ÉCOLE D'INGÉNIEURS ECE PARIS · LYON

ING4 SI01

NETWORK SECURITY

ENCRYPTION

Lab 3

Author(s):
Gabriel PADIS
Anastasia DUCHESNE

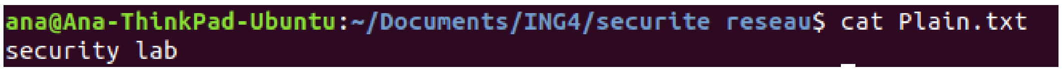
Teacher(s):
MR. FAYAD

Paris - Semester 7 - March 5, 2019

Contents

1 Hash function

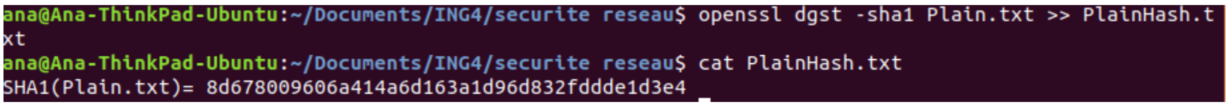
We created a file Plain.txt containing "security lab".



```
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ cat Plain.txt
security lab
```

Figure 1: Plain.txt

We generated the hash values for this file :



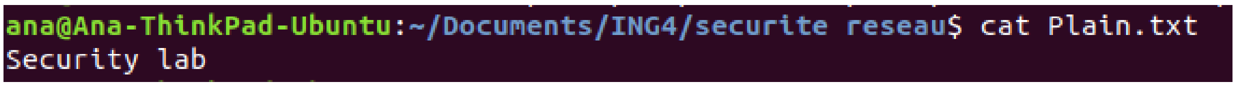
```
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ openssl dgst -sha1 Plain.txt >> PlainHash.t
xt
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ cat PlainHash.txt
SHA1(Plain.txt)= 8d678009606a414a6d163a1d96d832fddde1d3e4
```

Figure 2: Values H1

The **openssl dgst** command performs digest operation: A message digest is a cryptographic hash function containing a string of digits created by a one-way hashing formula.

- **-cipher**: name of the cipher used, here -sha1 hash algorithm

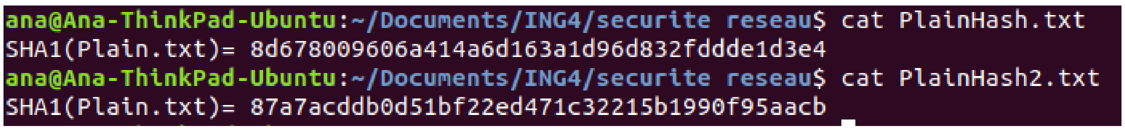
We changed the first s into a capital S of Plain.txt :



```
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ cat Plain.txt
Security lab
```

Figure 3: Plain.txt

Then, we generated values H2 for the modified file :



```
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ cat PlainHash.txt
SHA1(Plain.txt)= 8d678009606a414a6d163a1d96d832fddde1d3e4
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ cat PlainHash2.txt
SHA1(Plain.txt)= 87a7acddb0d51bf22ed471c32215b1990f95aacb
```

Figure 4: Description

They are completely different.

2 Symmetric encryption

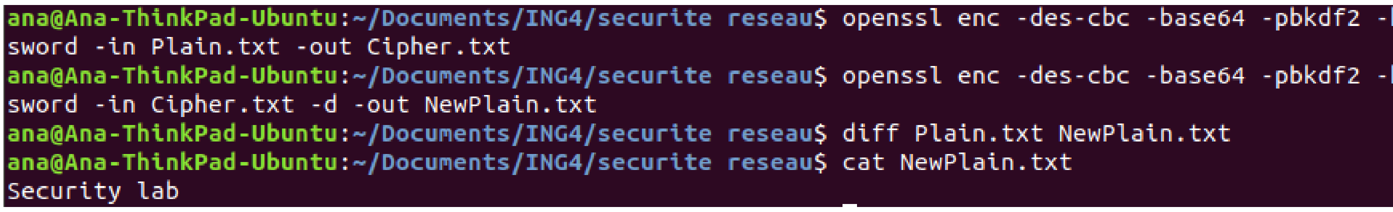
openssl:

- **enc**: allow data to be encrypted or decrypted using key
- **-cipher**: name of the cipher used, here des-cbc encryption method
- **-base64**: encode in a base64 format once it has been ecrypted. It is a binary-to-text encoding schemes that represent binary data in an ASCII string format.

