




# Lab 6: Trust and Digital Certificates

**Objective:** Digital certificates are used to define a trust infrastructure within PKI (Public Key Infrastructure). A certificate can hold a key pair, while a distributable certificate will only contain the public key. In this lab we will read-in digital certificates and analyse them.

 **Lab demo:** <https://youtu.be/-uNQFv0GTZc>

## A Introduction


No	Description	Result
A.1	<p>From:</p> <p> <b>Web link (Digital Certificate):</b> <a href="http://asecuritysite.com/encryption/digitalcert">http://asecuritysite.com/encryption/digitalcert</a></p> <p>Open up Certificate 1 and identify the following:</p>	<p>Serial number:</p> <p>Effective date:</p> <p>Name:</p> <p>Issuer:</p> <p>What is CN used for:</p> <p>What is ON used for:</p> <p>What is O used for:</p> <p>What is L used for:</p>
A.2	<p>Now open-up the ZIP file for the certificate (Certificate 3), and view the DER file.</p>	<p>What other information can you gain from the certificate:</p> <p>What is the size of the public key:</p> <p>Which hashing method has been used:</p> <p>Is the certificate trusted on your system: [Yes][No]</p>
A.3	<p>Make a connection to the <b>www.live.com</b> Web site:</p> <pre>openssl s_client -connect www.live.com:443</pre>	<p>Can you identify the certificate chain?</p> <p>What is the subject on the certificate?</p> <p>Who is the issuer on the certificate?</p>
A.4	<p>Google moved in July 2018 to mark sites as being insecure if they did not have a match between their digital certificate and the site. First open a browser and see if you can access</p>	<ul style="list-style-type: none"><li>Run a scan from SSL Labs on testfire.net. What do you observe from the result?</li></ul>

	testfire.net (you can try both https and http for the connection).	<ul style="list-style-type: none"> <li>What is the SSL Labs rating on this site? Is it "A", "B", "C", "D", "E" or "F"?</li> <li>What does a "T" rating identify?</li> <li>Can you locate another "T" rated site?</li> </ul> 
--	--	--

**A.5** Which the certificates in A.2, for Example 2 to Example 6. Complete the following table:

Cert	Organisation (Issued to)	Date range when valid	Size of public key	Issuer	Root CA	Hash method	Is it trusted?
2							
3							
4							
5							
6							

**A.6** Now download the DER files from:

 **Web link (Digital Certificate):** <http://asecuritysite.com/der.zip>

Now use openssl to read the certificates:

```
openssl x509 -inform der -in [certname] -noout -text
```

## B Creating certificates

Now we will create our own self-signed certificates.

No	Description	Result
----	-------------	--------

<b>B.1</b>	<p>Create your own certificate from:</p> <p> <b>Web link (Create Certificate):</b>  <a href="http://asecuritysite.com/encryption/createcert">http://asecuritysite.com/encryption/createcert</a></p> <p>Add in your own details.</p>	<p>View the certificate, and verify some of the details on the certificate.</p> <p>Can you view the DER file?</p>
------------	--	---

We have a root certificate authority of My Global Corp, which is based in Washington, US, and the administrator is admin@myglobalcorp.com and we are going to issue a certificate to My Little Corp, which is based in Glasgow, UK, and the administrator is admin@mylittlecorp.com.

No	Description	Result
<b>B.2</b>	<p>Create your RSA key pair with:</p> <pre>openssl genrsa -out ca.key 2048</pre> <p>Next create a self-signed root CA certificate ca.crt for <b>My Global Corp</b>:</p> <pre>openssl req -new -x509 -days 1826 -key ca.key -out ca.crt</pre>	<p>How many years will the certificate be valid for?</p> <p>Which details have you entered:</p>
<b>B.3</b>	<p>Next go to Places, and from your Home folder, open up ca.crt and view the details of the certificate.</p>	<p>Which Key Algorithm has been used:</p> <p>Which hashing methods have been used:</p> <p>When does the certificate expire:</p> <p>Who is it verified by:</p> <p>Who has it been issued to:</p>
<b>B.4</b>	<p>Next we will create a subordinate CA (<b>My Little Corp</b>), and which will be used for the signing of the certificate. First, generate the key:</p> <pre>openssl genrsa -out ia.key 2048</pre> <p>Next we will request a certificate for our newly created subordinate CA:</p>	<p>View the newly created certificate.</p> <p>When does it expire:</p> <p>Who is the subject of the certificate:</p> <p>Which is their country:</p> <p>Who signed the certificate:</p>

