

# how to obtain, deploy and verify an X.509 certificate

Created by Filipozzi, Luca, last modified by Heck, Aaron on Aug 12, 2020

- background
- how to obtain an X.509 certificate
  - requirements
  - step 1: generate the configuration file (CFG)
  - step 2: generate the private key file (KEY)
  - step 3: generate the certificate signing request (CSR)
  - step 4: obtain the end-entity certificate (CRT)
  - step 5: obtain the bundle of intermediate CA certificates
  - step 6: generate the chained certificate (for Apache httpd and nginx)
  - step 7: generate the PKCS12 archive (for Microsoft IIS or Apache Tomcat)
  - step 8: generate the Java keystore (for Apache Tomcat or Java-based apps)
  - epilog: resulting files
- how to deploy an X.509 certificate
  - Apache httpd
  - nginx
  - Microsoft IIS
  - Apache Tomcat
- how to verify an X.509 certificate
  - step 1: verify the certificate's integrity
  - step 2: verify the certificate's trust chain
  - step 3: verify the certificate's deployment with OpenSSL
  - step 4: verify the certificate's deployment with Qualys SSL Labs
- resources
  - regarding X.509 certificates
  - regarding OpenSSL
  - for Microsoft IIS administrators
  - for Apache Tomcat administrators

## background

This guide demonstrates how to obtain, deploy and verify an X.509 certificate. While written specifically in the context of a [domain-validation single-address certificate](#) for `www.obfuscate.xyz` from Entrust, the OpenSSL commands are valid for other vendors (e.g.: Gandi), for other certificate types (e.g.: wildcard), and for other validation modes (e.g.: organization-validation or extended-validation).

When following this guide, replace `www.obfuscate.xyz` with the common name for your website. Similarly, if purchasing a certificate from another vendor, replace the URLs for the bundle of intermediate CA certificates and the root certificates, as necessary.

When the single-domain certificate signing request is for `<domain>`, Entrust adds `www.<domain>` as a subject alternative name to the certificate. It is recommended to always set the common name to `<domain>` even if the website is intended to be hosted at `www.<domain>` (despite this guide using `www.obfuscate.xyz` throughout this guide).

When the wildcard certificate signing request is for `*.<domain>`, Entrust adds `<domain>` as a subject alternative name to the certificate.

## how to obtain an X.509 certificate

### requirements

To generate the required key, config, and signing request, you will need access to a system with the openssl command line tool installed on it. **This does not need to be the server or system on which the certificate will reside.** In fact, in most cases, it's best to generate the required files on a controlled, private system. The openssl command line tool is provided with OpenSSL or LibreSSL, which are installed by default on macOS, nearly all Linux/Unix/BSD systems, and available for download for Windows.

This guide focuses on openssl as it is consistent and platform agnostic. There are any number of other tools which you can use to generate keys and certificate request files for use with the UBC centralized certificate management service.

### step 1: generate the configuration file (CFG)


#### Single-Domain Certificate

Using a text editor, generate `www.obfuscate.xyz.cfg` containing the following configuration directives.

| www.obfuscate.xyz.cfg (Single Domain) |   |
|---------------------------------------|---|
| 1                                     | [ req ]                                 |
| 2                                     | distinguished_name = distinguished_name |
| 3                                     | default_md = sha256                     |
| 4                                     | prompt = no                             |
| 5                                     |   |
| 6                                     | [ distinguished_name ]                  |
| 7                                     | C = CA                                  |
| 8                                     | ST = British Columbia                   |
| 9                                     | L = Vancouver                           |
| 10                                    | O = The University of British Columbia  |
| 11                                    | CN = www.obfuscate.xyz                  |

Notes:

- Do not delete this file. If you inadvertently delete this file and are unable to restore it from an archive, then you may regenerate it manually using values extracted from the CSR or the CRT.
- For DV (domain validation) certificates, it is not necessary to specify country (C), state (ST), locality (L), organization (O) or organizational unit (OU) in the Distinguished Name. These values are only used for OV (organization validation) and EV (extended validation) certificates.

 The remainder of this document assumes that the CFG is for a single-domain certificate. Other options are available and contain recommendations for different CFG file names. The CFG file name is non-prescriptive

## Other Certificate Options

[Wildcard](#)   [Multi-Domain](#)   [Single-IP](#)   [Multi-IP](#)   [Multi-Domain / Multi-IP Combo](#)

### Wildcard Certificate

Using a text editor, generate `www.obfuscate.xyz.cfg` containing the following configuration directives.

