HUGO MELEIRO

Rapport des LABs de Nouvelles Technologies et Sociétés

2021

SOMMAIRE

LAB1

L'objectif de cette séance est d'apprendre à utiliser **openssl** *et ses outils.*

1. Introduction d'OpenSSL
2. Cryptographie Symétrique

Dans cet exercice, nous avons dû chiffrer un document avec une clé publique et le déchiffrer ensuite grâce à la clé privée.

```
urxvt
[hugo.meleiro@r01p08 Bob]$ ls
BobDocument  BobPrivateKey  BobPublicKey
[hugo.meleiro@r01p08 Bob]$ cat BobPublicKey
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvcG09r5ljrT0FAVMoyqN
fsEQ9dL4UbEa8ODej2inyzTo8sXVTVOs+V5pZL5M+cUDN2xsYkcCfeJvx6GDumGM
GnknKiku4v5iW/w+f75/s/tZLPV6gVJobbHt0YhazUO0QD9OVtZa8PUve8D94J5l
l6vLd4DU6l889kdHbiW7p1Oqb4bL1VgRrLRZptxPAlFY1hmykd5efkRqLeZPVKB7
xTSJmTlqLKVcXOLxnN750gasLarJvOCA0ATqCBp2l7bSdx9Qf71QK+GseljIUj0s
OA1RgKf1gV27XlT/EgMdzFBZjFTO2Kt7mhQ3c4J2yyK8WfZ5FKwwaLZ9x30FkdJY
awIDAQAB
-----END PUBLIC KEY-----
[hugo.meleiro@r01p08 Bob]$
```

fig. 1 - La paire de clés de Bob a été créée

```
[hugo.meleiro@r01p08 Bob]$ cp /home/hugo.meleiro/afs/LAB1/Bob/BobPublicKey /home/hugo.meleiro/afs/LAB1/Alice/
[hugo.meleiro@r01p08 Bob]$ cd /home/hugo.meleiro/afs/LAB1/Alice/
[hugo.meleiro@r01p08 Alice]$ ls
AliceDocument  AlicePrivateKey  AlicePublicKey  BobPublicKey
[hugo.meleiro@r01p08 Alice]$
```

fig. 2 - Dossier d'Alice contenant la clé publique de Bob

```
[hugo.meleiro@r01p08 Alice]$ openssl rsautl -encrypt -in AliceDocument -pubin -inkey BobPublicKey -out AliceDocumentEncrypted
[hugo.meleiro@r01p08 Alice]$ ls
AliceDocument  AliceDocumentEncrypted  AlicePrivateKey  AlicePublicKey  BobPublicKey
[hugo.meleiro@r01p08 Alice]$
```

fig. 3 - Dossier d'Alice avec le fichier AliceDocumentEncrypted créé

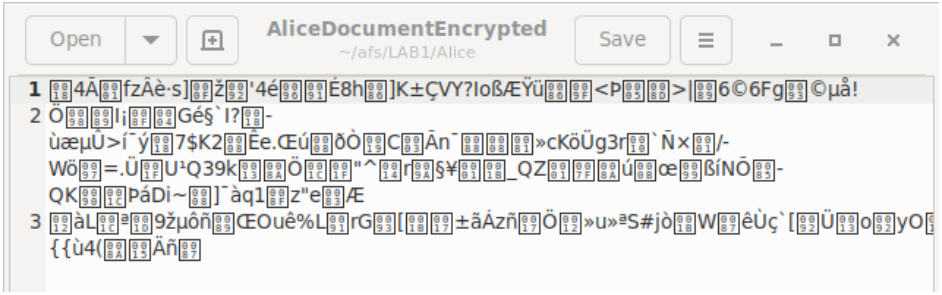

fig. 4 - Contenu du fichier AliceDocumentCrypted, on peut y voir que certains caractères sont étranges montrant que le document a bien été crypté

```
[hugo.meleiro@r01p08 Alice]$ cp /home/hugo.meleiro/afs/LAB1/Alice/AliceDocumentEncrypted /home/hugo.meleiro/afs/LAB1/Bob/
[hugo.meleiro@r01p08 Alice]$ cd /home/hugo.meleiro/afs/LAB1/Bob/
[hugo.meleiro@r01p08 Bob]$ ls
AliceDocumentEncrypted  BobDocument  BobPrivateKey  BobPublicKey
[hugo.meleiro@r01p08 Bob]$
```

fig. 5 - Dossier de Bob avec le fichier AliceDocumentEncrypted copié depuis le dossier d'Alice

```
[hugo.meleiro@r01p08 Bob]$ openssl rsautl -decrypt -in AliceDocumentEncrypted -inkey BobPrivateKey -out AliceDocumentDecrypted
[hugo.meleiro@r01p08 Bob]$ ls
AliceDocumentDecrypted  AliceDocumentEncrypted  BobDocument  BobPrivateKey  BobPublicKey
```

fig. 6 - Dossier de Bob avec le fichier AliceDocumentEncrypted et AliceDocumentDecrypted qui a été décrypté grâce à la clé privée de Bob

