# 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** | From:  🕮 **Web link (Digital Certificate):**  <http://asecuritysite.com/encryption/digitalcert>  Open up Certificate 1 and identify the following: | Serial number: 702958  Effective date: 4/24/2008 8:18:42 PM  Name: CN=Fred Smith, OU=None, E=fred@home, O=Nowhere, L=Edinburgh, S=Lothian, C=GB  Issuer: CN=Fred Smith, OU=None, E=fred@home, O=Nowhere, L=Edinburgh, S=Lothian, C=GB  What is CN used for: Common Name  [It is a part of a Distinguished Name (DN) that identifies a certificate subject or issuer](https://knowledge.digicert.com/solution/SO7239.html)  [In PKI, CN can be used to specify a domain name, an email address, a user name or a device name](https://knowledge.digicert.com/solution/SO7239.html)  What is ON used for:  ON stands for Organization Name. [It is a part of a Distinguished Name (DN) that identifies an organization that owns a certificate subject or issuer](https://knowledge.digicert.com/generalinformation/INFO1745.html). [In PKI, ON can be used to specify the name of a company, a department, or a group](https://knowledge.digicert.com/generalinformation/INFO1745.html). For example, ON=IBM means that the certificate is issued for IBM.  What is O used for:  O stands for Organization. [It is a part of a Distinguished Name (DN) that identifies an organization that owns a certificate subject or issuer](https://www.rfc-editor.org/rfc/rfc1779). [In PKI, O can be used to specify the name of a company, a department, or a group](https://www.rfc-editor.org/rfc/rfc1779). For example, O=IBM means that the certificate is issued for IBM.  What is L used for:  L stands for Locality Name. [It is a part of a Distinguished Name (DN) that identifies a locality or city where a certificate subject or issuer is located](https://www.rfc-editor.org/rfc/rfc1779). [In PKI, L can be used to specify the name of a city, town, or region](https://knowledge.digicert.com/generalinformation/INFO1745.html). For example, L=Basingstoke means that the certificate is issued for Basingstoke. |
| **A.2** | Now open-up the ZIP file for the certificate (Certificate 3), and view the DER file. | What other information can you gain from the certificate:  What is the size of the public key:  256 ECC  Which hashing method has been used:  Sha256  Is the certificate trusted on your system: [Yes~~][No]~~ |
| **A.3** | Make a connection to the **www.live.com** Web site:  openssl s\_client -connect www.live.com:443 | Can you identify the certificate chain?  DigiCert Inc Global Root CA  DigiCert Inc Cloud Services  Microsoft Corporation  What is the subject on the certificate?  subject=C = US, ST = Washington, L = Redmond, O = Microsoft Corporation, CN = outlook.live.com  Who is the issuer on the certificate?  DigiCert Inc, DigiCert Cloud Services |
| **A.4** | 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 testfire.net (you can try both https and http for the connection). | * Run a scan from SSLLabs on testfire.net. What do you observe from the result?   Certificate Name mismatch  Not Trusted   * What is the SSLLabs rating on this site? Is it "A", "B", "C", "D", "E" or "F"?   T but if trust issues are ignored, B   * What does a “T” rating identify?   [A T rating in SSL Labs means that the site certificate is not trusted by your browser or device](https://blog.qualys.com/product-tech/2014/06/17/ssl-labs-new-grades-for-trust-t-and-mismatch-m-issues). This could be because the certificate is self-signed, expired, revoked, or issued by an unknown authority. [A T rating does not reflect the actual security level of the server configuration](https://github.com/ssllabs/research/wiki/SSL-Server-Rating-Guide). [If you want to check the SSL configuration of a web server, you can use SSL Server Test](https://www.ssllabs.com/ssltest/).   * Can you locate another "T" rated site? |

**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 | No One | 29/10/2011 to 28/10/2013 | RSA 1024 | No One | No ne | SHA1 | No |
| 3 | \*.google.com | 08/02/2023 to 03/05/2023 | ECC 256 | GTS CA 1C3 | GTS Root R1 | SHA256 | No |
| 4 | [www.cisco.com](http://www.cisco.com) | 10/07/2012 to 11/07/2013 | RSA 1024 | www.cisco.com | www.cisco.com | SHA1 | No |
| 5 | Microsoft.com | 13/01/2023 to 08/01/2024 | RSA 2048 | Microsoft Azure TLS Issuing CA 05 | DigiCert Global Root G2 | SHA384 | No |
| 6 | Oracle.com | 14/02/2023 to 26/02/2024 | RSA 2048 | DigiCert TLS RSA SHA256 2020 CA1 | DigiCert | SHA256 | No |