#### `www.obfuscate.xyz.cfg` (Wildcard)

```
1 [ req ]
2 distinguished_name = distinguished_name
3 default_md = sha256
4 prompt = no
5
6 [ distinguished_name ]
7 C = CA
8 ST = British Columbia
9 L = Vancouver
10 O = The University of British Columbia
11 CN = *.obfuscate.xyz
```

Notes:

- See above notes for single-domain certificate.
- **Note the use of \* (asterisk) in the CN.**
- Consider a naming convention of `_wildcard_.obfuscate.xyz` for the files rather than `www.obfuscate.xyz`.

## step 2: generate the private key file (KEY)

ECDSA keys and certificates are recommended by Cybersecurity in alignment with the Mozilla Intermediate and Modern server side TLS standards. If you encounter compatibility challenges with ECDSA keys and certificates (typically this is with very old software), and your server supports concurrent RSA and ECDSA certs, that's the next best option. If that is not possible, go with RSA.

[ecdsa P-256 \(recommended\)](#)   [rsa 2048-bit](#)

Using the following command, generate `www.obfuscate.xyz.key` containing a P-256 ECDSA private key.

```
openssl ecparam -out www.obfuscate.xyz.key -name prime256v1 -genkey
```

Notes:

- Do not delete this file as it contains the private key that underpins the identity presented by the end-entity certificate. If you are unable to restore it from an archive, then you will need to purchase a new certificate, discarding the previously created CSR and previously obtained CRT.
- **Do not lose control of `www.obfuscate.xyz.key`.** In particular, do not store it in a location served by the web server. Further, the file permissions should read-write for user, read-only group, and none for other (i.e., 640); with user and group ownership set to 'root'.
- Please generate a new KEY when you need to renew the certificate as it is a recommended best practice by the certificate authorities.

## step 3: generate the certificate signing request (CSR)

Using the following command, generate `www.obfuscate.xyz.csr` containing a certificate signing request based on the private key and configuration files.

```
openssl req -config www.obfuscate.xyz.cfg -new -key www.obfuscate.xyz.key -out www.obfuscate.xyz.csr
```

Notes:

- Do not delete this file. If you inadvertently delete this file and are unable to restore it from an archive, then you may re-run the command to generate a new CSR, which is needed during renewals.
- While it is possible to combine the generation of a private key and the generation of a certificate signing request into a single command, it is advantageous to keep them separate for clarity.
- While it is possible to generate a certificate signing request using the default configuration, it is advantageous to capture, for future reference, the parameters used. Additionally, some CAs require the specification of subject alternative names via the CSR, which can only be accomplished via the configuration file.
- Please generate a new CSR, with the new KEY, when you need to renew the certificate as it is a recommended best practice by the certificate authorities.

## step 4: obtain the end-entity certificate (CRT)

Submit the certificate signing request using the instructions in the following article (login required): [how to submit a certificate request to the UBC Cybersecurity team](#)

Cybersecurity will provide you with the end-entity certificate issued by Entrust. Save this as `www.obfuscate.xyz.pem`.

 **Domain Validation**  
Domain control validation may be required. UBC Cybersecurity will contact you with further instructions if necessary.

Notes:

- Do not delete this file. If you inadvertently delete this file and are unable to restore it from an archive, then contact UBC Cybersecurity asking for it to be re-issued.

## Conversions

If needed, you can use the following openssl command to convert between formats.

### p7b to pem

```
openssl pkcs7 -print_certs -in www.obfuscate.xyz.p7b -out www.obfuscate.xyz.pem
```

### pem to p7b

You should have the intermediate cert first (see step 5), before performing this conversion.

```
openssl crl2pkcs7 -nocrl -certfile www.obfuscate.xyz.pem -out www.obfuscate.xyz.p7b -certfile www.obfuscate.xyz-cacerts.pem
```

## step 5: obtain the bundle of intermediate CA certificates

Obtain the bundle of intermediate CA certificates from Entrust. The command used will depend on whether you are using an ECDSA cert (recommended), or an RSA cert.

[ecdsa P-256 \(recommended\)](#)    [rsa 2048-bit](#)

ECDSA certificates make use of the Entrust L1F intermediate.

