Extended public key cryptography in OpenSSL
HW6-7 - CNS Sapienza

Giulia Muscarà 1743261

December 19, 2019

1 Objectives

The purpose of this report is to present and analyze some OpenSSL tools for public key cryptography. In particular, the focus is on RSA and DSA asymmetric encryption protocols. Public and private keys were generated, along with X.509 digital certificates and digital signatures. Then a virtual network was created using Netkit environment to simulate the exchange of messages between a Certificate Authority and a host, to request and receive certificates' signatures and revocations. To carry out the analysis, the commands were executed on a scriptable Bash shell running on a Lubuntu Netkit virtual box, as it provides a fast and lightweight operating system with a clean and easy-to-use user interface and it allows to emulate a network.

2 Private key generation

To generate a private key, OpenSSL offers, among others, the command *genpkey*. By command line, it requires the specification of the desired algorithm to be used and the name of the output file *.pem* where the key will be stored. In this case, two private keys were generated, one by using DSA and the other by using RSA, using the default cipher and operating mode for both of them. While for RSA private key the above said options are sufficient, DSA protocol also requires the specification of the *-paramfile*, with a *.pem* file containing the necessary parameters to compute the key. To randomly generate the parameters, beforehand the command *genpkey* was used with the option *-genparam* to produce the needed file *dsaparams.pem* with 1024-bit long parameters, so that the dsa key could be computed starting from them. After being created, the two private keys were stored in the respective files *private_rsa.pem* and *private_dsa.pem*.

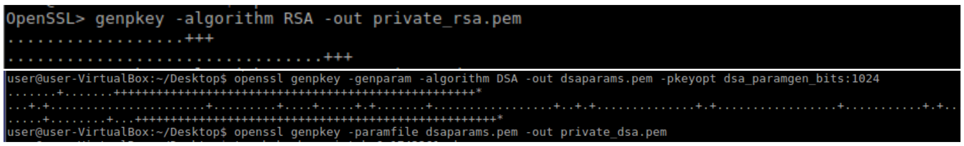


Figure 1: RSA and DSA private keys generation

3 Public key extraction

The previously generated private keys include the respective public ones, that were extracted from the files containing the private keys.
As far as the rsa private key was concerned, to extract the public RSA one, *rsa* OpenSSL command was used. In particular, through *-in* and *-pubout -out* options, from the file containing the private key, another *.pem* file was created, storing in it the associated RSA public key. The same procedure was carried out for the extraction of DSA public key. This time, *dsa* command was adopted with the same options specified for RSA. The two public keys were stored in the respective files *public_rsa.pem* and *public_dsa.pem*.

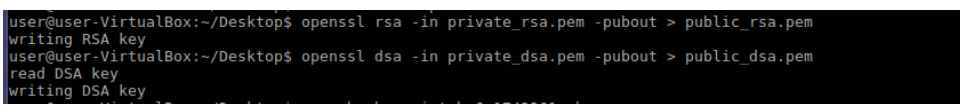


Figure 2: DSA and RSA public keys extraction

4 X.509 certificate generation

Then, X.509 digital certificates were created and verified by self-signing it. X.509 standard defines the format of public key certificates. Certificates contain a public key and the related identity, and it is either signed by a certificate authority or self-signed. In order to create an X.509 certificate, the *req* OpenSSL command was used. The *-new* option specified the request of a new Certificate Signing Request, the *-key* option allows to specify a file containing an existing key and *-out* is used to store in the *rsa_certificate_req.csr* file the generated CSR. It was needed to insert a series of details about who is requesting the certificate, that in this case were left on default values, and a password challenge for the key by command line.

