Security Architectures – 2023/10/09 – TP OpenSSL

Davide Caria & Tee Wei Teoh

Partial Version

Part 1 – OpenSSL Keys

1.1 – Random character file

1.1.1 Execution of the command: “dd if=/dev/random of=out1.rand bs=1024 count=16” The option bs=1024 specifies the block-size and 1024 has been chosen accordingly to the request, while the option count=16 generates 16 distinct blocks. The output is a file of 16kB with random data inside. The same was done for “/dev/urandom”.

1.1.2 The main difference is that /dev/random will only return random number from the entropy pool and it has a “blocking behavior” in case that there are not enough inputs in the entropy pool. On the contraty /dev/urandom will provide ulimited random numbers and when the pool is not enough it will generate extra randomness from other algorithms.

1.2 – Key generation

1.2.1 The command executed was: “openssl genrsa -aes256 -rand out1.rand -out secretkey” since it provides the encryption of the private key before outputting it, preventing any side-channel attack. Otherwise, there is the option to output the private key in clear and then encrypt it. To execute the command, it is required to insert a passphrase.

1.2.2 After using the openssl rsa to read the key with the command “openssl rsa -in secretkey -text” the output is composed by:
- modulus
- publicExponent
- privateExponent
- prime1
- prime2
- exponent1
- exponent2
- coefficient

The moduls and the publicExponent are the public part of the key.

1.2.3 According to the documentation of the OpenSSL library the -rand is used as a seed of the pseudo-random number generator. Then other sources are considered, and further mathematical tools are used so that it is technically possible but very unlikely that two iterations with the same random file end up in two identical keys. Anyway the random file should be keep as private.

1.3 – Enciphering and signature

1.3.1 First the private key must be recovered. Since it is encrypted with the pass phrase provided, the command: “openssl rsa -in ../Download/clefrsa.pem -out clefrsa-dectypt.pem” plus the use of the provided pass phrase is enough to retrive the private key. Then the key can be used to decipher the file “clef-sym-crypt.bin” with the command: “openssl pkeyutl -decrypt -in ../Download/clef-sym-crypt.bin -out clef-sym-decrypt -inkey clefrsa-dectypt.pem”. At this point the password is “quitelongsecretpassword”.

1.3.2 To extract the public key the command “openssl rsa -in clefrsa-decrypt.pem -pubout -out clefpub.pem” can be used. This way, starting from the rsa .pem fromat we get the public key.

1.3.3 To sign the message with the private key the command “openssl pkeyult -sign -inkey clefrsa-decrypt.pem -in decrypte -out signature.bin”. The error that we receive is related to the fact that we are not providing a hash as input, but a full size message.

1.3.4 To use the previous command we shoud first hash the clear file with “cat decrypte | openssl dgst -sha256 > hashedDecrypte” and then use the has as the input of the previously reported command

1.3.5 openssl dgst -sha256 -verify clefpub.pem -signature signature.bin hashedDecrypte

Part 2 – Certification Authority

2.a – CA Setting

2.a.1 To set up the CA we created the proper folders, files and assigned the right privilege with the following set of commands:

```
mkdir demoCA; cd demoCA
mkdir private certs crl newcerts
echo “02” > serial
echo “01” > crlnumber
touch index.txt ; touch index.txt.attr
chmod 600 serial crlnumber index.txt index.txt.attr
chmod 700 private certs crl newcerts
```

2.a.2 Using the command used in one of the previous exercise: “dd if=/dev/random of=out1.rand bs=1024 count=16” a random file was generated. Then the private/public key pair is generated with the following: “openssl genrsa -aes256 -rand out1.rand -out cakey_tem.pem 4096”
Then the cakey was encrypted with the command “openssl rsa -aes256 -in cakey_tem.pem -out ./private/ cakey.pem”

2.a.3 With used the command “openssl req -x509 -key demoCA/private/cakey.pem -out demoCA/cacert.pem -days 365” to create the self-signed certificate.

2.b – User Certificate

2.b.1 openssl req -new -key private/cakey.pem -out mycsr.csr -newkey rsa:2048 -nodes -days 60

2.b.2 openssl req -text -noout -verify -in mycsr.csr