Java SE Security
Lab 4

1 Lab setup

Make sure you have the following items installed

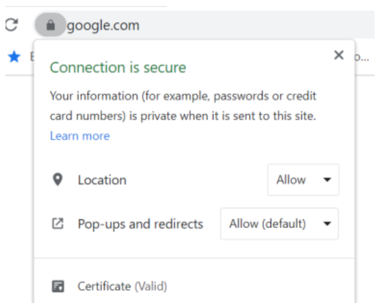
- Latest version of JDK 8 / 11 (note: labs are tested with JDK 8 but should work on JDK 11 with no or minimal changes)
- Eclipse Enterprise Edition or Spring Tool Suite (STS) IDE
- Latest version of Maven
- A suitable text editor (Notepad ++)
- A suitable hex editor (HxD Hex Editor)
- A utility to extract zip files

2 Accessing certificates

2.1 Server TLS certificates

When you use a browser to connect to a website whose hosting server supports HTTPS, you will be able to see a padlock at the left of the address bar. Different browsers have slightly different approaches to view the certificate that the server returns as part of the TLS/SSL protocol

In Chrome, click on the toolbar and select the option Certificate (Valid).


google.com

Connection is secure

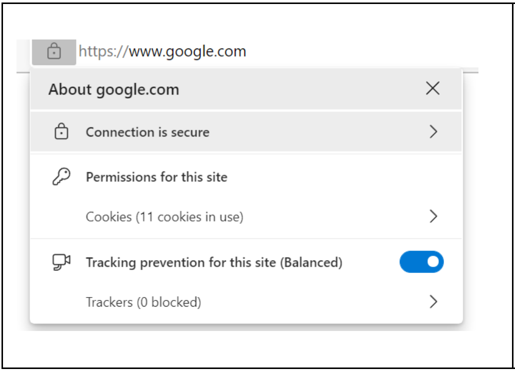
Your information (for example, passwords or credit card numbers) is private when it is sent to this site. Learn more

Location: Allow

Pop-ups and redirects: Allow (default)

Certificate (Valid)

On Edge, select Connection is secure, then select the Certificate icon.


https://www.google.com

About google.com

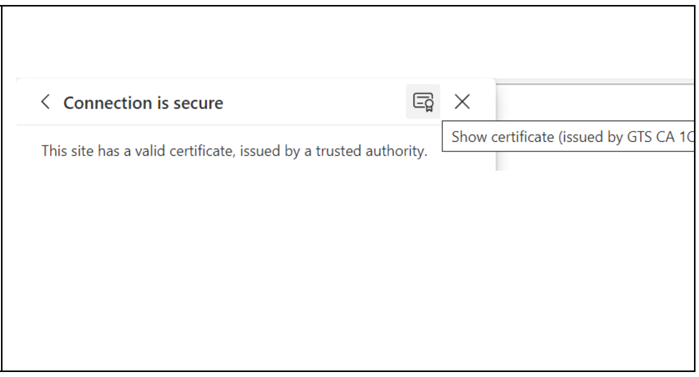
Connection is secure

Permissions for this site

Cookies (11 cookies in use)

Tracking prevention for this site (Balanced)

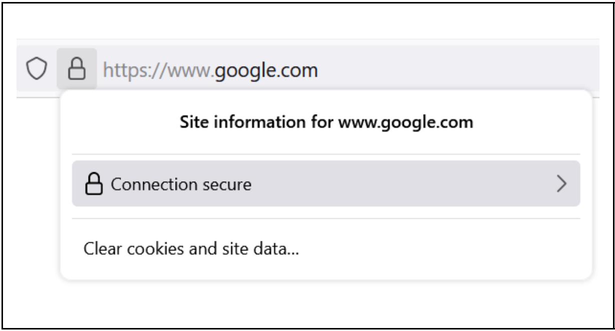
Trackers (0 blocked)


Connection is secure

This site has a valid certificate, issued by a trusted authority.

Show certificate (issued by GTS CA 1C

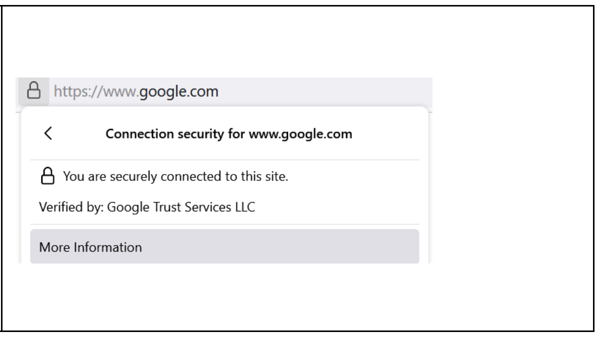
On Firefox, select Connection Secure -> More Information


https://www.google.com

Site information for www.google.com

Connection secure

Clear cookies and site data...


https://www.google.com

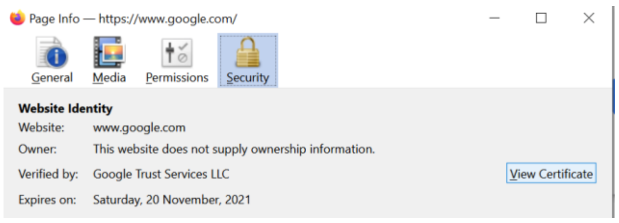
Connection security for www.google.com

You are securely connected to this site.