#### wget syntax

```
wget -q -O - https://entrust.com/root-certificates/entrust_l1f.cer | tr -d '\r' > www.obfuscate.xyz-cacerts.pem
```

#### curl syntax

```
curl -sL https://entrust.com/root-certificates/entrust_l1f.cer | tr -d '\r' > www.obfuscate.xyz-cacerts.pem
```

- If required, other intermediate certificates for Entrust can be downloaded from: <https://www.entrustdatacard.com/pages/root-certificates-download>

## step 6: generate the chained certificate (for Apache httpd and nginx)

Using the following command, generate `www.obfuscate.xyz-chained.pem` containing the concatenation of the end-entity certificate and the intermediate CA certificates, ordered from leaf towards root.

```
cat www.obfuscate.xyz.pem <(echo www.obfuscate.xyz-cacerts.pem) > www.obfuscate.xyz-chained.pem
```

Notes:

- The order of certificates in this file is normally from leaf (i.e.: end-entity certificate) towards root.
- This file should never contain root certificates.
- The `<(echo)` command in the middle is required because the pem file from Entrust does not contain a newline at the end of the file, causing the END and BEGIN certificate statements to merge into a single line, invalidating the chain.

## step 7: generate the PKCS12 archive (for Microsoft IIS or Apache Tomcat)

This step uses `www.obfuscate.xyz-chained.pem`, which was generated in the previous step.

Using the following command, generate `www.obfuscate.xyz-pfxpass.txt` containing a random 36-character string to be used as the passphrase for the PKCS12 archive.

```
openssl rand -base64 36 > www.obfuscate.xyz-pfxpass.txt
```

Using the following command, generate `www.obfuscate.xyz-chained.pfx` a PKCS12 archive containing the private key, the end-entity certificate, and the intermediate CA certificates.

```
openssl pkcs12 -export -inkey www.obfuscate.xyz.key -in www.obfuscate.xyz-chained.pem -out www.obfuscate.xyz-chained.pfx -passout file:www.obf
```

Notes:

- **Do not lose control of `www.obfuscate.xyz-pfxpass.txt`.** Since the PKCS12 archive contains the private key, the passphrase securing the PKCS12 archive should be held as securely as the private key itself.

## step 8: generate the Java keystore (for Apache Tomcat or Java-based apps)

This step uses `www.obfuscate.xyz-chained.pfx` and `www.obfuscate.xyz-pfxpass.txt`, which were generated in the previous step.

Using the following command, generate `www.obfuscate.xyz-jkspass.txt` containing a random 36-character string to be used as the passphrase to secure the Java keystore.

```
openssl rand -base64 36 > www.obfuscate.xyz-jkspass.txt
```

Using the following command, generate `www.obfuscate.xyz-keypass.txt` containing a random 36-character string to be used as the passphrase to secure the key stored in the Java keystore.

```
openssl rand -base64 36 > www.obfuscate.xyz-keypass.txt
```

Using the following command, generate `www.obfuscate.xyz-chained.jks`, a Java keystore containing the private key, the end-entity certificate, and the intermediate CA certificates.

```
keytool -importkeystore \
-srckeystore www.obfuscate.xyz-chained.pfx -srcstoretype pkcs12 -srcalias www.obfuscate.xyz \
-destkeystore www.obfuscate.xyz-chained.jks -deststoretype jks -destalias www.obfuscate.xyz \
-srcstorepass `cat www.obfuscate.xyz-pfxpass.txt` \
-deststorepass `cat www.obfuscate.xyz-jkspass.txt` \
-destkeypass `cat www.obfuscate.xyz-keypass.txt`
```

Notes:

- **Do not lose control of `www.obfuscate.xyz-jkspass.txt` and `www.obfuscate.xyz-keypass.txt`.** Since the Java keystore contains the private key, the passphrases securing the Java keystore should be held as securely as the private key itself.