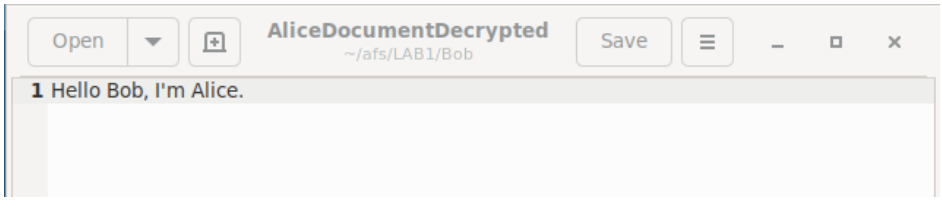

fig. 7 - Contenu du fichier AliceDocumentDecrypted, nous pouvons lire sans caractères étranges le message envoyé par Alice

```
[hugo.meleiro@r01p08 Alice]$ openssl rsautl -encrypt -in LargeFile -pubin -inkey BobPublicKey -out AliceLargeFileEncrypted
RSA operation error
140463180695360:error:04FFF06E:rsa routines:CRYPTO_internal:data too large for key size:/build/libressl-3.2.5/crypto/rsa/rsa_pk1.c:151:
[hugo.meleiro@r01p08 Alice]$
```

fig. 8 - Tentative d'encrypter LargeFile en utilisant la clé publique de Bob, résultant en une erreur

```
[hugo.meleiro@r01p08 Alice]$ cp /home/hugo.meleiro/afs/LAB1/Alice/AlicePublicKey /home/hugo.meleiro/afs/LAB1/Bob/
[hugo.meleiro@r01p08 Alice]$ ls
AliceDocument  AliceDocumentEncrypted  AliceLargeFileEncrypted  AlicePrivateKey  AlicePublicKey  AuthData  BobPublicKey  LargeFile
[hugo.meleiro@r01p08 Alice]$
```

fig. 9 - Copie de la clé publique d'Alice dans le dossier de Bob et création du fichier AuthData dans le dossier d'Alice

```
[hugo.meleiro@r01p08 Alice]$ openssl dgst -sha256 -out HashAuthData AuthData
[hugo.meleiro@r01p08 Alice]$ ls
AliceDocument  AliceDocumentEncrypted  AliceLargeFileEncrypted  AlicePrivateKey  AlicePublicKey  AuthData  BobPublicKey  HashAuthData  LargeFile
[hugo.meleiro@r01p08 Alice]$
```

fig. 10 - Hashage du fichier AuthData dans le dossier d'Alice

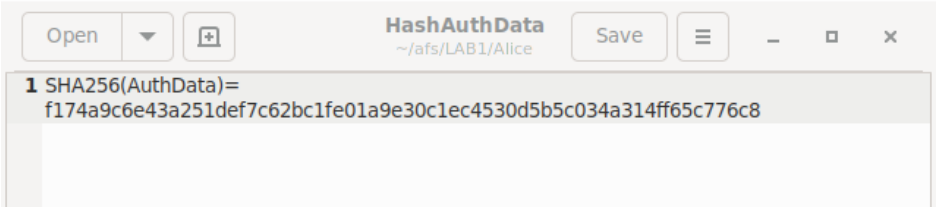

fig. 11 - Contenu du fichier HashAuthData

```
[hugo.meleiro@r01p08 Alice]$ openssl rsautl -sign -in HashAuthData -inkey AlicePrivateKey -out AliceSignature
[hugo.meleiro@r01p08 Alice]$ ls
AliceDocument           AliceLargeFileEncrypted  AlicePublicKey  AuthData      HashAuthData
AliceDocumentEncrypted  AlicePrivateKey          AliceSignature  BobPublicKey  LargeFile
[hugo.meleiro@r01p08 Alice]$
```

fig. 12 - Signature du fichier HashAuthData avec la clé privée d'Alice

Le sujet ainsi l'intégralité des fichiers sont disponibles sur :
https://git.hugofnm.fr/NTS/LAB1