Verified by: Google Trust Services LLC

More Information

Then click on the View Certificate button.


Page Info — https://www.google.com/

General | Media | Permissions | Security

Website Identity

Website: www.google.com

Owner: This website does not supply ownership information.

Verified by: Google Trust Services LLC

Expires on: Saturday, 20 November, 2021

View Certificate

TLS server certificates (the most common types of digital certificates) can be classified based on domain type:

- Single domain
- multi domain
- wildcard

and also on validation type:

- Domain validation (DV)
- Organization Validation (OV)
- Extended Validation (EV)

Certificates will feature combinations from these two categorization types.

Go to:

https://www.google.com/
https://hometaste.my/

To view the normal and extended fields of the certificate in the Certificate box (from either Chrome or Edge), select Details. In FireFox, the fields will appear in a separate tab which provides a more readable view.

Check the Subject field. Notice that there is only the CN field here shown for the subject DN for both these certificates. This indicates that these certificates only have DV (they do not feature all the remaining DN fields such as O, OU, L, S, etc.

Notice that the validity period of the certificate is roughly around a year or less (check the Valid Form and Valid to fields). This is inline with the current certification security standards which mandate a shorter period to guard against possible key exhaustion or key compromise.

Check the Subject Alternative Name extension field. Notice that there is also only one DNS name for both these certificates. This indicates that these certificates are single domain.

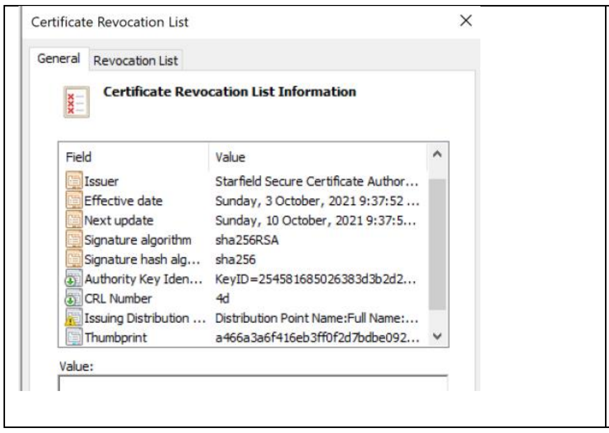
In the Public key field, you can see the specified public key provided in hex encoding. Notice that for ECC, the public key parameters has a specific value, while for RSA, the public key parameter is null.

The Enhanced Key usage indicates the applications in which the certificate may be used for, and this will nearly all the time be either server / client authentication or both.

The Key usage fields indicate what the public key in the certificate can be used for. Nearly all the time, this will be digital signature (as this is a core part of the SSL/TLS procedure) but can also include other actions such as key encipherment (where the public key is used to encrypt a symmetric session key). This will be the case if a RSA public key is being used for SSL RSA key exchange.

The Basic Constraints field clearly indicates the subject type as a end-entity

The CRL distribution point provides a URL from where you can download the CRL. If you do this, you can double click on the file in Windows to view its contents in a CRL dialog box.

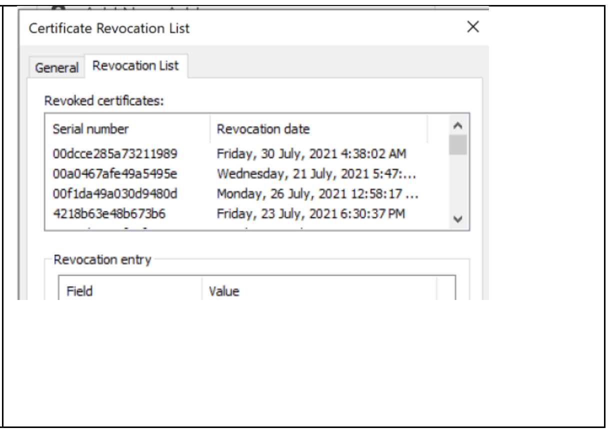

Certificate Revocation List

General | Revocation List

Certificate Revocation List Information

Field	Value
Issuer	Starfield Secure Certificate Author...
Effective date	Sunday, 3 October, 2021 9:37:52 ...
Next update	Sunday, 10 October, 2021 9:37:5...
Signature algorithm	sha256RSA
Signature hash alg...	sha256
Authority Key Iden...	KeyID=254581685026383d3b2d2...
CRL Number	4d
Issuing Distribution ...	Distribution Point Name:Full Name:...
Thumbprint	a466a3a6f416eb3ff0f2d7bdbe092...