## epilog: resulting files

Execution of the commands detailed in steps 1 through 8 will result in the generation of the following files:

- `www.obfuscate.xyz.cfg`: OpenSSL configuration (CFG)
- `www.obfuscate.xyz.key`: P-256 ECDSA private key or 2048-bit RSA private key (KEY) ⚠
- `www.obfuscate.xyz.csr`: certificate signing request (CSR)
- `www.obfuscate.xyz.pem`: end-entity certificate (CRT)
- `www.obfuscate.xyz-cacerts.pem`: bundle of intermediate CA certificates
- `www.obfuscate.xyz-chained.pem`: concatenation of end-entity and intermediate CA certificates
- `www.obfuscate.xyz-pfxpass.txt`: passphrase used to secure the PKCS12 archive ⚠
- `www.obfuscate.xyz-chained.pfx`: PKCS12 archive containing the private key, the end-entity certificate and the intermediate CA certificates
- `www.obfuscate.xyz-jkspass.txt`: passphrase used to secure the Java keystore ⚠
- `www.obfuscate.xyz-keypass.txt`: passphrase used to secure the private key within the Java keystore ⚠
- `www.obfuscate.xyz-chained.jks`: Java keystore containing the private key, the end-entity certificate and the intermediate CA certificates

Notes:

- **Do not lose control of the files marked ⚠ as they either contain or grant access to the private key.** Specifically, this means setting the file/group ownership and permission flags appropriately so that unauthorized users, locally and remotely, can not gain access to the private key.
- By following a naming convention, all the files associated with a particular certificate are kept together.
- One approach to managing these files is to store them outside of the context of the software that will use them (see next section on how to deploy an X.509 certificate). For example, these files could be kept in `/root/ssl` or `/home/sysadmin/ssl` and copied to locations needed by the software (with appropriate updates to ownership and permissions). In particular, software running on Linux machines configured with enhanced security policies, such as those employed by SELinux, will expect these files to be in specific locations.

## how to deploy an X.509 certificate

### Apache httpd

v2.4.7 or earlier    v2.4.8 or later

#### for versions up to 2.4.7 (inclusive)

Apache httpd versions up to 2.4.7 require the use of three configuration directives to specify the private key (`SSLCertificateKeyFile`), the end-entity certificate (`SSLCertificateFile`) and the bundle of intermediate CA certificates (`SSLCertificateChainFile`).

Example configuration:

```
1 <VirtualHost 192.168.1.1:443>
2     ServerName www.obfuscate.xyz
3     ServerAlias obfuscate.xyz
4
5     SSLEngine on
6     SSLCertificateFile      /path/to/www.obfuscate.xyz.pem
7     SSLCertificateKeyFile   /path/to/www.obfuscate.xyz.key
8     SSLCertificateChainFile /path/to/www.obfuscate.xyz-cacerts.pem
9     ...
10 </VirtualHost>
```

Notes:

- The configuration snippet, above, does not provide guidance regarding configuration directives regarding protocols and cipher suites.
- A common error is to use `SSLCACertificateFile` to specify the file containing the bundle of intermediate certificates. In Apache versions up to 2.4.7, use `SSLCertificateChainFile` for this purpose.

### nginx

Example configuration:

```
1 server {
2     listen 192.168.1.1:443;
```

```
4 | server_name www.obfuscate.xyz obfuscate.xyz;
5 |
6 | ssl on;
7 | ssl_certificate /path/to/www.obfuscate.xyz-chained.pem;
8 | ssl_certificate_key /path/to/www.obfuscate.xyz.key;
9 | ...
  | }
```

Notes:

- The configuration snippet, above, does not provide guidance regarding configuration directives regarding protocols and cipher suites.

Microsoft IIS

Using IIS management tools (Start -> Administrative Tools -> IIS), import the PKCS12 archive (`www.obfuscate.xyz-chained.pfx`) and edit the bindings to use the new certificate. The passphrase securing the PKCS12 archive is contained in `www.obfuscate.xyz-pfxpass.txt`.

Apache Tomcat

[pkcs12 \(recommended\)](#)   `jks`

Apache Tomcat provides support for multiple SSL/TLS implementations. The `server.xml` configuration snippet, below, assumes the use of JSSE rather than APR.

