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, and then

🕮 **Web link (Weekly activities):** https://asecuritysite.com/esecurity/unit06

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: Effective date: Name: Issuer: What is CN used for: What is ON used for: What is O used for: What is L used for:
A.2	Now open-up the ZIP file for the certificate, and view the CER file.	What other information can you gain from the certificate: What is the size of the public key: Which hashing method has been used: Is the certificate trusted on your system: [Yes][No]

A.3 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.4 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?

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.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 Little Corp:
	
	`openssl req -new -x509 -days 1826 -key ca.key -out ca.crt`
		How many years will the certificate be valid for?
		
		Which details have you entered:

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: Which hashing methods have been used: When does the certificate expire: Who is it verified by: Who has it been issued to:

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:
		
		Who is the subject of the certificate:
		
		Which is their country:
		
		Who signed the certificate:
		
		Which is their country:
		
		What is the serial number of the certificate:
		
		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:	Can you view ia.p12 in a text edit?

	`openssl pkcs12 -export -out ia.p12 -inkey ia.key -in ia.crt -chain -CAfile ca.crt`	

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:
		
		Which are the standard headers and footers used:

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
ggEKAoIBAQCuQE68qgssJ21OwGxfKjCX3PG/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/Ws95OoM5Ahn8K8HBdKubQ
-----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

D.8 Now check the signing on these certificate requests:

```
-----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-----
```

```
-----BEGIN CERTIFICATE REQUEST-----
MIIByjCCATMCAQAwgYkxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlh
MRYwFAYDVQQHEw1Nb3VudGFpbiBWaWV3MRMwEQYDVQQKEwpHb29nbGUgSW5jMR8w
HQYDVQQLExZJbmZvcm1hdGlvbiBUZWNobm9sb2d5MRcwFQYDVQQDEw53d3cuZ29v
Z2xlLmNvbTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEApZtYJCHJ4VpVXHfV
IlstQTlO4qC03hjX+ZkPyvdYd1Q4+qbAeTwXmCUKYHThVRd5aXSqlPzyIBwieMZr
WFlRQddZ1IzXAlVRDWwAo60KecqeAXnnUK+5fXoTI/UgWshre8tJ+x/TMHaQKR/J
cIWPhqaQhsJuzZbvAdGA8OBLxdMCAwEAAaAAMA0GCSqGSIb3DQEBBQUAA4GBAIhl
4PvFq+e7ipARgI5ZM+GZx6mpCz44DToOJkwfRDf+BtrsaC0q68eTf2XhYOsq4fkH
Q0uA0aVog3f5iJxCa3Hp5gxbJQ6zV6kJ0TEsuaaOhEko9sdpCoPOnRBm2i/XRD2D
6iNh8f8z0ShGsFqjDgFHyF3o+lUyj+UC6H1QW7bn
-----END CERTIFICATE REQUEST-----
```


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:

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 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?

D Simple Key Distribution Centre (KDC)

Rather than use PKI, we can setup a KDC, and where Bob and Alice can have long-term keys, and these can be used to generate a session key for them to use. Enter the following Python program, and prove its operation:

```
import hashlib
import sys
import binascii
import Padding
import random

from Crypto.Cipher import AES
from Crypto import Random

msg="test"

def encrypt(word,key, mode):
        plaintext=pad(word)
        encobj = AES.new(key,mode)
        return(encobj.encrypt(plaintext))

def decrypt(ciphertext,key, mode):
        encobj = AES.new(key,mode)
        rtn = encobj.decrypt(ciphertext)
        return(rtn)

def pad(s):
         return s
        extra = len(s) % 16
        if extra > 0:
                s = s + (' ' * (16 - extra))
        return s


rnd = random.randint(1,2**128)

keyA= hashlib.md5(str(rnd)).digest()

rnd = random.randint(1,2**128)

keyB= hashlib.md5(str(rnd)).digest()

print 'Long-term Key Alice=',binascii.hexlify(keyA)
print 'Long-term Key Bob=',binascii.hexlify(keyB)
```

```
rnd = random.randint(1,2**128)
keySession= hashlib.md5(str(rnd)).hexdigest()

ya = encrypt(keySession,keyA,AES.MODE_ECB)
yb = encrypt(keySession,keyB,AES.MODE_ECB)

print "Encrypted key sent to Alice:",binascii.hexlify(ya)
print "Encrypted key sent to Bob:",binascii.hexlify(yb)

decipherA = decrypt(ya,keyA,AES.MODE_ECB)
decipherB = decrypt(yb,keyB,AES.MODE_ECB)

print "Session key:",decipherA
print "Session key:",decipherB
```

🕮 **Web link (Simple KDC):** https://asecuritysite.com/encryption/kdc01

The program above uses a shared 128-bit session key (generated by MD5). Now change the program so that you generate a 256-bit session key. What are the changes made:

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 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: 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 What I should have learnt from this lab?

The key things learnt:

- Understand how digital certificates are generated and ported onto systems.
- Understand how we can create a key pair for RSA and Elliptic Curve.
- How to setup a simple KDC.

Notes

To setup your Python to run Python 2.7:

```
sudo update-alternatives --set python /usr/bin/python2.7
```

To install a Python library use:

```
easy_install libname
```

or:

```
pip install libname
```