LAB 2

*L'objectif de cette séance est d'approfondir les usages d'***openssl**, *notamment sur la Cryptographie Symétrique et les certificats X509.*

1. Cryptographie Symétrique

Dans cet exercice, nous avons dû reproduire le scénario comme montré sur la *fig. 13.*

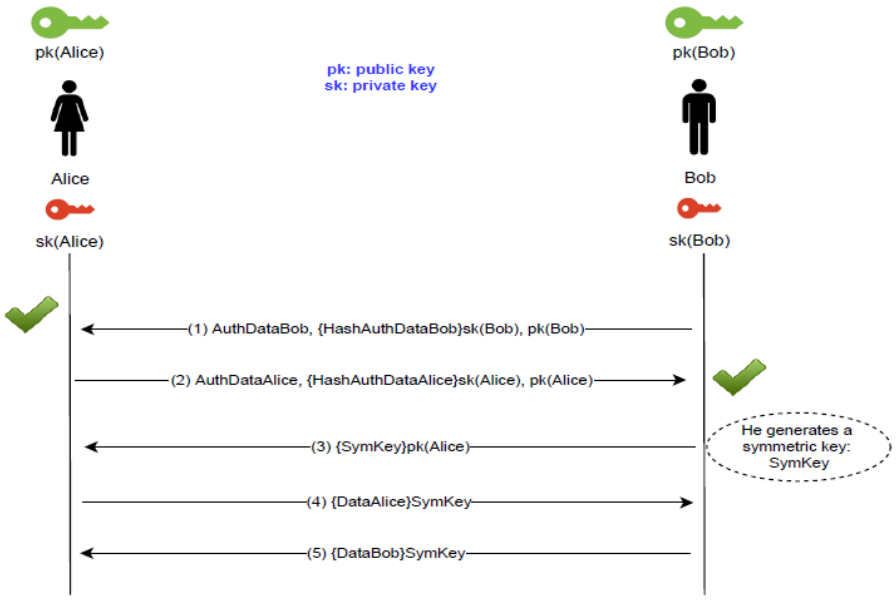

fig. 13 - Échanges entre Alice et Bob

```
[hugo.meleiro@localhost Bob]$ ls
AuthDataBob  AuthDataBob~  BobPrivateKey  BobPublicKey  BobSignature
[hugo.meleiro@localhost Bob]$
```

fig. 14 - Création du fichier AuthDataBob

```
[hugo.meleiro@localhost Alice]$ openssl dgst -sha256 -verify BobPublicKey -signature BobSignature AuthDat
aBob
Verified OK
[hugo.meleiro@localhost Alice]$
```

fig. 15 - Vérification de la signature BobSignature grâce à la clé publique de Bob.
La signature est valide, Bob est donc authentifié

```
[hugo.meleiro@localhost Alice]$ openssl dgst -sha256 -sign AlicePrivateKey -out
 AliceSignature AuthDataAlice
[hugo.meleiro@localhost Alice]$ ls
AlicePrivateKey  AliceSignature  AuthDataAlice~  BobPublicKey
AlicePublicKey   AuthDataAlice   AuthDataBob     BobSignature
```

fig. 16 - Génération de la signature d'Alice grâce à sa clé privée

```
[hugo.meleiro@localhost Bob]$ openssl dgst -sha256 -verify AlicePublicKey -signature AliceSignature AuthDataAlice
Verified OK
[hugo.meleiro@localhost Bob]$
```

fig. 17 - Vérification de la signature AliceSignature grâce à la clé publique de Alice.
La signature est valide, Alice est donc authentifiée

```
[hugo.meleiro@localhost Bob]$ openssl rsautl -encrypt -in SymKey -pubin -inkey AlicePublicKey -out SymKeyEncrypted
```

fig. 18 - Cryptage du fichier SymKey avec la clé publique d'Alice

LAB2

1. Cryptographie Symétrique (suite)

```
[hugo.meleiro@localhost Alice]$ openssl rsautl -decrypt -in SymKeyEncrypted -inkey AlicePrivateKey -out SymKey
[hugo.meleiro@localhost Alice]$ ls
AlicePrivateKey  AliceSignature  AuthDataAlice~  BobPublicKey  DataAlice   SymKey
AlicePublicKey   AuthDataAlice   AuthDataBob     BobSignature  DataAlice~  SymKeyEncrypted
```