- **-k**: supersesed by -pass pass:<password, provide a password to derive the key from

s

The **diff** command compares files line by line. The **-q** option only report when files differ. Here they don't differ so nothing is printed.

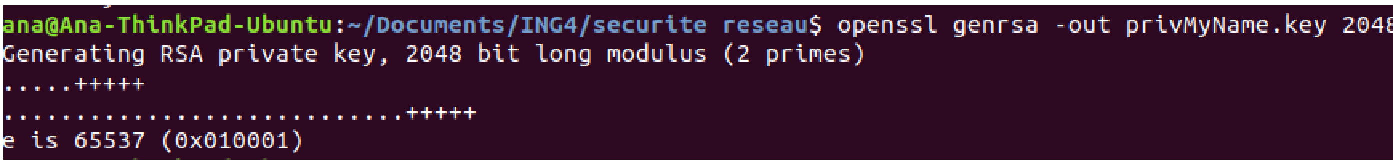


```
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ openssl enc -des-cbc -base64 -pbkdf2 -
sword -in Plain.txt -out Cipher.txt
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ openssl enc -des-cbc -base64 -pbkdf2 -
sword -in Cipher.txt -d -out NewPlain.txt
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ diff Plain.txt NewPlain.txt
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ cat NewPlain.txt
Security lab
```

Figure 5: Symmetrically Encoding and Decoding a file

3 Asymmetric encryption

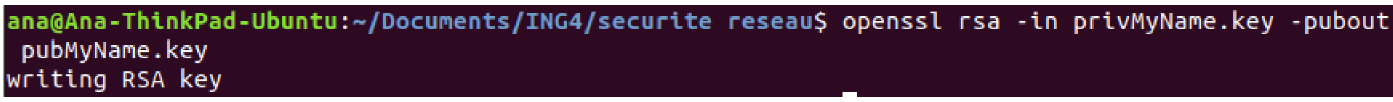
The **openssl genrsa** command generates an RSA private key. It takes as a mandatory parameter the size of the private key to generate in bits, here 2048 (it is also the default value).



```
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ openssl genrsa -out privMyName.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....+++++
...........................+++++
e is 65537 (0x010001)
```

Figure 6: Generate a private key

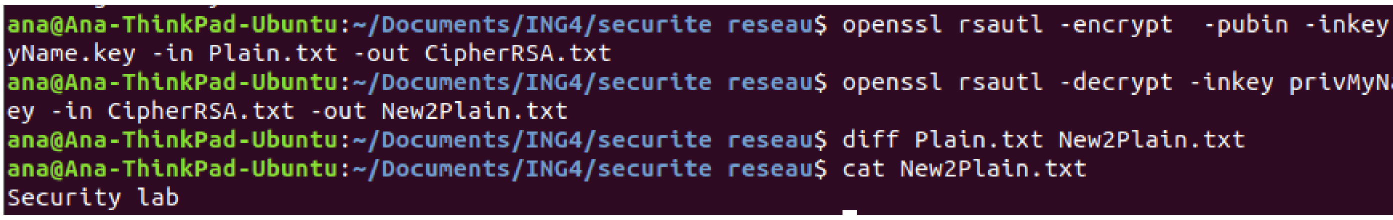
The **openssl rsa** command processes RSA keys. With it you can extract a public key from a private key.



```
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ openssl rsa -in privMyName.key -pubout
 pubMyName.key
writing RSA key
```

Figure 7: Generate a public key

The **openssl rsautl** command can be used to sign, verify, encrypt and decrypt data using the RSA algorithm.



```
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ openssl rsautl -encrypt  -pubin -inkey
yName.key -in Plain.txt -out CipherRSA.txt
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ openssl rsautl -decrypt -inkey privMyN
ey -in CipherRSA.txt -out New2Plain.txt
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ diff Plain.txt New2Plain.txt
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ cat New2Plain.txt
Security lab
```

