Public key cryptography in OpenSSL
HW6-CNS Sapienza

Giulia Muscarà 1743261

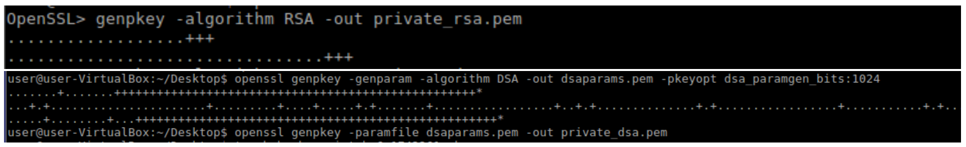
December 12, 2019

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. To carry out the analysis, the commands were executed on a scriptable Bash shell running on a Lubuntu virtual box, as it provides a fast and lightweight operating system with a clean and easy-to-use user interface.

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.*

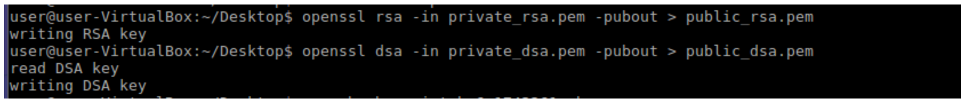


```
OpenSSL> genpkey -algorithm RSA -out private_rsa.pem
..................+++
................................+++
user@user-VirtualBox:~/Desktop$ openssl genpkey -genparam -algorithm DSA -out dsaparams.pem -pkeyopt dsa_paramgen_bits:1024
.......+.......++++++++++++++++++++++++++++++++++++++++++++++++++*
...+.+......................+.........+....+.....+.+.......+................+..+.+.............+.+................+..........+.+..
.....+........+...++++++++++++++++++++++++++++++++++++++++++++++++++*
user@user-VirtualBox:~/Desktop$ openssl genpkey -paramfile dsaparams.pem -out 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*.

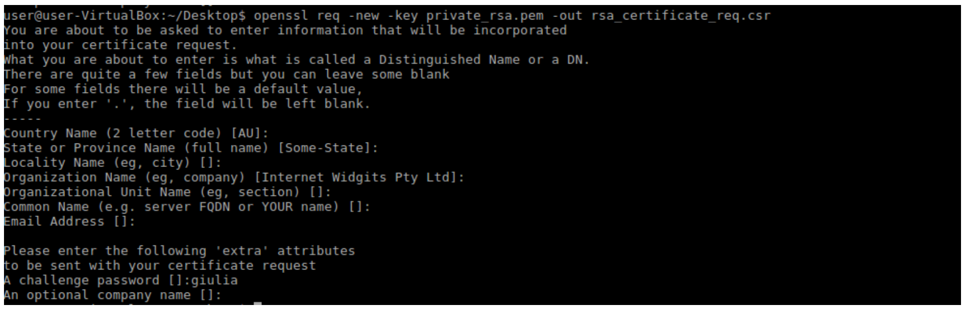


```
user@user-VirtualBox:~/Desktop$ openssl rsa -in private_rsa.pem -pubout > public_rsa.pem
writing RSA key
user@user-VirtualBox:~/Desktop$ openssl dsa -in private_dsa.pem -pubout > public_dsa.pem
read DSA key
writing DSA key
```

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.

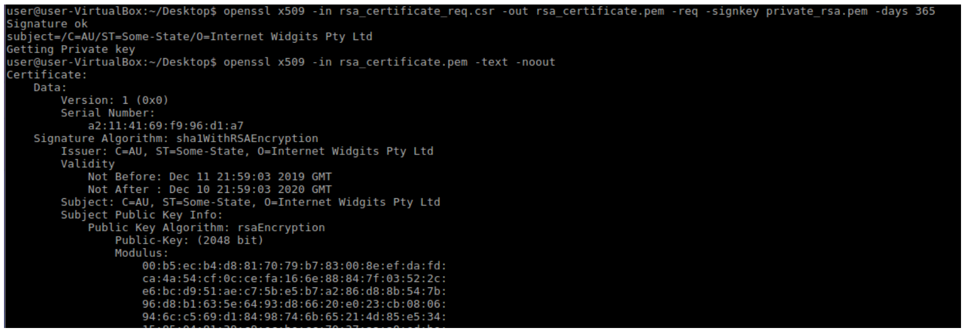


```
user@user-VirtualBox:~/Desktop$ openssl req -new -key private_rsa.pem -out rsa_certificate_req.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a 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 (e.g. server FQDN or YOUR name) []:
Email Address []:

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

Figure 3: RSA certificate request

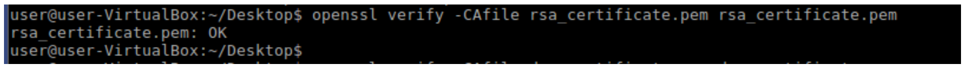
After creating a CSR, in order to self-sign the certificate, the private keys were used to prove the identity of the certificate owner. Through the command *x509*, the CSR was signed with the private key and the number of days of validity were specified. After the verification, the *rsa_certificate.pem*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



```
user@user-VirtualBox:~/Desktop$ openssl x509 -in rsa_certificate_req.csr -out rsa_certificate.pem -req -signkey private_rsa.pem -days 365
Signature ok
subject=/C=AU/ST=Some-State/O=Internet Widgits Pty Ltd
Getting Private key
user@user-VirtualBox:~/Desktop$ openssl x509 -in rsa_certificate.pem -text -noout
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            a2:11:41:69:f9:96:d1:a7
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=AU, ST=Some-State, O=Internet Widgits Pty Ltd
        Validity
            Not Before: Dec 11 21:59:03 2019 GMT
            Not After : Dec 10 21:59:03 2020 GMT
        Subject: C=AU, ST=Some-State, O=Internet Widgits Pty Ltd
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:b5:ec:b4:d8:81:70:79:b7:83:00:8e:ef:da:fd:
                    ca:4a:54:cf:0c:ce:fa:16:6e:88:84:7f:03:52:2c:
                    e6:bc:d9:51:ae:c7:5b:e5:b7:a2:86:d8:8b:54:7b:
                    96:d8:b1:63:5e:64:93:d8:66:20:e0:23:cb:08:06:
                    94:6c:c5:69:d1:84:98:74:6b:65:21:4d:85:e5:34:
```

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.



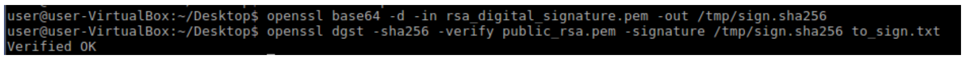
```
user@user-VirtualBox:~/Desktop$ openssl verify -CAfile rsa_certificate.pem rsa_certificate.pem
rsa_certificate.pem: OK
user@user-VirtualBox:~/Desktop$
```

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.



```
user@user-VirtualBox:~/Desktop$ openssl base64 -d -in rsa_digital_signature.pem -out /tmp/sign.sha256
user@user-VirtualBox:~/Desktop$ openssl dgst -sha256 -verify public_rsa.pem -signature /tmp/sign.sha256 to_sign.txt
Verified OK
```

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*.