# CAS Cyber Security
# A5 Crypto Assignment II
# Sample Solution

| | |
|---|---|
| **Document Name:** | cas_cyber_sec_crypto_assignment_II_sample_solution_v0.1.docx |
| **Version:** | v0.1 |
| **Date of Delivery:** | December 11th, 2020 |
| **Authors:** | Cyrill Brunschwiler, Compass Security Schweiz AG |
| **Classification:** | TLP RED |

**Compass Security Schweiz AG**, Werkstrasse 20, Postfach 2038, CH-8645 Jona
T +41 55 214 41 60, team.csch@compass-security.com, www.compass-security.com

## Table of Contents

# 1 A5 Elliptic Curve Cryptography

A quick excurse to elliptic curves cryptography (ECC).

**1. What mathematical problem does ECC rely on?**

You maybe remember the discrete logarithm problem which is the base for ElGamal or Diffie-Hellmann. Both work on mathematical groups with a generator g that could produce the integers in the group. For elliptic curves, most is very similar with the major difference that the elements produced are dots on the elliptic curve.

Components

- the curve C
- the point G that could be used to generate a large group of points on C
- the random chosen integer x (private key)
- the point Y (public key) =xG

Finding x such that a given point Y on curve C, Y=xG is a hard problem – the ECDLP.

**2. Name reasons why ECC keys are significantly shorter than for other approaches to public-key cryptography?**

Obviously, the problem seems to be much harder to solve and thus allows for smaller keys.

**3. What are good key sizes when working with ECC and keeping information confidential for another decade?**

Until 2018

| Method | Date | Symmetric | Factoring Modulus | Discrete Logarithm Key | Discrete Logarithm Group | Elliptic Curve | Hash |
|---|---|---|---|---|---|---|---|
| [1] Lenstra / Verheul | 2028 | 92 | 2362 1888 | 162 | 2362 | 173 | 183 |
| [2] Lenstra Updated | 2028 | 87 | 1633 1958 | 174 | 1633 | 174 | 174 |
| [3] ECRYPT | 2018 - 2028 | 128 | 3072 | 256 | 3072 | 256 | 256 |
| [4] NIST | 2019 - 2030 | 112 | 2048 | 224 | 2048 | 224 | 224 |
| [5] ANSSI | 2021 - 2030 | 128 | 2048 | 200 | 2048 | 256 | 256 |
| [6] NSA | - | 256 | 3072 | - | - | 384 | 384 |
| [7] RFC3766 | - | - | - | - | - | - | - |

Until 3031

| Method | Date | Symmetric | Factoring Modulus | Discrete Logarithm Key | Discrete Logarithm Group | Elliptic Curve | Hash |
|---|---|---|---|---|---|---|---|
| [1] Lenstra / Verheul | 2031 | 94 | 2560 2080 | 166 | 2560 | 178 | 188 |
| [2] Lenstra Updated | 2031 | 89 | 1732 2118 | 178 | 1732 | 178 | 178 |
| [3] ECRYPT | 2029 - 2068 | 256 | 15360 | 512 | 15360 | 512 | 512 |
| [4] NIST | 2019 - 2030 & beyond | 128 | 3072 | 256 | 3072 | 256 | 256 |
| [5] ANSSI | > 2030 | 128 | 3072 | 200 | 3072 | 256 | 256 |
| [6] NSA | - | 256 | 3072 | - | - | 384 | 384 |
| [7] RFC3766 | - | - | - | - | - | - | - |

**4. Can you choose any curve as a base or are there concerns when choosing a curve?**

Choosing good constants is vital for the security of ECC. Due to historical developments, many researchers are concerned that NISTs P curve constants have been chosen to deliberately weaken implementations.

Thus, trust into curves and constants is vital. Therefore, industry has mostly adopted towards Curve25519 in hope it is an unbiased alternative.

# 2 A5 Digital Signatures

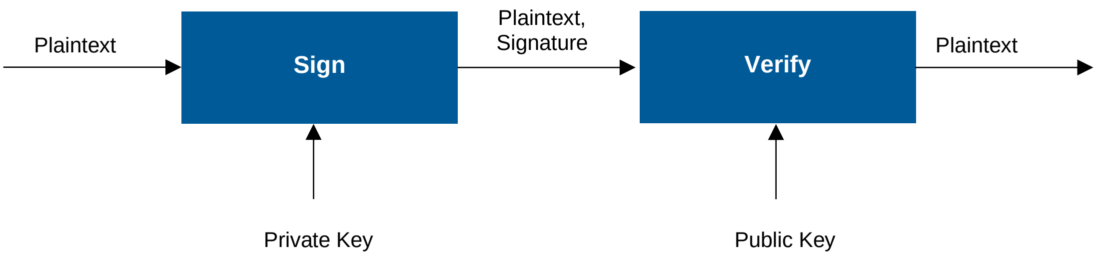
Digital signatures form the base of many real world applications. It is important to certify content with signatures to keep businesses around the world running and trusting each other. The service they could provide are authentication, integrity or non-repudiation.

**1. What does "non-repudiation of origin" and "non-repudiation of receipt" mean?**

These are often shortened as NRO and NRR. NRO means that the sender cannot deny having sent a specific message. NRR means that a recipient cannot deny having received a specific message.

Bottom line of this is, that if you send an e-mail message e.g. using S/MIME than the sender can proof you sent it. Following that, we have NRO. However, as long as you did not get a signed reply for your e-mail message, the criteria for NRR are not satisfied. Thus, the recipient could still deny having received your e-mail.

**2. Draw a simple signature scheme for the creation and verification of digital signatures. Label all relevant components and processes and explain how creation and verification basically works?**

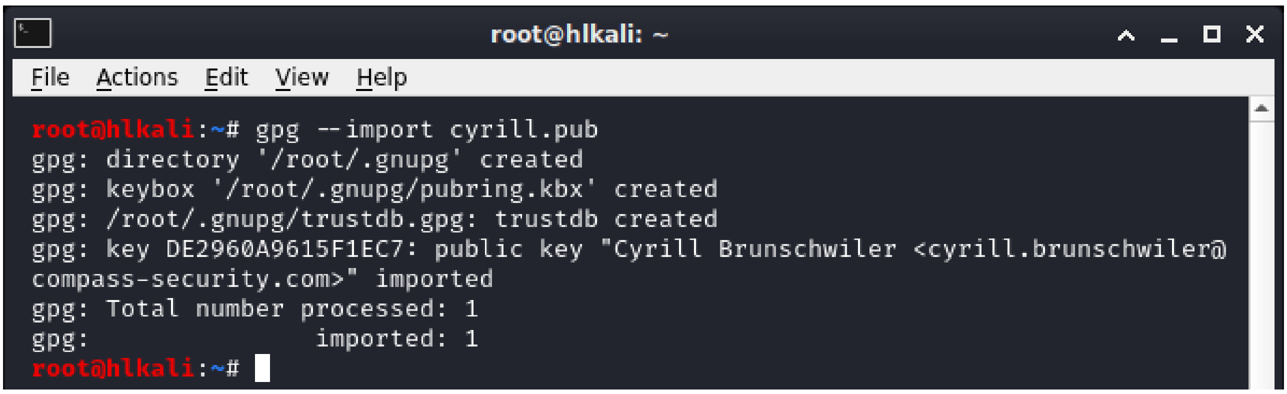



**3. Name three algorithms that could be used to create and verify signatures?**

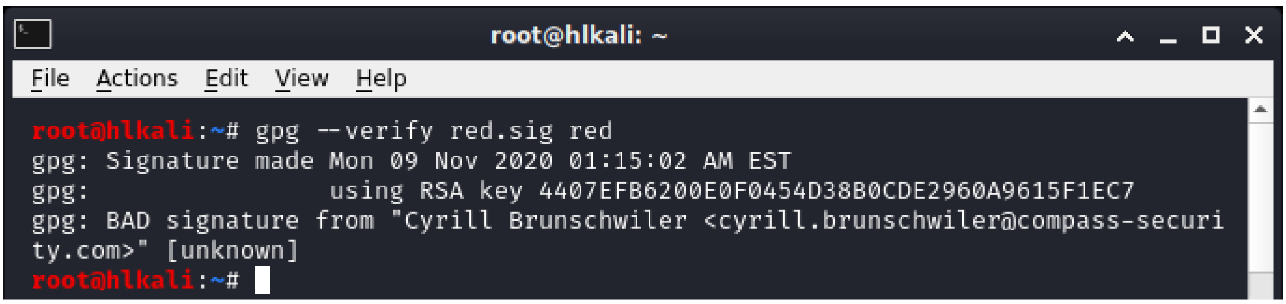
- RSA
- ElGamal
- ECDSA

**4. You asked me for my favourite colour. Can you verify which is my favourite colour using gpg - GNU Privacy Guard on your live CD?**

1. Import public key. We must assume we can trust that key or would need to contact the owner to verify the fingerprint.




2. Check signatures for red



3. Check signature for blue

```
root@hlkali:~# gpg --verify blue.sig blue
gpg: Signature made Mon 09 Nov 2020 01:14:36 AM EST
gpg:                using RSA key 4407EFB6200E0F0454D38B0CDE2960A9615F1EC7
gpg: Good signature from "Cyrill Brunschwiler <cyrill.brunschwiler@compass-secur
ity.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 4407 EFB6 200E 0F04 54D3  8B0C DE29 60A9 615F 1EC7
root@hlkali:~# 
```

4. Let's see what's in the blue file

```
root@hlkali:~# cat blue
my favourite colour is blue
root@hlkali:~# 
```

So, Cyrill's favorite color is blue. … just for the records … this was an exercise.

# 3 A5 Public Key Infrastructure

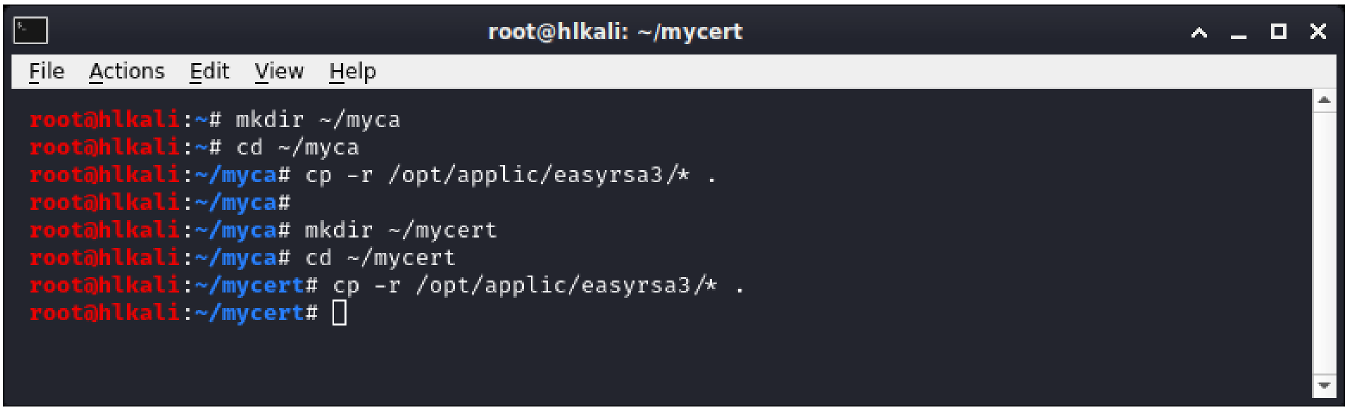
The whole Internet and many companies rely on public key infrastructure (PKI). For the Internet many so-called certificate authorities (CA) are being run as the root of trust. As there are many of them and all can work as the root of trust the risk of a malicious CA is evident and thus the mechanism of certificate transparancy (CT) has been introduced to detect deliberate and accidential misbehavior of a CA.

Let's create your own certificate authority using easy-rsa on the Kali Live CD. Moreover, we want to create server and client certificates to use these for mutual authentication with an Apache web server.

## 3.1 Step 1.1 Setup Workspace

Prepare a temporary space for you PKI. Open a shell and change into your home directory

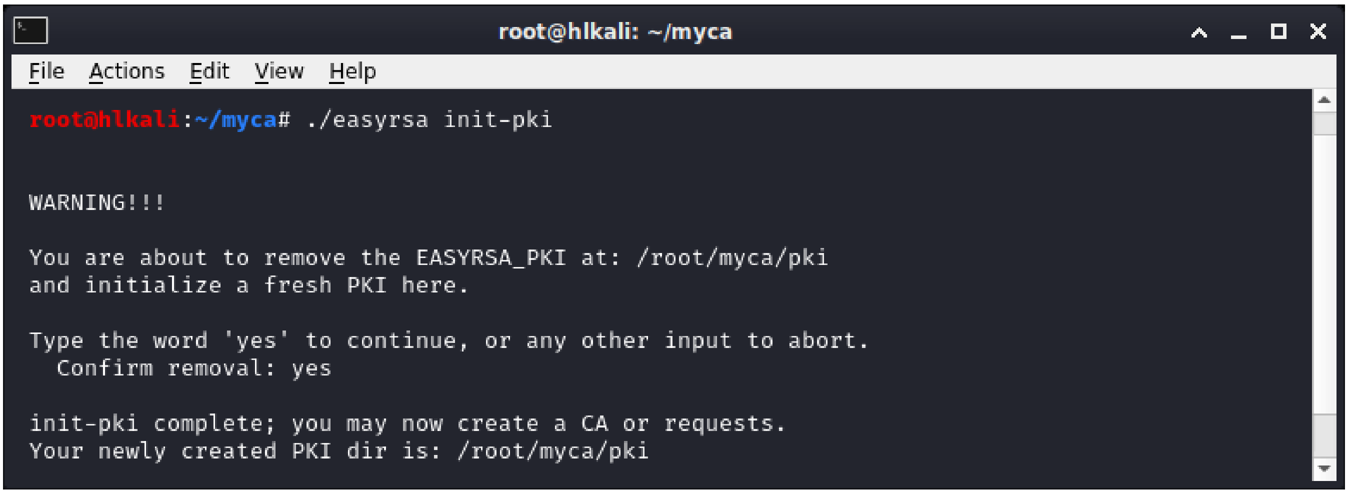
```
mkdir ~/myca
cd ~/myca
cp -r /opt/applic/easyrsa3/* .

mkdir ~/mycert
cd ~/mycert
cp -r /opt/applic/easyrsa3/* .
```

