Workshop 3 : Cryptography with OpenSSL

Objective

The objective of this workshop is to familiarize you with basic security services. Including symmetric encryption, asymmetric encryption, hashing, digital signature and its verification, and certification.

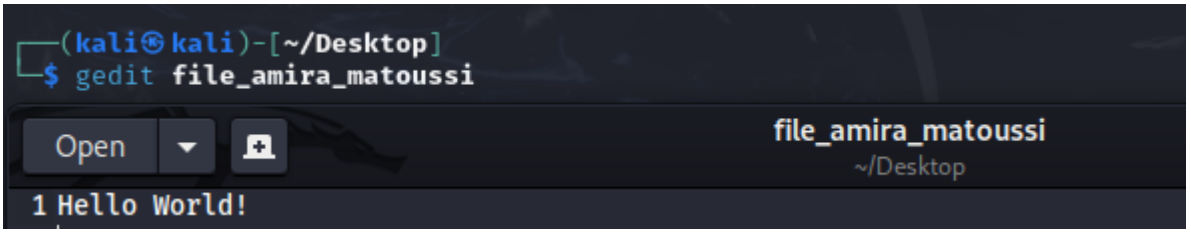
This practical work is based on the OpenSSL software suite.

OpenSSL is a cryptographic toolbox implementing the SSL and TLS protocols which offers a C programming library for creating secure client/server applications based on SSL/TLS.

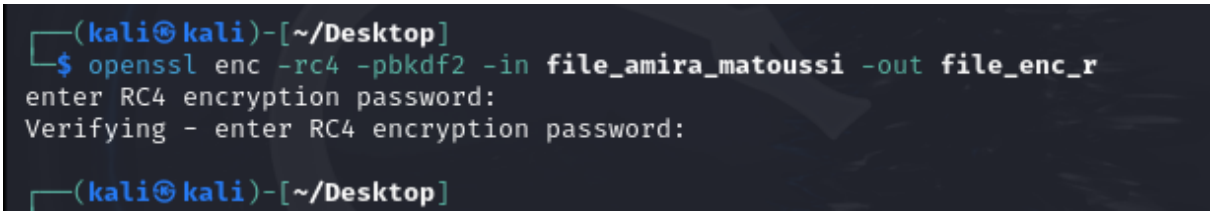
1. Symmetric encryption

The RC4 algorithm :

1. Create the file file_name_student containing clear text



2. Encrypt this file with the RC4 algorithm0



NB: openSSL gives us the possibility to give a password as input: given the password, openSSL derives an encryption key.

- Check that the message in file_name_student is unintellgible

```
┌──(kali㉿kali)-[~/Desktop]
└─$ cat file_enc_r
Salted__6◆橺◆◆(`q(+◆:}Xl◆◆
```

This encrypted file is transmitted to your friend who can also decrypt it.

- Write the command that decrypts it and thus produces the file
- Check that the decrypted message is indeed identical to the initial file

#diff file_name_student file_dec_rc

```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl enc -rc4 -d -pbkdf2 -in file_enc_r -out file_dec_r~
enter RC4 decryption password:

┌──(kali㉿kali)-[~/Desktop]
└─$ diff file_amira_matoussi file_dec_r~

┌──(kali㉿kali)-[~/Desktop]
└─$ cat file_dec_r~
Hello World!

┌──(kali㉿kali)-[~/Desktop]
└─$ cat file_amira_matoussi
Hello World!
```

The DES algorithm :

To encrypt the file file_name_student with the DES algorithm with an explicit key:

What command is used to decrypt file_enc_des?

- Check that the decrypted file is identical to the initial file.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl enc -des -pbkdf2 -in file_amira_matoussi -out file_enc_des -k 0123456789ABCDEF

┌──(kali㉿kali)-[~/Desktop]
└─$ cat file_enc_des
Salted__◆◆◆L◆?i◆◆◆◆W4$◆J1ŏ◆◆K

┌──(kali㉿kali)-[~/Desktop]
└─$ cat file_amira_matoussi
Hello World!