**pkcs12 (recommended)**

```
server.xml

<Connector address="192.168.1.1" port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
    maxThreads="150" SSLEnabled="true" scheme="https" secure="true" clientAuth="false" sslProtocol="TLS"
    keystoreFile="/path/to/www.obfuscate.xyz-chained.pfx"
    keystorePass="«passphrase contained in www.obfuscate.xyz-pfxpass.txt»"
    keystoreType="PKCS12"/>
```

Notes:

- The passphrase securing the pfx file is contained in `www.obfuscate.xyz-pfxpass.txt`. Use this passphrase with the `keystorePass` attribute.
- **Do not lose control of Tomcat's `server.xml`.** Since this configuration file contains the passphrases to the pfx file, which also contains the key, it should be held as securely as the private key itself.
- The configuration snippet, above, does not provide guidance regarding configuration directives regarding protocols and cipher suites.

how to verify an X.509 certificate

step 1: verify the certificate's integrity

Using the following commands, verify the integrity of the private key (KEY), the certificate signing request (CSR), and the end-entity certificate (CRT). The SHA1 hash value output by the three commands should be identical, indicating that the end-entity certificate is based on the certificate signing request over the private key.

```
openssl pkey -pubout -in www.obfuscate.xyz.key | openssl dgst -sha1 -binary | xxd -p
```

output of previous command

021d551fb04d52b18464ddb26ba95c79d94a3c3d

```
openssl req -noout -pubkey -in www.obfuscate.xyz.csr | openssl dgst -sha1 -binary | xxd -p
```

output of previous command

021d551fb04d52b18464ddb26ba95c79d94a3c3d

```
openssl x509 -noout -pubkey -in www.obfuscate.xyz.pem | openssl dgst -sha1 -binary | xxd -p
```

output of previous command

021d551fb04d52b18464ddb26ba95c79d94a3c3d

step 2: verify the certificate's trust chain

Using the following command, examine the end-entity certificate. Note that it has subject `CN=www.obfuscate.xyz` and is issued by `CN=Entrust Certification Authority - L1K`

```
openssl x509 -noout -subject -issuer -fingerprint -in www.obfuscate.xyz.pem
```

output of previous command

```
subject= /C=CA/ST=British Columbia/L=Vancouver/O=The University of British Columbia/CN=www.obfuscate.xyz
issuer= /C=US/O=Entrust, Inc./OU=See www.entrust.net/legal-terms/OU=(c) 2012 Entrust, Inc. - for authorized use only/CN=Entrust Certification
SHA1 Fingerprint=C2:1A:1D:78:9C:04:94:AF:3B:CE:FC:FB:89:B8:4B:72:12:CC:0E:62
```

Using the following command, examine the bundle of intermediate CA certificates. Note that the bundle contains just one certificate.

The certificate in the bundle has subject CN=Entrust Certification Authority - L1K and is issued by CN=Entrust Root Certification Authority - G2.

```
cat www.obfuscate.xyz-cacerts.pem | gawk 'BEGIN {RS="-----END CERTIFICATE-----\n";ORS=RS} NR==1 {print}' | openssl x509 -noout -subject -issue
```

output of previous command

```
subject= /C=US/O=Entrust, Inc./OU=See www.entrust.net/legal-terms/OU=(c) 2012 Entrust, Inc. - for authorized use only/CN=Entrust Certification
issuer= /C=US/O=Entrust, Inc./OU=See www.entrust.net/legal-terms/OU=(c) 2009 Entrust, Inc. - for authorized use only/CN=Entrust Root Certifica
SHA1 Fingerprint=F2:1C:12:F4:6C:DB:6B:2E:16:F0:9F:94:19:CD:FF:32:84:37:B2:D7
```

Using one the following commands, verify the end-entity certificate's trust chain.

For Debian and Debian-based distributions (e.g.: Ubuntu)

```
openssl verify -CApath /etc/ssl/certs -untrusted www.obfuscate.xyz-cacerts.pem www.obfuscate.xyz.pem
```

expected output of previous command

```
www.obfuscate.xyz.pem: OK
```

For RedHat and Redhat-based distributions (e.g.: CentOS):

```
openssl verify -CAfile /etc/ssl/certs/ca-bundle.trust.crt -untrusted www.obfuscate.xyz-cacerts.pem www.obfuscate.xyz.pem
```

expected output of previous command

```
www.obfuscate.xyz.pem: OK
```

If your trust store does not contain the Entrust G2 Root CA Certificate, then your trust store is not up to date. Please ask the system administrator to update the CA certificates in the trust store via the system's package management system.

Although *not recommended* since your operating system should provide one or both of these in the root store, you may install it. Contact the UBC Cybersecurity team for assistance.

### step 3: verify the certificate's deployment with OpenSSL

Using the following command, verify the deployment of the private key, end-point certificate and intermediate CA certificates.