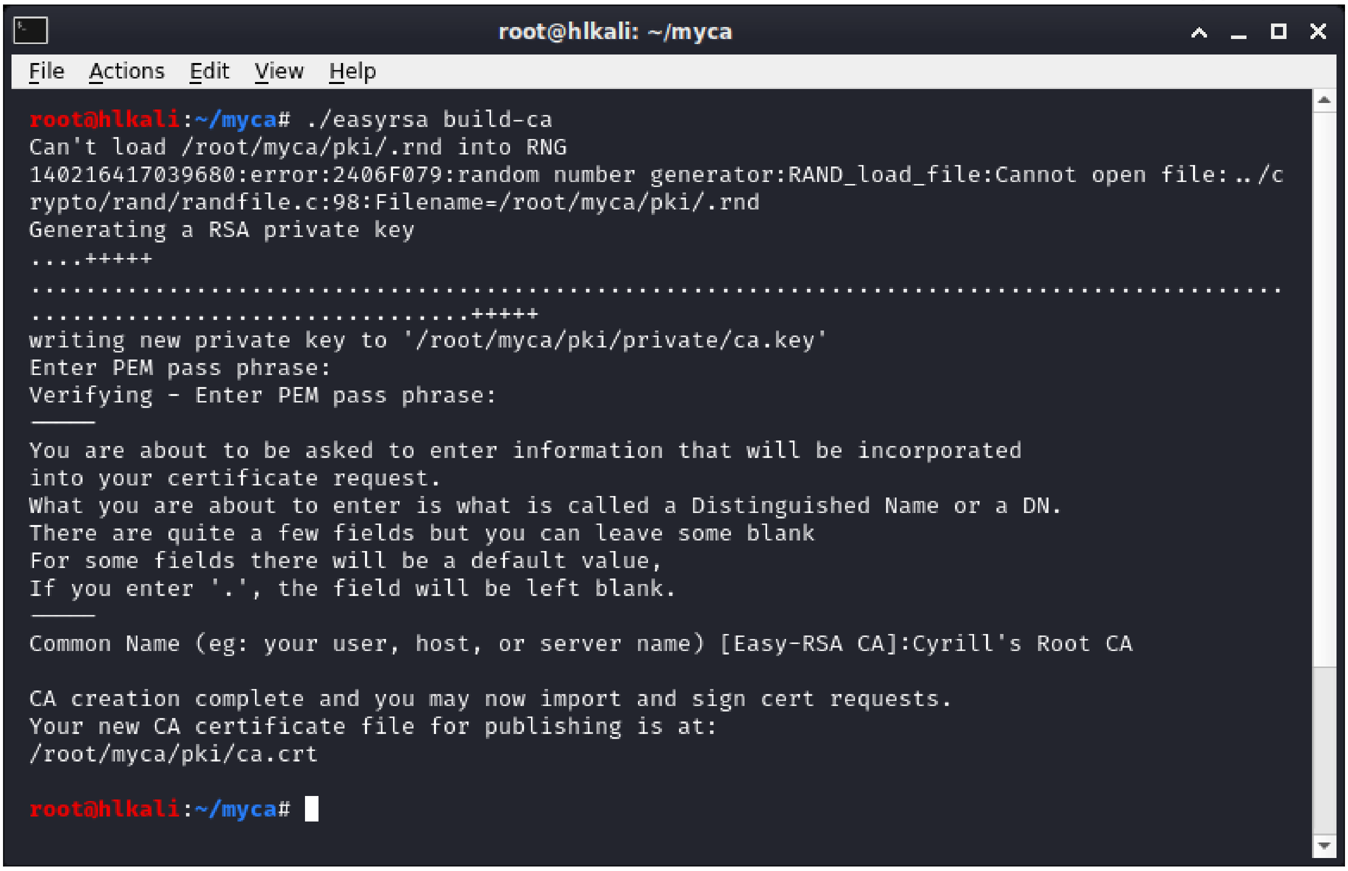



## 3.2 Step 1.2 Init PKI

Initialize your PKI as follows

```
cd ~/myca
./easyrsa init-pki (say yes)
./easyrsa build-ca (set a name for your CA. e.g. Cyrill's Root CA)
```





```
root@hlkali: ~/myca
File Actions Edit View Help
root@hlkali:~/myca# ./easyrsa build-ca
Can't load /root/myca/pki/.rnd into RNG
140216417039680:error:2406F079:random number generator:RAND_load_file:Cannot open file:../c
rypto/rand/randfile.c:98:Filename=/root/myca/pki/.rnd
Generating a RSA private key
....+++++
............................................................................................
.................................+++++
writing new private key to '/root/myca/pki/private/ca.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Common Name (eg: your user, host, or server name) [Easy-RSA CA]:Cyrill's Root CA

CA creation complete and you may now import and sign cert requests.
Your new CA certificate file for publishing is at:
/root/myca/pki/ca.crt

root@hlkali:~/myca#
```

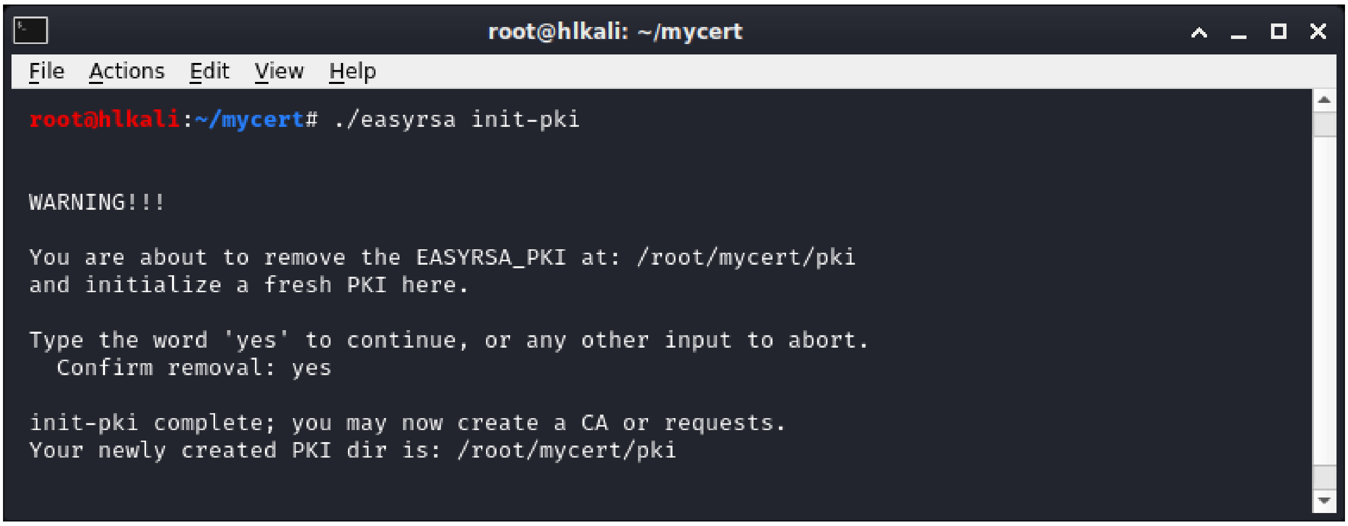
Btw, the use of the .rnd file is particularly important on systems with low entropy that often use the library. The .rnd file stores about 256 bytes of random information that is also considered as entropy. Kind of entropy CBC for invocations of OpenSSL.

## 3.3 Step 1.3 Create CSR

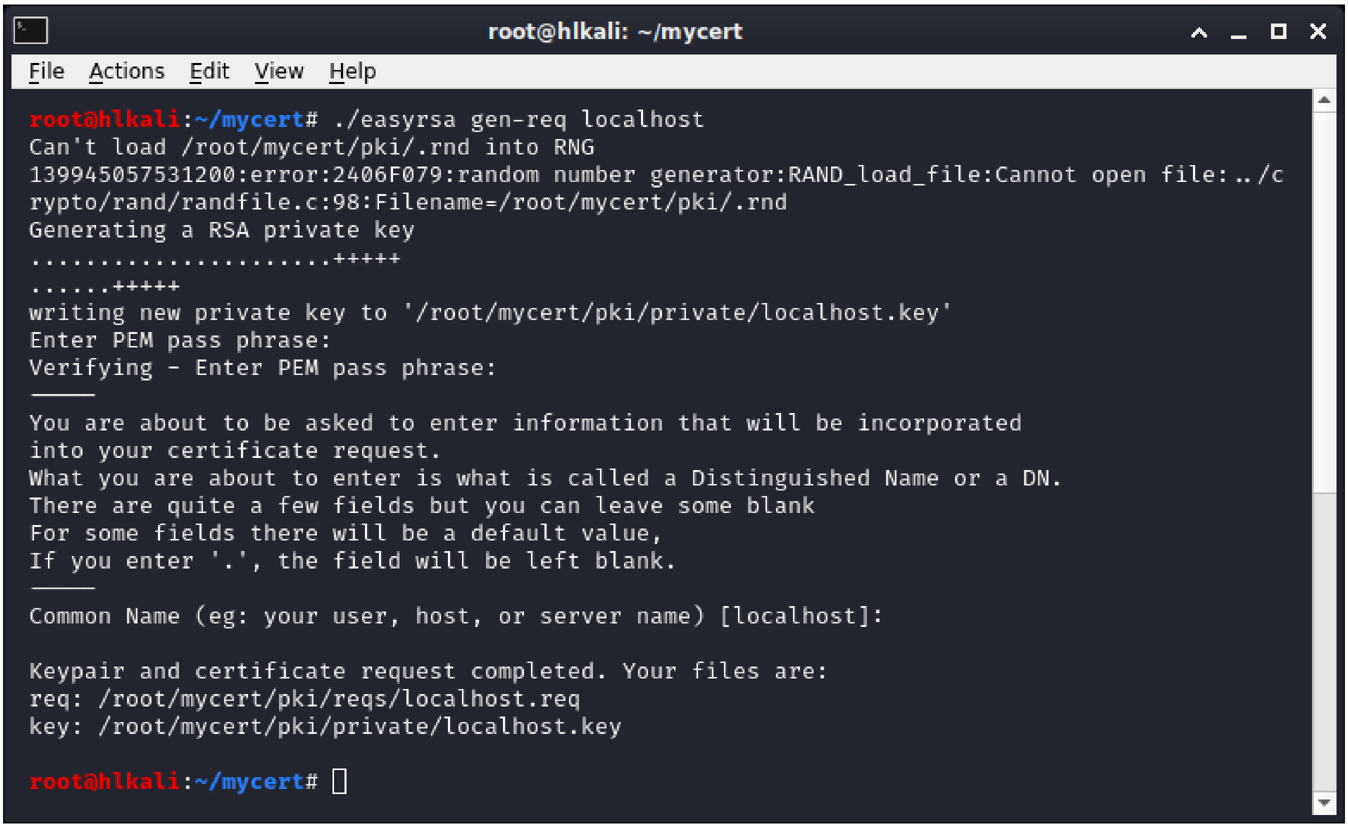
You will now create a certificate request. Usually, you would do this on the server or on the client who is requesting a certificate. This ensures that the key material is generated and held with the system/entity that requests a certificate. The sigining request (CSR) which is submitted to the CA does never contain the private key material.

Let's create a certificate for our "localhost" Apache server.

```
cd ~/mycert
./easyrsa init-pki (say yes)
./easyrsa gen-req localhost (choose a password you remember)
```



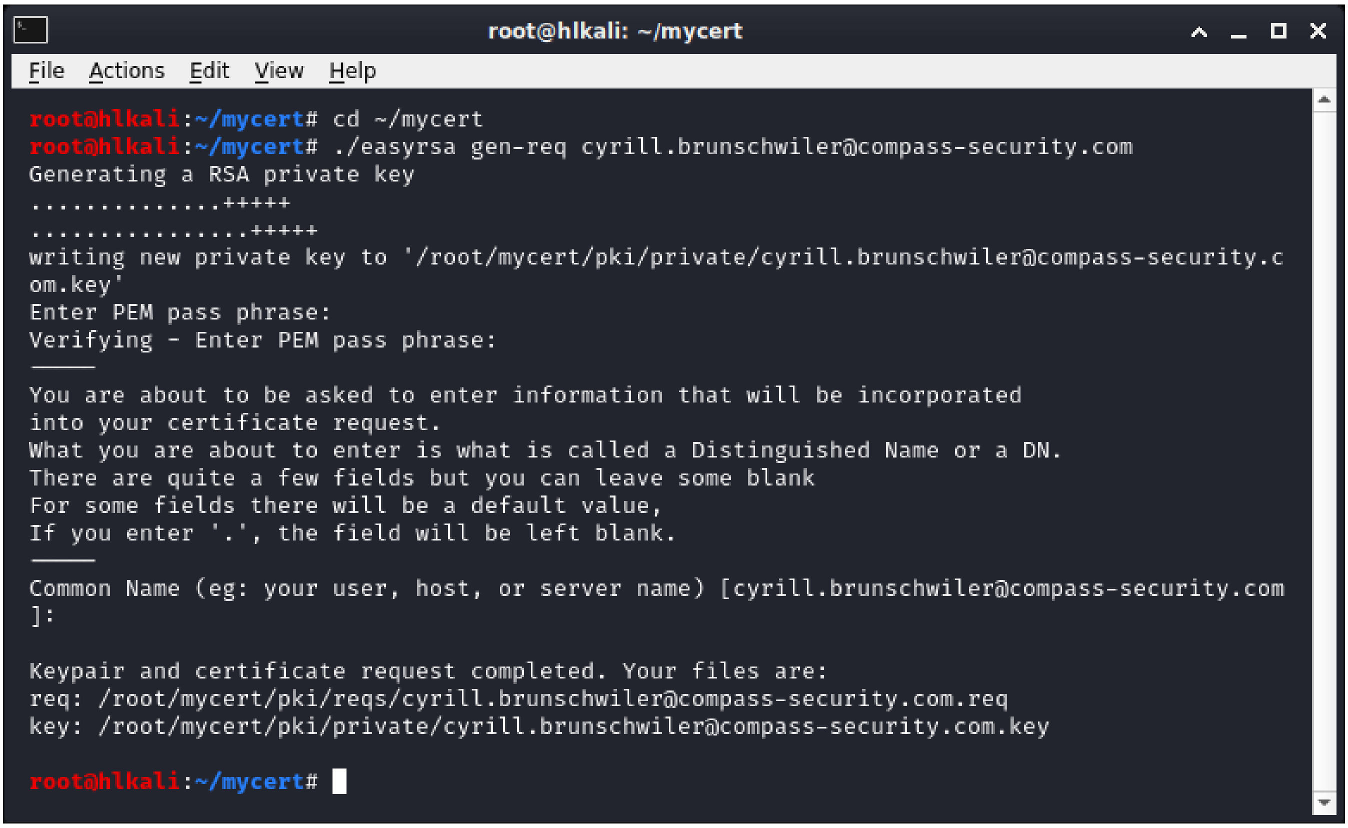
```
root@hlkali: ~/mycert
File Actions Edit View Help
root@hlkali:~/mycert# ./easyrsa init-pki


WARNING!!!

You are about to remove the EASYRSA_PKI at: /root/mycert/pki
and initialize a fresh PKI here.

Type the word 'yes' to continue, or any other input to abort.
  Confirm removal: yes

init-pki complete; you may now create a CA or requests.
Your newly created PKI dir is: /root/mycert/pki
```



```
root@hlkali: ~/mycert
File Actions Edit View Help
root@hlkali:~/mycert# ./easyrsa gen-req localhost
Can't load /root/mycert/pki/.rnd into RNG
139945057531200:error:2406F079:random number generator:RAND_load_file:Cannot open file:../c
rypto/rand/randfile.c:98:Filename=/root/mycert/pki/.rnd
Generating a RSA private key
.....................+++++
......+++++
writing new private key to '/root/mycert/pki/private/localhost.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Common Name (eg: your user, host, or server name) [localhost]:

Keypair and certificate request completed. Your files are:
req: /root/mycert/pki/reqs/localhost.req
key: /root/mycert/pki/private/localhost.key

root@hlkali:~/mycert#
```

Moreover, lets create a client certificate.

```
cd ~/mycert
./easyrsa gen-req cyrill.brunschwiler@compass-security.com (choose a password you remember)
```



```
root@hlkali: ~/mycert
File Actions Edit View Help
root@hlkali:~/mycert# cd ~/mycert
root@hlkali:~/mycert# ./easyrsa gen-req cyrill.brunschwiler@compass-security.com
Generating a RSA private key
..............+++++
...............+++++
writing new private key to '/root/mycert/pki/private/cyrill.brunschwiler@compass-security.c
om.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Common Name (eg: your user, host, or server name) [cyrill.brunschwiler@compass-security.com
]:

Keypair and certificate request completed. Your files are:
req: /root/mycert/pki/reqs/cyrill.brunschwiler@compass-security.com.req
key: /root/mycert/pki/private/cyrill.brunschwiler@compass-security.com.key

root@hlkali:~/mycert#
```

## 3.4 Step 1.4 Sign Certificates

Once the two signing requests are created you switch to the CA console and approve the two requests

```
cd ~/myca
```

```
./easyrsa import-req ~/mycert/pki/reqs/localhost.req localhost
./easyrsa sign-req server localhost
```

```
root@hlkali: ~/myca
File  Actions  Edit  View  Help
root@hlkali:~/myca# ./easyrsa sign-req server localhost


You are about to sign the following certificate.
Please check over the details shown below for accuracy. Note that this request
has not been cryptographically verified. Please be sure it came from a trusted
source or that you have verified the request checksum with the sender.

Request subject, to be signed as a server certificate for 3650 days:

subject=
    commonName                = localhost


Type the word 'yes' to continue, or any other input to abort.
  Confirm request details: yes
Using configuration from /root/myca/openssl-1.0.cnf
Enter pass phrase for /root/myca/pki/private/ca.key:
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'localhost'
Certificate is to be certified until Dec  8 07:58:17 2030 GMT (3650 days)

Write out database with 1 new entries
Data Base Updated

Certificate created at: /root/myca/pki/issued/localhost.crt
```

```
./easyrsa import-req ~/mycert/pki/reqs/cyrill.brunschwiler@compass-security.com.req cyrill
./easyrsa sign-req client cyrill
```

```
root@hlkali: ~/myca
File  Actions  Edit  View  Help
root@hlkali:~/myca# ./easyrsa sign-req client cyrill


You are about to sign the following certificate.
Please check over the details shown below for accuracy. Note that this request
has not been cryptographically verified. Please be sure it came from a trusted
source or that you have verified the request checksum with the sender.

Request subject, to be signed as a client certificate for 3650 days:

subject=
    commonName                = cyrill.brunschwiler@compass-security.com


Type the word 'yes' to continue, or any other input to abort.
  Confirm request details: yes
Using configuration from /root/myca/openssl-1.0.cnf
Enter pass phrase for /root/myca/pki/private/ca.key:
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'cyrill.brunschwiler@compass-security.com'
Certificate is to be certified until Dec  8 07:59:06 2030 GMT (3650 days)

Write out database with 1 new entries
Data Base Updated

Certificate created at: /root/myca/pki/issued/cyrill.crt
```

The two issued certificates need to be transferred to the client and server including a copy of the CA certificate.

```
cp -r ~/myca/pki/issued ~/mycert/pki/
cp ~/myca/pki/ca.crt ~/mycert/pki/
```

```
root@hlkali: ~/myca
File Actions Edit View Help
root@hlkali:~/myca# cp -r ~/myca/pki/issued ~/mycert/pki/
root@hlkali:~/myca# cp ~/myca/pki/ca.crt ~/mycert/pki/
root@hlkali:~/myca#
```

Well done. You just setup your first CA and signed a client and a server certificate. Let's bring 'em into production.