┌──(kali㉿kali)-[~/Desktop]
└─$ openssl enc -d -des -in file_enc_des -out file_amira_matoussi -k 0123456789ABCDEF

*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
bad decrypt
40976B04317F0000:error:1C800064:Provider routines:ossl_cipher_unpadblock:bad decrypt:../providers/implementations/c
iphers/ciphercommon_block.c:124:

┌──(kali㉿kali)-[~/Desktop]
└─$ openssl enc -d -des -pbkdf2 -in file_enc_des -out file_amira_matoussi_2 -k 0123456789ABCDEF


┌──(kali㉿kali)-[~/Desktop]
└─$ cat file_amira_matoussi_2
Hello World!
```

2. Asymmetric encryption

RSA asymmetric encryption requires the use of a pair of keys: a public key and a private key that must first be generated.

RSA private/public key generation :

In openSSL, the generated RSA keys are stored in a file with the extension .pem (Privacy Enhanced Mail). The statement to use is openssl genrsa.

Use this instruction to:

- Generate a 1024-bit key pair and store it in the file rsakey.pem

```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl genrsa -out rsakey.pem 1024
```

- View the file using the cat command. What do you notice

```
┌──(kali㉿kali)-[~/Desktop]
└─$ cat rsakey.pem
-----BEGIN PRIVATE KEY-----
MIICdwIBADANBgkqhkiG9w0BAQEFAASCAmEwggJdAgEAAoGBAOHIDaddkkYwhpjl
7QYYlR5oyX/r18CUfSxdhEM6kbLW9Z/dw0hQWF9+Q9RXZ64ktPo5YfAP9KkiJNFT
iBb5FETvplfg5gjasI5+HB1Dzt1hL4EbsQufsKULBZPROTbXTbIjxr7eO4qWHauT
0D6KA58E+PSA+sbXLHDiMYTiF/ozAgMBAAECgYEAuVipHMPL2KLtW4ck1ORJRfVv
tPu+2fEL8Oe0NBxYjx+388zxCuQqJ/pXZHF8Q5EOoHuK0q/GgTHuV4jjJzjCUWnn
rtb5vzlDWc1S3Suvf/79z8WgukhWwG+xv0WC4gE9jTL5pj3YRqLq2Yf5b2NwwGjF
9RcqAMbH6b8yvxG71oECQQD7neSAHduOfOi3QGTdnEOcfEqbXAZoCykDruE+tH0e
1ZLko6vTcd0i6kbHTFEiZEYYuIQIHR/yvYcbonxaF2a7AkEA5bbyJbMFMum+vteP
JD5Yh2J6li4dzbOHCZMAHkB5Sq+1CZUUOYDCESvflt1s+1wG1+TIzRL75FqVO7US
BtPO6QJACCuIT+CoUGgd5lVg8REx4NLoWUMTsi0ql8QF7FNP/MBSlLn7ylHYLFoP
EkmYGNXIsOTM5LaNYwRkbrFH+GTqIQJBAKaVgwi/yBdh1wfCfnWEGOMlEUzzaH2R
oW3aAo3tp4f338txJteEfjfBn0VikfO+it/pc4z0JWM2JV7FMqts0nECQHdEw3CB
k/WyZ74pJ0J6qkOsK6CCXkUkLxAfUN/v5XsJgpSI66Iao9fg/daAcRQMfqwjHSGa
Ln64PON2vYxqHZM=
-----END PRIVATE KEY-----
```

- One way to view keys in full format is to use the rsa command. Then print the keys in hexadecimal format, suppressing the output normally produced by the rsa instruction.

```
    51:22:64:46:18:b8:84:08:1d:1f:f2:bd:87:1b:a2:
    7c:5a:17:66:bb
prime2:
    00:e5:b6:f2:25:b3:05:32:e9:be:be:d7:8f:24:3e:
    58:87:62:7a:96:2e:1d:cd:b3:87:09:93:00:1e:40:
    79:4a:af:b5:09:95:14:39:80:c2:11:2b:df:96:dd:
    6c:fb:5c:06:d7:e4:c8:cd:12:fb:e4:5a:95:3b:b5:
    12:06:d3:ce:e9
exponent1:
    08:2b:88:4f:e0:a8:50:68:1d:e6:55:60:f1:11:31:
    e0:d2:e8:59:43:13:b2:2d:2a:97:c4:05:ec:53:4f:
    fc:c0:52:94:b9:fb:ca:51:d8:2c:5a:0f:12:49:98:
    18:d5:c8:b0:e4:cc:e4:b6:8d:63:04:64:6e:b1:47:
    f8:64:ea:21
exponent2:
    00:a6:95:83:08:bf:c8:17:61:d7:07:c2:7e:75:84:
    18:e3:25:11:4c:f3:68:7d:91:a1:6d:da:02:8d:ed:
    a7:87:f7:df:cb:71:26:d7:84:7e:37:c1:9f:45:62:
    91:f3:be:8a:df:e9:73:8c:f4:25:63:36:25:5e:c5:
    32:ab:6c:d2:71
coefficient:
    77:44:c3:70:81:93:f5:b2:67:be:29:27:42:7a:aa:
    43:ac:2b:a0:82:5e:45:24:2f:10:1f:50:df:ef:e5:
    7b:09:82:94:88:eb:a2:1a:a3:d7:e0:fd:d6:80:71:
    14:0c:7e:ac:23:1d:21:9a:2e:7e:b8:3c:e3:76:bd:
    8c:6a:1d:93
```

- Extract the public key from the private key and save the result to the file **rsapubkey.pem.**

