Data Protection

Lab work 3
Hashed ElGamal + Hybrid Encryption + HMAC

Students

Alejandro Capella del Solar

Cristian Fernández Jiménez

Professor

Jorge Luis Villar Santos

December 20, 2022

Contents

List of Figures

1 Introduction

As a practical exercise, we developed an encryption scheme based on Hashed EL-Gamal encryption, combined with a symmetric encription scheme and a MAC. Using a normal DH group with size of 256bits as common secret, and hashing it with SHA256 to obtain a 256 bits symmetric key. Finally, splitting this hash into two 128 bit parts, we produce two new keys: one, *k1*, for AES-128-CBC encryption and the other, *k2*, for SHA256-HMAC, used as message authentication method.

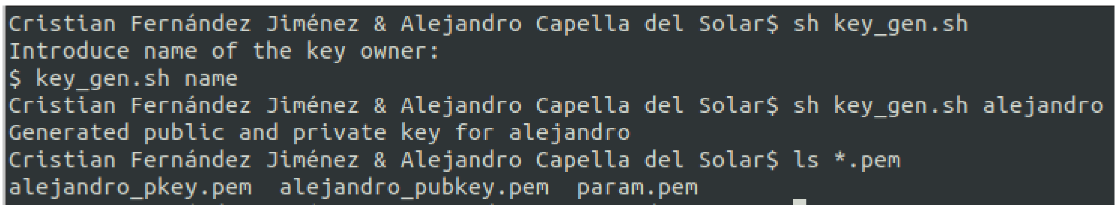
In addition, before encrypting a file, the recipient must have generated a parameter file for the DH group and a long-term Diffie Hellman keypair, and sharing the corresponding public key afterwards.

Therefore, we have developed three scripts that will let us generate the required parameters, encrypt files using hybrid encryption and recover these messages using a decryption script.

For the laboratory, we have made use of `OpenSSL 1.1.1f`.

2 Key generation

The long-term key and the DH parameter file are both generated by the script `key_gen.sh`. A simple bash script that requires the name of the owner to create all required credentials:



```
Cristian Fernández Jiménez & Alejandro Capella del Solar$ sh key_gen.sh
Introduce name of the key owner:
$ key_gen.sh name
Cristian Fernández Jiménez & Alejandro Capella del Solar$ sh key_gen.sh alejandro
Generated public and private key for alejandro
Cristian Fernández Jiménez & Alejandro Capella del Solar$ ls *.pem
alejandro_pkey.pem  alejandro_pubkey.pem  param.pem
```

Figure 1: Key gen script execution.

The script follows the next steps:

Generate the DH group. We chose the third group in RFC5114 and generated the corresponding parameters in `param.pem`. Diffie Hellman cryptosystems require a previous generation of some public parameters, which we will use to create a key pair out of, belonging to the same DH group.

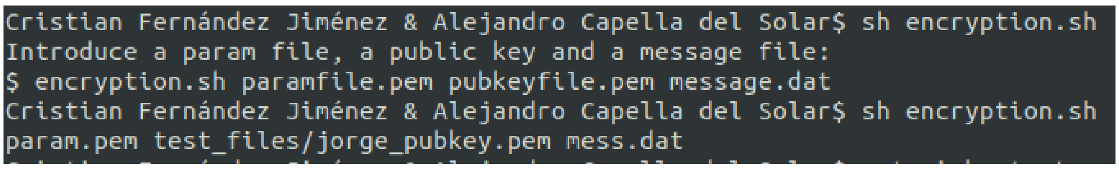
Generate the long-term keypair. Once the param file has been created, a long-term keypair can be generated, using the name given as parameter by the user. Later on, we may extract the public key, since will be transmitted for future secure communication.

Two files are generated, `<owner>_pkey.pem` and `<owner>_pubkey.pem`, containing the owner's private and public long-term keys respectively.

Moreover, the recipient would need to obtain a certificate for this action, in order to prevent impersonation attacks. We assume this as done for practical reasons.