## 3.5 Step 1.5 Configure Certs with Apache

Open the Apache configuration file at /opt/applic/httpd/conf/extra/httpd-ssl.conf and change the two entries related to the certificate and key to match your newly created server certificate.

```
SSLCertificateFile "/root/mycert/pki/issued/localhost.crt"
SSLCertificateKeyFile "/root/mycert/pki/private/localhost.key"
```

**root@hlkali:~/myca#** vim /opt/applic/httpd/conf/extra/httpd-ssl.conf

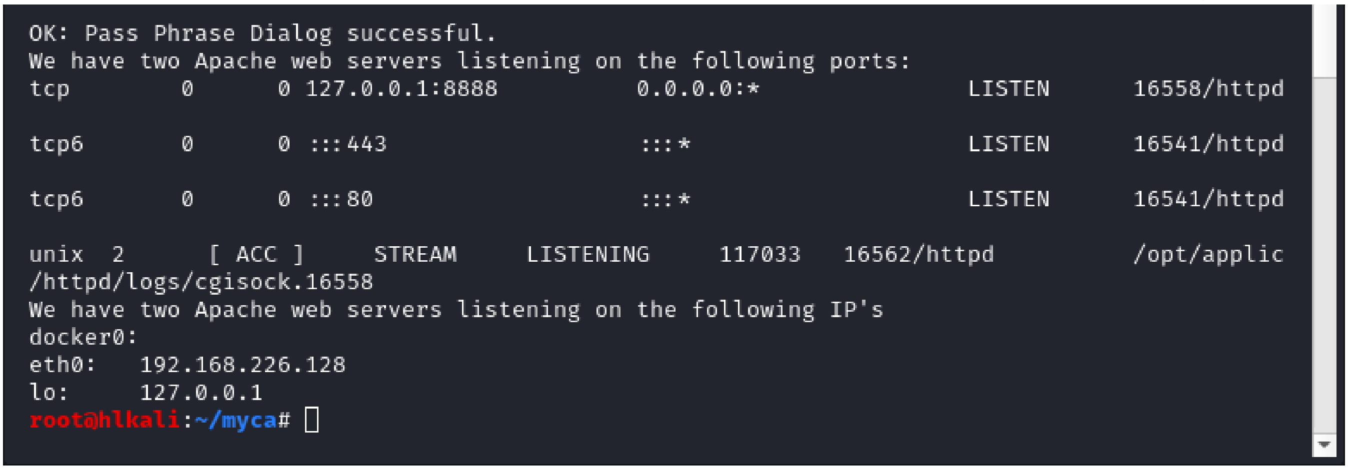
```
root@hlkali: ~/myca
File Actions Edit View Help
ServerName landingpage:443
ServerAdmin you@example.com
ErrorLog "/opt/applic/httpd/logs/ssl_error_log"
TransferLog "/opt/applic/httpd//logs/ssl_access_log"

SSLEngine on
SSLCertificateFile "/root/mycert/pki/issued/localhost.crt"
SSLCertificateKeyFile "/root/mycert/pki/private/localhost.key"


<FilesMatch "\.(cgi|shtml|phtml|php)$">
    SSLOptions +StdEnvVars
</FilesMatch>
<Directory "/opt/applic/httpd/cgi-bin">
                                                              64,1          70%
```

Hint: type :q to leave vim, type :wq to save and quit, type :q! to dismiss changes and quit

Now, try to restart the apache in the console

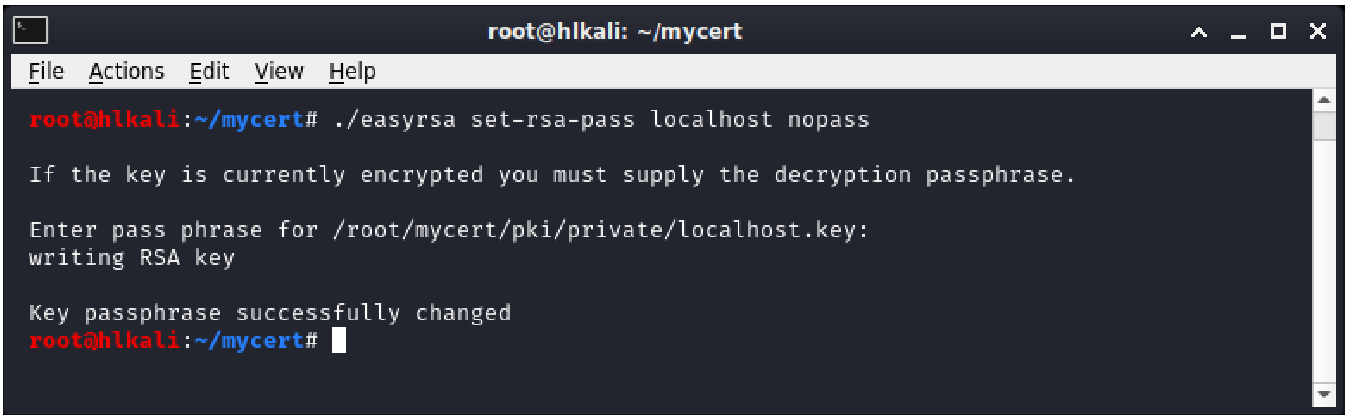
```
/etc/init.d/apache_but restart
```

```
root@hlkali: ~/myca
File Actions Edit View Help
root@hlkali:~/myca# /etc/init.d/apache_but restart
══════════════════════════════════
RESTART APACHE_BUT
══════════════════════════════════
httpd (no pid file) not running
httpd (no pid file) not running
Apache/2.4.33 mod_ssl (Pass Phrase Dialog)
Some of your private key files are encrypted for security reasons.
In order to read them you have to provide the pass phrases.

Private key landingpage:443:0 (/root/mycert/pki/private/localhost.key)
Enter pass phrase:
```




Common issues if the server does not restart are misconfigured paths, wrong permission and the key file still has a password set. You may type the password on Apache startup but you may get rid of the password like this

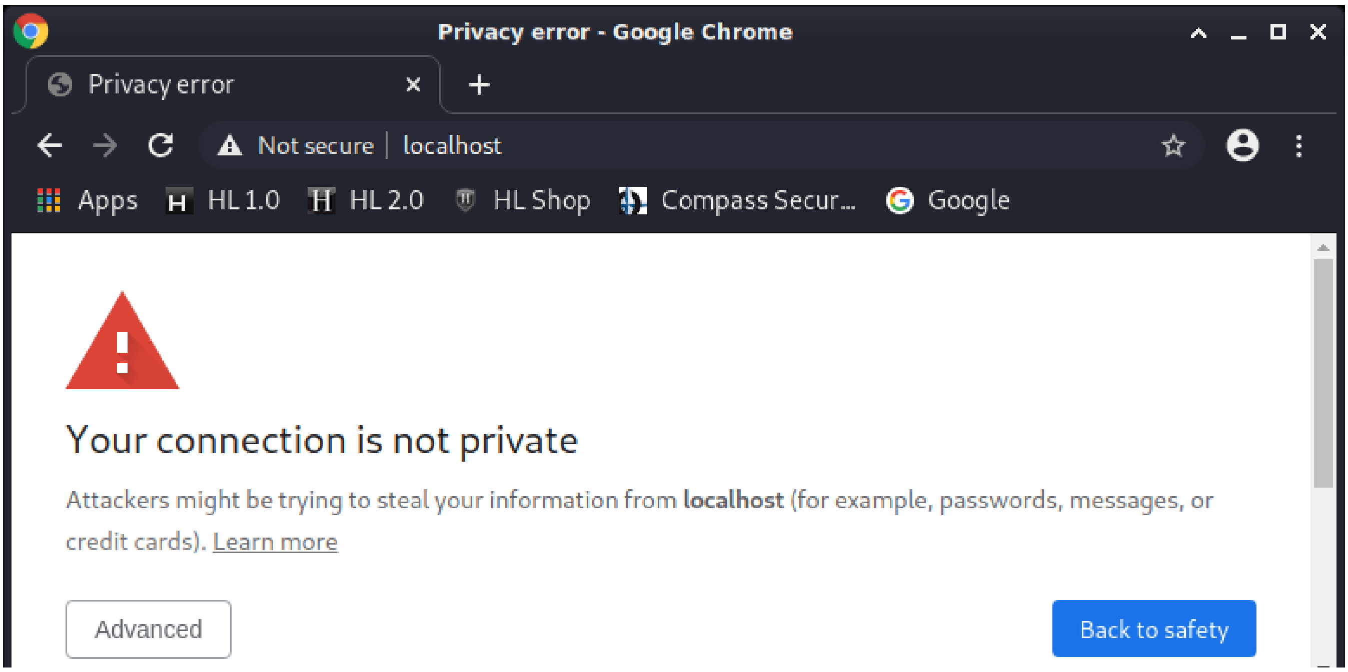
```
cd ~/mycert
./easyrsa set-rsa-pass localhost nopass
```




Btw, if you are interested how to create server keys w/o passphrase right from the start then read the easy-rsa docs.

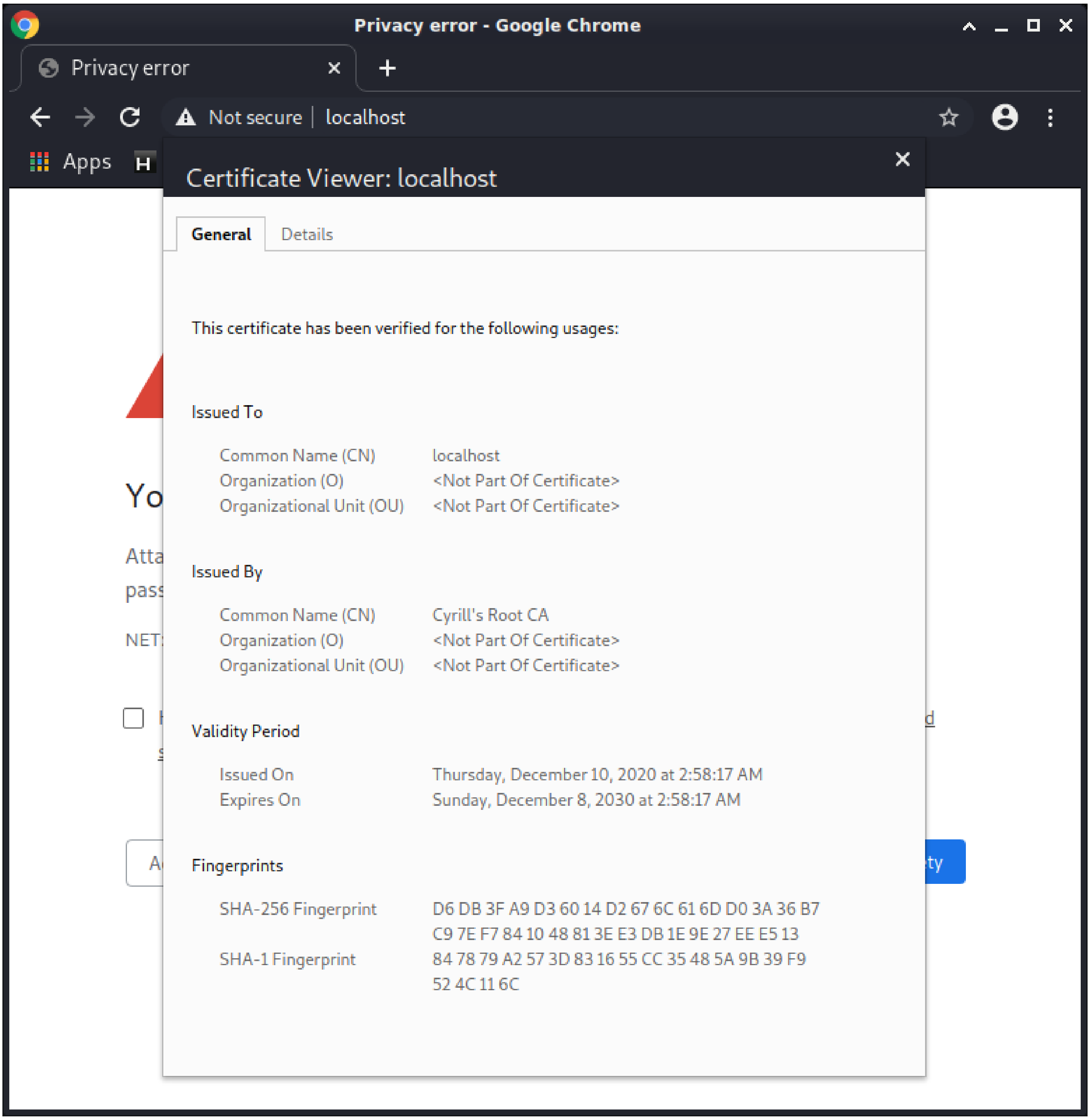
## 3.6 Step 1.6 Test in Browser, Collect Screenshot 1

Visit the server in order to check whether the server runs with your certificates. Open Chrome and surf on https://localhost/.



Oh, you still get a warning message as the certificate is not trusted. Anyways, does the certificate reflect your CA Name? If yes, we are good to proceed. If not, please check the previous steps.

Collect a screenshot of the current Certificate Dialog, Tab General as an evidence for this assignment.