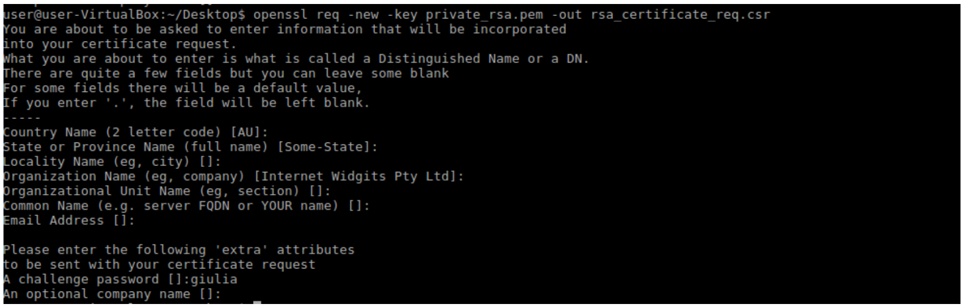


Figure 3: RSA certificate request

After creating a CSR, in order to self-sign the certificate, the private keys were used to prove the identity of the certificate owner. Through the command *x509*, the CSR was signed with the private key and the number of days of validity were specified. After the verification, the *rsa_certificate.pem* file was created, which is the actual certificate. The content of the certificate was displayed as follows.

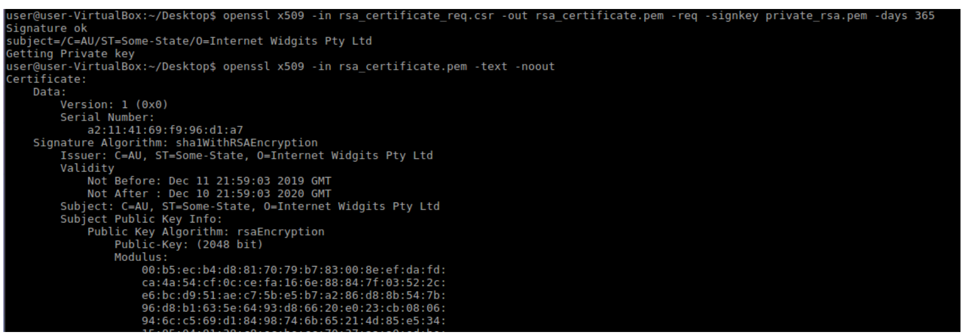


Figure 4: RSA certificate signing

The exact same procedure can be applied to create a certificate signed with the DSA private key.
Finally, in order to verify the created certificates, the *verify* command was used, as shown in Fig. 5.

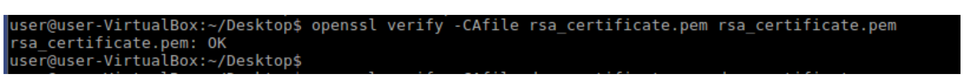


Figure 5: RSA certificate verification

The exact same procedure can be applied to verify a certificate signed with the DSA private key.

5 Digital signatures

Digital signatures securely associate a signer to its identity and a document. DSs come in the standard PKI format, to provide the highest levels of security and universal acceptance. While during a digital signature transaction the private key is not shared but it is used only by the signer to digitally sign documents, the public one is openly available and used by who needs to validate the authenticity of the digital signature. To digitally sign a sample empty document *to_sign.txt* in OpenSSl and to verify the signature, we need a pair *(privatekey, publickey)*, which was already generated. By using the OpenSSL command *dgst* with the option -*sign*, the file *sign.sha256* is created in the /tmp/ folder, containing the binary format of the digital signature, which was then encoded into the *rsa_digital_signature.pem* file. On the other hand, to verify the signature, the public key was used. The verification consists in the inverse process: the *rsa_digital_signature.pem* file was decoded into the *sign.sha256* file and the latter was decrypted by using the public key. *Verification OK* was printed, as the process ended correctly.

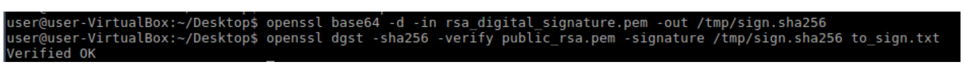


Figure 6: RSA signature verification

The exact same procedure can be applied to verify a digital signature created with a DSA private key.