Figure 8: Asymmetrically Encrypt and Decrypt a file

4 Digital Signature

4.1 Scenario

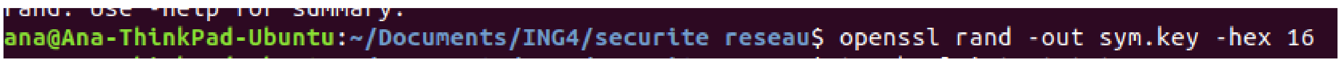
Alice has a plaintext M file. She symmetrically encrypts it with a secret key k that was just generated. And she sends the encrypted text as Ciphertext C to Bob.
Bob needs the secret key k to decrypt it back to plaintext M to read it.
To pass the secret key k securely Alice encrypts it with Bob's public key. She also hash it and sign it with it's own private key. She send both files : the encrypted secret key k and the hash that she signed to Bob. Bob verify the signature with Alice public key on the hash to prove that it is her that send it and decrypts the secret key k with it's own private key.
With the secret key k Bob decrypts the Ciphertext C and is able to read the plaintext M file.

4.2 Operation

The **openssl rand** command outputs a number of pseudo-random bytes precised by the number passed as parameter.

- -hex: the outputs is an hex string

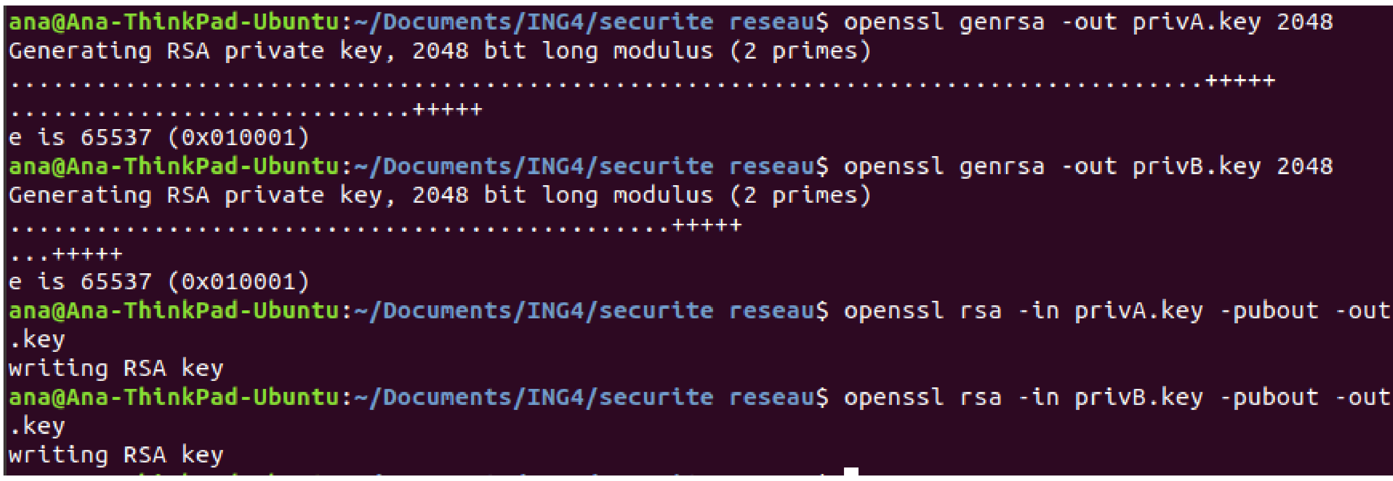
Here we create a hexadecimal symmetric key on 16 bytes so of 128 bits.



```
rand: use -help for summary.
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ openssl rand -out sym.key -hex 16
```

Figure 9: Symmetric key sym.key

We create a text file called PlaintTextM.txt and in it is : "My Security LAB: My name is Anastasia".
As we did before we generate asymmetric keys, 2 private keys A and B, from them 2 public key respectively.



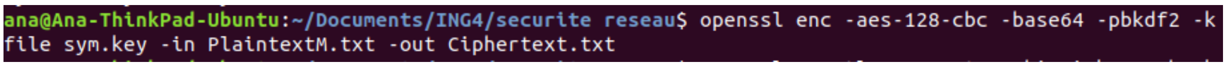
```
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ openssl genrsa -out privA.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
...................................................................................+++++
............................+++++
e is 65537 (0x010001)
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ openssl genrsa -out privB.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
..............................................+++++
...+++++
e is 65537 (0x010001)
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ openssl rsa -in privA.key -pubout -out
.key
writing RSA key
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ openssl rsa -in privB.key -pubout -out
.key
writing RSA key
```

Figure 10: Generate keys

We then exchanged the keys. Alice has her private key (privA) and Bob's public key (pubB), and Bob's has the opposites (pubA and privB).