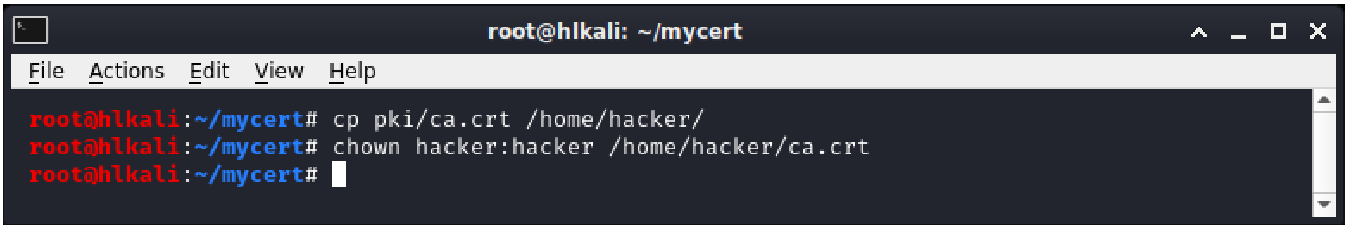


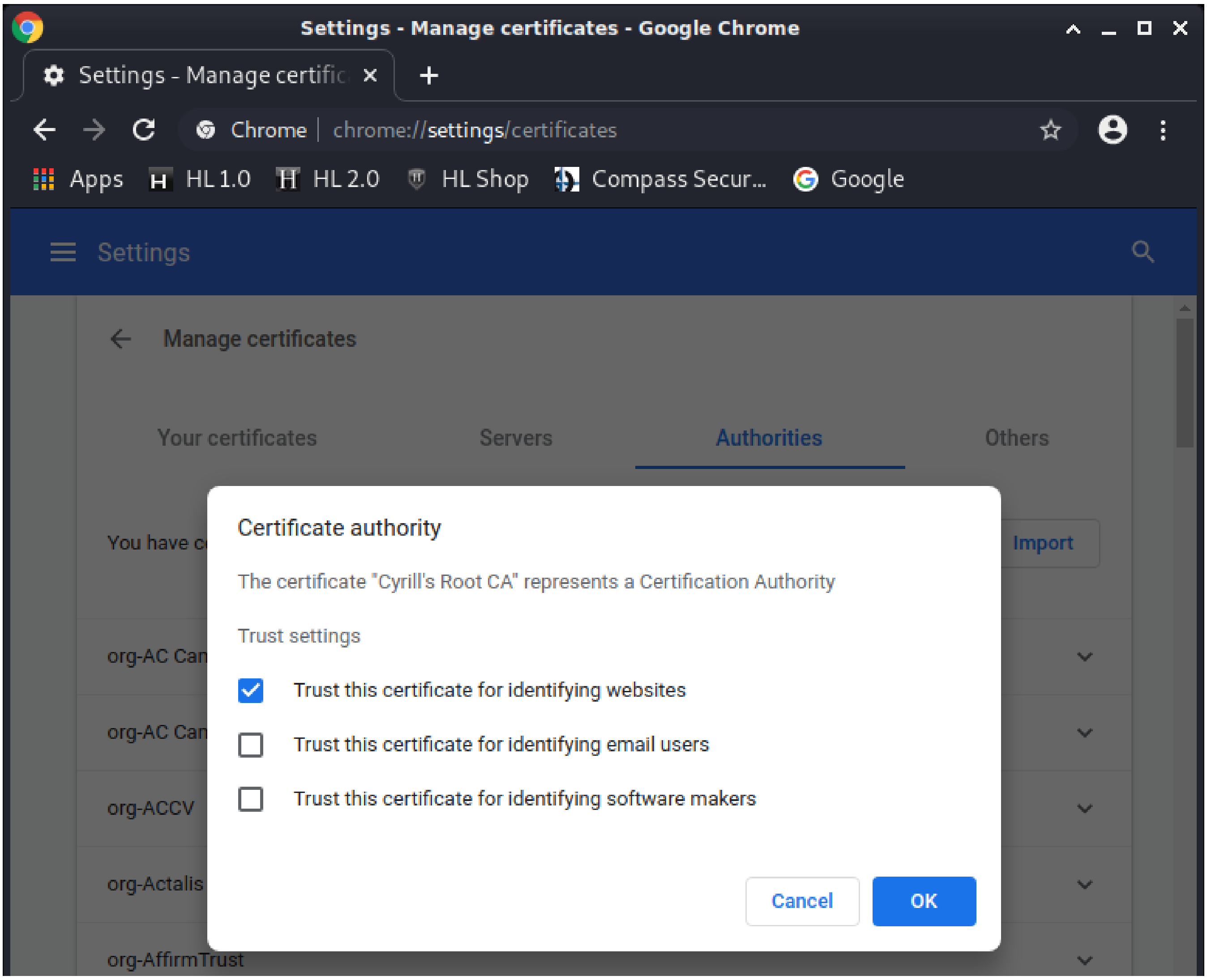

## 3.7 Step 1.7 Update Trusted Root Certificate Store, Collect Screenshot 2

Unfortunately, your certificate is not trusted. This is because your CA is not pre-packed with the browser. Therefore, let's import your own CA certificate to the trusted root CA store.

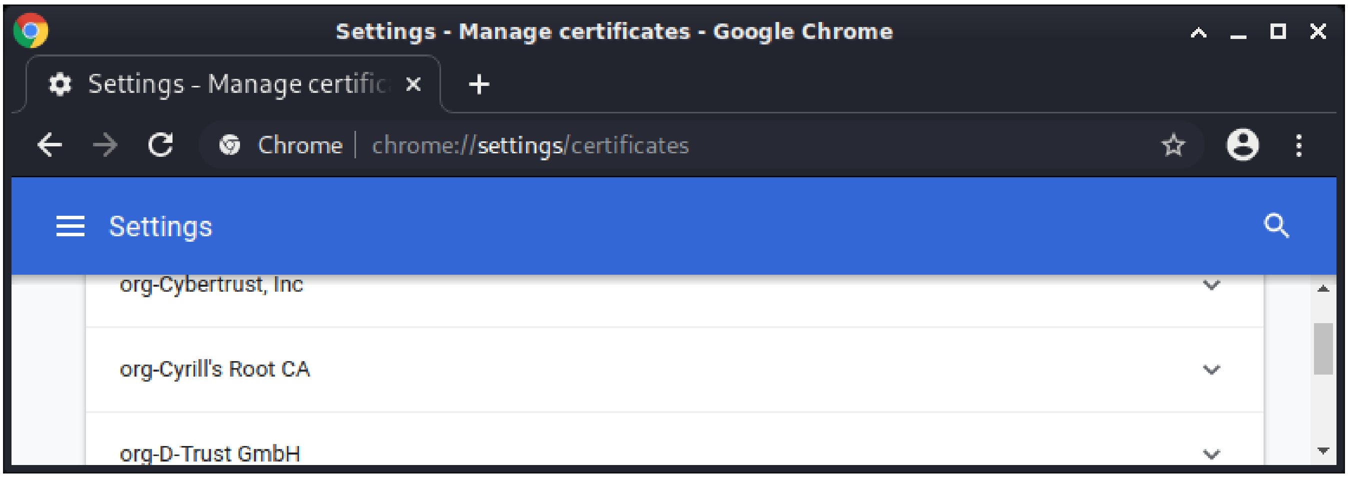
Go to chrome://settings/certificates Tab: Authorities and import your CA certificate from ~/myca/pki/ca.crt. If you created the CA as root then you maybe need to set permissions to allow the browser to access the cert file.

```
root@hlkali:~/mycert# cp pki/ca.crt /home/hacker/
root@hlkali:~/mycert# chown hacker:hacker /home/hacker/ca.crt
```

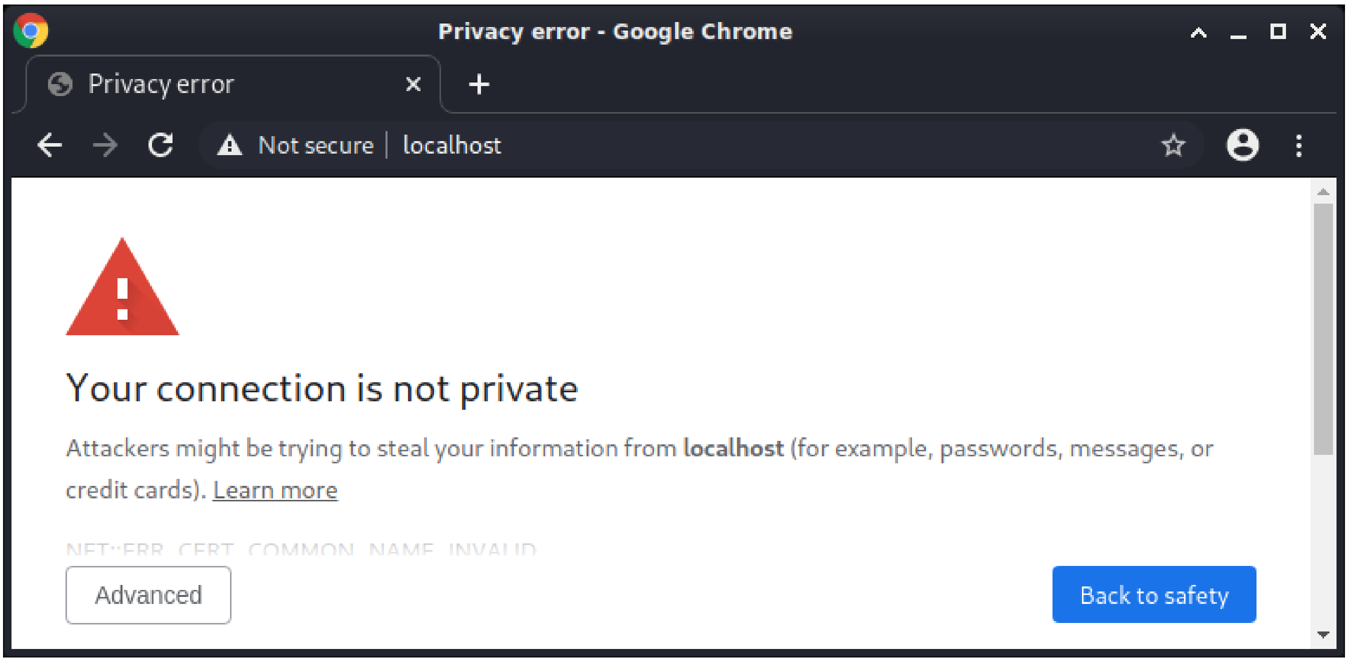



Once successfully imported. Capture a screenshot of the trusted root certificate store as evidence for this assignment.




Reload the tab and see whether Chrome properly identifies your web server?

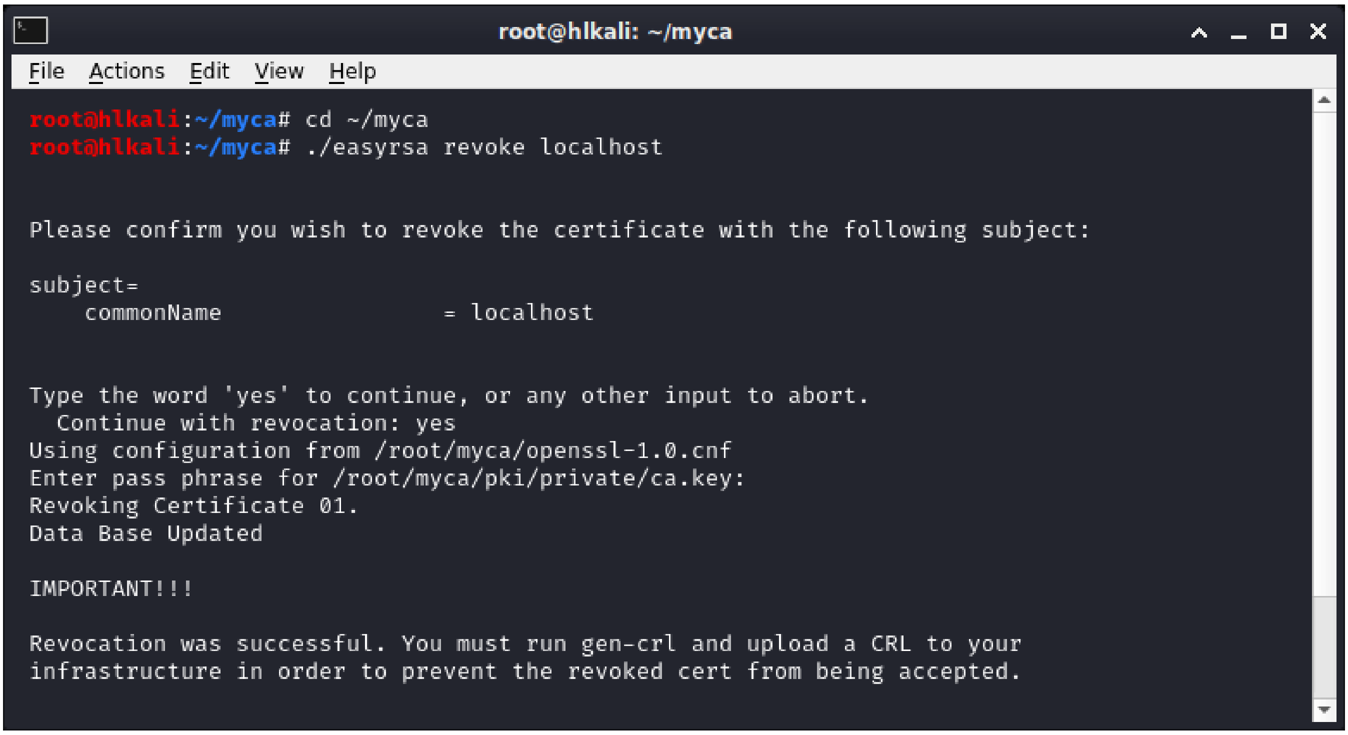


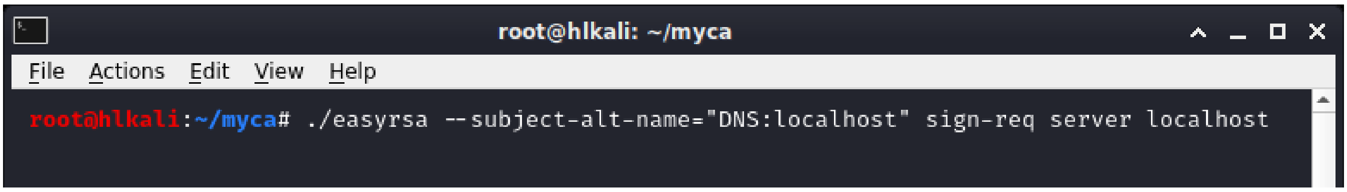

No? Why? Do you remember the mention regarding the subject alternative name?

### 3.8 Step 1.8 Update Certificate with SAN, Capture Screenshot 3

Now go and revoke the current certificate and create a new one including the relevant SAN.

```
cd ~/myca
./easyrsa revoke localhost
./easyrsa --subject-alt-name="DNS:localhost" sign-req server localhost
cp pki/issued/localhost.crt ~/mycert/pki/issued/localhost.crt
```



```
You are about to sign the following certificate.
Please check over the details shown below for accuracy. Note that this request
has not been cryptographically verified. Please be sure it came from a trusted
source or that you have verified the request checksum with the sender.

Request subject, to be signed as a server certificate for 3650 days:

subject=
    commonName                = localhost


Type the word 'yes' to continue, or any other input to abort.
  Confirm request details: yes
Using configuration from /root/myca/openssl-1.0.cnf
Enter pass phrase for /root/myca/pki/private/ca.key:
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'localhost'
Certificate is to be certified until Dec  8 08:16:10 2030 GMT (3650 days)

Write out database with 1 new entries
Data Base Updated

Certificate created at: /root/myca/pki/issued/localhost.crt
```

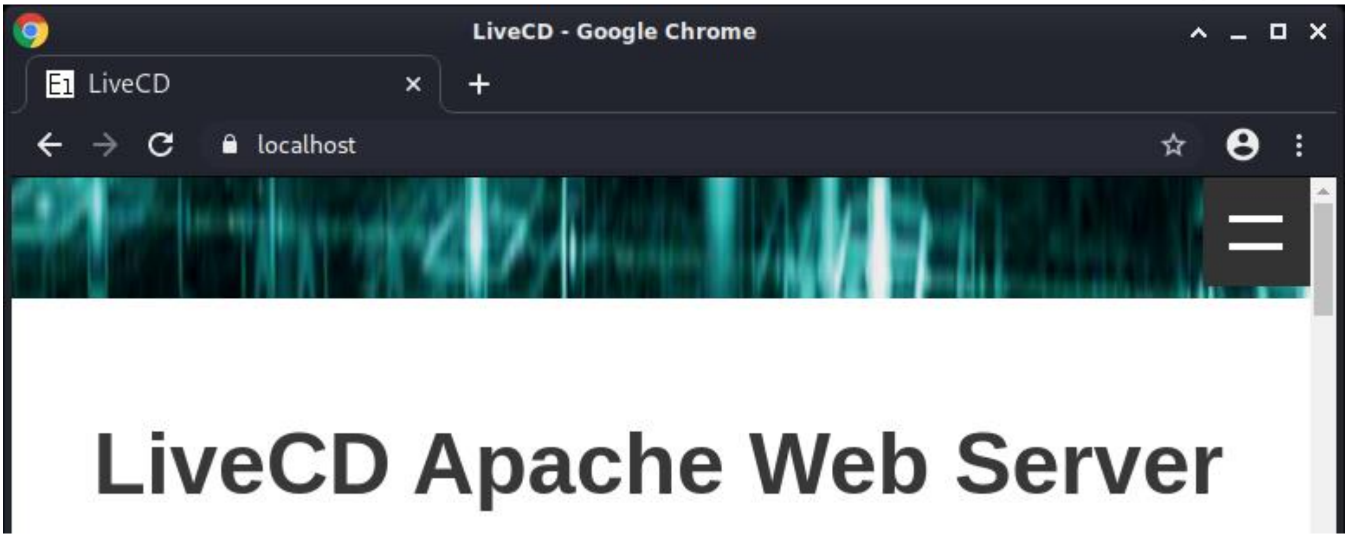
```
root@hlkali: ~/myca
File Actions Edit View Help
root@hlkali:~/myca# cp pki/issued/localhost.crt ~/mycert/pki/issued/localhost.crt
root@hlkali:~/myca#
```