fig. 19 - Décryptage du fichier SymKeyEncrypted à l'aide de la clé privée d'Alice

```
[hugo.meleiro@localhost ]$ openssl enc -e -aes-128-cbc -salt -pbkdf2 -kfile SymKey -in DataBob -out DataBobEncrypted
[hugo.meleiro@localhost ]$ ls
AlicePublicKey  AliceSignature  AuthDataAlice  AuthDataBob  BobPrivateKey  BobPublicKey  BobSignature  DataBob  DataBobEncrypted  SymKey  SymKeyEncrypted
```

fig. 20 - Cryptage du fichier DataBob avec la clé symétrique SymKey
Bob va envoyer le message crypté grâce à la clé SymKey

```
[hugo.meleiro@localhost Alice]$ openssl enc -d -aes-128-cbc -salt -pbkdf2 -kfile SymKey -in DataBobEncrypted -out DataBob
[hugo.meleiro@localhost Alice]$ cat DataBob
Just some important data[hugo.meleiro@localhost Alice]$
```

fig. 21 - Décryptage du fichier DataBob avec la clé symétrique SymKey
Alice a reçu le message et a réussi à le décrypter grâce à la clé SymKey

2. Certificats X509

```
[hugo.meleiro@localhost LAB2]$ openssl x509 -noout -in CertificateLCL -dates
notBefore=Dec 28 00:00:00 2020 GMT
notAfter=Dec 28 23:59:59 2021 GMT
```

*fig. 22 - Affichage de la date de validité du certificat du site **lcl.fr***

```
[hugo.meleiro@localhost LAB2]$ openssl x509 -noout -in CertificateLCL -fingerp
rint
SHA256 Fingerprint=DB:C0:36:51:5A:39:42:93:62:59:6E:7F:7A:C6:CE:B2:2A:E7:34:6A
:13:7E:72:A8:12:EF:47:4E:6E:64:8B:8F
```

*fig. 23 - Affichage de la signature du certificat du site **lcl.fr***

```
[hugo.meleiro@localhost LAB2]$ openssl x509 -noout -in CertificateLCL -serial
serial=AB6AD8276765FE51DC43493A4C7B6223
```

*fig. 24 - Affichage du numéro de série du certificat du site **lcl.fr***

```
[hugo.meleiro@localhost LAB2]$ openssl x509 -noout -in CertificateLCL -pubkey
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAwWquvcQxcMCjfwYsgPbg
Zd4RVMhwtow3ONvViwyo7jO4CNebZT8esC5pP1GFtqp2opZMr6Wczi+yhNJ2AfCB
UKC23bAGiAzoDG9Z6HtOqbCiqb1Je9GmW03ZuygF6RxZHl5oi2nsQ5lSZ8Oas+NG
i/3aE6khRTQfR8mV8cL3iAfgVlfsOD6rx0SZNWz4WwQgAermd1cTRXwcr0VHjhbK
3yHmlLbLy7PUbya/kTiAsWh+eHTGvE11bp9orHuxT+W6crRcyLhdmG+L+5NgAJsx
qJgFcjrjqAzrm5A/IBXIdUhCQRG55wTI/Y73ZMIrn1ujzHBT6YNP6RsB1BifDSYh
DQIDAQAB
-----END PUBLIC KEY-----
```

*fig. 25 - Affichage de la clé publique du certificat du site **lcl.fr***

```
[hugo.meleiro@localhost LAB2]$ openssl rsa -in ServerKeyPair -pubout -out ServerPublicKey
writing RSA key
```

fig. 26 - Extraction de la clé publique à partir du fichier ServerKeyPair

```
[hugo.meleiro@localhost LAB2]$ openssl genrsa -out CAKeyPair -passout pass:password 4096
Generating RSA private key, 4096 bit long modulus
.....
.....++++
.....++++
e is 65537 (0x10001)
```

fig. 27 - Génération d'une paire de clés (4096 bits) pour un CA protégée par un mot de passe

2. Certificats X509 (suite)

```
[hugo.meleiro@localhost LAB2]$ openssl verify -CAfile CACertificate.crt ServerCertificate.crt
ServerCertificate.crt: OK
```

fig. 28 - Vérification du certificat serveur grâce au certificat CA

Le sujet ainsi l'intégralité des fichiers sont disponibles sur :
https://git.hugofnm.fr/NTS/LAB2

LAB3

L'objectif de cette séance finale est d'utiliser les connaissances du LAB1 et LAB2 sur **openssl**.

1. Sécurité cryptographique

Dans cet exercice, nous avons dû reproduire les éléments de sécurité cryptographique comme montré sur la *fig. 29*.

