2019 oplossingen labo 5 - Lars Lemmens

Met dank aan de Github van Martijn en natuurlijk Lars Lemmens

LABO 5

Encrypting the self created file secret.txt using AES can be done like this: "

```
'user@:~$' • openssl enc -aes-128-cbc -in secret.txt -out secret.txt.aes
```

-The enc command is used for symmetric cipher routines

To decrypt the file created above and show the decrypted data on screen, use:

```
'user@:~$' • openssl enc -d -aes-128-cbc -in secret.txt.aes
```

To get a list of all available encryption algorithms in OpenSSL, type:

```
'user@:~$' • openssl list-cipher-commands
```

On more recent OpenSSL versions the command above will not work. The new

syntax is as follows:

```
'user@:~$' • openssl list -help
'user@:~$' • openssl list -cipher-commands
```

Thus we need to tell our application that ecrypted_picture.bmp is indeed a bitmap. Bitmaps begin with a 54-byte header.

The idea is to extract the first 54 bytes of the unencrypted bitmap and overwrite the first 54 bytes of the encrypted one with the real header. See the following examples: "

AES in ECB mode

```
'user@:~$' 1) openssl enc -aes-128-ecb -in slimmerik.bmp -out slimmerik_ECB.bmp
'user@:~$' 2) dd if=slimmerik.bmp of=slimmerik_ECB.bmp bs=1 count=54 conv=notrunc
```

"We could, of course, automate this: "

```
'user@:~$' 1) dd if=slimmerik.bmp bs=1 count=54 > slimmerik_ECB.bmp &&
 openssl enc -aes-128-ecb -k pass:t -in slimmerik.bmp | dd bs=1 skip=54 >> slimmerik_ECB.bmp
'user@:~$' 2) dd if=slimmerik.bmp of=slimmerik_ECB.bmp bs=1 count=54 &&
 dd if=slimmerik.bmp bs=1 skip=54 | openssl enc -aes-128-ecb -k pass:t >> slimmerik_ECB.bmp
```

- The dd command converts and copy a file
- The -k argument forces nc to stay listening for another connection after its current connection is completed

AES in CBC mode

```
'user@:~$' 1) openssl enc -aes-128-cbc -in slimmerik.bmp -out slimmerik_CBC.bmp
'user@:~$' 2) dd if=slimmerik.bmp of=slimmerik_CBC.bmp bs=1 count=54 conv=notrunc
```

ROT13 encryption

```
'user@:~$' 1) alias rot13="tr '[A-Za-z]' '[N-ZA-Mn-za-m]'"
'user@:~$' 2) cat slimmerik.bmp | rot13 > slimmerik_rot13.bmpw
```

- Match a single character present in the list below [N-ZA-Mn-za-m]
- N-Z matches a single character in the range between N (index 78) and Z (index 90) (case sensitive)
- A-M matches a single character in the range between A (index 65) and M (index 77) (case sensitive)
- n-z matches a single character in the range between n (index 110) and z (index 122) (case sensitive)
- a-m matches a single character in the range between a (index 97) and m (index 109) (case sensitive)

You need a key pair to be able to use GnuGP. You can generate one with: "

```
'user@:~$' • gpg --gen-key
```

After your keypair is created you should immediately generate a revocation certificate for the primary public key using the option --gen-revoke.

If you forget your passphrase or if your private key is compromised or lost, this

revocation certificate may be published to notify others that the public key should no longer be used

```
'user@:~$' • gpg --output revoke.asc --gen-revoke <mykey>
```

You can use your key pair to encrypt and sign, but without exchanging public keys this is useless. Others need your public key to verify your signatures and to send encrypted messages to you. You need their keys for the same purposes.

```
'user@:~$' • gpg --list-keys
```

You can export your public key using:

```
'user@:~$' • gpg --output <file> --export <email>
```

Now import some keys from classmates:

```
'user@:~$' • gpg --import <file>
```

You already have an automatically created encryption subkey. Now you will create another subkey for signing.

Instead of the master key the subkey will be used to verifying message signatures.

```
'user@:~$' • gpg --edit-key YOURMASTERKEYID
```

Exercise 1:

Bob needs to send a text file through an encrypted tunnel to Alice. Both already agreed on a shared secret 'secret' using the Diffie Hellman algorithm. Alice wants to display the contents of the file directly on her screen instead of storing it locally and then opening it. Use a suitable encryption algorithm. The data is sent over a medium which only allows ASCII text.

```
'Alice@leia:~$' • nc -l 10000 | openssl enc -a -d -aes-128-cbc -k pass:secret
'Bob@laptop:~$' • cat file | openssl enc -a -aes-128-cbc -k pass:secret | nc leia.uclllabs.be 10000
```

Exercise 2:

Bob needs to send a text file through an encrypted tunnel to Alice. Both already agreed on a shared secret 'secret' using the Diffie Hellman algorithm. Alice wants to display the contents of the file directly on her screen instead of storing it locally and then opening it. Use a suitable encryption algorithm. The data is sent over a medium which only allows ASCII text. "

Step 1:

As Bob needs to send an encrypted file to Alice, he will need her public key.

So in the first step Alice needs to generate a keypair, and export her public key so she could provide it to Bob. "

```
'Alice@Server ~ $' • gpg --gen-key
'Alice@Server ~ $' • gpg --output alice.gpg --export alice@uclllabs.be
```

Step 2:

Alice has exported her gpg public key to the file alice.gpg.

She sends this file e.g. by email to bob. Note that this file is no secret, it is just a public key. Now bob can import Alice's public key into gpg: "

```
'Bob@leia ~ $' • gpg --import alice.gpg
```

Step 3:

Bob is ready to sent his secret text file to Alice while providing confidentiality:

```
'Alice@Server ~ $' • nc -l -p 10000 | gpg --decrypt --quiet

'Bob@leia ~ $' • cat file.txt | gpg --encrypt --armor --output - --recipient alice@uclllabs.be |
 nc -N server.x.cnw2.uclllabs.be 10000
```

- The --encrypt argument encrypts data
- The --armor argument creates ASCII armored output
- The nc command is a TCP/IP swiss army knife

Exercise 3:

Same exercise as above, but now use gpg and netcat to create a simple chat application.

```
'Alice@Server ~ $' • nc -l -p 10000 | gpg --decrypt --quiet --allow-multiple-messages

'Bob@leia ~ $' • while read line; do echo $line | gpg --encrypt --armor --output - --recipient alice@uclllabs.be;
done | nc -N server.x.cnw2.uclll
```

Revision #1
Created 17 June 2021 14:14:57 by Jasper G.
Updated 3 December 2021 22:13:09 by Jasper G.