For Debian and Debian-based distributions (e.g.: Ubuntu):

```
echo "GET /" | openssl s_client -CApath /etc/ssl/certs -connect www.obfuscate.xyz:443 -servername www.obfuscate.xyz
```

expected output of previous command (truncated)

```
...
Verify return code: 0 (ok)
---
DONE
```

For RedHat and RedHat-based distributions (e.g.: CentOS):

```
echo "GET /" | openssl s_client -CAfile /etc/ssl/certs/ca-bundle.trust.crt -connect www.obfuscate.xyz:443 -servername www.obfuscate.xyz
```

expected output of previous command (truncated)

```
...
Verify return code: 0 (ok)
---
DONE
```

## step 4: verify the certificate's deployment with Qualys SSL Labs

Use [Qualys SSL Labs](#) to obtain a comprehensive analysis of the server's configuration in addition to verifying the certificate's deployment. If you need assistance in improving your rating, contact the UBC Cybersecurity team.

## resources

### regarding X.509 certificates

- [Wikipedia - X.509](#)
- [Zytrax Survival Guides - TLS/SSL and SSL \(X.509\) Certificates](#)
- [PKI Basics - A Business Perspective](#)
- [PKI Basics - A Technical Perspective](#)
- [Understanding Certificate Path Construction](#)

### regarding OpenSSL

- [OpenSSL Command-Line HOWTO](#)
- [Ivan Ristić's OpenSSL Cookbook](#)

### for Microsoft IIS administrators

- [OpenSSL for Windows](#)
- [PuTTY - an SSH Client for Windows](#)
- [MobaXterm - an SSH Client for Windows](#)
- [WinSCP - a graphical SCP/SFTP Client for Windows](#)

### for Apache Tomcat administrators

#### version 6

- [Configuration Reference - HTTP Connector](#)
- [SSL/TLS Configuration HOW-TO](#)

#### version 7

- [Configuration Reference - HTTP Connector](#)
- [SSL/TLS Configuration HOW-TO](#)

#### version 8

- [Configuration Reference - HTTP Connector](#)
- [SSL/TLS Configuration HOW-TO](#)

[kb-how-to-article](#)

## 17 Comments



Grigoryan, Armenak

Thanks, Luca. Much needed. This instruction is the example how documentation should be done!



Filipozzi, Luca

For DNS-based domain control validation (updated per John's comment on 28 Jul 2017 ), use the following steps.

**Step 1: Generate the "MD5 hash":**

```
openssl req -in www.obfuscate.xzy.csr -outform der | openssl dgst -md5
```

**Step 2: Generate the "split SHA256 hash" (64-character string split into two 32-character string separated by a dot):**

```
openssl req -in www.obfuscate.xzy.csr -outform der | openssl dgst -sha256 -r | awk '{printf "%s.%s\n",substr($1,1,32),substr($1,33,32)}
```

**Step 3: Create CNAME record:**

```
_«MD5 hash».www.obfuscate.xyz. IN CNAME «split SHA256 hash».comodoca.com.
```

⚠ Note the underscore in front of the the «MD5 hash».



Bratlien, John

Comodo has updated the DNS-based DCV scheme to use SHA-256 and a slightly different format:

```
_<MD5 hash>.www.obfuscate.xyz. CNAME <SHA-256 hash>.comodoca.com
```

Note that a "hex (base-16) encoded SHA-256 hash will not fit in a single DNS label because it is too long. The SHA-256 hash should therefore be split into two labels, each 32 characters long."

For more detail see 'DNS CNAME-based'

<https://support.comodo.com/index.php?/comodo/Knowledgebase/Article/View/791/0/>



Filipozzi, Luca

Thanks. I've updated the instructions.



Filipozzi, Luca

I have added example openssl configuration files for multi-domain and wildcard certificates.



Filipozzi, Luca

Some users complain that their operating system, browser or Java application (Tomcat, JBoss, Oracle Wallet or other) does not trust the end-entity certificate issued by Gandi.

This is neither a Gandi nor a Comodo issue (Gandi resells Comodo certificates). This is a trust store issue.

Typically, the trust store is provided by the operating system vendor or by the browser vendor. Each vendor (Apple for macOS/iOS/Safari, Microsoft for Windows/IE, Google for Android/Chrome, Oracle for Java) has their own policies / procedures for adding/removing the CA certificates from their respective trust stores. Most GNU/Linux distributions incorporate Mozilla's trust store as the operating system trust store.