6 Certificate formats convertion

There are different file extensions for X.509 certificates. Among the most used there are the *.pem* format, that stands for Privacy-enhanced Electronic Mail and Base64 encoded, *.cer, .crt, .der* formats, that indicate binary DER form, *.p7b, .p7c*, for PKCS7 SignedData structure without data, just certificates or CRLs, and *.p12*, which may contain certificates and private keys.
The certificates were previously created using PEM, in Base64 format, starting with "BEGIN CERTIFICATE" and "END CERTIFICATE". OpenSSL

allows to convert certificates in other formats, like DER, which has a binary form. In particular, by using the *x509* command PEM format files can be transformed into DER files and viceversa, by specifying the format of the file to convert with the option *-inform* and the format of the desired output with the option *-outform*.

7 Network simulation

To make a practical example of a host making requests and getting responses from an external CA, a virtual network was built up thanks to the Netkit environment. Netkit allows to set up and to perform networking experiments by creating several virtual network devices that can be interconnected.
The sample network is composed by two VMs, called v1 and v2, the former acting as a CA and the latter as a simple host, and by a router r that interconnects v1 and v2. To make sure that each VM can communicate with the others and exchange files and requests the ICMP protocol was exploited, pinging the IP addresses of the terminals, that were chosen statically and randomly. The topology is characterized by the fact that the CA and the host are situated in two different LANs. The first subnet has address 192.168.0.0/24 and it is formed by v1 and the interface eth0 of the router, while the second LAN has address 10.0.0.0/24 and is made of v2 and the interface eth1 of r. The topology is shown in Fig. 7.

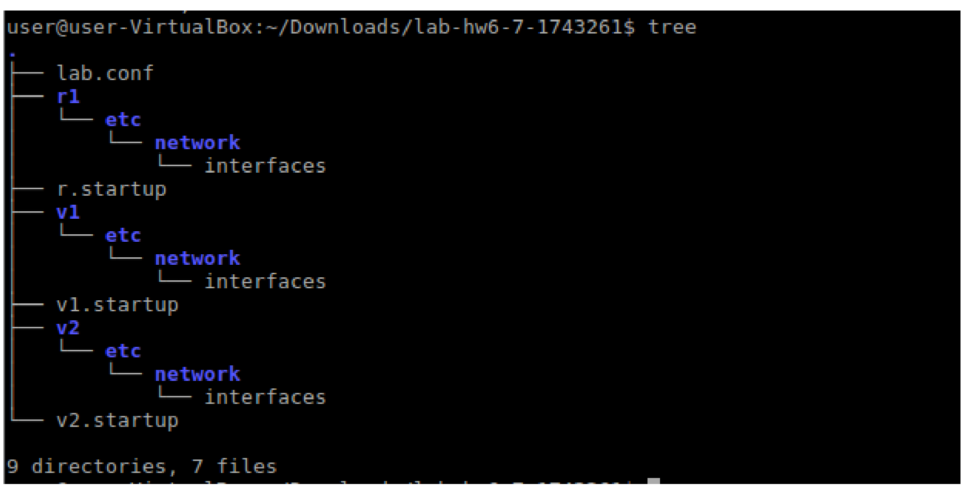
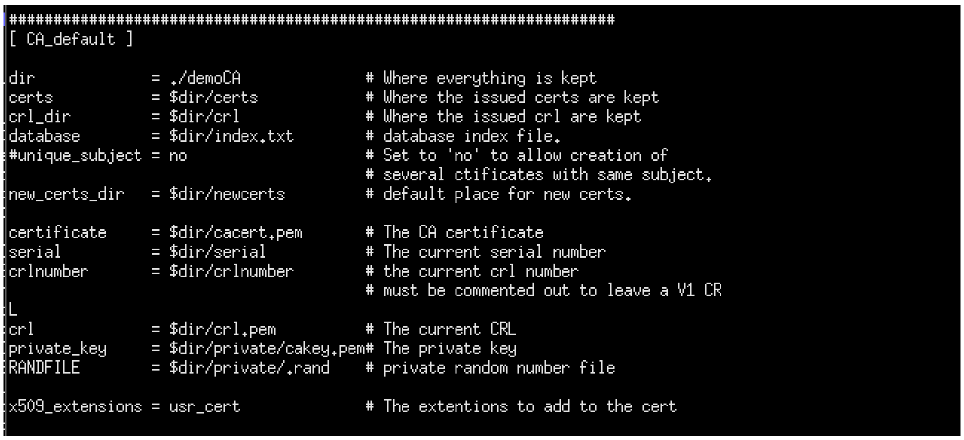