Value:


Certificate Revocation List

General | Revocation List

Revoked certificates:

Serial number	Revocation date
00dcce285a73211989	Friday, 30 July, 2021 4:38:02 AM
00a0467afe49a5495e	Wednesday, 21 July, 2021 5:47:...
00f1da49a030d9480d	Monday, 26 July, 2021 12:58:17 ...
4218b63e48b673b6	Friday, 23 July, 2021 6:30:37 PM

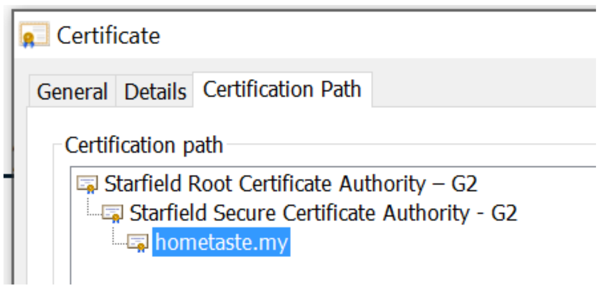
Revocation entry

Field	Value

In the general tab, you can see that the issuer of the CRL is also the same CA that issued the certificate. There are dates to indicate when the CRL takes effect and when the next update will be (typically this will be a 1 - 2 week interval period). The algorithm used to sign the CRL with the public key of the CA is also shown.
In the revocation list, you can see the serial number (which uniquely identifies that particular certificate in the CA's certificate database) of the revoked certificate as well as the revocation date.

The Authority Info Access field includes the URL of an OCSP responder (typically we will not be able to access this directly in a browser), as the protocol used is not HTTP.

The Certification Authority Issuer may provide a URL from which you can download the certificate chain starting from the root CA, proceeding through one or more intermediate CAs down to the current end-entity certificate. If you click on the Certification Path tab for a given certificate, you should be able to view the same certificate chain as well. Double clicking on one of the certificates in the certification path will allow you to view that particular certificate in another Certificate Dialog box.


Certificate

General | Details | Certification Path

Certification path

```
Starfield Root Certificate Authority – G2
  Starfield Secure Certificate Authority - G2
    hometaste.my
```

The Certificate Policies field provides a URL to download the policy governing the issuance and use of the certificate, as well as other relevant items. You can also view this by clicking on the Issuer Statement in the General tab of the Certificate Dialog box.

Most of the major CAs provide a dedicated URL where their policies are publicly viewable, for e.g.
https://www.digicert.com/legal-repository
https://sectigo.com/legal

The certificate may also feature OIDs (e.g. 2.23.140.1.2.1) which are used to uniquely identify an object (typically in certificate policies field). You can search for the specific object that they identify at:

http://www.oid-info.com/index.htm

In Firefox, you can see the name of the particular CT log that the certificates embedded SCTS are found at. The complete list of logs is available at:
https://ct.cloudflare.com/logs

You can choose to download the certificate in specific formats. In the details tab of the Certificate Dialog box (from Chrome or Edge), select Copy to File. This brings up the Certificate Export Wizard which typically offers 3 formats: DER, Base64 (both with .CER extensions) and PKCS7 (.P7B extension). Try saving the certificate to all 3 different formats and open up their contents with NotePad++. Notice that PKCS7 and DER are both binary. You can open up any of these files by simply double clicking on then, where you are also presented the additional option of installing the certificate. Clicking on this brings up the Certificate Import Wizard, which allows you to install the certificate into your local (either for user or computer) certificate store. We will explore this in more detail in a future lab.

Firefox only allows you to download the certificate itself (or the chain) in PEM format. You cannot open PEM format directly in Windows, so you have to rename the extension first to .CER, after which double clicking should directly open up the file.

To view examples of multi-domain certificates (where the SAN extended field has multiple domains), check the certificates at:

https://www.awam.org.my/
https://www.mercy.org.my/

Notice that these are subdomains of the main domains.

To view examples of wildcard certificates (where the SAN extended field has multiple domain names including some with *):

https://www.moh.gov.my/
https://www.foodpanda.my/

To view more examples of DV certificates (where only the CN field is shown for the subject DN), go to:

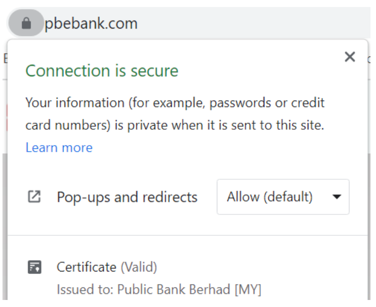
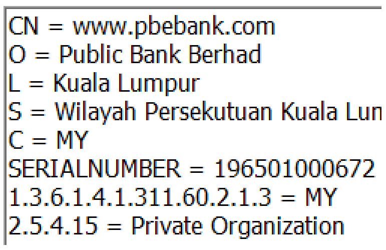
https://www.awam.org.my/
https://www.thestar.com.my/

To view examples of OV validated certificates, where the DN field includes complete details for most or all of the DN fields such as O, OU, L, S, etc.

https://www.moh.gov.my/
https://www.lazada.com.my/

To view examples of EV validated certificates, which feature a short description below the Certificate Valid section as well as detailed information on the Subject DN that includes a SERIALNUMBER and OID for the country and organization.

https://www.paypal.com
https://www.pbebank.com/

 pbebank.com
Connection is secure
Your information (for example, passwords or credit card numbers) is private when it is sent to this site.
Learn more
Pop-ups and redirects: Allow (default)
Certificate (Valid)
Issued to: Public Bank Berhad [MY]
	 CN = www.pbebank.com
	O = Public Bank Berhad
	L = Kuala Lumpur
	S = Wilayah Persekutuan Kuala Lun
	C = MY
	SERIALNUMBER = 196501000672
	1.3.6.1.4.1.311.60.2.1.3 = MY
	2.5.4.15 = Private Organization

To view certificates that are presented using Server Name Indication (SNI) when there are multiple domains hosted on a single server, see:

https://www.malaymail.com/
https://www.nst.com.my/

Notice here that the Subject DN does not contain the actual domain name of these websites. Instead this is contained in the extension SAN field.
Notice how this is different from another setup of a server hosting multiple domains but listing this in the SAN extension field (i.e. multi domain certificate) instead of using SNI:

https://www.pantai.com.my/

For the Pantai certificate, you can see all the other domains and if you attempt to connect to any one of them, you will be presented with the same certificate in return. This is compared to SNI, where none of the other domains are listed.