And reload Apache again

```
/etc/init.d/apache_but restart
```

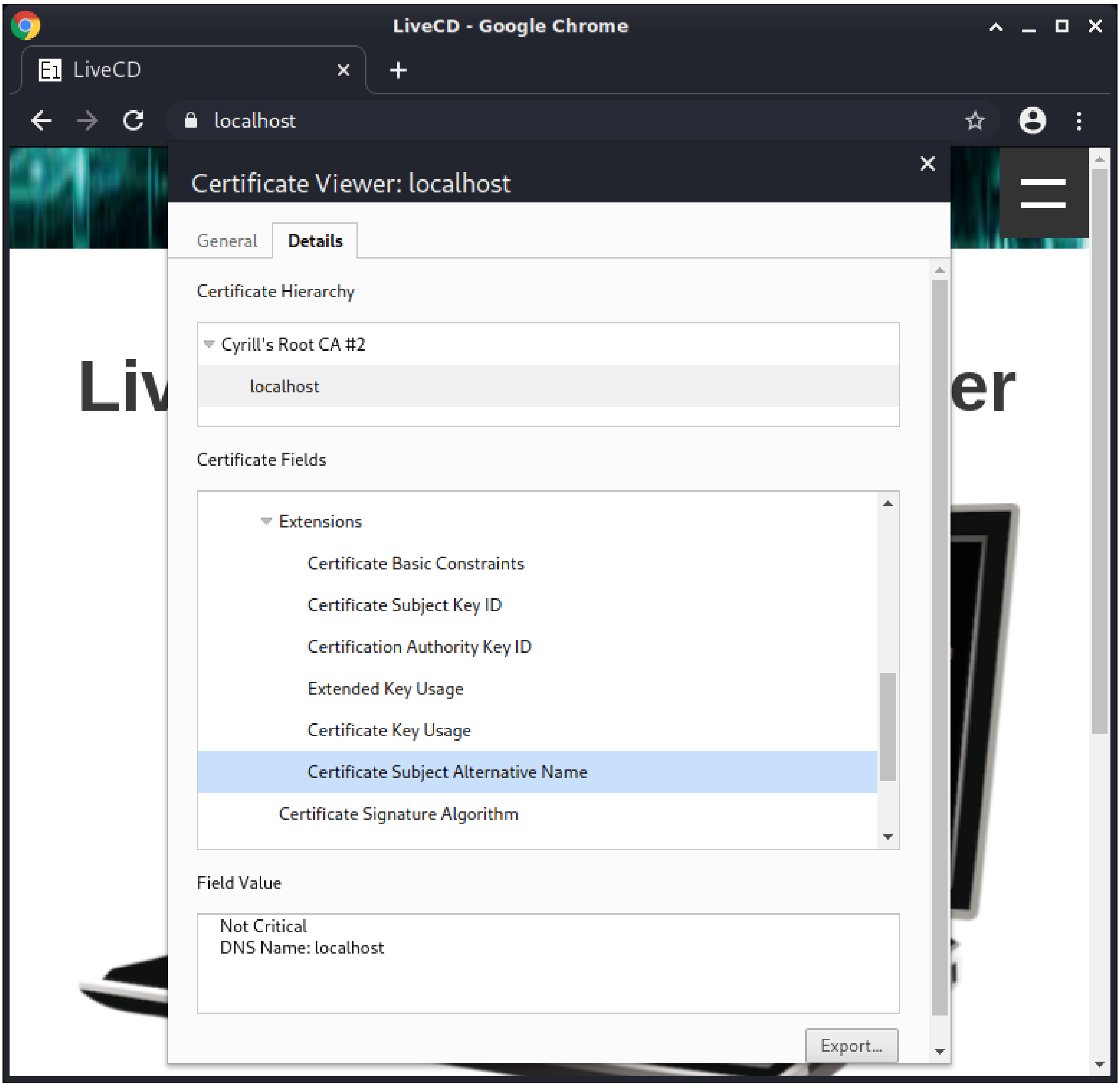
```
root@hlkali: ~/myca
File Actions Edit View Help
root@hlkali:~/myca# /etc/init.d/apache_but restart
=================================
RESTART APACHE_BUT
=================================
We have two Apache web servers listening on the following ports:
tcp        0      0 127.0.0.1:8888          0.0.0.0:*               LISTEN      17664/httpd

tcp6       0      0 :::443                  :::*                    LISTEN      17576/httpd

tcp6       0      0 :::80                   :::*                    LISTEN      17576/httpd

unix  2      [ ACC ]     STREAM     LISTENING     132325   17668/httpd          /opt/applic
/httpd/logs/cgisock.17664
We have two Apache web servers listening on the following IP's
docker0:
eth0:   192.168.226.128
lo:     127.0.0.1
```

Visit your browser and reload the tab. All good?




Well done! The SAN is really relevant nowadays!

Capture the last screenshot as evidence. I'd like to see the subject alternative name from the certificate dialog details tab.



# 4 A5 Certificates

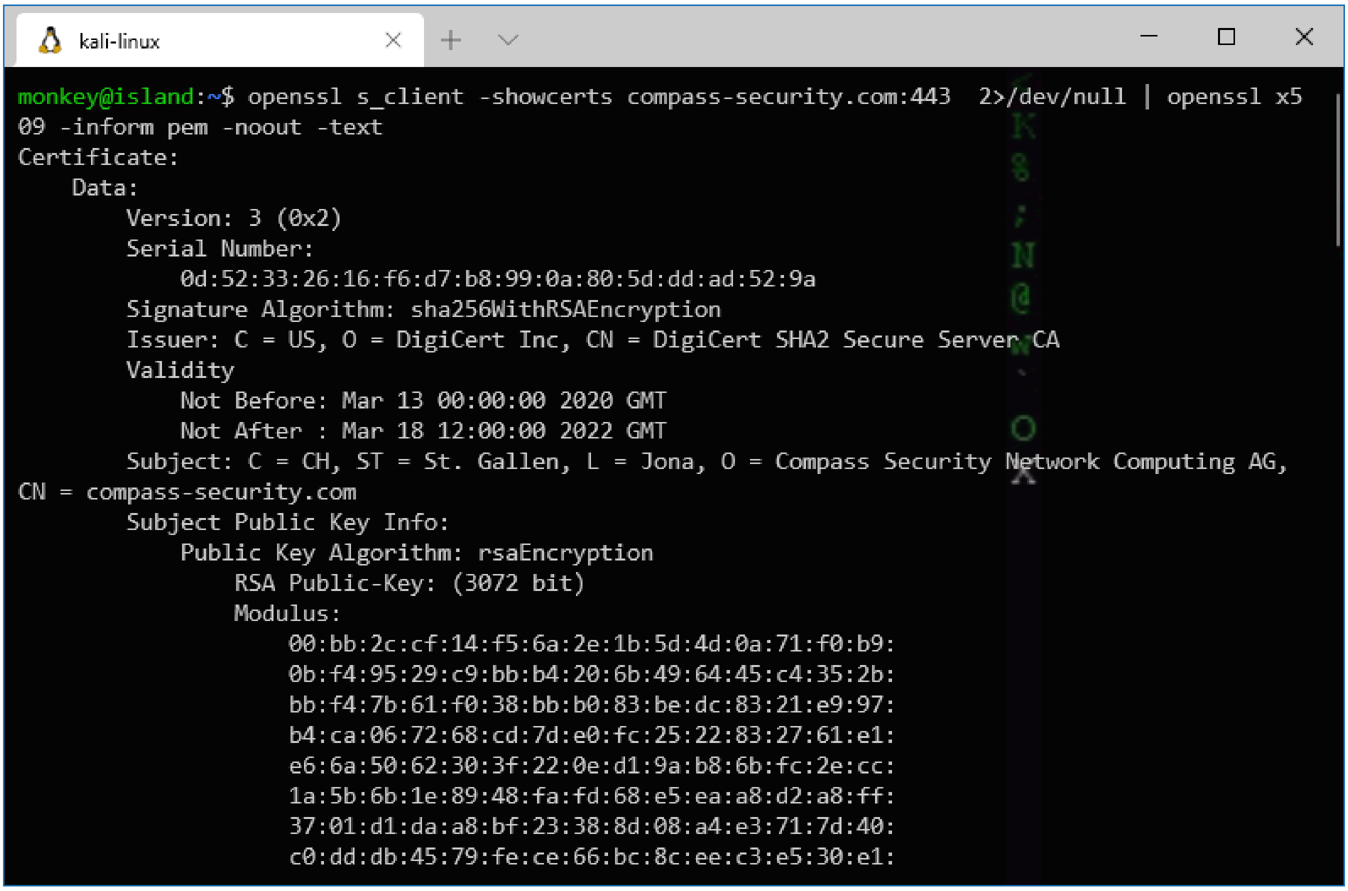
This part dives a bit into the nifty details of certificates. You should bring some background with hashes, signatures, RSA or ECC and Diffie-Hellmann. In your role as a cyber professional you should be capable to understand whether a server or client certificate's properties are reasonably set.

## 4.1 Step 1.1 Dump a Certificate

Use openssl to dump a certificate and note the following properties:

1. Issuer
2. Expiration time
3. Common Name
4. Subject Alternative Name
5. Public Key Algorithm
6. Public Key Size
7. Signature Algorithm used for the end-entity (leaf, last)
8. Signature Algorithm used for root and intermediate CAs
9. OCSP Supported
10. CRL Supported
11. Purpose of the key (extended key usage)
12. Basic constraint extensions

```
monkey@island:~$ openssl s_client -showcerts compass-security.com:443  2>/dev/null | openssl
x509 -inform pem -noout -text
```




```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            0d:52:33:26:16:f6:d7:b8:99:0a:80:5d:dd:ad:52:9a
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, O = DigiCert Inc, CN = DigiCert SHA2 Secure Server CA
        Validity
            Not Before: Mar 13 00:00:00 2020 GMT
            Not After : Mar 18 12:00:00 2022 GMT
```

```
        Subject: C = CH, ST = St. Gallen, L = Jona, O = Compass Security Network Computing
AG, CN = compass-security.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (3072 bit)
                Modulus:
                    00:bb:2c:cf:14:f5:6a:2e:1b:5d:4d:0a:71:f0:b9:
                    0b:f4:95:29:c9:bb:b4:20:6b:49:64:45:c4:35:2b:
                    bb:f4:7b:61:f0:38:bb:b0:83:be:dc:83:21:e9:97:
                    b4:ca:06:72:68:cd:7d:e0:fc:25:22:83:27:61:e1:
                    e6:6a:50:62:30:3f:22:0e:d1:9a:b8:6b:fc:2e:cc:
                    1a:5b:6b:1e:89:48:fa:fd:68:e5:ea:a8:d2:a8:ff:
                    37:01:d1:da:a8:bf:23:38:8d:08:a4:e3:71:7d:40:
                    c0:dd:db:45:79:fe:ce:66:bc:8c:ee:c3:e5:30:e1:
                    a2:55:da:39:78:e9:2c:27:a1:88:e2:6e:71:05:15:
                    59:22:d9:ce:36:67:98:86:30:0a:50:bc:ee:46:5e:
                    d1:7d:dc:0f:4f:7c:c8:d0:79:36:25:75:ee:48:d0:
                    b0:78:5f:78:98:34:97:06:a6:16:f5:f6:7a:8b:19:
                    93:ca:2b:f0:80:66:60:24:e8:61:cd:8b:95:5c:3e:
                    97:0a:b9:aa:14:55:33:8e:17:88:e4:50:78:4b:5e:
                    ee:ac:86:74:1e:50:52:bb:01:91:d4:8f:72:e5:38:
                    c1:80:02:d4:7b:4a:35:79:2a:b0:b7:3c:87:9e:1f:
                    e0:a8:e3:6f:49:e3:fe:98:42:55:fc:bc:95:93:72:
                    3c:37:69:d0:94:5c:72:fe:e6:75:f1:88:18:64:a2:
                    ba:52:e1:fe:02:35:b0:b0:03:a4:ef:dc:7a:30:7c:
                    26:79:47:4c:61:0a:ad:4e:42:e5:04:aa:3b:ce:b8:
                    5a:ea:f0:3c:c6:de:d4:5b:cb:e1:27:53:4e:68:f4:
                    2c:2f:ee:60:5f:a8:d2:bd:dd:b2:f3:23:ff:64:38:
                    db:65:80:78:0c:a4:44:8a:a2:57:e2:0a:03:77:0a:
                    5a:a7:84:a2:cb:bc:14:16:49:29:51:8e:e4:9b:21:
                    39:43:7f:08:c5:0c:98:7f:7e:2a:02:cf:fb:06:64:
                    e3:3e:db:73:74:73:59:c2:74:2b
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Authority Key Identifier:
                keyid:0F:80:61:1C:82:31:61:D5:2F:28:E7:8D:46:38:B4:2C:E1:C6:D9:E2

            X509v3 Subject Key Identifier:
                86:6F:58:80:1C:74:1D:D9:F8:C3:34:6A:2F:0B:DA:1D:0D:96:5C:EA
            X509v3 Subject Alternative Name:
                DNS:compass-security.com, DNS:www.compass-security.com
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 CRL Distribution Points:

                Full Name:
                  URI:http://crl3.digicert.com/ssca-sha2-g6.crl

                Full Name:
                  URI:http://crl4.digicert.com/ssca-sha2-g6.crl

            X509v3 Certificate Policies:
                Policy: 2.16.840.1.114412.1.1
                  CPS: https://www.digicert.com/CPS
                Policy: 2.23.140.1.2.2

            Authority Information Access:
                OCSP - URI:http://ocsp.digicert.com
                CA Issuers - URI:http://cacerts.digicert.com/DigiCertSHA2SecureServerCA.crt

            X509v3 Basic Constraints: critical
                CA:FALSE
            CT Precertificate SCTs:
                Signed Certificate Timestamp:
                    Version   : v1 (0x0)
                    Log ID    : A4:B9:09:90:B4:18:58:14:87:BB:13:A2:CC:67:70:0A:
                                3C:35:98:04:F9:1B:DF:B8:E3:77:CD:0E:C8:0D:DC:10
                    Timestamp : Mar 13 08:40:39.759 2020 GMT
                    Extensions: none
                    Signature : ecdsa-with-SHA256
```

```
                                30:44:02:20:70:0A:F7:AB:F2:F9:90:76:F6:B9:30:0D:
                                28:B9:19:6D:D8:BF:20:88:34:5F:08:C0:D4:EA:AC:81:
                                FF:89:80:1E:02:20:09:07:9B:3C:04:E5:DE:DA:C3:B1:
                                25:31:7C:5D:6A:FB:F6:65:8F:A3:B8:77:20:82:C1:E7:
                                6D:3D:C6:BD:F0:D4
                Signed Certificate Timestamp:
                    Version   : v1 (0x0)
                    Log ID    : 22:45:45:07:59:55:24:56:96:3F:A1:2F:F1:F7:6D:86:
                                E0:23:26:63:AD:C0:4B:7F:5D:C6:83:5C:6E:E2:0F:02
                    Timestamp : Mar 13 08:40:39.827 2020 GMT
                    Extensions: none
                    Signature : ecdsa-with-SHA256
                                30:45:02:21:00:FE:9E:81:A9:4A:E2:78:36:FB:FB:6B:
                                EB:D3:3F:ED:D8:A0:8E:4A:22:58:1F:CD:EC:F6:62:95:
                                DF:2D:58:F4:9E:02:20:6F:A9:1E:EE:DE:03:BC:B7:AA:
                                43:6F:18:B8:48:02:24:69:19:A8:52:CD:85:AE:6E:7A:
                                CA:15:A6:4A:53:08:CB
                Signed Certificate Timestamp:
                    Version   : v1 (0x0)
                    Log ID    : 51:A3:B0:F5:FD:01:79:9C:56:6D:B8:37:78:8F:0C:A4:
                                7A:CC:1B:27:CB:F7:9E:88:42:9A:0D:FE:D4:8B:05:E5
                    Timestamp : Mar 13 08:40:39.855 2020 GMT
                    Extensions: none
                    Signature : ecdsa-with-SHA256
                                30:45:02:20:01:83:A9:9C:2C:A1:81:B0:0F:7D:DD:5D:
                                90:B9:78:3B:10:D8:06:86:AD:5F:DD:6A:AD:CB:93:C6:
                                FC:7F:2D:EB:02:21:00:F7:17:C1:7B:36:E6:D0:02:15:
                                89:39:1D:E9:3B:52:C8:FF:17:06:14:97:C5:B7:92:34:
                                D7:98:63:5D:36:0B:F5
    Signature Algorithm: sha256WithRSAEncryption
         db:6f:29:c3:68:c0:2f:aa:20:ae:ab:70:28:ce:2f:6d:cc:71:
         06:7f:13:5b:ea:a2:e2:e3:dc:1e:f4:f2:04:d8:0b:9b:cd:84:
         14:4f:49:fd:6e:ed:a3:9d:fb:93:97:2a:fa:6a:e5:c3:1a:d8:
         5a:e7:aa:62:5a:56:d6:8a:1f:e7:a5:15:40:5c:d2:78:91:11:
         cc:16:aa:a3:c5:84:e4:ff:57:5b:9f:cb:46:c1:3c:8a:da:13:
         33:38:73:f1:ba:f8:8e:00:50:82:34:1f:bb:e7:d7:01:64:8f:
         43:c9:8e:dd:ef:1b:bc:d8:33:1f:fd:a1:a7:2e:a3:7d:fe:3d:
         b7:ff:47:23:c6:71:c8:4f:07:f5:d6:a6:4b:54:47:e4:97:6e:
         e9:43:c2:49:f2:ea:f8:07:55:6e:11:04:36:08:5a:4f:db:55:
         6c:13:f7:9d:21:27:63:8a:cd:fd:a5:7c:02:97:8a:16:0c:58:
         7a:f0:1b:4e:47:9c:95:a0:bf:db:db:55:86:e2:53:d3:e7:8b:
         d7:43:a3:d2:05:c8:a9:fe:31:9e:75:47:bc:39:b4:44:75:26:
         3f:3e:ee:aa:a0:0a:c7:e2:fb:fa:2f:63:9a:1d:75:9c:98:5f:
         2d:4d:b4:52:ea:59:ef:60:f1:1f:d3:5e:6d:10:94:d5:cf:6e:
         95:43:72:a7
```

| No. | Property | Value |
|---|---|---|
| 1. | Issuer | Issuer: C = US, O = DigiCert Inc,<br>CN = DigiCert SHA2 Secure Server CA |
| 2. | Expiration time | Not Before: Mar 13 00:00:00 2020 GMT<br>Not After : Mar 18 12:00:00 2022 GMT |
| 3. | Common Name | compass-security.com |
| 4. | Subject Alternative Name | DNS:compass-security.com, DNS:www.compass-security.com |
| 5. | Public Key Algorithm | rsaEncryption |
| 6. | Public Key Size | 3072 bit |
| 7. | Signature Algorithm used for the end-entity (leaf, last) | sha256WithRSAEncryption |
| 8. | Signature Algorithm used for root and intermediate CAs | sha256WithRSAEncryption |
| 9. | OCSP Supported | OCSP - URI:http://ocsp.digicert.com |

| No. | Property | Value |
|---|---|---|
| 10. | CRL Supported | URI:http://crl3.digicert.com/ssca-sha2-g6.crl<br>URI:http://crl4.digicert.com/ssca-sha2-g6.crl |
| 11. | Purpose of the key (extended key usage) | TLS Web Server Authentication,<br>TLS Web Client Authentication |
| 12. | Basic constraint extension | CA:FALSE |