Figure 7: Network topology

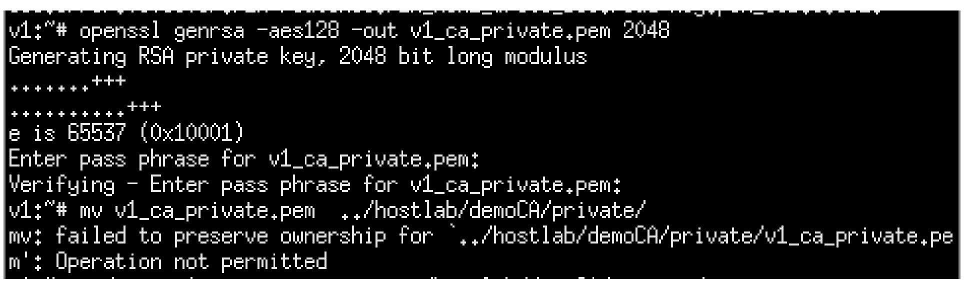
After verifying that v1 was reachable from v2 through pings, because the *openssl.cnf* file at *v1/etc/ssl/openssl.cnf* specifies where the certificates, keys, crl, databases should be stored to be correctly managed when using OpenSSL, the necessary folders were created as well.



```
####################################################################
[ CA_default ]

dir             = ./demoCA              # Where everything is kept
certs           = $dir/certs            # Where the issued certs are kept
crl_dir         = $dir/crl              # Where the issued crl are kept
database        = $dir/index.txt        # database index file.
#unique_subject = no                    # Set to 'no' to allow creation of
                                        # several ctificates with same subject.
new_certs_dir   = $dir/newcerts         # default place for new certs.

certificate     = $dir/cacert.pem       # The CA certificate
serial          = $dir/serial           # The current serial number
crlnumber       = $dir/crlnumber        # the current crl number
                                        # must be commented out to leave a V1 CR
L
crl             = $dir/crl.pem          # The current CRL
private_key     = $dir/private/cakey.pem# The private key
RANDFILE        = $dir/private/.rand    # private random number file

x509_extensions = usr_cert              # The extentions to add to the cert
```

Figure 8: Folders structure

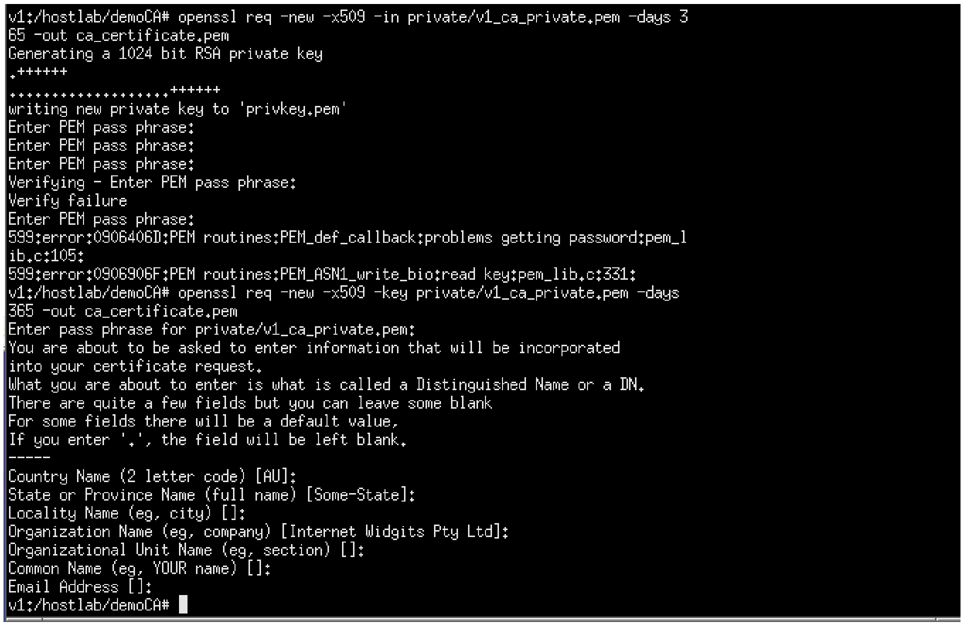
To make v1 a CA, its private key was generated using the *genrsa* function and it was moved to the */CADemo/private/* folder.



```
v1:~# openssl genrsa -aes128 -out v1_ca_private.pem 2048
Generating RSA private key, 2048 bit long modulus
.......+++
..........+++
e is 65537 (0x10001)
Enter pass phrase for v1_ca_private.pem:
Verifying - Enter pass phrase for v1_ca_private.pem:
v1:~# mv v1_ca_private.pem  ../hostlab/demoCA/private/
mv: failed to preserve ownership for `../hostlab/demoCA/private/v1_ca_private.pe
m': Operation not permitted
```