You can search for certificate details at

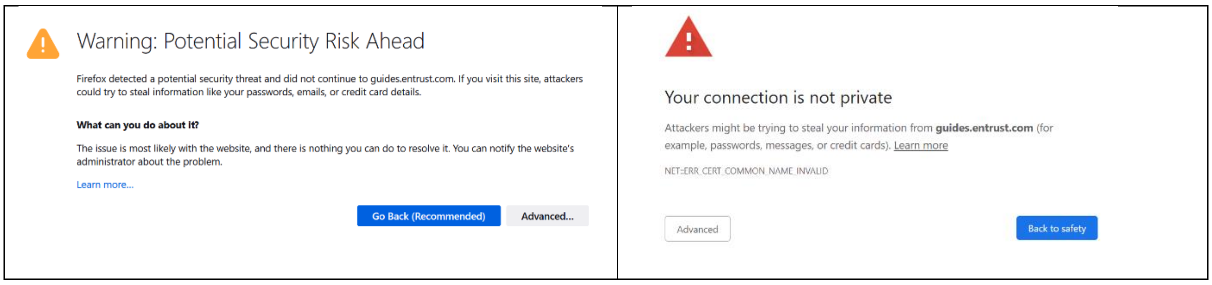
https://search.censys.io/certificates?q=

by typing in relevant fields such CN from subject DN or certificate thumbprint

Below is an example of a certificate that produces a browser warning. This is due to the certificate subject CN and SAN not matching the domain name of the server returning the certificate.

https://guides.entrust.com/g/entrust-1492151/43114

The warning differs on the browser used and typically offers an Advanced option for you to click to connect to the insecure domain.



Warning: Potential Security Risk Ahead

Firefox detected a potential security threat and did not continue to guides.entrust.com. If you visit this site, attackers could try to steal information like your passwords, emails, or credit card details.

What can you do about it?

The issue is most likely with the website, and there is nothing you can do to resolve it. You can notify the website's administrator about the problem.

Learn more...

Your connection is not private

Attackers might be trying to steal your information from **guides.entrust.com** (for example, passwords, messages, or credit cards). Learn more

NET::ERR_CERT_COMMON_NAME_INVALID

Even if you do this, the browser will still display Not secure or a padlock with an afflicted icon to indicate that you have set a security exception for that particular domain. You can choose to disable this exception, whereupon the error message will be displayed again.

A comprehensive list of certificate issues that results in browser warnings can be seen at:

https://badssl.com/

You can click on the options to view some of these.

To check revocation status of a particular certificate, use:
https://certificate.revocationcheck.com/

To check the SSL status of a server hosting a particular domain name, you can use:
https://www.sslshopper.com/ssl-checker.html

SSL Labs provides online tests for verifying various aspects of SSL functionality:
https://www.ssllabs.com/projects/index.html?_ga=2.246766532.1110731722.1633325878-1098056815.1631077691

You can try out the Server and Client Test

Another site to test the security functionality of your browser with respect to Server Name Indication (SNI)
https://www.cloudflare.com/en-gb/ssl/encrypted-sni/

2.2 Trusted root CA certificates

The list of trusted root CA certificates (as well as other relevant certificates) are maintained either by the OS or browser. Windows / MacOS have their own trusted CA store (Microsoft or Apple Root Program), browsers (Edge, Safari and Chrome) use this store. Firefox uses its own CA store (independent of underlying OS) - Mozilla Root Program. Java apps uses the bundled trust store that comes with the JDK (cacerts) - Oracle Java Root Program

To view certificates from the browser:

Chrome: Settings -> Privacy and Security -> Security (Safe Browsing) -> Manage Certificates
Firefox: Settings -> Privacy and security -> Security Certificates -> View Certificates

To view certificates installed for current user / current computer in Windoww:

https://www.top-password.com/blog/view-installed-certificates-in-windows-10-8-7/

Select a few of the certificates in the Trusted Root CA section to view. The key points that differentiate these from end-entity certificates we have previously examined are:

a) Each CA may have one or more root certificates (for e.g. DigiCert, Entrust, GlobalSign, etc) that covers different areas of authorization (e.g. EV) or different areas of the world (Global Root)
b) The Subject and the Issuer in the root certificate is exactly the same, i.e they are self-signed certificates
c) The validity period is long (anywhere between 15 - 30 years)
d) The Basic Constraints field clearly indicates the subject type as a CA
e) Key usage includes activities specific only to a CA: Certificate Signing, Off-line CRL Signing, CRL Signing while enhanced key usage includes all relevant functionality that a certificate can be used for: Code Signing, OCSP signing, Secure Email, Time Stamping, etc

