Lab Work 3: Public Key Encryption

Aleix Galan Figueras - 26062500G
Data Protection
Universitat Politècnica de Catalunya

10 de novembre de 2020

1 Introduction

This laboratory work describes how to operate with a hybrid encryption scheme with MAC authentication. This scheme will use DH as asymmetric encryption algorithm, AES-128-CBC as symmetric encryption algorithm and SHA256-HMAC to generate the MAC authentication.

In all the scripts, the treatment of the DH ephemeral private keys once derived a common secret is left to the user. As a rule of thumb, when derived the common secret the private keys used in DH should be deleted to guarantee perfect forward secrecy.

The corresponding code for this exercise can be found at: `github.com/Algafix/problemes-dprot/tree/master/practica3/jorge`

2 `keygen.sh` script

The `keygen.sh` script takes as parameter the name of the user for who we want to generate the keys. Then, using the default DH group configuration for this work generates the private key r (¡name¿_pkey.pem) and use it to generate the public key g^r (¡name¿_pubkey.pem).

```
# DH group
openssl genpkey -genparam -algorithm dh -pkeyopt dh_rfc5114:3 -
    out param.pem

# Generating keys
openssl genpkey -paramfile param.pem -out $1_pkey.pem
openssl pkey -in $1_pkey.pem -pubout -out $1_pubkey.pem
```

The script can be found at `github.com/Algafix/problemes-dprot/blob/master/practica3/jorge/keygen.sh`.

3 `encrypt.sh` script

The `encrypt.sh` script takes as input 2 arguments. The first one is the name of the peer who wants to encrypt and send the message. The former one is the name of the peer to who we want to send the message. This names are used to locate the keys to encrypt.

First of all we generate a common secret using the public key from the peer and our private key. Then we hash the common secret for security reasons and create a common key with 32 byte length.

Then from this key we derive an encryption key (first 16 bytes) and a authentication key (last 16 bytes).

```
# Generae common secret
openssl pkeyutl -inkey ${my_keys}_pkey.pem -peerkey ${peer_keys}_pubkey.pem -derive -out common.bin
cat common.bin | openssl dgst -sha256 -binary > commonkey.bin
head -c 16 commonkey.bin > k1.bin
tail -c 16 commonkey.bin > k2.bin
```

Now we can perform the encryption with AES-128-CBC and a random IV. For the authentication we will authenticate the ciphertext and the IV, as are the 2 parameters that need the other peer to decrypt the message. We are using SHA256-HMAC.

```
openssl rand 16 > iv.bin
openssl enc -aes-128-cbc -K `cat k1.bin | xxd -p` -iv `cat iv.bin | xxd -p` -in secret.txt -out ciphertext.bin
cat iv.bin ciphertext.bin | openssl dgst -sha256 -binary -mac hmac -macopt hexkey:`cat k2.bin | xxd -p` -out tag.bin
```

Finally, we print the all the needed values for the other peer in a PEM file in base64 encoding. These values are our public key, the IV, the ciphertext and the tag.

The script can be found at `github.com/Algafix/problemes-dprot/blob/master/practica3/jorge/encrypt.sh`.

4 decrypt.sh script

The `decrypt.sh` script takes as input 3 parameters. The first is the name of the peer performing the decryption, the second the name of the peer who send the message (just for file administration purposes) and the last parameter is the name of the cipher file received.

First of all we need to parse the cipher file to extract all the needed values and convert them again to binary from base64.

```
# Parse file
sed -n '/^-----BEGIN PUBLIC KEY-----/,/^-----END PUBLIC KEY
   -----/p' $cipherFile > d_${peer_keys}_pubkey.pem
sed '/^-----BEGIN AES-128-CBC IV-----/,/^-----END AES-128-CBC
   IV-----/{//!b};d' $cipherFile | openssl enc -d -a > d_iv.bin
sed '/^-----BEGIN AES-128-CBC CIPHERTEXT-----/,/^-----END AES
   -128-CBC CIPHERTEXT-----/{//!b};d' $cipherFile | openssl enc
    -d -a > d_ciphertext.bin
sed '/^-----BEGIN SHA256-HMAC TAG-----/,/^-----END SHA256-HMAC
   TAG-----/{//!b};d' $cipherFile | openssl enc -d -a > d_tag.
   bin
```

Now we can compute the common secret an derive the common keys as in the last section.

```
# Generate common secret
openssl pkeyutl -inkey ${my_keys}_pkey.pem -peerkey d_${peer_
   keys}_pubkey.pem -derive -out d_common.bin
cat d_common.bin | openssl dgst -sha256 -binary > d_commonkey.
   bin
head -c 16 d_commonkey.bin > d_k1.bin
tail -c 16 d_commonkey.bin > d_k2.bin
```

Finally we can decrypt the message. First we need to ensure that the message hasn't been mangled and for that purpose we first check the authentication tag. If the tag is the same, we can go ahead and decrypt the message with AES-128-CBC.

```
# Authenticate and decrypt
cat d_iv.bin d_ciphertext.bin | openssl dgst -sha256 -binary -
   mac hmac -macopt hexkey:‘cat d_k2.bin | xxd -p‘ -out
   recomputed_tag.bin

```

```
STATUS="$(cmp --silent d_tag.bin recomputed_tag.bin; echo $?)"
if [[ $STATUS -ne 0 ]]
then
    echo "Diferent Tag! Not authenticated."
else
    echo "Same Tag."
    openssl enc -d -aes-128-cbc -K `cat d_k1.bin | xxd -p` -iv
       `cat d_iv.bin | xxd -p` -in d_ciphertext.bin -out
       decrypted.txt
    echo "Decrypted text in decrypted.txt"
fi
```

The script can be found at `github.com/Algafix/problemes-dprot/blob/master/practica3/jorge/decrypt.sh`.