	<p><code>openssl req -new -key ia.key -out ia.csr</code></p> <p>We can then create a <b>certificate from the subordinate CA</b> certificate and <b>signed by the root CA</b>.</p> <p><code>openssl x509 -req -days 730 -in ia.csr -CA ca.crt -CAkey ca.key -set_serial 01 -out ia.crt</code></p>	<p>Which is their country:</p> <p>What is the serial number of the certificate:</p> <p>Check the serial number for the root certificate. What is its serial number:</p>
<b>B.5</b>	<p>If we want to use this certificate to digitally sign files and verify the signatures, we need to convert it to a PKCS12 file:</p> <p><code>openssl pkcs12 -export -out ia.p12 -inkey ia.key -in ia.crt -chain -CAfile ca.crt</code></p>	<p>Can you view ia.p12 in a text edit?</p>
<b>B.6</b>	<p>The crt format is in encoded in binary. If we want to export to a Base64 format, we can use DER:</p> <p><code>openssl x509 -inform pem -outform pem -in ca.crt -out ca.cer</code></p> <p>and for My Little Corp:</p> <p><code>openssl x509 -inform pem -outform pem -in ia.crt -out ia.cer</code></p>	<p>View each of the output files in a text editor (ca.cer and then ia.cer). What can you observe from the format:</p> <p>Which are the standard headers and footers used:</p>

**B.7** Enter and run the following program, and verify its operation:

```
import OpenSSL.crypto
from OpenSSL.crypto import load_certificate_request, FILETYPE_PEM

csr = '''-----BEGIN NEW CERTIFICATE REQUEST-----
MIICyTCCABECAQAwajELMAkGA1UEBhMCVUSxDTAhBgNVBAGTBEE5vbmUXEjAQBGNV
BACTCUVkaw5idXJnaDEXMBUGA1UEChMOTXkgTG10dGx1IENvcnAxDDAKBgNVBAST
A01MQZERMA8GA1UEAxMITUXDLM5vbmUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAw
ggEKAoIBAQCuQE68qgssJ210wGxfKjCX3PG/RgSb5VpAp2rzavx71M9Bhg9kuORE
OP7BQC3E6DGu+xba3NdnhrHAFNa+hh9dntZr1xb98aM5q9+Tum76V1toIseOMDdu
UE9IpxXoFvD6b0inbFZnhrjFj3XUuzIIqvviZW4rIOxzgbwqZ5+F7YpP8d59ewW0
6ixzJKoeE/+Gw7Slsdr1+QQAUAx05MHTweMYbZEhir2M8f1RA4o81zEd2twCK85F
6VS/EkCzUG1cqDBQQ7D2S9MwN8Zk2P7CS8/yZx7uRTmT1t3UWKLuyIN0TU3IjCeY
t53P6C+9DT6UD0fDFZRCmPOH+qb6/YBAgMBAAGGjAYBgkqhkiG9w0BCQcxCMJ
UXdlcnR5MTIzMA0GCSqGSIb3DQEBBQUAA4IBAQCqpXjmaQf2/o/xbNZG5ggAV8yv
d6rSabnov5zIkciT9NQXSPJEi84u7CbcRiYqY5h7XlMwJv476mAGbgAVZB2Zh1p
qLa1+lx9xwhFbuLHNRxZcUMM0g9KQZaZTKAQd1DVU/vPZRjq+EHGoPfG7R9QKGD0
k1b4DqOvInWLOs+yuWT7YYtwdr2TNKPPcBqBzCYzrWL6UaUN7LYFpNn4BbqXRgVw
iManUh9fvLMe7oreYfTaevXT/506Sj9WvQFXTCLtRhs+M30q22/wUK0ZZ8APjpwf
rQMegvZXxEIO3xEGRBi5/wXJxsawRLCM3ZSGPu/ws950oM5Ahn8K8HBdkubQ
-----END NEW CERTIFICATE REQUEST-----'''
```



```
4PvFq+e7ipARgI5ZM+GZx6mpCz44DT00JkwfRdf+BtrsaC0q68eTf2XhY0sq4fkH
Q0uA0aVog3f5iJxCa3Hp5gxbJQ6zV6kJ0TESuaa0hEko9sdpCoPOnRBm2i/XRD2D
6iNh8f8z0ShGsFqjDgFHyF3o+lUyj+UC6H1QW7bn
-----END CERTIFICATE REQUEST-----
```

What are the details on the requests?