3 Encryption

The encryption part has been specified on `encryption.sh`. This bash scripts requires a DH parameter file, a public key and a file to encrypt. The execution way is as follows:



```
Cristian Fernández Jiménez & Alejandro Capella del Solar$ sh encryption.sh
Introduce a param file, a public key and a message file:
$ encryption.sh paramfile.pem pubkeyfile.pem message.dat
Cristian Fernández Jiménez & Alejandro Capella del Solar$ sh encryption.sh
param.pem test_files/jorge_pubkey.pem mess.dat
```

Figure 2: Message encryption using test public key.

This scripts creates a `ciphertext.pem` file, specially composed to contain all auxiliary credentials to decrypt the message within afterwards. The encryption itself is achieved after the following steps:

Compute an ephemeral Diffie-Hellman keypair. Derived from the given parameter file and used after to derive a common secret, the pair is saved in `eph_pkey.pem` and `eph_pubkey.pem`.

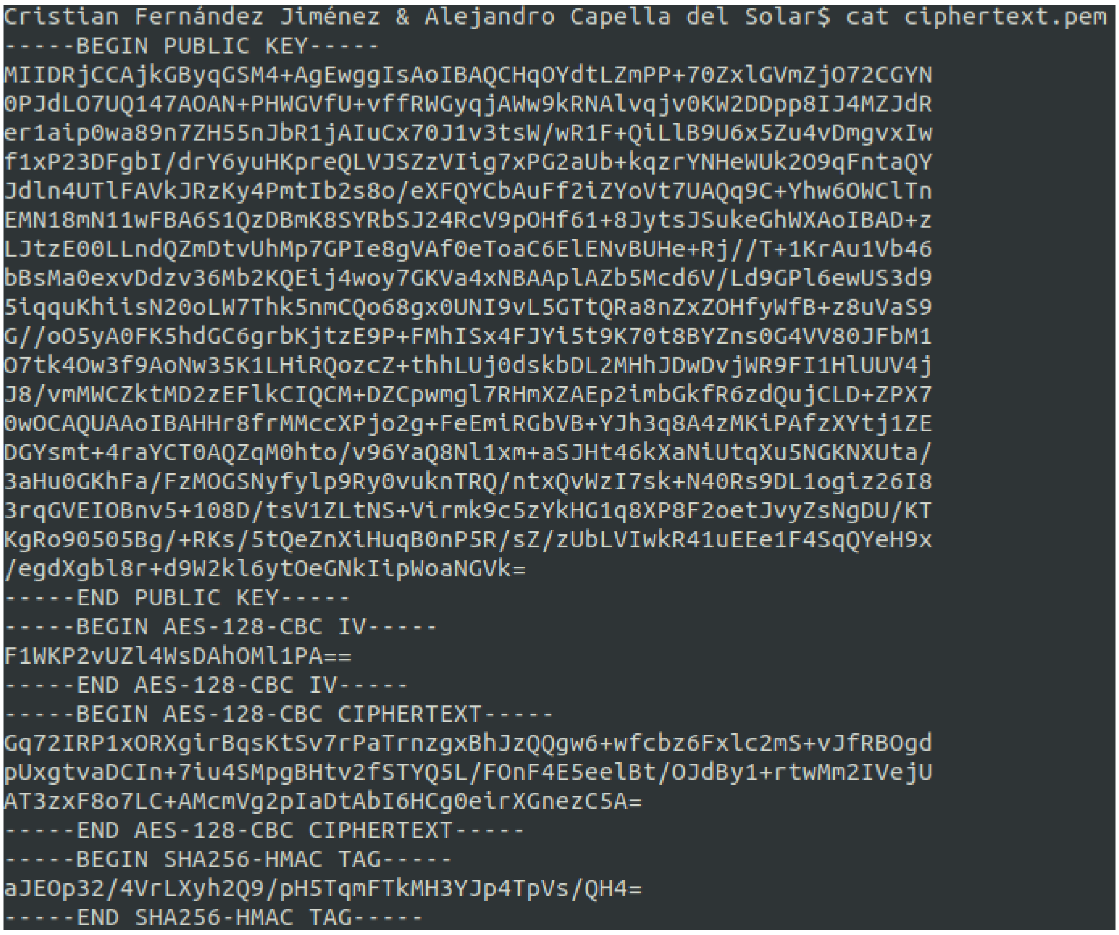
Derive a common secret from the secret ephemeral key, contained in `eph_pkey.pem`, and the long-term public key of the recipient, contained in `jorge_pubkey.pem`, in our case. The common secret is save in binary form in `commonsecret.bin`.