Comodo's responsibility is to work with vendors to ensure that the Comodo root certificates are included in each vendor's master trust store.

The vendors' responsibility is to manage their master trust store so that it **does include** the root certificates from trusted CAs and **does not include** the root certificates from untrusted CAs. CAs become untrusted when their operations are determined to be in contravention of the CA/B Forum's Baseline Requirements: recent examples include Symantec and StartCom/WoSign. Since each vendor has different policies/procedures, there are differences between the vendors' trust stores.

**The users' responsibility is to keep her local copy of the trust stores up-to-date. It does no good for Comodo to work with vendors, nor for the vendors to actively manage their master trust store if users never refresh their local trust stores from the vendor master trust store.**

Typically, users accomplish this local trust store update by patching their operating systems or updating their browsers to the latest version.

Where this breaks, frequently, is when an application embeds a trust store, such as with Java Runtime Environments. System administrators may remember to update the operating system's trust store but often forget/neglect to update the JRE's trust store.

**Rule of Thumb: keep your systems up to date, kids.**



Poyser, John

UBC IT needs an annual award for Best KB submission. This gets my vote.



Heck, Aaron

We've had a question come up with regards to the SHA1 certification path going EOL in 2020. I've added a small "alert" section in the pathing area to let people know the timeline of the deprecation of the SHA1 path, and the impact for users.



Heck, Aaron

In Step 5, when performing the wget, the response from Gandi will be a redirect to a different location.

Gandi is using a 302 (Temporary) redirect, instead of a 301 (Permanent) redirect to point to the new location. Assuming that Gandi knows their stuff, and is following the [standards defined in the RFC](#):

- If the redirect is a 301, then "...any future references to this resource ought to use one of the enclosed URIs". In other words, update your links and documentation.
- If the redirect is a 302, then "...the client ought to continue to use the effective request URI for future requests". In other words, no documentation change.

The documentation is intentionally not being updated to reflect this new URL.



Heck, Aaron

@Bratlien, John Pointed out that the intermediate cert he fetched from Gandi today no longer contains the SHA1 intermediate cert. Will update the page accordingly.



Sherrington, Ryan

The Tomcat docs for version 7 [say that PKCS12 is supported](#). As well, the JDK 8 version of keytool throws a warning when I try and use jks:

Warning:

The JKS keystore uses a proprietary **format**. It is recommended to migrate to PKCS12 **which** is an industry standard **format** using "**keytool**

Should I use pkcs12 instead of jks as the confluence instructions say?



Heck, Aaron

I don't see any security considerations to be concerned about in a move from jks to pkcs12. I would agree moving to pkcs12 is a better option. JKS can be a real pain and I would recommend getting away from it wherever possible.





Rolland, Stuart

Can I do this in windows or is it Linux only?

I found this on the gandi website so I guess that answers my question

### What You Need¶

To generate the CSR you will need access to a unix terminal on a machine with OpenSSL, or an equivalent, installed. You do not need to use the machine where you will install the certificate to generate the CSR.

Some options may be:

- Your regular computer, if you use Linux or OSx
- A Gandi cloud server
- A Gandi simple hosting instance (even if you intend to use the certificate outside of Gandi)
- An accessible production server running on Linux or Unix
- A Windows computer with OpenSSL for Windows installed

I suggest making this document more open so we can crowdsource the improvement on it - like wikipedia



Heck, Aaron

Thanks, Stuart. Added a requirements section to address this question.



Sherrington, Ryan

@ Rolland, Stuart I did this in [MAS-3315](#) which is Windows Server 2012 R2, but you can do it on any windows machine that allows local application installation. See [openssl.org binaries page](#).



Rolland, Stuart

Might also be worth including a link to this [https://docs.gandi.net/en/ssl/common\\_operations/csr.html](https://docs.gandi.net/en/ssl/common_operations/csr.html)



Heck, Aaron

Thanks, Stuart. The document is carefully crafted to support the UBC centralized certificate service, so changes are currently curated and don't include links to vendor-specific guides that might not align with our practices or expectations. To be fair, at the end of the day sysadmins are free to generate their CSRs any way they see fit and we will process them. We provide the guide in an attempt to help with consistency and alignment with standards.

---