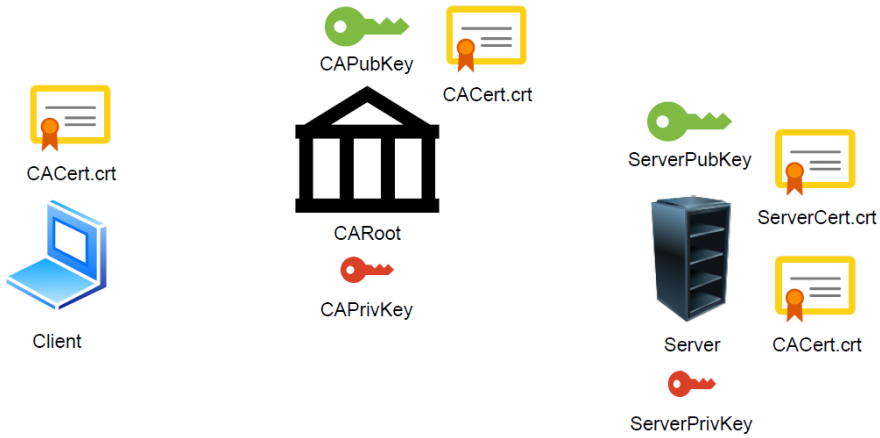

fig. 29 - Éléments de sécurité cryptographique pour le client, le serveur et l'Autorité de Certification

```
[hugo.meleiro@localhost CARoot]$ openssl genrsa -out CAKeyPair -passout pass:password 4096
Generating RSA private key, 4096 bit long modulus
.....................................................................++++
..........++++
e is 65537 (0x10001)
[hugo.meleiro@localhost CARoot]$ openssl rsa -in CAKeyPair -pubout -out CAPublicKey
writing RSA key
[hugo.meleiro@localhost CARoot]$ mv CAKeyPair CAPrivateKey
[hugo.meleiro@localhost CARoot]$ ls
CAPrivateKey  CAPublicKey
[hugo.meleiro@localhost CARoot]$
```

fig. 30 - Génération de la clé publique, de la clé privée du CARoot (Autorité de Certification)

```
[hugo.meleiro@localhost CARoot]$ openssl req -x509 -new -key CAPrivateKey -out CACertificate.crt -days 500
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) []:FR
State or Province Name (full name) []:Haute-Garonne
Locality Name (eg, city) []:Toulouse
Organization Name (eg, company) []:EPITA
Organizational Unit Name (eg, section) []:First Year
Common Name (eg, fully qualified host name) []:MyCA
Email Address []:hugo.meleiro@epita.fr
[hugo.meleiro@localhost CARoot]$ ls
CACertificate.crt  CAPrivateKey  CAPublicKey  ServerRequest.crt
```

fig. 31 - Génération du certificat du CARoot (Autorité de Certification) à l'aide de la clé privée

```
[hugo.meleiro@localhost Server]$ openssl genrsa -out ServerKeyPair -passout pass:password 2048
Generating RSA private key, 2048 bit long modulus
.....................+++++
.....................+++++
e is 65537 (0x10001)
[hugo.meleiro@localhost Server]$ openssl rsa -in ServerKeyPair -pubout -out ServerPublicKey
writing RSA key
[hugo.meleiro@localhost Server]$ mv ServerKeyPair ServerPrivateKey
[hugo.meleiro@localhost Server]$ ls
ServerPrivateKey  ServerPublicKey
[hugo.meleiro@localhost Server]$
```

fig. 32 - Génération de la clé publique et clé privée du serveur

```
[hugo.meleiro@localhost Server]$ openssl req -new -key ServerPrivateKey -out ServerRequest.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) []:FR
State or Province Name (full name) []:Haute-Garonne
Locality Name (eg, city) []:Toulouse
Organization Name (eg, company) []:EPITA
Organizational Unit Name (eg, section) []:First Year
Common Name (eg, fully qualified host name) []:myserver.fr
Email Address []:hugo.meleiro@epita.fr

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
[hugo.meleiro@localhost Server]$ ls
ServerPrivateKey  ServerPublicKey  ServerRequest.crt
[hugo.meleiro@localhost Server]$ cp ServerRequest.crt ../CARoot/
```