Apply SHA256 to the common secret. This extra layer of security is done to generate two 16 byte keys. A SHA256 is applied to the common secret and saved in `sha256.txt`. Then, the hash is split in two halves, one to encrypt the message, `k1.bin`, and the other, `k2.bin`, for the HMAC code generation.

Encrypt the message file with AES-128-CBC. Using key *k1* we compute a CBC and store it in `ciphertext.bin`. Previously, we would need to provide an iv for the encryption operation, generated randomly with `openssl`, 16 bytes are stored in `iv.bin`.

Compute a SHA256-HMAC tag. The concatenation of iv and ciphertext binaries are used to obtain a tag, `tag.bin`, for message authentication before decryption.

Parse a ciphered file. The final ciphertext consists of the four files `eph_pubkey.pem`, `iv.bin` , `ciphertext.bin` and `tag.bin`, storing the last three in BASE64 format and all together in a PEM file: `ciphertext.pem`

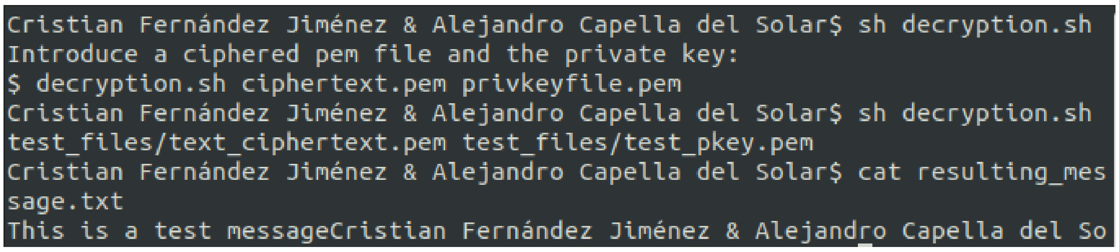


```
Cristian Fernández Jiménez & Alejandro Capella del Solar$ cat ciphertext.pem
-----BEGIN PUBLIC KEY-----
MIIDRjCCAjkGByqGSM4+AgEwggIsAoIBAQCHqOYdtLZmPP+70ZxlGVmZjO72CGYN
0PJdLO7UQ147AOAN+PHWGVfU+vffRWGyqjAWw9kRNAlvqjv0KW2DDpp8IJ4MZJdR
er1aip0wa89n7ZH55nJbR1jAIuCx70J1v3tsW/wR1F+QiLlB9U6x5Zu4vDmgvxIw
f1xP23DFgbI/drY6yuHKpreQLVJSZzVIig7xPG2aUb+kqzrYNHeWUk2O9qFntaQY
Jdln4UTlFAVkJRzKy4PmtIb2s8o/eXFQYCbAuFf2iZYoVt7UAQq9C+Yhw6OWClTn
EMN18mN11wFBA6S1QzDBmK8SYRbSJ24RcV9pOHf61+8JytsJSukeGhWXAoIBAD+z
LJtzE00LLndQZmDtvUhMp7GPIe8gVAf0eToaC6ElENvBUHe+Rj//T+1KrAu1Vb46
bBsMa0exvDdzv36Mb2KQEij4woy7GKVa4xNBAAplAZb5Mcd6V/Ld9GPl6ewUS3d9
5iqquKhiisN20oLW7Thk5nmCQo68gx0UNI9vL5GTtQRa8nZxZOHfyWfB+z8uVaS9
G//oO5yA0FK5hdGC6grbKjtzE9P+FMhISx4FJYi5t9K70t8BYZns0G4VV80JFbM1
O7tk4Ow3f9AoNw35K1LHiRQozcZ+thhLUj0dskbDL2MHhJDwDvjWR9FI1HlUUV4j
J8/vmMWCZktMD2zEFlkCIQCM+DZCpwmgl7RHmXZAEp2imbGkfR6zdQujCLD+ZPX7
0wOCAQUAAoIBAHHr8frMMccXPjo2g+FeEmiRGbVB+YJh3q8A4zMKiPAfzXYtj1ZE
DGYsmt+4raYCT0AQZqM0hto/v96YaQ8Nl1xm+aSJHt46kXaNiUtqXu5NGKNXUta/
3aHu0GKhFa/FzMOGSNyfylp9Ry0vuknTRQ/ntxQvWzI7sk+N40Rs9DL1ogiz26I8
3rqGVEIOBnv5+108D/tsV1ZLtNS+Virmk9c5zYkHG1q8XP8F2oetJvyZsNgDU/KT
KgRo90505Bg/+RKs/5tQeZnXiHuqB0nP5R/sZ/zUbLVIwkR41uEEe1F4SqQYeH9x
/egdXgbl8r+d9W2kl6ytOeGNkIipWoaNGVk=
-----END PUBLIC KEY-----
-----BEGIN AES-128-CBC IV-----
F1WKP2vUZl4WsDAhOMl1PA==
-----END AES-128-CBC IV-----
-----BEGIN AES-128-CBC CIPHERTEXT-----
Gq72IRP1xORXgirBqsKtSv7rPaTrnzgxBhJzQQgw6+wfcbz6Fxlc2mS+vJfRBOgd
pUxgtvaDCIn+7iu4SMpgBHtv2fSTYQ5L/FOnF4E5eelBt/OJdBy1+rtwMm2IVejU
AT3zxF8o7LC+AMcmVg2pIaDtAbI6HCg0eirXGnezC5A=
-----END AES-128-CBC CIPHERTEXT-----
-----BEGIN SHA256-HMAC TAG-----
aJEOp32/4VrLXyh2Q9/pH5TqmFTkMH3YJp4TpVs/QH4=
-----END SHA256-HMAC TAG-----
```