```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl rsa -in rsakey.pem -pubout -out rsapubkey.pem
writing RSA key
```

Encryption of the RSA key by the algorithm

- We will now use the AES256 algorithm to encrypt the private key Write the command that encrypts the rsakey.pem file and thus produces an **rsakeyencaes.pem** file

```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl enc -pbkdf2 -AES256 -in rsakey.pem -out rsakeyencaes.pem
enter AES-256-CBC encryption password:
Verifying - enter AES-256-CBC encryption password:
```

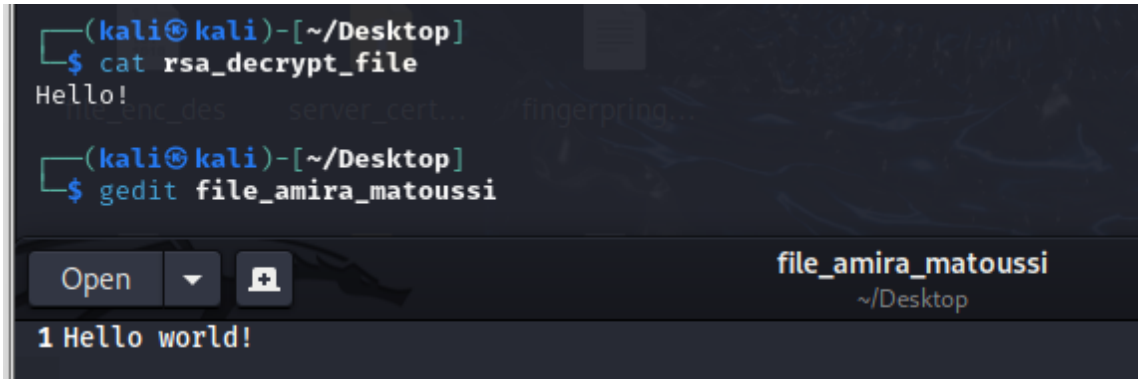
Noticed: the file we want to encrypt is the one that contains the private/public key pair! This will ensure that each time we want to use the information stored in the key file, we will have to enter the password

Data encryption/decryption with RSA

- Write the command that encrypts the initial file file_name_student with the public key rsapubkey.pem and thus produces a file file_name_student.rsaenc (use the instruction openssl rsautl).

- Write the command that decrypts the file file_name_student.rsaenc and thus produces a file file_name_student.rsadec

- Check the equality of the two files file_name_student and file_dechiff_rsa.



3. Digital signature

Generate a fingerprint of a file

To sign a document, a fingerprint of this document is first calculated. The instruction to use to calculate the fingerprint is:

#openssl dgst ←algo> -out <output> <input>

- Calculate the value of the fingerprint of the file file_name_student with the MD5 algorithm and put it in a file fingerprint.md5
- How big is this file ?

```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl dgst -md5 -out fingerpring.md5 file_amira_matoussi

┌──(kali㉿kali)-[~/Desktop]
└─$ ls -l
total 69328
-rw-r--r-- 1 kali kali     1322 Apr 23 17:24  ca.cr
-rw------- 1 kali kali     1704 Apr 23 17:18  cakey.pem
-rw-r--r-- 1 kali kali      765 Apr 23 17:01  cert_server.pem
drwxr-xr-x 3 kali kali     4096 Apr 13 17:49  environments
-rw-r--r-- 1 kali kali        8 Apr 23 06:37  file_amira_matoussi
-rw-r--r-- 1 kali kali       13 Apr 23 06:37  file_amira_matoussi_2
-rw-r--r-- 1 kali kali       13 Apr 23 05:21  file_dec_r~
-rw-r--r-- 1 kali kali       45 Apr 23 04:56  file_dec_rc~
-rw-r--r-- 1 kali kali       32 Apr 23 06:36  file_enc_des
-rw-r--r-- 1 kali kali       29 Apr 23 04:47  file_enc_r
-rw-r--r-- 1 kali kali       59 Apr 23 17:27  fingerpring.md5
-rwxrw-rw- 1 kali kali  2100987 Apr 13 16:28 'ML book.pdf'
-rwxrw-rw- 1 kali kali 68834474 Apr  5 05:18  Nessus-10.7.2-debian10_amd64.deb
-rw-r--r-- 1 kali kali     1103 Apr 23 17:05  server_cert.crt
-rw------- 1 kali kali      916 Apr 23 16:59  server_cle.pem
```

TYPO : fingerprint* .

- Calculate the value of the hash of the same file with the SHA1 algorithm and put it in a file fingerprint.sha1