You can also click on Certificate Properties to set specific properties for the certificates for e.g.
- Adding / removing purposes for the certificates
- Setting cross certificates
- Specifying addition OCSP download locations
- Specifying OIDS to mark a root certificate as an EV root certificate

Select a few of the certificates in the Intermediate CA section to view. The key points that differentiate these from trusted root CA certificates:

a) The Issuer is nearly always one of the Trusted Root CAs, which results in a certification path length of 2
b) Most of the intermediate CAs are from the same organization as the Trusted Root CA, or to be more technically accurate, their certificates are used as additional security layer to protect the signing key of the Trusted Root CA which is only used to sign them and not end-entity certificates. Each of these intermediate certificate serve a clear purpose based on their names (e.g. DigiCert intermediate CAs)
c) There are also cross signed certificates. For e.g. GeoTrust is a Trusted Root CA but its intermediate certificates are signed by DigiCert instead.

All of the well-known global root CAs publish their root, intermediary and cross-signed certificates for download on their official website, for e.g.:

https://www.digicert.com/kb/digicert-root-certificates.htm
https://www.entrust.com/resources/certificate-solutions/tools/root-certificate-downloads

Most of these are already preinstalled in the Trusted Root CA section for an OS (through the Root programs of Microsoft or Apple). It may be possibly that an malicious party (for e.g. a malware that has gained residence on the system) may substitute the certificate in the Trusted Root CA section with a certificate of its own that has identical content as the original CA certificate, with the exception that it contains the public key of the malicious party instead. In that case, when a browser connects to site using a certificate that was signed using the private key of the malicious party, no warning will be flagged.

To circumvent this issue, some of the global CAs provide a web page which is signed using an intermediate certificate that links to all of their different root certificates (e.g. Digicert).

Alternatively, you can download the root certificates themselves and then compare their contents directly with the corresponding certificate present in the Trusted Root CA section (they should have identical contents, in particular the public key field value). You can easily save the public key Hex string encoding for the two certificates that you wish to compare into a text file and run a comparison using a standard diff tool (for e.g. on Linux using diff) or tools like Winmerge (https://winmerge.org/) on Windows. Try this out with a few certificates from Digicert as practice.

In the course of development work on apps that utilize certificates as part of their security infrastructure, we will encounter certificates that are signed by root CAs that are not currently in the Trusted Root CA section of the local certificate store. In order to allow our apps to recognize and use these certificates, we will need to import the certificates of these unknown root CAs into the Trusted Root CA section.

For e.g. MSC Trustgate (one of the official CAs in Malaysia) provides an official list of their CA certs at:
https://www.msctrustgate.com/cacerts.php

Notice that in addition to the various categories of utilization for their own intermediate CAs (such as ECC Individual CA, RSA Individual CA, Corporate ID (Mobile), etc), MSC Trustgate also issues intermediate CAs for organizations such as banks and payment processors (Bank Negara Malaysia, RHB, ABMB, Paynet), national bodies (National Cyber Security Agency (NACSA), MAMPU), etc. These intermediate CAs can then be used as root CAs within that organization's / banks internal PKI, which is useful for creating client and server certificates for mutual authentication in a zero-trust security environment.

Below is a case study for zero trust authentication at Paynet using the security services provided by CloudFlare:
https://www.cloudflare.com/en-gb/case-studies/paynet/

3 Using Keytool

The Java Keytool is a command line tool which can perform a variety of key generation and management functionality for both symmetric and public / private key pairs in a key store. To run keytool from the command line, make sure you have the `%JAVA_HOME%\bin` on your system path.

https://mkyong.com/java/how-to-set-java_home-on-windows-10/

Most commands involve actions on a certificate or private key (generating, importing, exporting, etc) within a new or existing key store. You can specify passwords for the individual private / public keys (`-keypass`) as well as a password for the store as a whole (`-storepass`). If the `-keypass` argument is omitted, the password for the key is the same as the keystore password. At minimum, keytool will prompt you for the keystore password.

Most commands will involve the use of the `-keystore` argument to specify the keystore file. If this is omitted, then the default keystore file is used. This is typically named `.keystore` in the user's home directory is used (or created, if it doesn't already exist).

The keytool command can create key stores in a variety of formats. The most widely used are either JKS (a proprietary Java format) or PKCS12, which is also a certificate format but which allows the storage of multiple cryptography objects within a single file (which serves the purpose of a key store). We will use the PCKS12 format since this a popular format that can be worked on by other tools such as OpenSSL.

The reference for the full list of commands for keytool is available at:
https://docs.oracle.com/en/java/javase/13/docs/specs/man/keytool.html

Navigate to an empty directory in a command prompt shell.