**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.1** | Create your own certificate from:  🕮 **Web link (Create Certificate):**  http://asecuritysite.com/encryption/createcert  Add in your own details. | View the certificate, and verify some of the details on the certificate.  Can you view the DER file?  No, it’s invalid |

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](mailto:admin@mylittlecorp.com).

|  |  |  |
| --- | --- | --- |
| **No** | **Description** | **Result** |
| **B.2** | Create your RSA key pair with:  openssl genrsa -out ca.key 2048  Next create a self-signed root CA certificate ca.crt for **My Global Corp:**  openssl req -new -x509 -days 1826 -key ca.key -out ca.crt | How many years will the certificate be valid for?  5  Which details have you entered:  E = admin@mylittlecorp.com  CN = My Little Company  OU = Business  O = My Little Corp  L = Glasgow  S = Scotland  C = UK |
| **B.3** | Next go to Places, and from your Home folder, open up ca.crt and view the details of the certificate. | Which Key Algorithm has been used:  RSA 2048  Which hashing methods have been used:  SHA256  When does the certificate expire:  04/03/2028  Who is it verified by:  My Little Company  Who has it been issued to:  admin@mylittlecorp.com |
| **B.4** | Next we will create a subordinate CA (**My Little Corp**), and which will be used for the signing of the certificate. First, generate the key:  openssl genrsa -out ia.key 2048  Next we will request a certificate for our newly created subordinate CA:  openssl req -new -key ia.key -out ia.csr  We can then create a **certificate from the subordinate CA** certificate and **signed by the root CA**.  openssl x509 -req -days 730 -in ia.csr -CA ca.crt -CAkey ca.key -set\_serial 01 -out ia.crt | View the newly created certificate.  When does it expire:  04/03/2025  Who is the subject of the certificate:  Intermediate Rip Offs  Which is their country:  US  Who signed the certificate:  admin@mylittlecorp.com  Which is their country:  UK  What is the serial number of the certificate:  01  Check the serial number for the root certificate. What is its serial number: |
| **B.5** | If we want to use this certificate to digitally sign files and verify the signatures, we need to convert it to a PKCS12 file:  openssl pkcs12 -export -out ia.p12 -inkey ia.key -in ia.crt -chain -CAfile ca.crt | Can you view ia.p12 in a text edit?  No |
| **B.6** | The crt format is in encoded in binary. If we want to export to a Base64 format, we can use DER:  openssl x509 -inform pem -outform pem -in ca.crt -out ca.cer  and for My Little Corp:  openssl x509 -inform pem -outform pem -in ia.crt -out ia.cer | View each of the output files in a text editor (ca.cer and then ia.cer). What can you observe from the format:  ia.cer is smaller than the resulting ca.cer, and the content is totally different  Which are the standard headers and footers used:  -----BEGIN CERTIFICATE-----  -----END CERTIFICATE----- |

**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-----

MIICyTCCAbECAQAwajELMAkGA1UEBhMCVUsxDTALBgNVBAgTBE5vbmUxEjAQBgNV

BAcTCUVkaW5idXJnaDEXMBUGA1UEChMOTXkgTGl0dGxlIENvcnAxDDAKBgNVBAsT

A01MQzERMA8GA1UEAxMITUxDLm5vbmUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAw

ggEKAoIBAQCuQE68qgssJ210wGxfKjCX3PG/RgSb5VpAp2rzavx71M9Bhg9kUORE

OP7BQC3E6DGu+xba3NdnhrHAFNa+hH9dnTZrlxb98aM5q9+TUm76V1toIseOMDdU

UE9IpxXoFvD6b0inbFZnbrjFj3XUUzIIqvvizw4rIOxzgbWqZ5+F7YpP8d59eWW0

6iXzJKoeE/+Gw7Slsdr1+QQAUaX05MHTweMYbZEHir2M8f1RA4o81zEd2tWCK85F

6VS/EkCzUG1cqDBQQ7D2S9MWN8Zk2P7CS8/yZx7uRTmT1t3UWKLUyIN0TU3IjCeY

t53P6C+9DT6UD0fDFZRBCmPOH+qb6/YBAgMBAAGgGjAYBgkqhkiG9w0BCQcxCxMJ

UXdlcnR5MTIzMA0GCSqGSIb3DQEBBQUAA4IBAQCqpXjmaQf2/o/xbNZG5ggAV8yV

d6rSabnov5zIkcit9NQXsPJEi84u7CbcriYqY5h7XlMWjv476mAGbgAVZB2ZhIlp

qLal+lx9xwhFbuLHNRxZcUMM0g9KQZaZTkAQdlDVU/vPzRjq+EHGoPfG7R9QKGD0

k1b4DqOvInWLOs+yuWT7YYtWdr2TNKPpcBqbzCYzrWL6UaUN7LYFpNn4BbqXRgVw

iMAnUh9fvLMe7oreYfTaevXT/506Sj9WvQFXTcLtRhs+M30q22/wUK0ZZ8APjpwf

rQMegvzXXEIO3xEGrBi5/wXJxsawRLcM3ZSGPu/Ws950oM5Ahn8K8HBdKubQ

-----END NEW CERTIFICATE REQUEST-----'''

req = load\_certificate\_request(FILETYPE\_PEM, csr)

key = req.get\_pubkey()

key\_type = 'RSA' if key.type() == OpenSSL.crypto.TYPE\_RSA else 'DSA'

subject = req.get\_subject()

components = dict(subject.get\_components())

print "Key algorithm:", key\_type

print "Key size:", key.bits()

print "Common name:", components['CN']

print "Organisation:", components['O']

print "Organisational unit", components['OU']

print "City/locality:", components['L']

print "State/province:", components['ST']

print "Country:", components['C']

🕮 **Web link (CSR):**  
https://asecuritysite.com/encryption/csr

**B.8** Now check the signing on these certificate requests:

-----BEGIN NEW CERTIFICATE REQUEST-----

MIICyTCCAbECAQAwajELMAkGA1UEBhMCVUsxDTALBgNVBAgTBE5vbmUxEjAQBgNV

BAcTCUVkaW5idXJnaDEXMBUGA1UEChMOTXkgTGl0dGxlIENvcnAxDDAKBgNVBAsT

A01MQzERMA8GA1UEAxMITUxDLm5vbmUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAw

ggEKAoIBAQCuQE68qgssJ210wGxfKjCX3PG/RgSb5VpAp2rzavx71M9Bhg9kUORE

OP7BQC3E6DGu+xba3NdnhrHAFNa+hH9dnTZrlxb98aM5q9+TUm76V1toIseOMDdU

UE9IpxXoFvD6b0inbFZnbrjFj3XUUzIIqvvizw4rIOxzgbWqZ5+F7YpP8d59eWW0

6iXzJKoeE/+Gw7Slsdr1+QQAUaX05MHTweMYbZEHir2M8f1RA4o81zEd2tWCK85F

6VS/EkCzUG1cqDBQQ7D2S9MWN8Zk2P7CS8/yZx7uRTmT1t3UWKLUyIN0TU3IjCeY

t53P6C+9DT6UD0fDFZRBCmPOH+qb6/YBAgMBAAGgGjAYBgkqhkiG9w0BCQcxCxMJ

UXdlcnR5MTIzMA0GCSqGSIb3DQEBBQUAA4IBAQCqpXjmaQf2/o/xbNZG5ggAV8yV

d6rSabnov5zIkcit9NQXsPJEi84u7CbcriYqY5h7XlMWjv476mAGbgAVZB2ZhIlp

qLal+lx9xwhFbuLHNRxZcUMM0g9KQZaZTkAQdlDVU/vPzRjq+EHGoPfG7R9QKGD0

k1b4DqOvInWLOs+yuWT7YYtWdr2TNKPpcBqbzCYzrWL6UaUN7LYFpNn4BbqXRgVw

iMAnUh9fvLMe7oreYfTaevXT/506Sj9WvQFXTcLtRhs+M30q22/wUK0ZZ8APjpwf

rQMegvzXXEIO3xEGrBi5/wXJxsawRLcM3ZSGPu/Ws950oM5Ahn8K8HBdKubQ

-----END NEW CERTIFICATE REQUEST-----

Subject: C = UK, ST = None, L = Edinburgh, O = My Little Corp, OU = MLC, CN = MLC.none

