

DATA PROTECTION

Lab 3 / Hashed ElGamal + Hybrid Encryption + HMAC

Team: Oriol Lalaguna Royo, Sofia Tsagiopoulou

The python version used to develop the following scripts is >3.0.0.

1. Key generation

We developed a python script for the key generation. In order to run this script, you need to pass a name into the program.

For example, **'python3 genkey.py alice'**

```
(kali㉿kali)-[~]  
$ python3 genkey.py marc
```

```
(kali㉿kali)-[~/dp]  
$ ls  
marc_pkey.pem  marc_pubkey.pem  param.pem
```

This python script generates the public and private long-term key for alice (the generated files are like these ones: alice pkey.pem and alice pubkey.pem). The public and private key of Alice is generated using DiffieHellman algorithm with the third cyclic group defined by RFC5114 and stored in PEM format.

```
(kali㉿kali)-[~]  
$ python3 genkey.py marc
```

```
(kali㉿kali)-[~/dp]  
$ ls  
marc_pkey.pem  marc_pubkey.pem  param.pem
```

2. Encryption

For the encryption we developed the python script encrypt.py. To execute this python script, you should pass 2 variables, the name that you gave in the generation key script and a message to encrypt and.

```
(kali㉿kali)-[~/dp]  
$ python3 encrypt.py marc 'merry christmas'
```

This script encrypts a message using alice's public key. A file 'ciphertext.pem' is generated. This file contains the ephemeral public key in PEM format, the Initialization Vector (16 bytes generated randomly, in base64), the encryption of the message using AES-128-CBC (in base64), and the HMAC code (in based64). Hashed ElGamal is used as public key scheme, to encrypt the

message and to generate the HMAC. The common secret key is hashed using SHA-256. The first 16 bytes are used to encrypt the message and the last 16 for the HMAC code generation.

```
#read the message to encrypt
messagetoencrypt = sys.argv[2]

#generate the ephkeys
os.system("openssl genpkey -paramfile param.pem -out tempephkey.pem")
os.system("openssl pkey -in tempephkey.pem -pubout -out ephpubkey.pem")

os.system("openssl pkeyutl -inkey tempephkey.pem -peerkey " + sys.argv[1] + "_pubkey.pem -derive -out common.bin")

os.system("cat common.bin | openssl dgst -sha256 -binary | head -c 16 > k1.bin")
os.system("cat common.bin | openssl dgst -sha256 -binary | tail -c 16 > k2.bin")

#random iv of 16 bytes
os.system("openssl rand 16 > iv.bin")

#encryption
os.system("echo -n \"" + sys.argv[2] + "\"" | openssl enc -aes-128-cbc -K `cat k1.bin | xxd -p` -iv `cat iv.bin | xxd -p` > ciphertext.bin")

#tag
os.system("cat iv.bin ciphertext.bin | openssl dgst -sha256 -mac hmac -macopt hexkey:`cat k2.bin | xxd -p` -binary > tag.bin")
```

3. Decryption

In this script we get the four parts ephpubkey.pem, iv.bin, ciphertext.bin, and tag.bin by parsing the ciphertext file:

```
#We open the file and read the 4 variables
f = open(sys.argv[1], "r")
variable = f.read()
f.close()

ephpubkey_pos = variable.find("-----END PUBLIC KEY-----")
ephpubkey = variable[:ephpubkey_pos+24]
os.system("echo -n \"" + ephpubkey + "\"" > ephpubkey.pem")

iv_pos = variable.find("-----END AES-128-CBC IV-----")
iv = variable[ephpubkey_pos+56:iv_pos]
os.system("echo -n \"" + iv + "\"" | openssl base64 -d -out iv.bin")

ciphertext_position = variable.find("-----END AES-128-CBC CIPHERTEXT-----")
ciphertext = variable[iv_pos+68:ciphertext_position]
os.system("echo -n \"" + ciphertext + "\"" | openssl base64 -d -out ciphertext.bin")

tag_position = variable.find("-----END SHA256-HMAC TAG-----")
tag = variable[ciphertext_position+69:tag_position]
os.system("echo -n \"" + tag + "\"" | openssl base64 -d -out tag.bin")
```

Then, with openssl pkeyutl -derive, we recover the common secret by using the files [name] pkey.pem and ephpubkey.pem:

```
os.system("openssl pkeyutl -inkey " + sys.argv[2] + "_pkey.pem -peerkey ephpubkey.pem -derive -out common.bin")
```

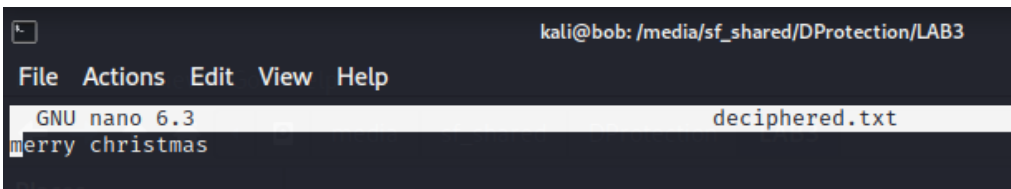
After this we divide the value into the two keys k1 and k2 by applying SHA256 to the preceding result:

```
#Split the key values
os.system("cat common.bin | openssl dgst -sha256 -binary | head -c 16 > k1.bin")
os.system("cat common.bin | openssl dgst -sha256 -binary | tail -c 16 > k2.bin")
```

Utilizing the key k2, we recalculate the SHA256-HMAC from the union of the files iv.bin and ciphertext.bin. The code aborts the decryption process and notify the error if the outcome differs from the file tag.bin. Finally, we use openssl enc -d -aes-128-cbc with the key k1 and the iv given in iv.bin to decrypt the file ciphertext.bin and recover the plaintext:

```
#Abort the decryption process and notify the error if the outcome differs from the file tag.bin.
if os.popen("cat tag.bin | openssl base64").read() == os.popen("cat outcome_tag.bin | openssl base64").read():
    os.system("openssl enc -aes-128-cbc -d -in ciphertext.bin -iv `cat iv.bin | xxd -p` -K `cat k1.bin | xxd -p` -out deciphered.txt")
else:
    print("Decryption process aborted, tag.bin differs from file")
```

As the result of the decrypted text:



```
kali@bob: /media/sf_shared/DProtection/LAB3
File Actions Edit View Help
GNU nano 6.3 deciphered.txt
Merry christmas
```