To create a symmetric secret key using AES-128 and store it in a PKCS12 keystore, type:

```
keytool -genseckey -alias firstAESkey -keyalg AES -keysize 128 -storetype PKCS12 -keystore firstkeystore.p12 -storepass changeit
```

To list the key we just created, type:

```
keytool -list -alias firstAESkey -keystore firstkeystore.p12 -storepass changeit -storetype PKCS12 -v
```

To create a new key pair and place the public key in a self-signed certificate, type:

```
keytool -genkeypair -alias awesome -keyalg RSA -keysize 2048 -sigalg SHA256withRSA -validity 365 -storetype PKCS12 -storepass changeit -keystore firstkeystore.p12
```

Enter these values for the following prompts that appear:

```
What is your first and last name?
  [Unknown]:  awesome.com
What is the name of your organizational unit?
  [Unknown]:  awesome developers
What is the name of your organization?
  [Unknown]:  cool developers
What is the name of your City or Locality?
  [Unknown]:  Petaling Jaya
What is the name of your State or Province?
  [Unknown]:  Selangor
What is the two-letter country code for this unit?
  [Unknown]:  MY
Is CN=awesome.com, OU=awesome developers, O=cool developers,
L=petaling jaya, ST=selangor, C=MY correct?
  [no]:  yes
```

To view this newly created certificate, type:

```
keytool -list -v -alias awesome  -keystore firstkeystore.p12 -storepass changeit -storetype PKCS12
```

You can see that this certificate has a chain length of 1, indicating that this is a self-signed certificate. Notice that the Entry type indicates a PrivateKeyEntry, since the alias is for the private key and its corresponding public key embedded within a certificate.

For commands that use the -keyalg options (such as -genkeypair, -genseckey) , the list of options that can be used are shown in:
https://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html#KeyGenerator
https://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html#KeyPairGenerator

We will create two more self-signed certificates with a SAN extension field of localhost which we will use in a subsequent lab for a Spring boot app running in an embedded Tomcat server. We also specify the DN as a hardcoded string argument.

```
keytool -genkeypair -alias Marvel -keyalg RSA -keysize 2048 -dname "cn=marvel.com,ou=marvel comics,o=marvel inc,l=NY city,ST=New York,C=US" -storetype PKCS12 -keystore firstkeystore.p12 -validity 365 -ext SAN=dns:localhost,ip:127.0.0.1 -storepass changeit
```

```
keytool -genkeypair -alias DC -keyalg RSA -keysize 2048 -dname "cn=dcuniverse.com,ou=DC comics,o=DC inc,l=San Francisco,ST=California,C=US" -storetype PKCS12 -keystore firstkeystore.p12 -validity 365 -ext SAN=dns:localhost,ip:127.0.0.1 -storepass changeit
```

To view all the certificates and private keys in the key store at this point of time, type:

```
keytool -list -v -keystore firstkeystore.p12 -storepass changeit -storetype PKCS12
```

You should see a listing of 4 entries.

Next, we create another key pair and generate a Certificate Signing Request (CSR) from it:

```
keytool -genkeypair -alias ironman -keyalg RSA -keysize 2048 -dname "cn=ironman.com,ou=supersuits,o=Stark Industries inc,l=Boston,ST=Massachusetts,C=US" -storetype PKCS12 -keystore firstkeystore.p12 -validity 365 -ext SAN=dns:localhost,ip:127.0.0.1 -storepass changeit

keytool -certreq -alias ironman -storetype PKCS12 -keystore firstkeystore.p12 -storepass changeit -file ironmanrequest.csr
```

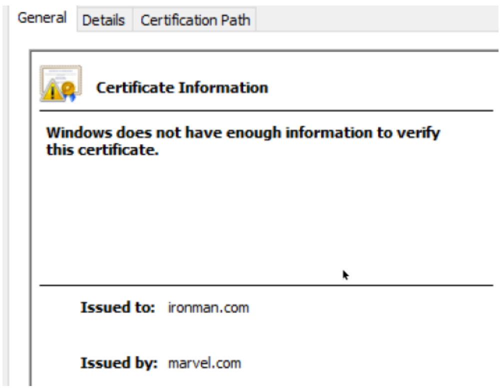
Creating CSRs is a standard way to submit a request to a CA in order to get a TLS certificate to install on a server, for e.g. as detailed in:
https://www.digicert.com/kb/csr-ssl-installation/tomcat-keytool.htm

Let us assume that the self-signed certificate with the alias `Marvel` that we created earlier is a trusted root CA certificate. We can create a certificate from the CSR file `ironmanrequest.csr` with:

```
keytool -alias marvel -gencert -sigalg SHA256withRSA -ext
san=dns:localhost,ip:127.0.0.1 -infile ironmanrequest.csr -outfile
signedcert.cer -storetype PKCS12 -keystore firstkeystore.p12 -
storepass changeit
```

For commands that use the -sigalg options (such as -gencert, -genkeypair, -certreq) , the list of options that can be used are shown in:
https://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html#Signature

Now if you open the generated certificate `signedcert.cer` by double clicking on it, you will be able to see that the certificate is issued to `ironman` and was issued by `marvel`. If we inspect the certification path, we only see a single certificate (this one) shown, and the issuer's (`marvel`) certificate is not shown. Windows flags the certificate as not containing enough information to validate since the issuer's certificate is not currently installed in the Trusted Root CA section for either the user or local computer.



Certificate Information

Windows does not have enough information to verify this certificate.