challengePassword :Qwerty123

-----BEGIN NEW CERTIFICATE REQUEST-----

MIIDPzCCAqgCAQAwZDELMAkGA1UEBhMCQ04xCzAJBgNVBAgTAmJqMQswCQYDVQQH

EwJiajERMA8GA1UEChMIbXhjei5uZXQxETAPBgNVBAsTCG14Y3oubmV0MRUwEwYD

VQQDEwx3d3cubXhjei5uZXQwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMQ7

an4v6pHRusBA0prMWXMWJCXY1AO1H0X8pvZj96T5GWg++JPCQE9guPgGwlD02U0B

NDoEABeD1fwyKZ+JV5UFiOeSjO5sWrzIupdMI7hf34UaPNxHo6r4bLYEykw/Rnmb

GKnNcD4QlPkypE+mLR4p0bnHZhe3lOlNtgd6NpXbAgMBAAGgggGZMBoGCisGAQQB

gjcNAgMxDBYKNS4yLjM3OTAuMjB7BgorBgEEAYI3AgEOMW0wazAOBgNVHQ8BAf8E

BAMCBPAwRAYJKoZIhvcNAQkPBDcwNTAOBggqhkiG9w0DAgICAIAwDgYIKoZIhvcN

AwQCAgCAMAcGBSsOAwIHMAoGCCqGSIb3DQMHMBMGA1UdJQQMMAoGCCsGAQUFBwMB

MIH9BgorBgEEAYI3DQICMYHuMIHrAgEBHloATQBpAGMAcgBvAHMAbwBmAHQAIABS

AFMAQQAgAFMAQwBoAGEAbgBuAGUAbAAgAEMAcgB5AHAAdABvAGcAcgBhAHAAaABp

AGMAIABQAHIAbwB2AGkAZABlAHIDgYkAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAADANBgkqhkiG9w0BAQUFAAOBgQBIKHVhHb9FZdVLV4VZ

9DK4aBSuYY//jlIpvsfMIdHXfAsuan7w7PH87asp1wdb6lD9snvLZix1UGK7VQg6

wUFYNlMqJh1m7ITVvzhjdnx7EzCKkBXSxEom4mwbvSNvzqOKAWsDE0gvHQ9aCSby

NFBQQMoW94LqrG/kuIQtjwVdZA==

-----END NEW CERTIFICATE REQUEST-----

Subject: C = CN, ST = bj, L = bj, O = mxcz.net, OU = mxcz.net, CN = [www.mxcz.net](http://www.mxcz.net)

-----BEGIN CERTIFICATE REQUEST-----

MIIByjCCATMCAQAwgYkxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlh

MRYwFAYDVQQHEw1Nb3VudGFpbiBWaWV3MRMwEQYDVQQKEwpHb29nbGUgSW5jMR8w

HQYDVQQLExZJbmZvcm1hdGlvbiBUZWNobm9sb2d5MRcwFQYDVQQDEw53d3cuZ29v

Z2xlLmNvbTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEApZtYJCHJ4VpVXHfV

IlstQTlO4qC03hjX+ZkPyvdYd1Q4+qbAeTwXmCUKYHThVRd5aXSqlPzyIBwieMZr

WFlRQddZ1IzXAlVRDWwAo60KecqeAXnnUK+5fXoTI/UgWshre8tJ+x/TMHaQKR/J

cIWPhqaQhsJuzZbvAdGA80BLxdMCAwEAAaAAMA0GCSqGSIb3DQEBBQUAA4GBAIhl

4PvFq+e7ipARgI5ZM+GZx6mpCz44DTo0JkwfRDf+BtrsaC0q68eTf2XhYOsq4fkH

Q0uA0aVog3f5iJxCa3Hp5gxbJQ6zV6kJ0TEsuaaOhEko9sdpCoPOnRBm2i/XRD2D

6iNh8f8z0ShGsFqjDgFHyF3o+lUyj+UC6H1QW7bn

-----END CERTIFICATE REQUEST-----

Subject: C = US, ST = California, L = Mountain View, O = Google Inc, OU = Information Technology, CN = www.google.com

What are the details on the requests?