## 4.2 Step 1.2 Challenge Properties

| No. | Question | Answer | Considerations |
|---|---|---|---|
| 1. | Would a certificate validity for another 5 years be a good idea? It would significantly simplify key handling on servers. | Although, there are mechanisms like OCSP and CRL it is recommended to expire certificate better sooner than later (1 year) which would lower the risk in case of loss of the private key material.<br><br>If the browser cannot reach the CRL or OCSP service it just connects with servers presenting a revoked but still valid certificate. | Certificate is not expired.<br><br>Certificate is valid for not more than 5 years in the future. |
| 2. | Are the algorithms strong enough for the lifetime of the certificate? If yes, what would you consider a weak algorithm? | The signature algorithm is fine.<br><br>Actually, I am not aware of an insufficient signature algorithm specified in https://tools.ietf.org/html/rfc3279 | RSA, DSA, ECDSA are okay. |
| 3. | Is the key strong enough? If yes, what would be insufficient? | The key is strong enough. Minimum requirements see "Considerations" column. | For RSA:<br>▪ Public Key (>= 2048-bit)<br>▪ Exponent (= 0x10001)<br><br>For ECC:<br>▪ Public Key (>= 256-bit) |
| 4. | Would it be okay if the OCSP and CRL info is missing? What would be the impact? | No. If the private key material is lost or gets stolen then there is no means for a connecting client to verify if the certificate is still valid. | Ideally, both OCSP and CRL info is provided.<br><br>In best case, the server supports OCSP stapling. |
| 5. | Would it be okay if the CA signed the request for the analysed certificate if key usage would also include "Code Signing" or "Email Protection" or "Time Stamping" or "OCSP Signing"? | No. Certificates should be restricted to a specific purpose to avoid misuse for other purpose. We do not want someone getting its hands on a server certificate could maybe sign binaries that would the be trusted to run. The definition of key usage helps to segregate duties. | For server certificates....<br><br>Good:<br>▪ Digital Signature<br>▪ Key Encipherment<br><br>Bad:<br>▪ Code Signing<br>▪ OCSP Signing |
| 6. | Is there are reason why the basic constraint states CA:FALSE? What if it would be CA:TRUE? | If CA would be TRUE than the leaf certificate key material could be used as an intermediate CA and could therefore be used to create certificates for arbitrary subjects. | For server certificates….<br><br>Good:<br>▪ End Entity<br><br>Bad:<br>▪ CA |

## 4.3 Step 1.3 Client Certificate Properties

Also dump the cert you created in the PKI challenge and point out the major difference to the above certificate properties in the same write-up.

```
openssl x509 -in ~/mycert/pki/issued/cyrill.crt -text
```

Mind, you maybe choose different names.


This certificate is not meant to be used as a server certificated or for key encipherment. The major differences are highlighted.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 2 (0x2)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN = Cyrill's Root CA
        Validity
            Not Before: Dec 10 07:59:06 2020 GMT
            Not After : Dec  8 07:59:06 2030 GMT
        Subject: CN = cyrill.brunschwiler@compass-security.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:d8:2f:70:0a:f6:05:59:7d:25:c8:b0:1a:7c:87:
                    ...
                    b9:73
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Subject Key Identifier:
                B4:89:52:CA:E3:A9:75:79:D5:B3:26:F1:12:30:B8:21:C7:8E:B2:F7
            X509v3 Authority Key Identifier:
                keyid:D3:16:3D:12:DF:D3:22:E9:50:F9:80:4E:32:3D:AB:E7:1B:2A:4C:A9
                DirName:/CN=Cyrill's Root CA
                serial:58:82:BC:CE:7F:D5:B0:5E:2C:0A:35:DD:63:60:24:DF:F4:52:76:A6

            X509v3 Extended Key Usage:
                TLS Web Client Authentication
            X509v3 Key Usage:
                Digital Signature
    Signature Algorithm: sha256WithRSAEncryption
         65:80:1f:d9:1a:84:10:ee:08:aa:28:7b:1d:14:9d:02:d4:0d:
         ...
         b5:a6:d0:9a
-----BEGIN CERTIFICATE-----
MIIDcTCCAlmgAwIBAgIBAjANBgkqhkiG9w0BAQsFADAbMRkwFwYDVQQDDBBDeXJp
...
3Y7//OstqViLBV3dzpbo7lu1ptCa
-----END CERTIFICATE-----
```

# 5 A5 Protocols

There is a general approach to secure protocols. The aim to secure protocols is to assure parties are communicating over secure channels and adversaries cannot interfere or eavesdrop on the channel. Thus, key exchange is a central part when securing protocols.

1. **Explain the two major concepts of authentication and its purpose in basic words?**

   **Data origin authentication** aims to provide integrity of data including that a specific entity has given consent of the integrity of the data. Thus, a receiver is sure the data was created or approved by a trusted party, user or system and that the data was not tampered with since.

   **Entity authentication** is used to verify a communication party. Usually a user or a system. It is often loosely referred as "authentication" or sometimes "login".

   What is authentication?
   - Data origin authentication (ISO 7498-2 origin verification)
   - Entity authentication (ISO 7498-2 identity verification)

   Entity authentication
   - Unilateral (entity) authentication
   - Mutual (entity) authentication
   - Both might be encryption, MAC or signature-based

2. **If you are presented a protocol what are the criteria you would judge it with?**

   - **Authenticity of data origin** to assure data was not tampered with and we receive the data from a specific party
   - **Freshness** to assure the data was not used earlier and is just being sent again (replay attacks). This could be enforced by incorporating nonces into the protocol
   - **Liveness** to assure we are seeing data that is currently being generated and acted upon. Liveness can be enforced by introducing time-stamps, logical time stamps or sequence numbers.
   - **Protocol must embed identities** to enable senders and receivers can check the involved parties and messages cannot be used with other parties than intended ones (impersonation attacks, ticket reuse attacks).

3. **Referring to the following agreement. Does it fulfil all of the above criteria**

   1. **Alice => Bob: {Session-Key}Public-KeyBob**
   2. **KeyMAC = HMAC(Session-Key || 'MAC')**
   3. **KeyENC = HMAC(Session-Key || 'ENC')**

   In 1) Alice sends Bob a message that is an encrypted session-key using Bob's public key. This is basically what happens if a web server sends the pre-master secret to the client using the RSA public key procedure to setup a TLS session.

   2) and 3) are just derivations of necessary key material from the Session-Key and not very relevant for the question.

   Let's judge the key agreement (mainly message 1)

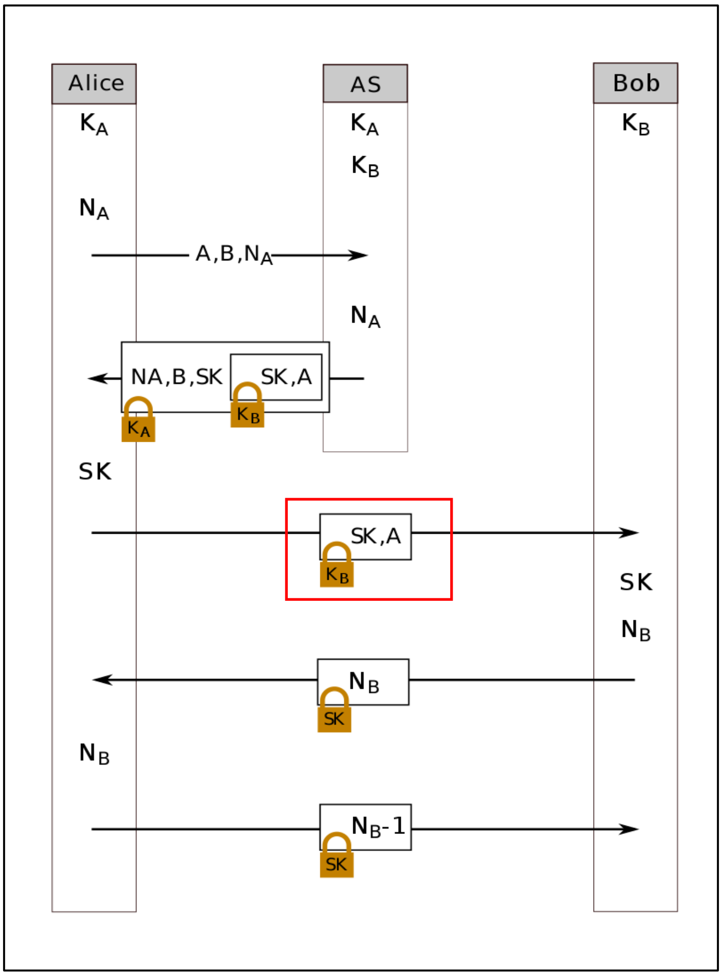
   **Authenticity of data origin**
   Bob cannot tell who sent the message and whether the contained data's integrity is okay as everyone in possession of the public key could send him such a message. Data origin authentication is not fulfilled.

   **Freshness and Liveness**
   Unless bob keeps a history of previously received Session-Keys he cannot verify if the message is fresh or not. Note that, if the message was intercepted by Mallory. Bob would assume its fresh but would have no means to figure the message was originally sent by Alice. So, the key agreement lacks freshness and liveness mechanisms.

   **Protocol must embed identities**
   See answers above

   Concluding this exercise. The public private key agreement lacks quite some requirements and it becomes obvious that without wrapping a PKI and CA trust construct around it, the protocol would be quite useless.

4. **Referring to the Needham-Schroeder protocol. How can Bob tell the first message he receives of Alice {Session-Key, Alice}KeyBob is a fresh one?**




Bob cannot tell if the said message is fresh or not. Mallory, who intercepted the message before could forward it to Bob and Bob would assume Alice is trying to start a new conversation.

However, to exploit the attack effectively we must assume that Mallory got hold of SK sometime.

The attack is described in "Timestamps in key distribution protocols", http://pages.cs.wisc.edu/~remzi/Classes/736/Spring2005/Papers/data-encryption-denning.pdf

# 6 A5 Transport Layer Security

The transport layer protocol is omnipresent and thus important to be understood. Cyber professionals need to be able to judge TLS configurations and thus you are going to learn a bit about the SSLyze toolkit that helps to query services for their configuration and judge the result respectively define minimum requirements.

## 6.1 Step 1.1 Run SSLyze

You may run SSLyze against your local Apache or any other public service as follows:

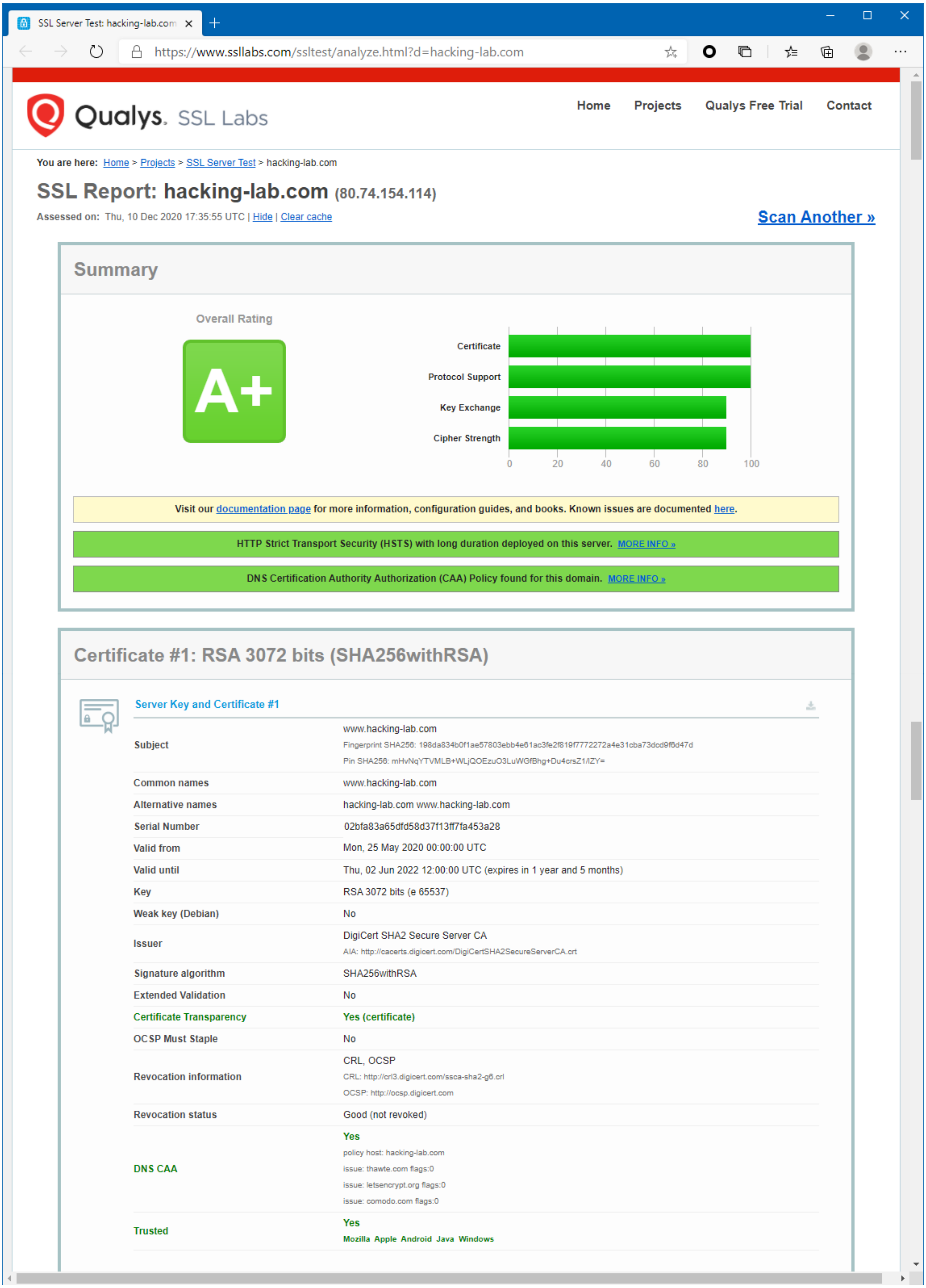

| No. | Description of Test | Actual Result |
|---|---|---|
| 1. | Are only strong ciphers supported? | Yes. However, some do not support perfect forward secrecy. See orange marked suites |
| 2. | Does the server support and prefer cipher suites with forward secrecy? | Yes. |
| 3. | Does the server support strong protocol versions? | TLS 1.2, 1.1 and 1.0 supported. Strong would mean 1.2+ |
| 4. | Does the server support downgrade detection? | Supported. |
| 5. | What is TLS_FALLBACK_SCSV? | Signaling Cipher Suite Value (SCSV) can be employed to prevent unintended protocol downgrades between clients and servers. The client uses the value to signal he is using a fallback and the server may return a fatal alert to signal it could cope with better versions. |
| 6. | Does the server support secure TLS renegotiation? | Yes. |
| 7. | Does the server support client-initiated renegotiation? | No. |
| 8. | Is TLS compression support enabled? | No. |
| 9. | Is the server vulnerable to the Heartbleed attack? | No. |
| 10. | Is the server vulnerable to the OpenSSL CCS injection attack? | No. |
| 11. | Is the server vulnerable to the ROBOT attack?<br><br>Reference: [https://robotattack.org](https://robotattack.org) | No. |
| 12. | Does the Domain use CAA to specify CAs, which can be used to issue certificates for it? | No. |

```
root@hlkali:~# sslyze --regular localhost

 CHECKING HOST(S) AVAILABILITY
 -----------------------------

   localhost:443                       => 127.0.0.1


 SCAN RESULTS FOR LOCALHOST:443 - 127.0.0.1
 ------------------------------------------

 * OpenSSL Heartbleed:
                                          OK - Not vulnerable to Heartbleed

 * TLS 1.3 Cipher suites:
     Attempted to connect using 5 cipher suites; the server rejected all cipher suites.