Issued to: ironman.com

Issued by: marvel.com

Next we can import this certificate into the key store under the same alias that the CSR was originally created from:

```
keytool -importcert -alias ironman -storetype PKCS12 -keystore
firstkeystore.p12 -file signedcert.cer -rfc -storepass changeit
```

Now if we inspect this alias with:

```
keytool -list -rfc -alias ironman -keystore firstkeystore.p12 -
storepass changeit -storetype PKCS12
```

you should now see a certificate chain length of 2 with both certificates displayed (`ironman` as well as the issuing certificate `marvel`), since this issuing certificate is present in the key store.

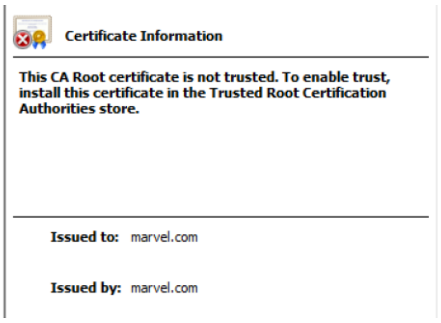
Let's export the `ironman` certificate as well as the `marvel` self-signed certificate that we are using in the role of a trusted root CA:

```
keytool -exportcert -rfc -alias ironman -file ironman.cer -keystore
firstkeystore.p12 -storepass changeit -storetype PKCS12

keytool -exportcert -rfc -alias marvel -file marvel.cer -keystore
firstkeystore.p12 -storepass changeit -storetype PKCS12
```

To view the contents of these two certificates that we have exported, type:

```
keytool -printcert -file ironman.cer -v

keytool -printcert -file marvel.cer -v
```

If we open `marvel.cer`, Windows flags its as an unrecognized CA since any self-signed certificate by default is assumed to be a root certificate (all other certificates such as end-entity or intermediate certificate are signed by some other certificate).



This CA Root certificate is not trusted. To enable trust, install this certificate in the Trusted Root Certification Authorities store.

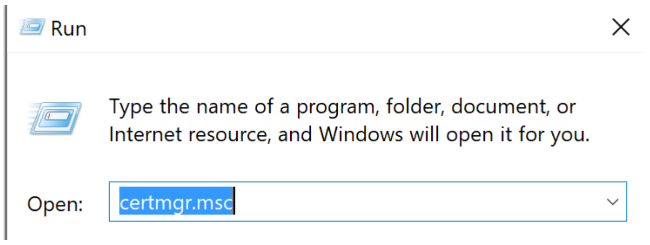
Issued to: marvel.com

Issued by: marvel.com

If we open `ironman.cer`, we get the same situation as `signedcert.cer` whereby Windows flags this certificate as not having enough information to verify.

Open marvel.cer, then click Install Certificate. This opens the Certificate Import Wizard. Select Current User for the store location, then in the certificate store section, select the option Place all certificates in the following store. Then click Browse and select Trusted Root Certification Authorities, select OK and select Next. Then click Finish.
A security warning should appear about installing the certificate. Click Yes.

Now open the certificate store for the current user, by typing `certmgr.msc` at the run command from the Windows Start icon.



Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

Open: certmgr.msc

You should now be able to see the `marvel` certificate installed there.

If you now attempt to open `ironman.cer`, Windows displays the certificate as being validated correctly with `marvel` appearing as the root certificate in the Certification Path tab.

Return back to the certificate store for the local user and delete `marvel` for now. Opening `ironman.cer` will now result in the previous warning message.

As noted earlier, Java apps use the bundled key store that comes with the JDK which is part of the Oracle Java Root Program, and this will be different from the trusted Root CA store in Windows that is used by Edge and Chrome.

The name of this key store is `cacerts` (we term it the trust store) and it is located by default at:

```
%JAVA_HOME%\jre\lib\security\cacerts
```

The default password for the trust store is `changeit`, and usually a system admin is supposed to change it upon installation of the JDK ! 😉

To view all the root CA certificates here, we can pipe the output from keytool to a text file for later inspection:

```
keytool -list -keystore "C:\Program Files\Java\jdk1.8.0_251\jre\lib\security\cacerts" -storepass changeit -v > cacertslist.txt
```

Open this file to view the contents of all the certificates.

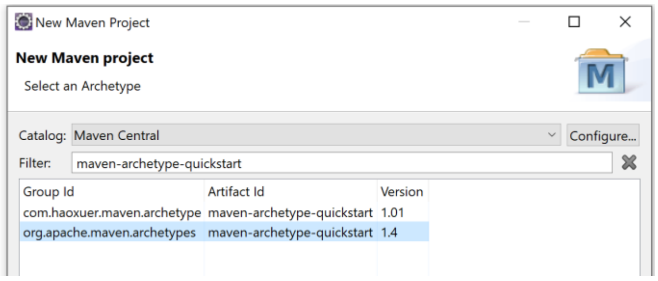
Keep in mind that any Java app (including Spring Boot apps) that you create will rely on this store to validate certificates that it receives as part of its application functionality, unless you explicitly specify a different store. We will explore this later in a future lab.

4 Working with Keystore programmatically

The source code for this lab is in the `Keystore-Basics-Demo` folder.

Switch to Java EE perspective.