Figure 9: CA private key generation

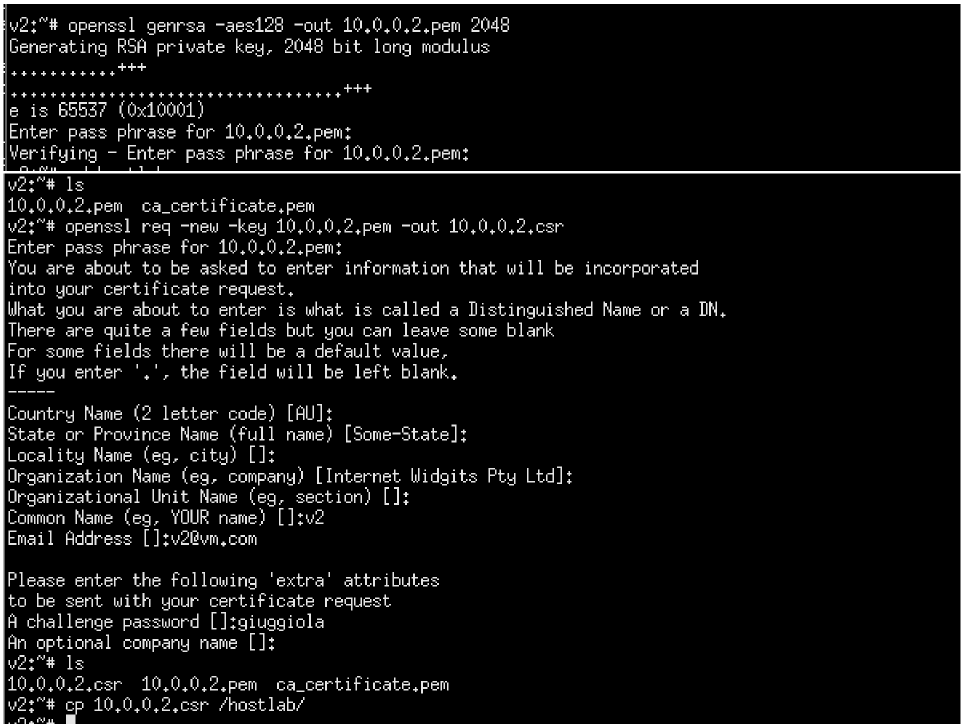
Then a self-signed certificate for the CA was created, to distribute it to all the hosts of the network that need to trust v1, propagating it to v2 through the network.



```
v1:/hostlab/demoCA# openssl req -new -x509 -in private/v1_ca_private.pem -days 3
65 -out ca_certificate.pem
Generating a 1024 bit RSA private key
.++++++
.................++++++
writing new private key to 'privkey.pem'
Enter PEM pass phrase:
Enter PEM pass phrase:
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
Verify failure
Enter PEM pass phrase:
599:error:0906406D:PEM routines:PEM_def_callback:problems getting password:pem_l
ib.c:105:
599:error:0906906F:PEM routines:PEM_ASN1_write_bio:read key:pem_lib.c:331:
v1:/hostlab/demoCA# openssl req -new -x509 -key private/v1_ca_private.pem -days
365 -out ca_certificate.pem
Enter pass phrase for private/v1_ca_private.pem:
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.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:
Email Address []:
v1:/hostlab/demoCA#
```