```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl dgst -sha1 -out fingerpring.sha1 file_amira_matoussi

┌──(kali㉿kali)-[~/Desktop]
└─$ ls -l
total 69332
-rw-r--r-- 1 kali kali     1322 Apr 23 17:24  ca.cr
-rw------- 1 kali kali     1704 Apr 23 17:18  cakey.pem
-rw-r--r-- 1 kali kali      765 Apr 23 17:01  cert_server.pem
drwxr-xr-x 3 kali kali     4096 Apr 13 17:49  environments
-rw-r--r-- 1 kali kali        8 Apr 23 06:37  file_amira_matoussi
-rw-r--r-- 1 kali kali       13 Apr 23 06:37  file_amira_matoussi_2
-rw-r--r-- 1 kali kali       13 Apr 23 05:21  file_dec_r~
-rw-r--r-- 1 kali kali       45 Apr 23 04:56  file_dec_rc~
-rw-r--r-- 1 kali kali       32 Apr 23 06:36  file_enc_des
-rw-r--r-- 1 kali kali       29 Apr 23 04:47  file_enc_r
-rw-r--r-- 1 kali kali       59 Apr 23 17:27  fingerpring.md5
-rw-r--r-- 1 kali kali       68 Apr 23 17:28  fingerpring.sha1
-rwxrw-rw- 1 kali kali  2100987 Apr 13 16:28 'ML book.pdf'
-rwxrw-rw- 1 kali kali 68834474 Apr  5 05:18  Nessus-10.7.2-debian10_amd64.deb
-rw-r--r-- 1 kali kali     1103 Apr 23 17:05  server_cert.crt
-rw------- 1 kali kali      916 Apr 23 16:59  server_cle.pem
```

- Compare the result of the two hash functions. What do you notice

MD5 and SHA1 are both cryptographic hash functions, but MD5 produces a 52-bytes hash value while SHA1 produces a 61-bytes hash value.In this scenario, we notice that the SHA1 hash value is longer than the MD5 hash value. Additionally, SHA1 is considered more secure than MD5 due to its larger hash size and resistance to collision attacks.

Signing a file

Signing a document is like signing your fingerprint. The instruction to use in this case is:

`#openssl rsautl -sign -in fingerprint_file -inkey rsaprivkey.pem -out file_sig`

- Sign the file fingerprint.sha1 and put the result in the file

  sig_file. If so, what key should you use to sign?

```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl pkeyutl -sign -in fingerprint.sha1 -inkey rsakey.pem -out file_si
g
```

- It then remains to check that the fingerprint thus produced in the file fingerprint.sha1 is the same as can be calculated. Use the openssl rsautl -verify statement

```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl pkeyutl -verify -in file_sig -out fingerprint2 -pubin -inkey rsap
ubkey.pem
pkeyutl: No signature file specified for verify
```

What is the key you need to use to verify the signature of the file sig_file?

the public key associated to the private key

4. Digital certificate

Generation of the private key

- Generate a 1024-bit RSA key pair and store the result in the server_cle.pem file

```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl genpkey -algorithm RSA -out server_cle.pem -pkeyopt rsa_keygen_bi
ts:1024
.+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
.....................................................+++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++
```

Generating a certificate creation request

- Create a CSR Certificate Signing Request file:

```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl req -new -key server_cle.pem -out cert_server.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]:TN
State or Province Name (full name) [Some-State]:Manouba
Locality Name (eg, city) []:Beau séjour
Organization Name (eg, company) [Internet Widgits Pty Ltd]:esprit
Organizational Unit Name (eg, section) []:tn
Common Name (e.g. server FQDN or YOUR name) []:Amira
Email Address []:amira.matoussi@esprit.tn

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:amira
An optional company name []:amira
```

Signing the certificate

- Self sign the certificate
- Sign the certificate by a certificate authority (CA)

Self signing a certificate

- Sign the server.cert file using the private key contained in the server_cle.pem file and store the result in the server_cert.crt file. The certificate must have a validity period of one year.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl req -new -x509 -days 365 -key server_cle.pem -out server_cert.crt
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]:tn
State or Province Name (full name) [Some-State]:Manouba
Locality Name (eg, city) []:beau séjour
Organization Name (eg, company) [Internet Widgits Pty Ltd]:esprit
Organizational Unit Name (eg, section) []:amira
Common Name (e.g. server FQDN or YOUR name) []:amira
Email Address []:amira.matoussi@esprit.tn
```

