS format (like the Java KeyStore API) is technically agnostic and can use any type of certificate for which the installed crypto-providers offer a CertificateFactory object, but in practice the only CertificateFactory implemented is X.509 (or PKIX) and the main applications in Java for keystores - code signing, S/MIME, XML/SOAP, and SSL/TLS - use only X.509. .

jks key stores are password protected, using a proprietary (and weak) cipher Sun created back during the munitions-list era, but they have been deprecated since 2017 [in favor of PKCS#12](https://bugs.openjdk.java.net/browse/JDK-8178828) with [transitional support](https://openjdk.java.net/jeps/166). For Java, JKS has the default password “changeit” and is usually found here: jdkx.y.z\lib\security\cacerts

### sha256

A picture containing graphical user interface

Description automatically generated

SHA 256 is a part of the SHA 2 family of algorithms, where SHA stands for Secure Hash Algorithm. Published in 2001, it was a joint effort between the NSA and NIST to introduce a successor to the SHA 1 family, which was slowly losing strength against brute force attacks. Hashing is the process of scrambling raw information to the extent that it cannot reproduce it back to its original form. It is irreversible and a convenient way to represent uniquely large chunks of data with a small footprint. Example use: checksum for downloaded files.

From a security perspective, sha512 is overkill: In practical terms, SHA-256 is just as secure as SHA-384 or SHA-512. We can't produce collisions in any of them with current or foreseeable technology, so the security you get is identical. Collisions occur when different chunks of data produce an exact same SHA value.

Reasons to choose SHA-256 over the longer digests: smaller packets, requiring less bandwidth, less memory, and less processing power. Also, there are likely compatibility issues, since virtually no one uses certs with SHA-384 or SHA-512, you're far more likely to run into systems that don't understand them.

### WHAT IS RSA?

Invented by Ron **R**ivest, Adi **S**hamir, and Leonard **A**dleman in 1977, RSA is an algorithm for public-key cryptography. RSA works on the basis of a [public](https://www.ssl247.co.uk/kb/glossary/p#Public%20Key) and [private](https://www.ssl247.co.uk/kb/glossary/p#Private%20Key) key.

Your public key is used to encrypt data before it's sent to the server on which the certificate is located. Every internet user attempting to connect with the site is sent the public key. The private key, generated along with the [CSR](https://www.ssl247.co.uk/kb/glossary/c#CSR), is used to decrypt the data encrypted by the public key. No one should have access to your private key - your SSL security depends on it.

1024-bit used to be the standard for key lengths, However SYMANTEC has required all their customers to upgrade to certificates with 2048-bit RSA key lengths in the end of 2014. [Get in touch to find out more about this upgrade.](https://www.ssl247.com/about/contact)

### WHAT IS DSA?

Digital Signature Algorithm, or DSA, uses a different algorithm for signing and encryption to RSA, yet provides the same level of security. It was proposed in 1991 by the National Institute of Standards and Technology (NIST) and adopted by the Federal Information Processing Standard (FIPS) in 1993. Since then it has gone under four revisions.

A DSA certificate makes it easier to keep up with government standards as it's endorsed by federal agencies - including the impending move to 2048-bit key lengths. You can even run RSA and DSA simultaneously to enhance your security further. Apache servers, for example, can run RSA and DSA certificates simultaneously on just one web server. This will benefit businesses seeking to maximise their ecosystem reach for their business correspondence.

### WHAT IS ECC?

ECC is the latest encryption method. It stands for Elliptic Curve Cryptography and promises **stronger security**, **increased performance**, yet **shorter key lengths**. This makes it ideal for the increasingly mobile world.

Just for a comparison: 256-bit ECC key equates to the same security as 3,072-bit RSA key.

The shorter key lengths require less computing power, meaning faster, secure connections to the likes of smart phones and tablets on-the-go. Plus, despite being new, Symantec's ECC roots have been in place for over 5 years, so your ECC certificate will work throughout your ecosystem. Again, ECC is FIPS-certified, like DSA, and endorsed by the National Security Agency.