fig. 33 - Génération de la demande de certificat du serveur à l'aide de la clé privée

```
[hugo.meleiro@localhost CARoot]$ openssl x509 -req -in ServerRequest.crt -CA CACertificate.crt -CAkey CAPrivateKey -CAcreateserial -out ServerCertificate.crt
-days 500 -sha256
Signature ok
subject=/C=FR/ST=Haute-Garonne/L=Toulouse/O=EPITA/OU=First Year/CN=myserver.fr/emailAddress=hugo.meleiro@epita.fr
Getting CA Private Key
[hugo.meleiro@localhost CARoot]$ ls
CACertificate.crt  CACertificate.srl  CAPrivateKey  CAPublicKey  ServerCertificate.crt  ServerRequest.crt
[hugo.meleiro@localhost CARoot]$
```

fig. 34 - Génération du certificat du serveur à l'aide de la demande effectuée par le serveur ainsi que la clé privée et le certificat du CARoot

```
[hugo.meleiro@localhost Client]$ ls
CACertificate.crt
```

```
[hugo.meleiro@localhost Server]$ ls
CACertificate.crt  ServerCertificate.crt  ServerPrivateKey  ServerPublicKey  ServerRequest.crt
```

fig. 35 & 36 - Le client et le serveur ont désormais confiance (Trusted Third Party) en l'Autorité de Certification

2. Protocole TLS

Dans cet exercice, nous avons dû reproduire le scénario comme montré sur la *fig. 37*.

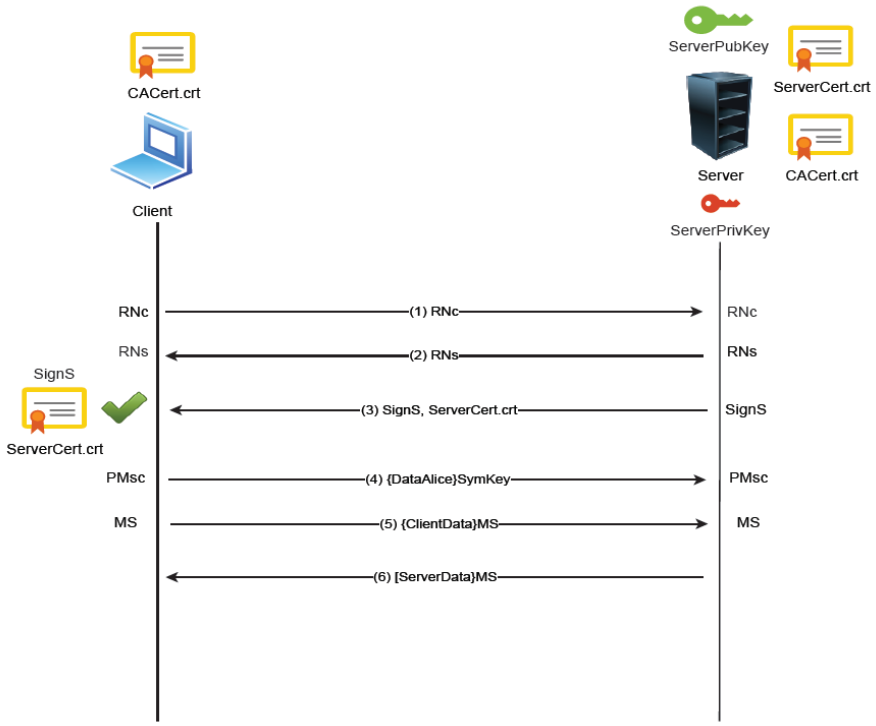

fig. 37 - Schéma de communication bidirectionnelle sécurisée utilisant le protocole TLS entre le client et le serveur

```
[hugo.meleiro@localhost Client]$ gedit RNc
[hugo.meleiro@localhost Client]$ ls
CACertificate.crt  RNc
[hugo.meleiro@localhost Client]$ cp RNc ../Server/
[hugo.meleiro@localhost Client]$
```

fig. 38 - Génération par le client du fichier RNc qui est ensuite transféré au serveur

```
[hugo.meleiro@localhost Server]$ gedit RNs
[hugo.meleiro@localhost Server]$ ls
CACertificate.crt  RNc  RNs  ServerCertificate.crt  ServerPrivateKey  ServerPublicKey  ServerRequest.crt
[hugo.meleiro@localhost Server]$ cp RNs ../Client/
[hugo.meleiro@localhost Server]$
```

fig. 39 - Génération par le client du fichier RNc qui est ensuite transféré au serveur

```
[hugo.meleiro@localhost Server]$ gedit RNs
[hugo.meleiro@localhost Server]$ ls
CACertificate.crt  RNc  RNs  ServerCertificate.crt  ServerPrivateKey  ServerPublicKey  ServerRequest.crt
[hugo.meleiro@localhost Server]$ cp RNs ../Client/
[hugo.meleiro@localhost Server]$
```