Figure 3: Encrypted message resulting.

4 Decryption

For the decryption part, a `decryption.sh` script has been created to recover a message given a cipher and a private key. The execution is as follows:



```
Cristian Fernández Jiménez & Alejandro Capella del Solar$ sh decryption.sh
Introduce a ciphered pem file and the private key:
$ decryption.sh ciphertext.pem privkeyfile.pem
Cristian Fernández Jiménez & Alejandro Capella del Solar$ sh decryption.sh
test_files/text_ciphertext.pem test_files/test_pkey.pem
Cristian Fernández Jiménez & Alejandro Capella del Solar$ cat resulting_mes
sage.txt
This is a test messageCristian Fernández Jiménez & Alejandro Capella del So
```

Figure 4: Decryption of given test message.

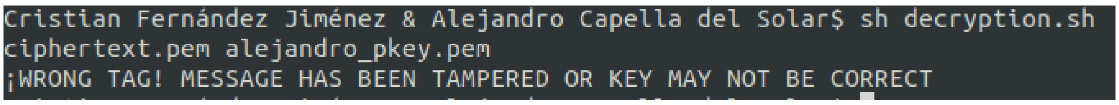
In the script, we have performed the following steps:

Parse the ciphertext file. Obtaining the four components `eph_pubkey.pem`, `iv.bin` , `ciphertext.bin` and `tag.bin`. Taking special care of removing headers and recover the binary representation from the BASE64 encoding.

Recover the common secret using the private key of the owner, `test_pkey.pem` in our case, and the public ephemeral key that we just parsed from the ciphertext.

Apply SHA256 to the common secret result in order to split the value into the two 16-byte keys *k1* and *k2*, necessaries to decrypt the text and generate the HMAC code verification.

Recompute the SHA256-HMAC from the concatenation of the files `iv.bin` and `ciphertext.bin` using *k2*. If the result is different from the recoverd tag, we will abort the decryption operation and raise the following error:



```
Cristian Fernández Jiménez & Alejandro Capella del Solar$ sh decryption.sh
ciphertext.pem alejandro_pkey.pem
¡WRONG TAG! MESSAGE HAS BEEN TAMPERED OR KEY MAY NOT BE CORRECT
```

Figure 5: Modifying the ciphertext results in tag error.

Decrypt the ciphertext using *k1* and `iv.bin`, to recover the plain text into `resulting_message.txt`.

A Resulting scripts

A github repository with all scripts is available online. All 3 scripts and an encrypted message, with `jorge_pubkey.pem`, are inside LAB3 folder. If necessary, please follow the execution instructions given on this document.