We then encrypt symmetrically the file that Alice wants to send. To do so be use the previous command **openssl aec**:

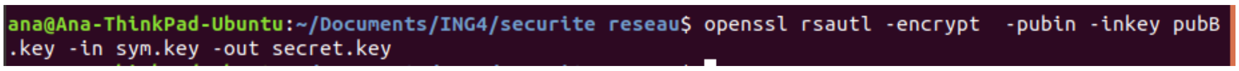
- **-cipher**: name of the cipher used, here aes-128-cbc encryption method
- **-base64**: encode in a base64 format once it has been ecrypted. It is a binary-to-text encoding schemes that represent binary data in an ASCII string format.
- **-kfile**: provide a password file to derive the key from



```
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ openssl enc -aes-128-cbc -base64 -pbkdf2 -k
file sym.key -in PlaintextM.txt -out Ciphertext.txt
```

Figure 11: File encryption

We now encrypt the symmetric key using a asymmetric algorithm with Bob's public key. With it we know that only him can decrypt it.

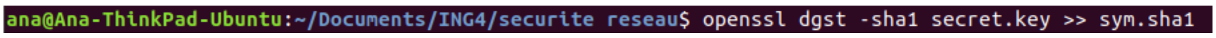


```
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ openssl rsautl -encrypt  -pubin -inkey pubB
.key -in sym.key -out secret.key
```

Figure 12: Key encryption

To reassure Bob that the key and file come from Alice:

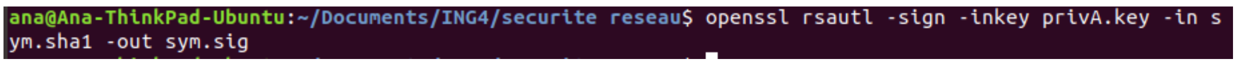
- We hash the secret symmetric key:



```
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ openssl dgst -sha1 secret.key >> sym.sha1
```

Figure 13: Hashing the secret key

- Sign the hashed secret symmetric key:

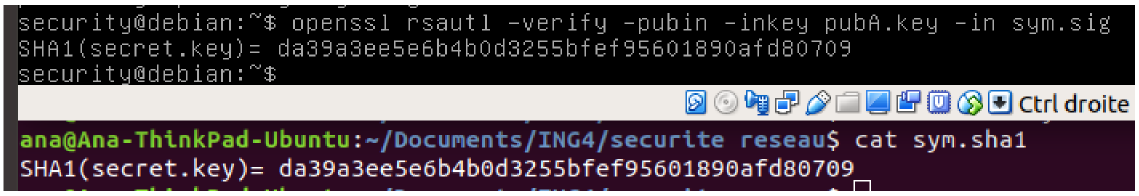


```
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ openssl rsautl -sign -inkey privA.key -in s
ym.sha1 -out sym.sig
```

Figure 14: Singing the hash

We then send the encrypted text file, the encrypted symmetric key and the signed hash of that key from Alice to Bob.

Once Bob has received all the files, he first has to verify their authenticity. To do so, he verifies the signed hash with Alice's public key.

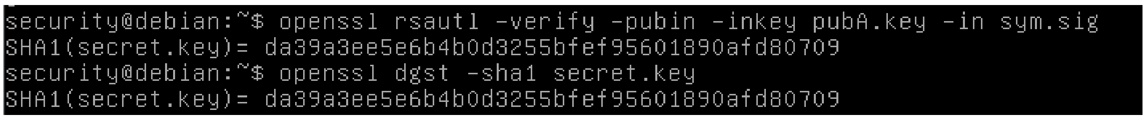


```
security@debian:~$ openssl rsautl -verify -pubin -inkey pubA.key -in sym.sig
SHA1(secret.key)= da39a3ee5e6b4b0d3255bfef95601890afd80709
security@debian:~$
```

```
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ cat sym.sha1
SHA1(secret.key)= da39a3ee5e6b4b0d3255bfef95601890afd80709
```