Start with File -> New -> Maven Project. Select Next and type `maven-archetype-quickstart` in the filter. Eclipse may freeze temporarily while it attempts to filter through all the archetypes available at Maven Central. Select the entry with group id: `org.apache.maven.archetypes` and click Next



Catalog: Maven Central
Filter: maven-archetype-quickstart

Group Id	Artifact Id	Version
com.haoxuer.maven.archetype	maven-archetype-quickstart	1.01
org.apache.maven.archetypes	maven-archetype-quickstart	1.4

Enter in the following details and click Finish.

Group id: `com.workshop.basics`
Artifact id: `KeystoreBasicsDemo`
Version: `0.0.1-SNAPSHOT`
Package: `com.workshop.basics`

Replace the contents of the `pom.xml` in the project with `pom.xml` from `changes`. Right click on the project, select Maven -> Update Project, and click OK.

In the package `com.workshop.basics`, place the following files:

```
CryptoUtils
BasicUtils
FileUtils
UseKeyStoreDemo
```

Place this key store that we created earlier in the project root folder and a plaintext file:

```
firstkeystore.p12
plaintext.txt
```

We use the Keystore class to retrieve and store the contents of the key store file. We can then extract the secret key and key pairs (public key / private key) that we had earlier stored in them using Keytool. Notice that in order to extract the public key, we need to obtain the certificate first using the keypair alias since the public key is embedded in the certificate.

Run the app and verify that the extracted secret key, public key and private key can be used in the usual manner to perform standard encryption, creation and verification of a digital signature.

In the package `com.workshop.basics`, place the following files:

```
CreateKeyStoreDemo
```

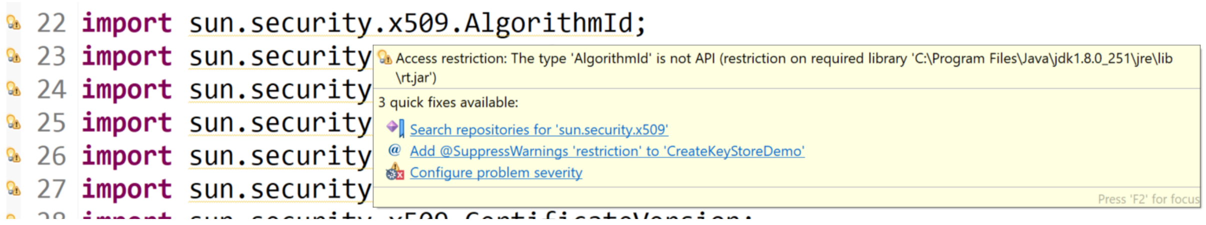
Here we demonstrate how to create a key store programmatically and place a symmetric key as well as a key pair (private key + public key in a self-signed certificate) in the key store.

Creating a key store and storing a symmetric secret key in it is pretty straightforward.

Generating a key pair as well as a self-signed certificate to hold the public key of the key pair is a bit more complicated. There are generally two ways to accomplish this:

- Using JDK only
- Using a 3rd party security provider (e.g. BouncyCastle)

Using only the JDK requires us to access the `sun.security.x509` classes. These classes are not officially documented since this API is not a public API (i.e. they are part of the Java runtime in rt.jar, and are not generally accessible for developers to work with directly). Eclipse will flag this as an access restriction warning since we should not generally be working with non-public APIs due to the possible problems it can cause with the Java runtime if it is not executed properly.



```
import sun.security.x509.AlgorithmId;
import sun.security
import sun.security
import sun.security
import sun.security
import sun.security
```

Access restriction: The type 'AlgorithmId' is not API (restriction on required library 'C:\Program Files\Java\jdk1.8.0_251\jre\lib \rt.jar')

3 quick fixes available:

- Search repositories for 'sun.security.x509'
- Add @SuppressWarnings 'restriction' to 'CreateKeyStoreDemo'
- Configure problem severity

We generate two keys pairs for two DNs (MOH and RHB) using two different approaches. The first approach is implemented in `genCertWithDetails` and is more detailed, requiring specification of all the relevant X509 standard fields. The second approach using the CertAndKeyGen class is more simplified. In both cases, you will need to create a certificate chain to store the private key under the same alias as the certificate. In this case, the certificate chain is of length 1, since it is a self-signed certificate.

The list of algorithms for a Signature class instance are:
https://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html#Signature
To identify the various algorithms for AlgorithmId class in sun.security.x509
http://www.docjar.com/docs/api/sun/security/x509/AlgorithmId.html

Run the app.

This should generate a new keystore: `secondkeystore.p12` in the root of the project folder. Open a command line terminal in this folder to view the contents of the key store and extract the 2 self-signed certificates contained within to separate standalone certificates:

```
keytool -list -v -keystore secondkeystore.p12 -storepass changeit -
storetype PKCS12

keytool -exportcert -rfc -alias moh -file moh.cer -keystore
secondkeystore.p12 -storepass changeit -storetype PKCS12

keytool -exportcert -rfc -alias rhb -file rhb.cer -keystore
secondkeystore.p12 -storepass changeit -storetype PKCS12
```

Open both `moh.cer` and `rhb.cer` by double clicking on them and verifying that their contents align with the values used in generating them.