Figure 10: CA's self-signed certificate

As far as v2 is concerned, after receiving the CA's certificate, its private key was generated as well as a CSR for the CA, using the ip address of v2 as domain name. During the creation of the CSR, the fields of the common name and the email were specified as v2 and v2@vm.com respectively. After its creation, the *.csr* file was sent to the CA through the network.



```
v2:~# openssl genrsa -aes128 -out 10.0.0.2.pem 2048
Generating RSA private key, 2048 bit long modulus
..........+++
..............................+++
e is 65537 (0x10001)
Enter pass phrase for 10.0.0.2.pem:
Verifying - Enter pass phrase for 10.0.0.2.pem:
v2:~# ls
10.0.0.2.pem  ca_certificate.pem
v2:~# openssl req -new -key 10.0.0.2.pem -out 10.0.0.2.csr
Enter pass phrase for 10.0.0.2.pem:
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.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:v2
Email Address []:v2@vm.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:giuggiola
An optional company name []:
v2:~# ls
10.0.0.2.csr  10.0.0.2.pem  ca_certificate.pem
v2:~# cp 10.0.0.2.csr /hostlab/
```

Figure 11: Host private key and csr creation

8 Signature

Now the CA can sign v2's certificate. By pressing "y" the certificate was signed and the database updated with an entry representing this certificate. The CA saved automatically a copy of the certificate, stored in the folder *newcerts*, where each certificate is named with its timestamp, so in this case we have 01.pem.

```
v1:/hostlab# openssl ca -in 10.0.0.2.csr -out /hostlab/demoCA/certs/10.0.0.2.pem
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for ./demoCA/private/cakey.pem:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 1 (0x1)
        Validity
            Not Before: Dec 20 00:09:20 2019 GMT
            Not After : Dec 19 00:09:20 2020 GMT
        Subject:
            countryName               = AU
            stateOrProvinceName       = Some-State
            organizationName          = Internet Widgits Pty Ltd
            commonName                = v2
            emailAddress              = v2@vm.com
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                D4:36:7B:C7:5E:2C:3F:C5:63:95:17:D3:AF:1F:84:94:E8:5C:81:01
            X509v3 Authority Key Identifier:
                keyid:B5:FB:D9:59:93:0B:E5:1F:75:05:91:2E:06:C1:BE:DB:FD:B3:CA:7E

Certificate is to be certified until Dec 19 00:09:20 2020 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
v1:/hostlab#
```

Figure 12: Certificate signing

After sending a copy of the signed certicate to V2, it can finally verify the certificate by using the public key of v1 contained in the cacert.pem certificate.

```
v2:/hostlab# openssl verify -CAfile cacert.pem 10.0.0.2.pem 10.0.0.2.pem
10.0.0.2.pem: OK
10.0.0.2.pem: OK
v2:/hostlab#
```

Figure 13: Certificate verification

9 Revocation

To revocate v2's certificate the following command were executed.

```
v1:/hostlab# openssl ca -revoke ./demoCA/certs/10.0.0.2.pem
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for ./demoCA/private/cakey.pem:
Revoking Certificate 01.
Data Base Updated
```

Figure 14: Certificate revocation

The file *index.txt* was updated, indicating that the certificate in question has been revoked. Finally, a new CRL was created.

```
v1:/hostlab# openssl ca -gencrl -out newca.crl
Using configuration from /usr/lib/ssl/openssl.cnf
Enter pass phrase for ./demoCA/private/cakey.pem:
v1:/hostlab# cp newca.crl demoCA/crl/
```

Figure 15: Certificate creation