fig. 40 - Génération par le serveur du fichier RNs qui est ensuite transféré au client

```
[hugo.meleiro@localhost Server]$ cat RNc RNs > RNcRNs
[hugo.meleiro@localhost Server]$ openssl dgst -sha256 -out HashRNcRNs RNcRNs
[hugo.meleiro@localhost Server]$ ls
CACertificate.crt  HashRNcRNs  RNc  RNcRNs  RNs  ServerCertificate.crt  ServerPrivateKey  ServerPublicKey  ServerRequest.crt
[hugo.meleiro@localhost Server]$ openssl rsautl -sign -in HashRNcRNs -inkey ServerPrivateKey -out SignS
[hugo.meleiro@localhost Server]$ ls
CACertificate.crt  HashRNcRNs  RNc  RNcRNs  RNs  ServerCertificate.crt  ServerPrivateKey  ServerPublicKey  ServerRequest.crt  SignS
[hugo.meleiro@localhost Server]$
```

fig. 41 - Concaténation de RNc et RNs, application du hash SHA-256 à ce fichier et génération de la signature SignS grâce a la clé privée du serveur

```
[hugo.meleiro@localhost Server]$ cp SignS ../Client/
[hugo.meleiro@localhost Server]$ cp ServerCertificate.crt ../Client/
[hugo.meleiro@localhost Server]$ cd ../Client/
[hugo.meleiro@localhost Client]$ ls
CACertificate.crt  RNc  RNs  ServerCertificate.crt  SignS
[hugo.meleiro@localhost Client]$
```

fig. 42 - La signature SignS et le certificat ServerCert du serveur sont envoyés au client

```
[hugo.meleiro@localhost Client]$ openssl verify -CAfile CACertificate.crt ServerCertificate.crt
ServerCertificate.crt: OK
[hugo.meleiro@localhost Client]$
```

fig. 43 - Le client vérifie le certificat ServerCertificate du serveur, la validité de ce certificat garantie l'authentification et la non répudiation du serveur

```
[hugo.meleiro@localhost Client]$ openssl x509 -noout -in ServerCertificate.crt -pubkey > ServerPublicKey
[hugo.meleiro@localhost Client]$ ls
CACertificate.crt  RNc  RNs  ServerCertificate.crt  ServerPublicKey  SignS
[hugo.meleiro@localhost Client]$
```

fig. 44 - Extraction de la clé publique à partir du certificat ServerCertificate

```
[hugo.meleiro@localhost Client]$ openssl rsautl -verify -in SignS -pubin -inkey ServerPublicKey -out HashSign
[hugo.meleiro@localhost Client]$ cat RNc RNs > RNcRNs
[hugo.meleiro@localhost Client]$ openssl dgst -sha256 -out HashRNcRNs RNcRNs
[hugo.meleiro@localhost Client]$ diff HashRNcRNs HashSign
[hugo.meleiro@localhost Client]$
```

fig. 45 - Vérification de la signature SignS, l'absence d'erreur confirme que la signature est valide

```
[hugo.meleiro@localhost Client]$ openssl rand -hex -out PMsc 64
[hugo.meleiro@localhost Client]$ ls
CACertificate.crt  HashRNcRNs  HashSign  PMsc  RNc  RNcRNs  RNs  ServerCertificate.crt  ServerPublicKey  SignS
```

fig. 46 - Génération de la clé symétrique PMsc

```
[hugo.meleiro@localhost Client]$ openssl rsautl -encrypt -in PMsc -pubin -inkey ServerPublicKey -out PMscEncrypted
[hugo.meleiro@localhost Client]$ ls
CACertificate.crt  HashRNcRNs  HashSign  PMsc  PMscEncrypted  RNc  RNcRNs  RNs  ServerCertificate.crt  ServerPublicKey  SignS
[hugo.meleiro@localhost Client]$ cp PMscEncrypted ../Server/
```

fig. 47 - Cryptage de PMsc grâce à la clé publique du serveur et envoi de la clé cryptée au serveur

```
hugo.meleiro@localhost Client]$ cd ../Server/
hugo.meleiro@localhost Server]$ openssl rsautl -decrypt -in PMscEncrypted -inkey ServerPrivateKey -out PMsc
hugo.meleiro@localhost Server]$ ls
CACertificate.crt  HashRNcRNs  PMsc  PMscEncrypted  RNc  RNcRNs  RNs  ServerCertificate.crt  ServerPrivateKey  ServerPublicKey  ServerRequest.crt  SignS
hugo.meleiro@localhost Server]$
```