- Display the contents of the certificate in text format:

```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl x509 -in server_cert.crt -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            6e:9a:7f:4b:9b:98:8f:39:8a:e4:ac:58:0c:96:f7:df:c4:10:fd:6b
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = tn, ST = Manouba, L = beau s\C3\83\C2\A9jour, O = esprit, OU = amira, CN = amira, emailAddress =
 amira.matoussi@esprit.tn
        Validity
            Not Before: Apr 23 21:05:18 2024 GMT
            Not After : Apr 23 21:05:18 2025 GMT
        Subject: C = tn, ST = Manouba, L = beau s\C3\83\C2\A9jour, O = esprit, OU = amira, CN = amira, emailAddress
= amira.matoussi@esprit.tn
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (1024 bit)
                Modulus:
                    00:d1:b3:ab:d0:cc:91:0e:9a:7a:ec:37:a2:bb:cb:
                    17:9e:c5:5b:e0:c9:e8:06:cf:b5:1a:ed:75:be:29:
                    b9:90:c1:96:02:eb:80:df:b6:dd:02:ff:f4:06:fd:
                    f1:17:d0:cd:91:99:24:00:4b:af:00:78:3a:98:7a:
                    4c:55:42:ca:84:0d:f8:56:ae:de:09:cd:7a:71:e0:
                    21:a5:4f:91:1f:d5:b7:4e:93:30:91:82:26:7a:d7:
                    a6:77:43:1d:fc:1d:b3:e2:7a:87:94:fa:de:19:a2:
                    71:56:a0:11:3c:91:73:c5:4b:20:c4:65:96:be:e2:
                    ea:1d:42:f8:88:db:5b:d1:9d
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                6F:F9:71:A8:9C:18:1E:17:6A:B4:D8:E3:12:F9:74:5F:29:7D:B8:A9
            X509v3 Authority Key Identifier:
                6F:F9:71:A8:9C:18:1E:17:6A:B4:D8:E3:12:F9:74:5F:29:7D:B8:A9
            X509v3 Basic Constraints: critical
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        9c:ee:a2:63:e8:93:f2:08:1a:3e:0d:66:69:8d:4a:95:48:21:
        dc:76:e7:d1:37:6d:29:0c:70:73:9f:b6:f2:f7:76:9d:6e:a9:
```

```
83:a6:6e:d8:87:97:1d:ba:8c:7a:df:21:d4:ea:50:b8:e5:3a:
59:76:43:73:13:07:98:f7:27:39:b0:72:b5:9f:c6:fa:a9:1c:
97:40:b9:91:de:2f:a9:7b:1a:52:6e:1e:12:31:cc:22:dd:aa:
77:9e:f0:4b:0e:4d:ae:a2:f1:5c:1b:2c:3f:eb:94:8b:93:8f:
65:a1:73:a9:46:3e:58:ff:43:35:6c:cd:5d:0a:38:d6:ef:4c:
01:c6
```

Signature by a certification authority (CA)

- The first step is to generate an RSA private key for the CA of size 2048 bits and store the result in the file cakey.pem

```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl genrsa -out cakey.pem 2048
```

- Generate a certificate for the CA with a validity period of 730 days and store the result in the ca.crt file.

```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl req -new -x509 -days 730 -key cakey.pem -out ca.cr
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]:tn
State or Province Name (full name) [Some-State]:manouba
Locality Name (eg, city) []:bs
Organization Name (eg, company) [Internet Widgits Pty Ltd]:esprit
Organizational Unit Name (eg, section) []:amira
Common Name (e.g. server FQDN or YOUR name) []:amira
Email Address []:amira.matoussi@esprit.tn
```

ca.crt is the self-signed certificate of the certification authority which will allow the certificates created to be signed.

- Sign the server certificate request (the server.csr file) by the CA certificate authority using the following statement

```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl x509 -req -in server.csr -out server.crt -CA ca.crt -CAkey cakey.
pem -CAcreateserial -CAserial ca.srl
Certificate request self-signature ok
subject=C = Tn, ST = tunis, L = El Manar, O = leila, OU = sk, CN = leila skou
ri, emailAddress = leila.skouri@gmail.com
Could not open file or uri for loading CA certificate from ca.crt: No such fi
le or directory
```

```
┌──(kali㉿kali)-[~/Desktop]
└─$ openssl x509 -req -in server.csr -out server.crt -CA ca.crt -CAkey cakey.
pem -CAcreateserial -CAserial ca.srl
Certificate request self-signature ok
```