 * ROBOT Attack:
                                          OK - Not vulnerable.

 * TLS 1.2 Session Resumption Support:
      With Session IDs: OK - Supported (5 successful resumptions out of 5 attempts).
      With TLS Tickets: OK - Supported.

 * OpenSSL CCS Injection:
                                          OK - Not vulnerable to OpenSSL CCS injection

 * TLS 1.2 Cipher suites:
     Attempted to connect using 158 cipher suites.

     The server accepted the following 46 cipher suites:
        TLS_RSA_WITH_SEED_CBC_SHA                        128
        TLS_RSA_WITH_CAMELLIA_256_CBC_SHA256             256
        TLS_RSA_WITH_CAMELLIA_256_CBC_SHA                256
        TLS_RSA_WITH_CAMELLIA_128_CBC_SHA256             128
        TLS_RSA_WITH_CAMELLIA_128_CBC_SHA                128
        TLS_RSA_WITH_ARIA_256_GCM_SHA384                 256
        TLS_RSA_WITH_ARIA_128_GCM_SHA256                 128
        TLS_RSA_WITH_AES_256_GCM_SHA384                  256
        TLS_RSA_WITH_AES_256_CCM_8                       128
        TLS_RSA_WITH_AES_256_CCM                         256
        TLS_RSA_WITH_AES_256_CBC_SHA256                  256
        TLS_RSA_WITH_AES_256_CBC_SHA                     256
        TLS_RSA_WITH_AES_128_GCM_SHA256                  128
        TLS_RSA_WITH_AES_128_CCM_8                       128
        TLS_RSA_WITH_AES_128_CCM                         128
        TLS_RSA_WITH_AES_128_CBC_SHA256                  128
        TLS_RSA_WITH_AES_128_CBC_SHA                     128
        TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256      256       ECDH: x25519 (253 bits)
        TLS_ECDHE_RSA_WITH_CAMELLIA_256_CBC_SHA384       256       ECDH: x25519 (253 bits)
        TLS_ECDHE_RSA_WITH_CAMELLIA_128_CBC_SHA256       128       ECDH: x25519 (253 bits)
        TLS_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384           256       ECDH: x25519 (253 bits)
        TLS_ECDHE_RSA_WITH_ARIA_128_GCM_SHA256           128       ECDH: x25519 (253 bits)
        TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384            256       ECDH: prime256v1 (256
bits)
        TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384            256       ECDH: prime256v1 (256
bits)
        TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA               256       ECDH: prime256v1 (256
bits)
        TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256            128       ECDH: prime256v1 (256
bits)
        TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256            128       ECDH: prime256v1 (256
bits)
        TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA               128       ECDH: prime256v1 (256
bits)
        TLS_DHE_RSA_WITH_SEED_CBC_SHA                    128       DH (2048 bits)
        TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256        256       DH (2048 bits)
        TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA256         256       DH (2048 bits)
        TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA            256       DH (2048 bits)
        TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA256         128       DH (2048 bits)
        TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA            128       DH (2048 bits)
```

```
        TLS_DHE_RSA_WITH_ARIA_256_GCM_SHA384              256       DH (2048 bits)
        TLS_DHE_RSA_WITH_ARIA_128_GCM_SHA256              256       DH (2048 bits)
        TLS_DHE_RSA_WITH_AES_256_GCM_SHA384               256       DH (2048 bits)
        TLS_DHE_RSA_WITH_AES_256_CCM_8                    256       DH (2048 bits)
        TLS_DHE_RSA_WITH_AES_256_CCM                      256       DH (2048 bits)
        TLS_DHE_RSA_WITH_AES_256_CBC_SHA256               256       DH (2048 bits)
        TLS_DHE_RSA_WITH_AES_256_CBC_SHA                  256       DH (2048 bits)
        TLS_DHE_RSA_WITH_AES_128_GCM_SHA256               128       DH (2048 bits)
        TLS_DHE_RSA_WITH_AES_128_CCM_8                    128       DH (2048 bits)
        TLS_DHE_RSA_WITH_AES_128_CCM                      128       DH (2048 bits)
        TLS_DHE_RSA_WITH_AES_128_CBC_SHA256               128       DH (2048 bits)
        TLS_DHE_RSA_WITH_AES_128_CBC_SHA                  128       DH (2048 bits)

     The group of cipher suites supported by the server has the following properties:
       Forward Secrecy                   OK - Supported
       Legacy RC4 Algorithm              OK - Not Supported

     The server is configured to prefer the following cipher suite:
        TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384             256       ECDH: prime256v1 (256
bits)


 * TLS 1.0 Cipher suites:
     Attempted to connect using 80 cipher suites.

     The server accepted the following 12 cipher suites:
        TLS_RSA_WITH_SEED_CBC_SHA                         128
        TLS_RSA_WITH_CAMELLIA_256_CBC_SHA                 256
        TLS_RSA_WITH_CAMELLIA_128_CBC_SHA                 128
        TLS_RSA_WITH_AES_256_CBC_SHA                      256
        TLS_RSA_WITH_AES_128_CBC_SHA                      128
        TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA                256       ECDH: prime256v1 (256
bits)
        TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA                128       ECDH: prime256v1 (256
bits)
        TLS_DHE_RSA_WITH_SEED_CBC_SHA                     128       DH (2048 bits)
        TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA             256       DH (2048 bits)
        TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA             128       DH (2048 bits)
        TLS_DHE_RSA_WITH_AES_256_CBC_SHA                  256       DH (2048 bits)
        TLS_DHE_RSA_WITH_AES_128_CBC_SHA                  128       DH (2048 bits)

     The group of cipher suites supported by the server has the following properties:
       Forward Secrecy                   OK - Supported
       Legacy RC4 Algorithm              OK - Not Supported

     The server is configured to prefer the following cipher suite:
        TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA                256       ECDH: prime256v1 (256
bits)


 * Session Renegotiation:
       Client-initiated Renegotiation:   OK - Rejected
       Secure Renegotiation:             OK - Supported

 * TLS 1.1 Cipher suites:
     Attempted to connect using 80 cipher suites.

     The server accepted the following 12 cipher suites:
        TLS_RSA_WITH_SEED_CBC_SHA                         128
        TLS_RSA_WITH_CAMELLIA_256_CBC_SHA                 256
        TLS_RSA_WITH_CAMELLIA_128_CBC_SHA                 128
        TLS_RSA_WITH_AES_256_CBC_SHA                      256
        TLS_RSA_WITH_AES_128_CBC_SHA                      128
        TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA                256       ECDH: prime256v1 (256
bits)
        TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA                128       ECDH: prime256v1 (256
bits)
        TLS_DHE_RSA_WITH_SEED_CBC_SHA                     128       DH (2048 bits)
        TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA             256       DH (2048 bits)
        TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA             128       DH (2048 bits)
        TLS_DHE_RSA_WITH_AES_256_CBC_SHA                  256       DH (2048 bits)
        TLS_DHE_RSA_WITH_AES_128_CBC_SHA                  128       DH (2048 bits)
```

```
     The group of cipher suites supported by the server has the following properties:
       Forward Secrecy                    OK - Supported
       Legacy RC4 Algorithm               OK - Not Supported

     The server is configured to prefer the following cipher suite:
        TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA                256       ECDH: prime256v1 (256
bits)


 * Deflate Compression:
                                          OK - Compression disabled

 * Certificates Information:
       Hostname sent for SNI:             localhost
       Number of certificates detected:   1


     Certificate #0 ( _RSAPublicKey )
       SHA1 Fingerprint:                  8ce110996253299743f3bca357c07c4448f02ff2
       Common Name:                       localhost
       Issuer:                            Cyrill's Root CA
       Serial Number:                     3
       Not Before:                        2020-12-10
       Not After:                         2030-12-08
       Public Key Algorithm:              _RSAPublicKey
       Signature Algorithm:               sha256
       Key Size:                          2048
       Exponent:                          65537
       DNS Subject Alternative Names:     ['localhost']

     Certificate #0 - Trust
       Hostname Validation:               OK - Certificate matches server hostname
       Android CA Store (9.0.0_r9):       FAILED - Certificate is NOT Trusted: unable to get
local issuer certificate
       Apple CA Store (iOS 13, iPadOS 13, macOS 10.15, watchOS 6, and tvOS 13):FAILED -
Certificate is NOT Trusted: unable to get local issuer certificate
       Java CA Store (jdk-13.0.2):        FAILED - Certificate is NOT Trusted: unable to get
local issuer certificate
       Mozilla CA Store (2020-06-21):     FAILED - Certificate is NOT Trusted: unable to get
local issuer certificate
       Windows CA Store (2020-05-04):     FAILED - Certificate is NOT Trusted: unable to get
local issuer certificate
       Symantec 2018 Deprecation:         ERROR - Could not build verified chain
(certificate untrusted?)
       Received Chain:                    localhost
       Verified Chain:                    ERROR - Could not build verified chain
(certificate untrusted?)
       Received Chain Contains Anchor:    ERROR - Could not build verified chain
(certificate untrusted?)
       Received Chain Order:              OK - Order is valid
       Verified Chain contains SHA1:      ERROR - Could not build verified chain
(certificate untrusted?)

     Certificate #0 - Extensions
       OCSP Must-Staple:                  NOT SUPPORTED - Extension not found
       Certificate Transparency:          NOT SUPPORTED - Extension not found

     Certificate #0 - OCSP Stapling
                                          NOT SUPPORTED - Server did not send back an OCSP
response

 * SSL 2.0 Cipher suites:
     Attempted to connect using 7 cipher suites; the server rejected all cipher suites.