fig. 48 - Décryptage de la clé PMscEncrypted grâce à la clé privée du serveur

```
[hugo.meleiro@localhost Server]$ cat RNc RNs PMsc > RNcRNsPMsc

[hugo.meleiro@localhost Server]$ openssl dgst -sha256 -out MS RNcRNsPMsc
[hugo.meleiro@localhost Server]$
```

fig. 49 - Calcul de la clé MS (Master key) à l'aide du hash de la concaténation de PMsc, RNc et RNs (côté serveur)

```
[hugo.meleiro@localhost Client]$ cat RNc RNs PMsc > RNcRNsPMsc
[hugo.meleiro@localhost Client]$ openssl dgst -sha256 -out MS RNcRNsPMsc
[hugo.meleiro@localhost Client]$ ls
CACertificate.crt  HashRNcRNs  HashSign  MS  PMsc  PMscEncrypted  RNc  RNcRNs  RNcRNsPMsc  RNs  ServerCertificate.crt  ServerPublicKey  SignS
[hugo.meleiro@localhost Client]$
```

fig. 50 - Calcul de la clé MS (Master key) à l'aide du hash de la concaténation de PMsc, RNc et RNs (côté client)

```
[hugo.meleiro@localhost Client]$ openssl enc -e -aes-128-cbc -salt -pbkdf2 -kfile MS -in ClientData -out
ClientDataEncrypted
[hugo.meleiro@localhost Client]$ ls
CACertificate.crt  ClientDataEncrypted  HashSign  PMsc          RNc     RNcRNsPMsc  ServerCertificate.cr
t  SignS
ClientData         HashRNcRNs           MS        PMscEncrypted  RNcRNs  RNs         ServerPublicKey
```

fig. 51 - Création du fichier ClientData et cryptage de celui-ci grâce à la clé symétrique MS

```
[hugo.meleiro@localhost Server]$ openssl enc -d -aes-128-cbc -salt -pbkdf2 -kfile MS -in ClientDataEncrypted -out ClientData
[hugo.meleiro@localhost Server]$ cat ClientData
Very Important Data
[hugo.meleiro@localhost Server]$
```

fig. 52 - Décryptage de ClientDataEncrypted grâce à la clé symétrique MS, le contenu est lisible, le serveur a bien reçu le message

```
[hugo.meleiro@localhost Server]$ gedit ServerData
[hugo.meleiro@localhost Server]$ cat ServerData
Very Important Data from the Server 😎
[hugo.meleiro@localhost Server]$ openssl enc -e -aes-128-cbc -salt -pbkdf2 -kfile MS -in ServerData -out ServerDataEncrypted
[hugo.meleiro@localhost Server]$ ls
CACertificate.crt  ClientDataEncrypted  MS    PMscEncrypted  RNcRNs      RNs                 ServerData           ServerPrivateKey  ServerRequest.crt
ClientData         HashRNcRNs           PMsc  RNc            RNcRNsPMsc  ServerCertificate.crt  ServerDataEncrypted  ServerPublicKey   SignS
[hugo.meleiro@localhost Server]$
```

fig. 53 - Création du fichier ServerData et cryptage de celui-ci grâce à la clé symétrique MS

```
[hugo.meleiro@localhost Client]$ openssl enc -d -aes-128-cbc -salt -pbkdf2 -kfile MS -in ServerDataEncrypted -out ServerData
[hugo.meleiro@localhost Client]$ cat ServerData
Very Important Data from the Server 😎
[hugo.meleiro@localhost Client]$
```

fig. 54 - Décryptage de ServerDataEncrypted grâce à la clé symétrique MS, le contenu est lisible, le client a bien reçu le message

Le sujet ainsi l'intégralité des fichiers sont disponibles sur :
https://git.hugofnm.fr/NTS/LAB3

CONCLUSION

Les outils d'***openssl*** peuvent être très utiles afin de simuler des transferts entre un serveur et un client. Ces simulation nous permettent d'observer et de comprendre tous les mécanismes mis en places afin de protéger le client et le serveur.

COORDONNÉES

Hugo Meleiro
Étudiant en 1e Année - SUP

hugofnm.fr
hugo.meleiro@epita.fr

Document créé à des fins éducatifs. Réalisé sur NixOS