## C Elliptic Curve Key Creation

Elliptic curve key pairs are increasing 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:

secp256k1 : SECG curve over a 256 bit prime field

prime256v1: X9.62/SECG curve over a 256 bit prime field

sect113r1 : SECG curve over a 113 bit binary field

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

Curve25519: This curve was designed by Daniel J. Bernstein and is widely used in the cryptographic community. It has a 128-bit security level and is used in applications such as the Signal messaging protocol and the Tor anonymity network.

NIST P-256: This is one of the elliptic curves recommended by the National Institute of Standards and Technology (NIST) for use in cryptographic applications. It has a 128-bit security level and is used in applications such as SSL/TLS (and the Bitcoin protocol)=wrong.

NIST P-384: This is another elliptic curve recommended by NIST, with a 192-bit security level. It is used in applications such as SSL/TLS and the US government's Suite B cryptographic algorithms.

Curve448: This is another elliptic curve designed by Daniel J. Bernstein, with a 224-bit security level. It is used in applications such as the WireGuard VPN protocol.

Brainpool curves: These are a set of elliptic curves standardized by the German Federal Office for Information Security (BSI). They come in various sizes, with security levels ranging from 80 bits to 512 bits, and are used in applications such as SSL/TLS and the German electronic identity card.

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?

EC-Parameters: (256 bit)

Field Type: prime-field

Prime:

00:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:

ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:fe:ff:

ff:fc:2f

A: 0

B: 7 (0x7)

Generator (uncompressed):

04:79:be:66:7e:f9:dc:bb:ac:55:a0:62:95:ce:87:

0b:07:02:9b:fc:db:2d:ce:28:d9:59:f2:81:5b:16:

f8:17:98:48:3a:da:77:26:a3:c4:65:5d:a4:fb:fc:

0e:11:08:a8:fd:17:b4:48:a6:85:54:19:9c:47:d0:

8f:fb:10:d4:b8

Order:

00:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:ff:

ff:fe:ba:ae:dc:e6:af:48:a0:3b:bf:d2:5e:8c:d0:

36:41:41

Cofactor: 1 (0x1)

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:

cat mykey.pem

-----BEGIN EC PRIVATE KEY-----

MHQCAQEEIGe7u8nnBtJ8KOh4kyfOzTkqN3Q6JErjQ3Q8FH4ltb0QoAcGBSuBBAAK

oUQDQgAETD5rTzxzse4reXdFG1cb42rRO+edMYKA1v5E3X2SrHkSxljR27qsztcP

KJP6Kl/1k2ZAi3ndmq5HzEiCZgBZ3w==

-----END EC PRIVATE KEY-----

cat enckey.pem

-----BEGIN EC PRIVATE KEY-----

Proc-Type: 4,ENCRYPTED

DEK-Info: AES-128-CBC,158F110BC6CB88FC138274BF0AE49FB9

enDNhqfJZDwBbs5Uxfzsi3Gec3gFg59Z9vPkukzEZZm8JMoFCvaqby5ASR1jFdjX

QKlx/FyTlW92d3KkEjaEviNVv5eOLAprM4JwDsSDlqIh7IdYJfGeM1ojtMajM2zY

sqNy1WeFJb9Pp3dPczRjWnuqjvJzZiZ+FwIQs6uQLD8=

-----END EC PRIVATE KEY-----

cat enckey.der

0t g�����|(�x�'��9\*7t:$J�Ct<~%���+�

�DBL>kO<s��+ywEj�;�1����D�}��y�X�ۺ���(��\*\_��f@�yݚ�G�H�fY�

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

brainpoolP512t1

Outline the commands used:

openssl ecparam -name brainpoolP512t1 -out brainpoolP512t1.pem

cat brainpoolP512t1.pem

-----BEGIN EC PARAMETERS-----

BgkrJAMDAggBAQ4=

-----END EC PARAMETERS-----

openssl ecparam -in brainpoolP512t1.pem -text -param\_enc explicit -noout

openssl ecparam -in brainpoolP512t1.pem -genkey -noout -out brainkey.pem

openssl ec -aes-128-cbc -in brainkey.pem -out brainenckey.pem

openssl ec -in brainenckey.pem -outform DER -out brainenckey.der

cat brainkey.pem

-----BEGIN EC PRIVATE KEY-----

MIHaAgEBBEB2pPmANdlLfuJkXB8lWxvsoyKaGWit6pY13WV+MPtwedUUPEGWpr3e

eQnl4OrQCWL7qpzJLTC0SBBN6FiRr3Y0oAsGCSskAwMCCAEBDqGBhQOBggAEVwAB

pDliDPyDxuJGFC38hlHV7dEz1zVnAGYFBzNb7A0fDwmQ+Bv+1jnhjJRnuXc9y5WG

ifvTRGKL+PeRFiF9uTgvwKUbGYhotxkGHjimTBjtyur8KHLMITVynsnZ3Ip050Vp

+yDdCkfhmIplGSxVNGdbaVRThqC2VUUB7ceNA0w=

-----END EC PRIVATE KEY-----

cat brainenckey.pem

-----BEGIN EC PRIVATE KEY-----

Proc-Type: 4,ENCRYPTED

DEK-Info: AES-128-CBC,C18F13467BBE331934049AA1DF7BBC5F

Zcp98ZIEwK/scErKs24rmPDb9zCozqkfwIcKWE/93aEc3w0kkT2wwEHxurSofzTr

03s1LIg+N8eVGeG1iwlYhr6pahHsvSkML9V8x4tn+6DseSo9eClV/4sw7mrZMrOB

KtZJrwyMoI96GTp9oXTr3l9Euv9V2RnsCsawgeHhu8bwu/0FCMv9SBdgO6TB+9wq

WTiOD4q40Ej6Pu4SAf5b5v3yrT6ve3Zwjk/OUt9dOZwiMtA/ZcVhr3pZ+DSyWUtA

Ie//ZGRgeLIXkeTea0G9E8S8ZIm3soYiLQt1sThyHX0=

-----END EC PRIVATE KEY-----

cat brainenckey.der

0��@v���5�K~�d\­%["�h��5�e~0�py�<A����y ���� b����-0�HM�X��v4�

+$�����W�9b

­ ���9ጔg�w=˕����Db����!}�8/��h�‑8�L����(r�!5r���܊t�Ei� � ����F-��Q���3�5gf3[�

Gᘊe,U4g[iTS���UE�ǍLubuntu@ip-172-31-62-50:~$

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

openssl pkcs12 -export -out cert.pfx -inkey private\_key.pem -in certificate.pem

pkcs12 is the OpenSSL command for creating and manipulating PKCS#12 files.

-export specifies that the PKCS#12 file should be created.

-out cert.pfx specifies the output file name and location for the PKCS#12 file.

-inkey private\_key.pem specifies the path to the private key file in PEM format.

-in certificate.pem specifies the path to the certificate file in PEM format.

You will be prompted to enter and confirm a password for the PFX file. Once you have entered the password, the command will create a PFX file containing the private key and certificate in encrypted form, protected by the password you provided.

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

What is the public key method used?

Elliptic Curve Public Key

What is the size of the public key?

00 04 17 82 5D 60 4F 52 82 5F D1 71 4A CE FA F5 28 6E 90 C0 F3 43 8C 81 D9 CE A6 C9 DB 7A 05 73 5E D0 6D 64 20 BF DD 89 CE 1A 4B 31 9B 8C 31 90 7B A6 54 00 CF 55 56 26 51 A0 B9 8A C9 5B 58 D4 E4 F3

520 bits

What is the curve type used?

secp256r1 (also known as prime256v1)

## 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:  🕮 **Web link (Digital Certificates):**  <http://asecuritysite.com/encryption/digitalcert2> | For Certificate 1, can you open it in the Web browser with an incorrect password:  No  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.  The password you entered is incorrect  Serial: 00FCA689  Serial: 00FCA689 | Was the import successful?  yes  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

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:

Fred=apples

Bill01 = orange

Bill02=lemon

Bill03=kiwi

Bill04 = strawberry

Bill05=raspberry

Bill06=blackberry

Bill07=

Bill08=grapes

Bill09=pineapple

Bill10=blueberry

Bill11=pear

Bill12=coconut

Bill13=apricot

Bill14=guava

Bill15=tangerine

Bill16=pomegranate

Bill17=

Bill18=

Country01=Germany

Country02=Finland

Country03=Russia

Country04=Iceland

Country05=Estonia

Country06=France

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

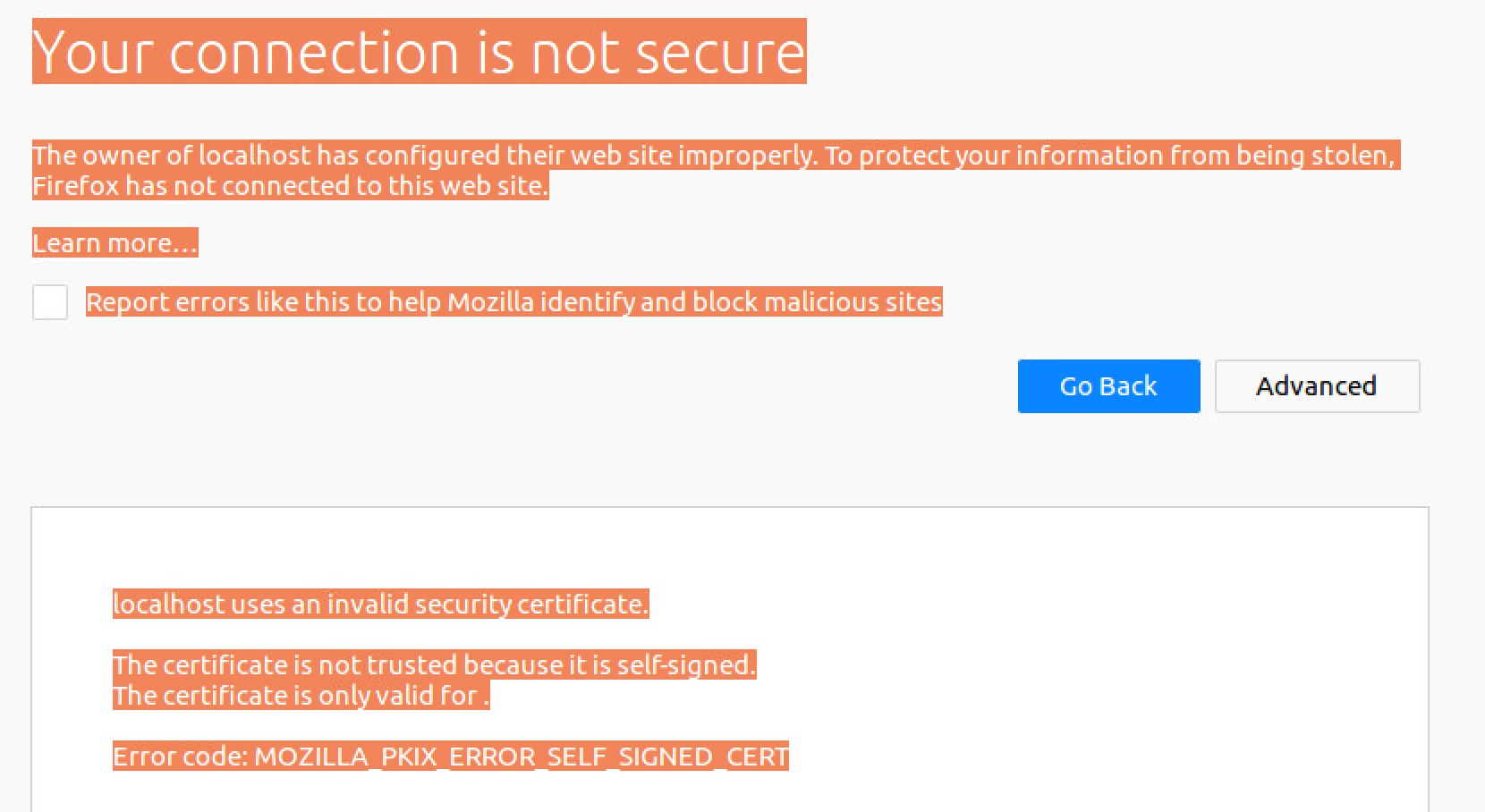
sudo mkdir /etc/apache2/ssl

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:



## 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.