## C Elliptic Curve Key Creation

Elliptic curve key pairs are increasingly used within corporate Web sites.

In Openssl we can view the curves with the `ecparam` option:

```
openssl ecparam -list_curves
```

Outline some of the curve names:

By performing an Internet search, which are the most popular curves (and where are they used)?

We can create our elliptic parameter file with:

```
openssl ecparam -name secp256k1 -out secp256k1.pem
```

Now view the details with:

```
openssl ecparam -in secp256k1.pem -text -param_enc explicit -noout
```

What are the details of the key?

Now we can create our key pair:

```
openssl ecparam -in secp256k1.pem -genkey -noout -out mykey.pem
```

Now we will encrypt your key pair (and add a password), and convert it into a format which is ready to be converted into a digital certificate:

```
openssl ec -aes-128-cbc -in mykey.pem -out enckey.pem
```

Finally we will convert into a DER format, so that we can import the keys into a system:

```
openssl ec -in enckey.pem -outform DER -out enckey.der
```

Examine each of the files created and outline what they contain:

Now pick another elliptic curve type and perform the same operations as above. Which type did you use?

Outline the commands used:

If you want to create a non-encrypted version (PFX), which command would you use:

Go to [www.cloudflare.com](http://www.cloudflare.com) and examine the digital certificate on the site.

What is the public key method used?


What is the size of the public key?

What is the curve type used?

## **E PFX files**

We have a root certificate authority of My Global Corp, which is based in Washington, US, and the administrator is [admin@myglobalcorp.com](mailto:admin@myglobalcorp.com) and we are going to issue a certificate to

My Little Corp, which is based in Glasgow, UK, and the administrator is admin@mylittlecorp.com.

No	Description	Result
E.1	We will now view some PFX certificate files, and which are protected with a password:   <b>Web link (Digital Certificates):</b> <a href="http://asecuritysite.com/encryption/digitalcert2">http://asecuritysite.com/encryption/digitalcert2</a>	For Certificate 1, can you open it in the Web browser with an incorrect password:  Now enter “apples” as a password, and record some of the key details of the certificate:  Now repeat for Certificate 2:
E.2	Now with the PFX files (contained in the ZIP files from the Web site), try and import them onto your computer. Try to enter an incorrect password first and observe the message.	Was the import successful?  If successful, outline some of the details of the certificates:

## F Cracking Certificates

Digital certificates are often protected with a simple password. With this we can use a Python program to try various passwords on the certificate, and if it does not create an exception, then we have found the required password. First download the following pfx files:

 [https://asecuritysite.com/cert\\_crack.zip](https://asecuritysite.com/cert_crack.zip)

Now for fred.pfx, crack the password with the following code:

```
import OpenSSL
from cryptography import x509
from cryptography.hazmat.backends import default_backend

str="fred.pfx"
passwords=["ankle","battery","password","bill","apple","apples","orange"]

for password in passwords:
    try:
        pfx = open(str, 'rb').read()
```



```

p12 = OpenSSL.crypto.load_pkcs12(pfx, password)
print "Found: ",password

privkey=OpenSSL.crypto.dump_privatekey(OpenSSL.crypto.FILETYPE_PEM,
p12.get_privatekey())

cert=OpenSSL.crypto.dump_certificate(OpenSSL.crypto.FILETYPE_PEM,
p12.get_certificate())

cert = x509.load_pem_x509_certificate(cert, default_backend())

print " Issuer: ",cert.issuer
print " Subect: ",cert.subject
print " Serial number: ",cert.serial_number
print " Hash: ",cert.signature_hash_algorithm.name
print privkey
print certificate

except:

    print "Not working: ",password

```

What is the password?