Figure 15: Verifying the received files

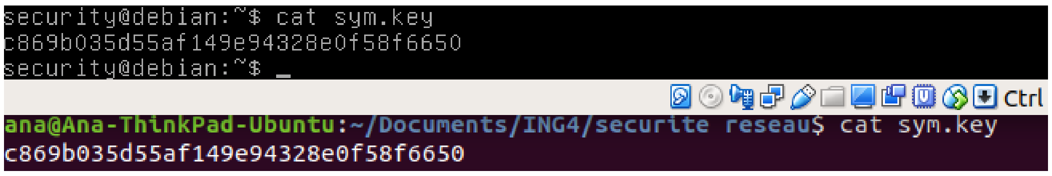
As you can see, the two hash are the same :



```
security@debian:~$ openssl rsautl -verify -pubin -inkey pubA.key -in sym.sig
SHA1(secret.key)= da39a3ee5e6b4b0d3255bfef95601890afd80709
security@debian:~$ openssl dgst -sha1 secret.key
SHA1(secret.key)= da39a3ee5e6b4b0d3255bfef95601890afd80709
```

Figure 16: Authenticating the key

Once the authentication is done, he has to decrypt the secret key with his public key (pubB).

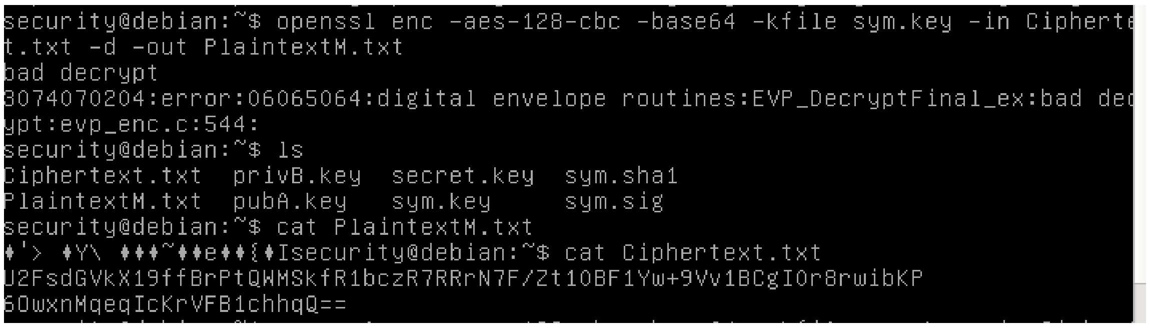


```
security@debian:~$ cat sym.key
c869b035d55af149e94328e0f58f6650
security@debian:~$ _
```

```
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ cat sym.key
c869b035d55af149e94328e0f58f6650
```

Figure 17: Decrypting the secret key

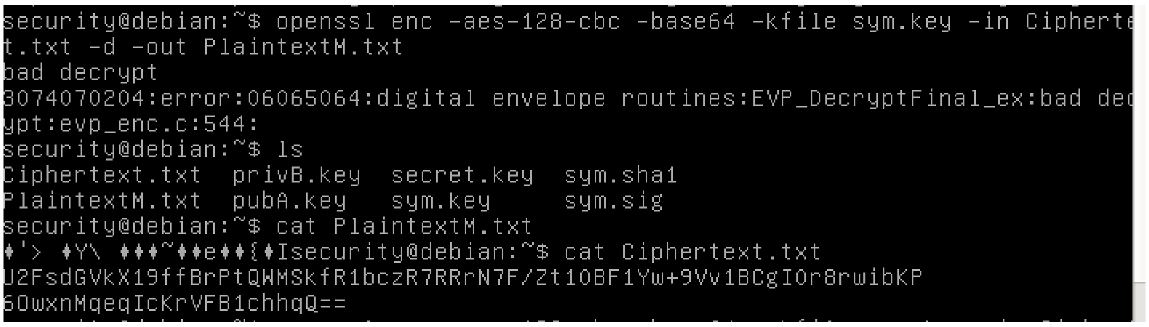
Finally, he can decrypt the encrypted file with the secret key.



```
security@debian:~$ openssl enc -aes-128-cbc -base64 -kfile sym.key -in Ciphert
t.txt -d -out PlaintextM.txt
bad decrypt
3074070204:error:06065064:digital envelope routines:EVP_DecryptFinal_ex:bad dec
ypt:evp_enc.c:544:
security@debian:~$ ls
Ciphertext.txt  privB.key  secret.key  sym.sha1
PlaintextM.txt  pubA.key   sym.key     sym.sig
security@debian:~$ cat PlaintextM.txt
♦'> ♦Y\ ♦♦♦~♦♦e♦♦{♦Isecurity@debian:~$ cat Ciphertext.txt
U2FsdGVkX19ffBrPtQWMSkfR1bczR7RRrN7F/Zt10BF1Yw+9Vv1BCgI0r8rwibKP
6OwxnMqeqIcKrVFB1chhqQ==
```