 * Downgrade Attacks:
       TLS_FALLBACK_SCSV:                 OK - Supported

 * SSL 3.0 Cipher suites:
     Attempted to connect using 80 cipher suites; the server rejected all cipher suites.
 SCAN COMPLETED IN 25.74 S
 -------------------------
```

Alternatively, you may visit SSL Labs from Qualys https://www.ssllabs.com/ssltest/analyze.html?d=hacking%2dlab.com&latest



SSL Server Test: hacking-lab.com

https://www.ssllabs.com/ssltest/analyze.html?d=hacking-lab.com

## Configuration

### Protocols

| | |
|---|---|
| TLS 1.3 | No |
| TLS 1.2 | Yes |
| TLS 1.1 | No |
| TLS 1.0 | No |
| SSL 3 | No |
| SSL 2 | No |

### Cipher Suites

# TLS 1.2 (suites in server-preferred order)

| | |
|---|---|
| TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f) ECDH secp256r1 (eq. 3072 bits RSA) FS | 128 |
| TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030) ECDH secp256r1 (eq. 3072 bits RSA) FS | 256 |
| TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0x9e) DH 3072 bits FS | 128 |
| TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0x9f) DH 3072 bits FS | 256 |

### Handshake Simulation

| | | | | |
|---|---|---|---|---|
| Android 4.4.2 | RSA 3072 (SHA256) | TLS 1.2 | TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 | ECDH secp256r1 FS |
| Android 5.0.0 | RSA 3072 (SHA256) | TLS 1.2 | TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 | ECDH secp256r1 FS |
| Android 6.0 | RSA 3072 (SHA256) | TLS 1.2 | TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 | ECDH secp256r1 FS |
| Android 7.0 | RSA 3072 (SHA256) | TLS 1.2 | TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 | ECDH secp256r1 FS |
| Android 8.0 | RSA 3072 (SHA256) | TLS 1.2 | TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 | ECDH secp256r1 FS |

SSL Server Test: hacking-lab.com

https://www.ssllabs.com/ssltest/analyze.html?d=hacking-lab.com

### Protocol Details

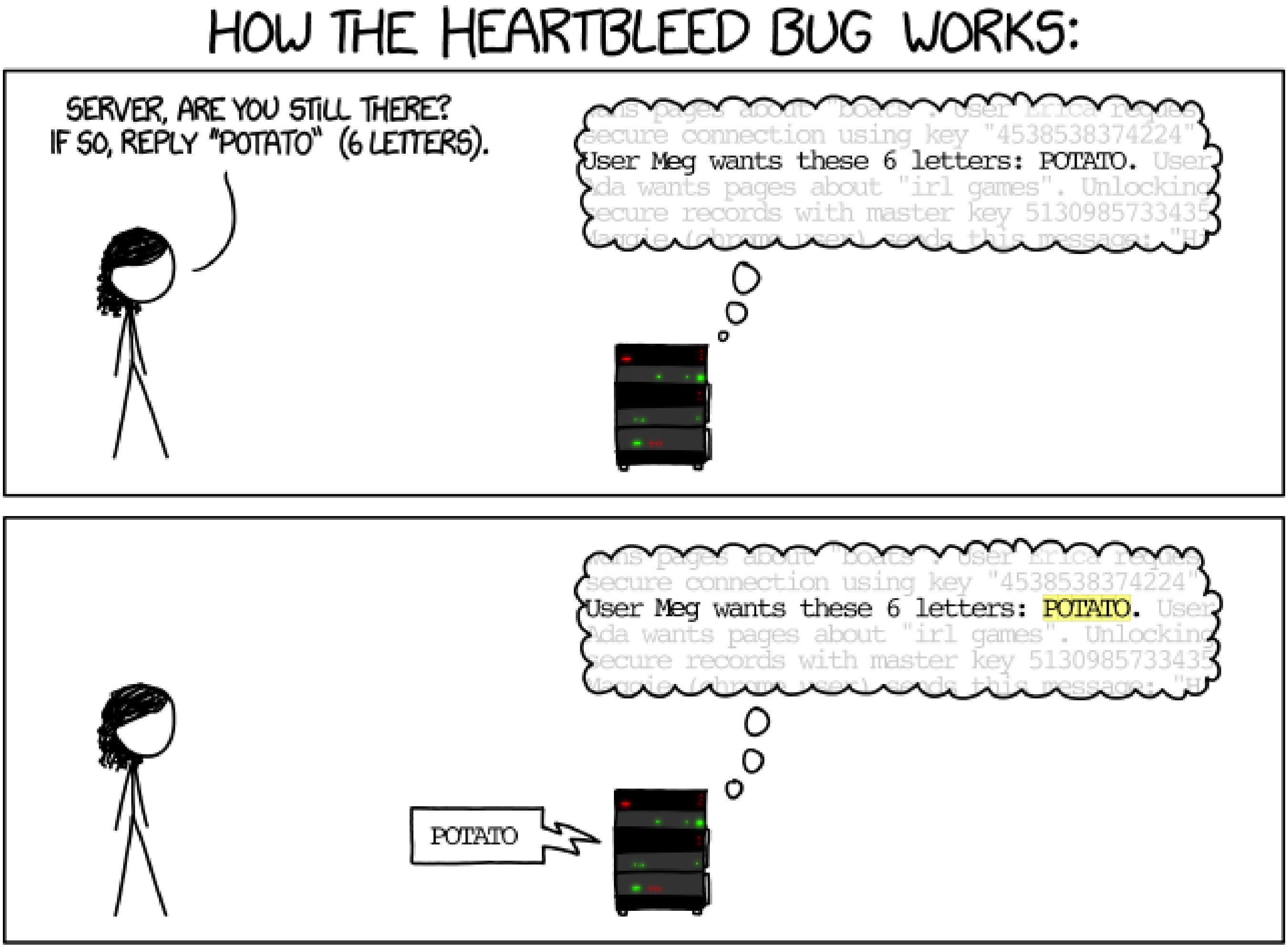
| | |
|---|---|
| DROWN | No, server keys and hostname not seen elsewhere with SSLv2<br>(1) For a better understanding of this test, please read this longer explanation<br>(2) Key usage data kindly provided by the Censys network search engine; original DROWN website here<br>(3) Censys data is only indicative of possible key and certificate reuse; possibly out-of-date and not complete |
| Secure Renegotiation | Supported |
| Secure Client-Initiated Renegotiation | No |
| Insecure Client-Initiated Renegotiation | No |
| BEAST attack | Mitigated server-side (more info) |
| POODLE (SSLv3) | No, SSL 3 not supported (more info) |
| POODLE (TLS) | No (more info) |
| Zombie POODLE | No (more info) |
| GOLDENDOODLE | No (more info) |
| OpenSSL 0-Length | No (more info) |
| Sleeping POODLE | No (more info) |
| Downgrade attack prevention | Unknown (requires support for at least two protocols, excl. SSL2) |
| SSL/TLS compression | No |
| RC4 | No |
| Heartbeat (extension) | Yes |
| Heartbleed (vulnerability) | No (more info) |
| Ticketbleed (vulnerability) | No (more info) |
| OpenSSL CCS vuln. (CVE-2014-0224) | No (more info) |
| OpenSSL Padding Oracle vuln. (CVE-2016-2107) | No (more info) |
| ROBOT (vulnerability) | No (more info) |
| Forward Secrecy | Yes (with most browsers) ROBUST (more info) |
| ALPN | No |
| NPN | No |
| Session resumption (caching) | Yes |
| Session resumption (tickets) | Yes |

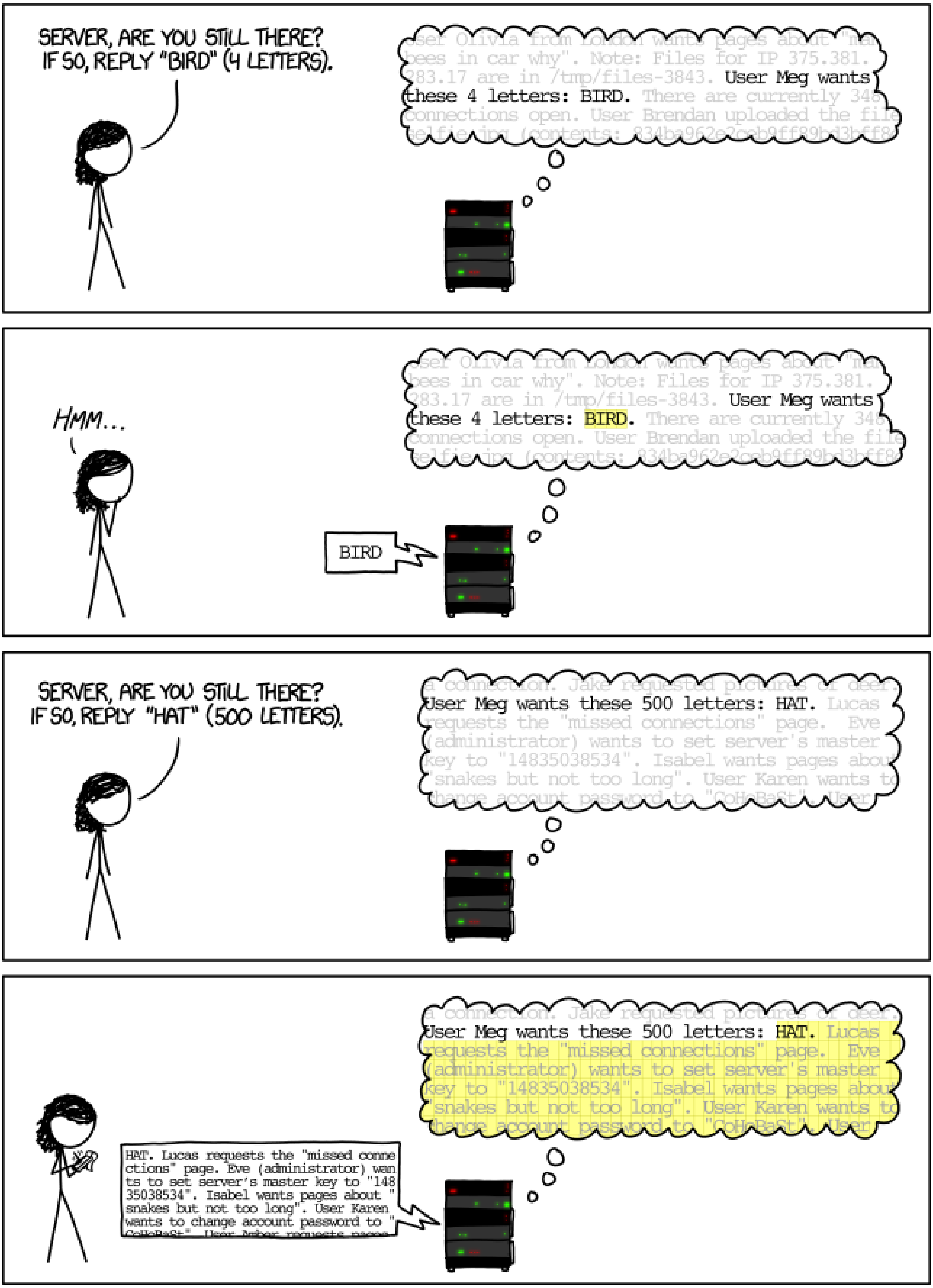
## 6.2 Step 1.2 Justify Questions

Please give a bit of background what each of the above questions aims at. Maybe describe what attack should be prevented by proper configuration for each of the questions.

| No. | Description of Test | Justification |
|---|---|---|
| 1. | Are only strong ciphers supported? | We want to avoid weak ciphers. Nowadays, we aim to have at least 128-bit key size for encryption and 256-bit hash size.<br><br>Ideally, we also employ DH to enable forward secrecy and use a block mode such as GCM to avoid all the issues we had with mac-then-encrypt (CBC, padding oracles). |
| 2. | Does the server support and prefer cipher suites with forward secrecy? | Mallory could capture TLS connections for a long time and hope that one day the key material of a specific server is leaked/stolen and would therefore allow to unwrap session keys which could be used to decrypt an entire TLS sessions.<br><br>To avoid this scenario, forward secrecy should be enforced. For that purpose, DH is used to agree on keys. If DH is used then DH groups should match or exceed the RSA key size.<br><br>DH ≥ 2048-bit<br>ECDH ≥ 256-bit |
| 3. | Does the server support strong protocol versions? | In 2020 TLS 1.2 and 1.3 are state of the art.<br><br>TLS 1.1, 1.0 should not be used for application with high security requirements. Mind that Windows XP and Java 7 do not by default support TLS1.1 or higher which might cause problems or block a large client base.<br><br>SSLv3 should not be supported, as it has known weaknesses.<br><br>SSLv2 should never be supported by the server. It's totally flawed. |
| 4. | Does the server support downgrade detection? | Downgrade detection is important to prevent Mallory of forcing the client to fallback to less secure versions of the protocol and exploit these easily.<br><br>https://tools.ietf.org/html/rfc7507 |
| 5. | What is TLS_FALLBACK_SCSV? | See above. |
| 6. | Does the server support secure TLS renegotiation? | Mallory could abuse renegotiation to injection its own plaintext into a victim TLS session. It could for example injection GET requests or SMTP commands.<br><br>Note: Even if secure TLS renegotiation is supported, this does not necessarily mean that insecure renegotiation is disabled. |
| 7. | Does the server support client-initiated renegotiation? | Client initiated renegotiation can be abused to perform a DoS attack on the server. |
| 8. | Is TLS compression support enabled? | There is a side-channel attack known as CRIME. It abuses the TLS compression support or compression in SPDY to e.g. extract session tokens. |
| 9. | Is the server vulnerable to the Heartbleed attack? | The Heartbleed bug allowed extraction of large server memory portions due to OpenSSL implementation flaws in the heart beat functionality. See xkcd pic below for a simple description. |

| No. | Description of Test | Justification |
|---|---|---|
| 10. | Is the server vulnerable to the OpenSSL CCS injection attack? | Mallory could force hosts that use OpenSSL as the library for SSL/TLS to use weak key material. This was possible due to a flaw in the ChangeCipherSpec implementation. |
| 11. | Is the server vulnerable to the ROBOT attack?<br><br>Reference: [https://robotattack.org](https://robotattack.org) | The ROBOT attack is a slightly tuned version of very long know oracle attack against PKCS#1 v1.5 padding. It can be mounted against TLS_RSA cipher suites but not DHE/ECDHE enabled ones. It affected quite some SSL/TLS implementations and ultimately enabled an attacker to decipher traffic, or sign messages as with being in possession of the server private key material. |
| 12. | Does the Domain use CAA to specify CAs, which can be used to issue certificates for it? | "The Certification Authority Authorization (CAA) DNS Resource Record allows a DNS domain name holder to specify one or more Certification Authorities (CAs) authorized to issue certificates for that domain name.  CAA Resource Records allow a public CA to implement additional controls to reduce the risk of unintended certificate mis-issue.", [https://tools.ietf.org/html/rfc8659](https://tools.ietf.org/html/rfc8659)<br><br>Assuming Mallory would like to social engineer an arbitrary CA to sign a certificate for a domain that would actually belong to you, than the CA has additional means to check your DNS entries whether it is allowed to sign certs for your domain at all. |






https://xkcd.com/1354/

# 7 A5 Key Management

Why do you need to have a clue about key management? Well, if channels and data are properly encrypted the only thing left for an attacker is going after the keys. Moreover, if keys get lost or are misplaced an entire system, service or dataset may break or remain unrecoverable. We really want to avoid such scenario. Key management principles provide you with some general ideas that need to be followed to assure CIA for the environments you take care off.

The OWASP Cheat Sheet on Key Management provides a rough overview on important aspects when implementing key management in practice - specifically with applications. As we have already covered bits and bytes in previous challenges, we rather focus on the processes which are generic.

Consult the questions and answer those based on the OWASP Cheat Sheet.

**1. Name major stages of the key lifecycle**

- Generation
- Distribution
- Storage
- Backup
- Recovery
- Escrow
- Destruction

**2. Name places where keys could be present/stored. Sort them best to worst location**

My take on this would be
- In files
- In application
- On disk
- In persistent memory
- In volatile memory
- In isolated cryptographic services
- In hardware security module (HSM)

**3. Name techniques to protect stored keys?**

- Hide in software… ever tried to command "strings" on a binary?
- Scramble… I am sure you had your hands on a debugger yet?
- Key wrapping using key encryption keys (KEK). The question is where you actually store the KEK. You may note that we just move the problem around. However, sometimes security is about making it difficult for the adversary.
- Hardware security module. Same here. Where are the credentials stored to access the HSM?

Ultimately where you store and how you protect keys is mainly governed by potential threats or by the risk appetite.

**4. What is important when generating keys?**

- Keys should be generated using strong RNGs or PRNGs
- Weak or semi-weak keys must be avoided depend on the algorithm the key is ultimately used for

**5. How long should a key be valid?**

Cryptographers refer to the key validity as the "cryptoperiod". Keys should basically only be used for that specific lifetime or cryptoperiod. Obviously short keys have usually shorter cryptoperiods. However, the cryptoperiod depends on the use case and considered risks.

**6. What needs to be considered when keys are distributed?**

Keys need to be distributed over secure channels whereby the protection of the channel should be at least as good as the transmitted key. Same applies for KEKs.

Mind regular updates where you will be required to roll-over keys. It is important to understand whether there are requirements to have seamless roll-over in running systems or if services could be halted for the update. You may also need to consider a roll-back scenario.

**7. Assuming you need a backup of your keys. Where would you place it and how would you protect the keys from illegitimate access. Who guards the guards?**

Key material backups need to employ stronger key material than the contained keys. The recovery of keys should require multiple parties to give consent before such keys could be recovered.

Mind that, signing keys should never be recovered/escrowed as this would render the signatures useless. The previous owner of the signing key could claim you had forged all of the signature due to you are in possession of the signing key material. This is a bit of a drawback considering backups of certificates used for S/MIME as you would like to recover encrypted messages. Unfortunately, the same key is actually being used for mail signatures. You would need to have proper logs of your mail infrastructure to claim certain messages had been sent by a specific person regardless of S/MIME signatures.

**8. Name concepts on how you could prevent a single person to access a system or key?**

This is very difficult as someone actually needs to maintain the backup system. However, having tight monitoring, logging, auditing and session recording around the system makes responsible administrators very much accountable and aware of everything they do on such system. See below for approaches to require consent of further parties.

**9. Name concepts to reveal keys in an emergency but only with approval of multiple persons?**

Secret sharing is used to distribute a secret over multiple parties. To avoid denial of service in case of one of the party loses its share the scheme could be tuned to maybe allow 5 of the 7 shares being accepted as valid secret. This is how the DNSSEC root key is protected.

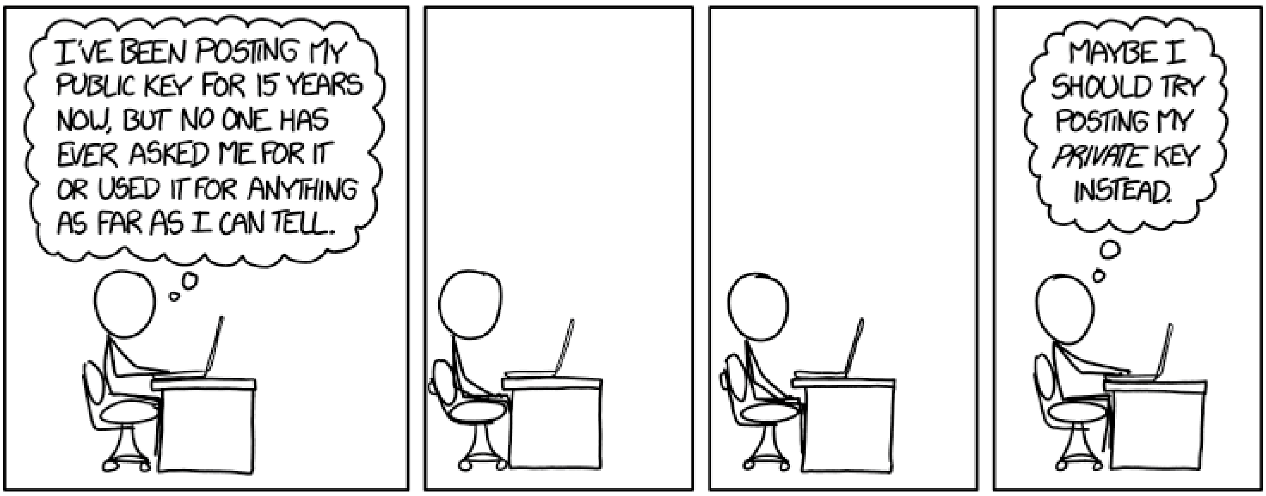
https://www.iana.org/dnssec/tcrs

| | |
|---|---|
| Moussa Guebre, BF | Recovery Key Share Holder (2010-) |
| Dan Kaminsky, US | Recovery Key Share Holder (2010-) |
| Kristian Ørmen, DK | Recovery Key Share Holder (2017-) |
| Norm Ritchie, CA | Recovery Key Share Holder (2010-) |
| Ondřej Surý, CZ | Recovery Key Share Holder (2010-) |
| Bevil Wooding, TT | Recovery Key Share Holder (2010-) |
| Jiankang Yao, CN | Recovery Key Share Holder (2010-) |

**10. Assuming we lost a key. What needs to be considered? Provide a bullet-list.**

- Panic 😊
- Is the key still valid or already expired?
- Are the attackers still in?
- Can we replace the key?
- Do we know all deployments of the key?
- What was the key intended use? Encryption, integrity, authentication, signatures …?
- Might the key be used for something else than the intended purpose?
- Does someone having the key know how and where to use it?
- What could an attacker do? Decipher single packets, corporate secrets, impersonate employees, access systems?
- Is the public aware of the leak?

Maybe better create a playbook for such scenario.




https://xkcd.com/1553/