The files bill01.pfx, bill02.pfx ... bill18.pfx have a password which are fruits. Can you determine the fruits used:

The files country01.pfx, country02.pfx ... country06.pfx have a password which are countries. Can you determine the countries used:

## G Setting up a certificate on a Web site

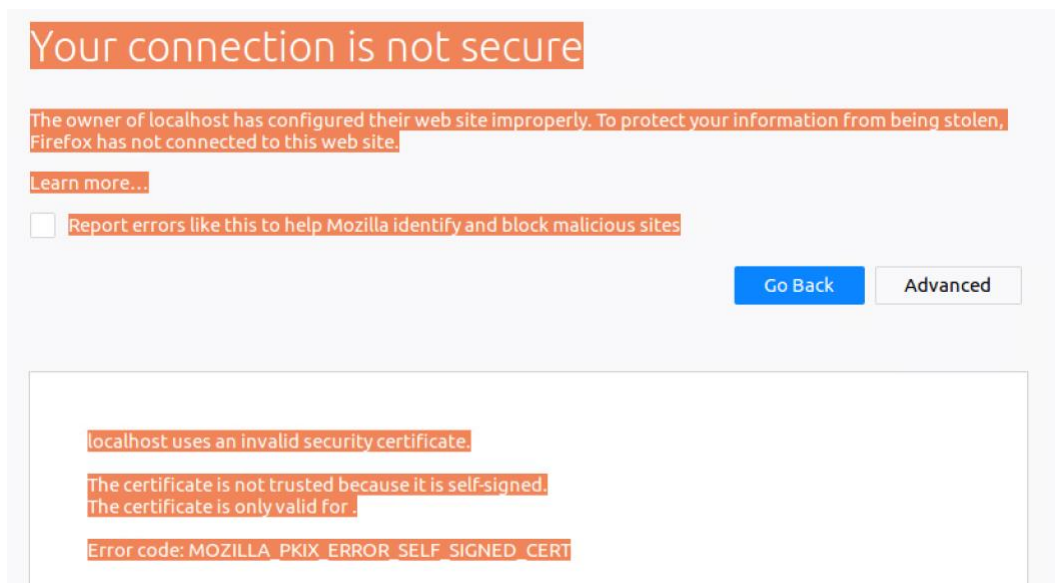
**G.1** Now we will enable HTTPs on an Apache Web Server, and install a digital certificate. Execute the following commands:

```

sudo a2enmod ssl
service apache2 restart
openssl genrsa -out ca.key 2048
sudo openssl req -nodes -new -key ca.key -out ca.csr
sudo openssl x509 -req -days 365 -in ca.csr -signkey ca.key -out ca.crt
sudo mkdir /etc/apache2/ssl
sudo cp ca.crt ca.key ca.csr /etc/apache2/ssl/
sudo nano /etc/apache2/sites-enabled/000-default.conf
sudo /etc/init.d/apache2 restart

```

HTTPs should now be enabled with a self-signed certificate. If you try <https://localhost>, you will have to add an exception to view the page, as we are using a self-signed certificate:

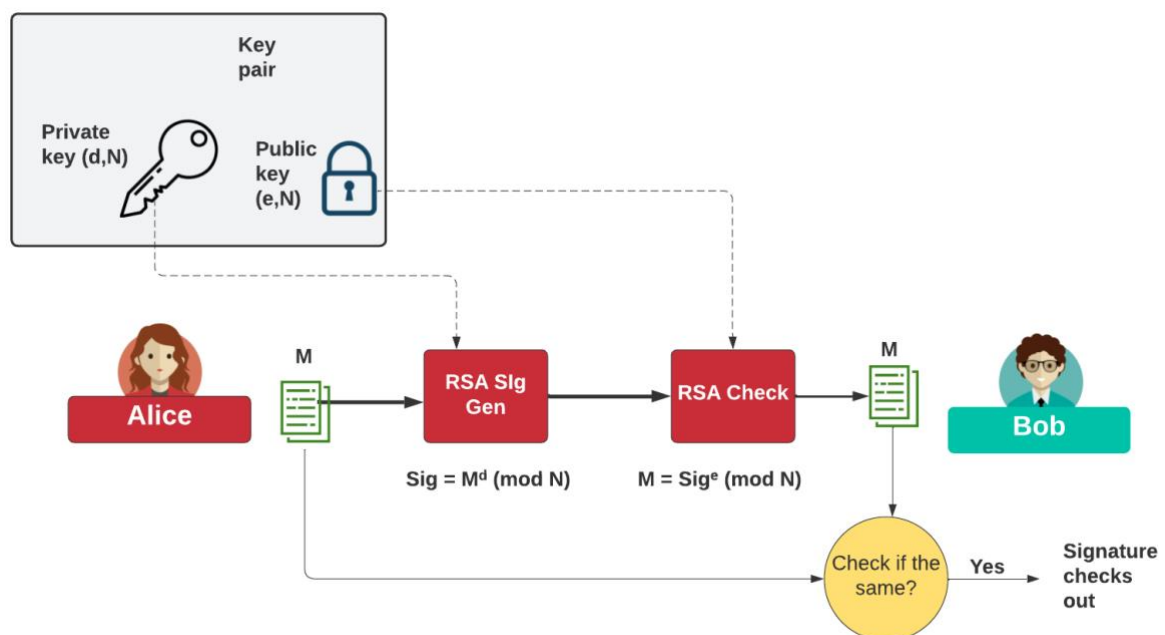


## AWS Digital Signing

Within digital signatures, we have two main signatures: RSA, ECDSA and EdDSA. In AWS, we can implement RSA (with PSS) and ECDSA.

### RSA PSS Signatures

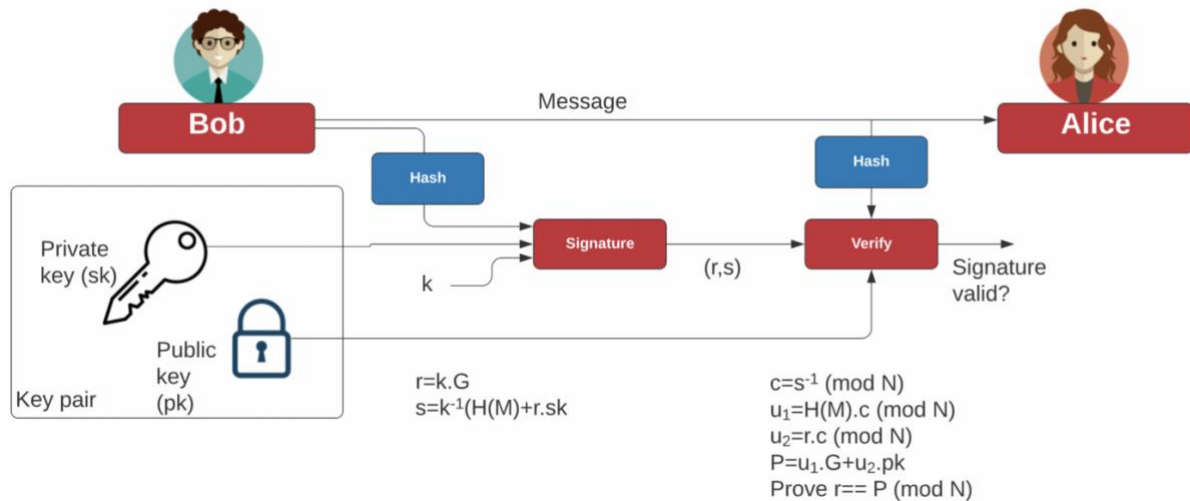
In digital signing, we use our private key to sign for a message, and then the proof of signing is done with our public key. This happens for a Bitcoin transaction, and where we take the private key from our wallet and then sign for a transaction. The public key is then used to prove that the user signing the transaction. While Bitcoin uses ECDSA, we can also use RSA signing. A common method is RSASSA\_PSS\_SHA\_256.



Undertake the lab here.

## ECDSA Signatures

In digital signing, we use our private key to sign for a message, and then the proof of signing is done with our public key. This happens for a Bitcoin transaction, and where we take the private key from our wallet and then sign for a transaction. The public key is then used to prove that the user signing the transaction.



Undertake the lab here.

## What I should have learnt from this lab?

The key things learnt:

- Understand how digital certificates are generated and ported onto systems.
- Identifying problems with digital certificates on sites.
- Understand how Python could be used in the analysis of certificates.