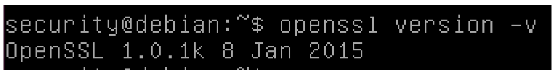
Figure 18: Decrypting the file

An error occurred since the versions of openssl are not the same on the two machines :



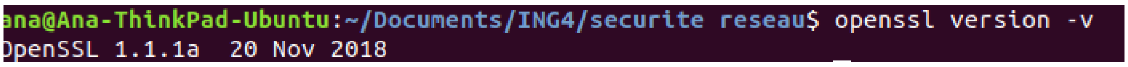
```
security@debian:~$ openssl enc -aes-128-cbc -base64 -kfile sym.key -in Cipherte
t.txt -d -out PlaintextM.txt
bad decrypt
3074070204:error:06065064:digital envelope routines:EVP_DecryptFinal_ex:bad dec
ypt:evp_enc.c:544:
security@debian:~$ ls
Ciphertext.txt  privB.key  secret.key  sym.sha1
PlaintextM.txt  pubA.key   sym.key     sym.sig
security@debian:~$ cat PlaintextM.txt
♦'> ♦Y\ ♦♦♦~♦♦e♦♦{♦Isecurity@debian:~$ cat Ciphertext.txt
U2FsdGVkX19ffBrPtQWMSkfR1bczR7RRrN7F/Zt10BF1Yw+9Vv1BCgIOr8rwibKP
60wxnMqeqIcKrVFB1chhqQ==
```

Figure 19: On the virtual machine



```
security@debian:~$ openssl version -v
OpenSSL 1.0.1k 8 Jan 2015
```

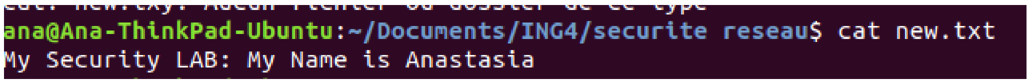
Figure 20: Version on VM



```
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ openssl version -v
OpenSSL 1.1.1a  20 Nov 2018
```

Figure 21: Version on Host

But the decryption works fine !



```
ana@Ana-ThinkPad-Ubuntu:~/Documents/ING4/securite reseau$ cat new.txt
My Security LAB: My Name is Anastasia
```

Figure 22: The transmitted file

4.3 Used command lines

Here are all the commands in the order used for this scenario:

```
openssl rand -out sym.key -hex 16

openssl genrsa -out privA.key 2048
openssl genrsa -out privB.key 2048
openssl rsa -in privA.key -pubout -out pubA.key
openssl rsa -in privB.key -pubout -out pubB.key

openssl enc -aes-128-cbc -base64 -kfile sym.key -in PlaintextM.txt -out Ciphertext.tx

openssl rsault -encrypt -pubin -inkey pubB.key -in sym.key -out secret.key
openssl dgst -sha1 secret.key >> sym.sha1
openssl rsault -sign -inkey privA -in sym.sha1 -out sym.sig
openssl rsault -verify -pubin -inkey pubA.key -in sym.sig -out sym.sha1
openssl dgst -sha1 secret.key
openssl rsault -decrypt -inkey privB.key -in secret.key -out sym.key
openssl enc -aes-128-cbc -d -base64 -kfile sym.key -in Ciphertext.txt -out Plaintext
```

4.4 Security services

The different security services assured by this scenario are:

- confidentiality and secure message format
- authentication and nonrepudiation : the receiver knows the sender is the only one who could encrypt the file thansk to the signature
- message integrity