Lab 1: Security Services, Algorithms with toolkit OpenSSL

Alexander Hoffmann

February 20, 2020

1 Hash function

a. Using vim, we create a new file called `plain.txt` and type in ”security lab”. We can now print out the context of the file using the `cat` command.

b. We generate the hash values for the `plain.txt` file using `openssl`. The digest functions output the message digest of a supplied file or files in hexadecimal. The digest functions also generate and verify digital signatures using message digests.

```
openssl dgst -sha1 plain.txt >> hash.txt
```

Now if we print the content of the file, we get the following output:

```
SHA1(plain.txt)= 8d678009606a414a6d163a1d96d832fddde1d3e4
```

c. Now that we modified the content of the file, we generate another hash called `hash2.txt` using the same command as in question a.

```
openssl dgst -sha1 plain.txt >> hash2.txt
```

d. We use `cat` to display the content of the new file. This time, we get another result.

```
SHA1(plain.txt)= 87a7acddb0d51bf22ed471c32215b1990f95aacb
```

Hence, for every input, the hash function gives a different output. Note that for the same input, the SHA1 algorithm will give the same output.

2 Symmetric Encryption

Enc is used for various block and stream ciphers using keys based on passwords or explicitly provided. It can also be used for Base64 encoding or decoding.

Here are the arguments used:
-cipher - name of the cipher used, here des-cbc encryption method.
-base64 - encode in a base64 format once it has been ecrypted.
-k password - Specify a password or a file containing the password which is used for key derivation.
-in filename - This specifies the input file.
-out filename - This specifies the output file. It will be created or overwritten if it already exists.

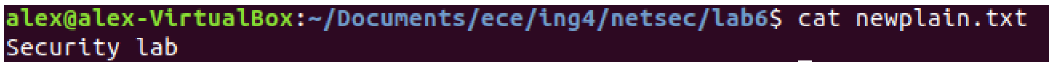
a. To encode a file `plain.txt` using password `ece`, we use the following command:

```
openssl enc -des-cbc -base64 -pbkdf2 -k ece -in plain.txt -out cipher.txt
```

b. Now we want to decode the file we previously encoded. To do this, use:

```
openssl enc -des-cbc -base64 -pbkdf2 -k ece -in cipher.txt -d -out newplain.txt
```

Finally, verify that both file are the same using `diff plain.txt newplain.txt` which yields no result at all, meaning both files are identical hence the encoding and decoding worked perfectly.



```
alex@alex-VirtualBox:~/Documents/ece/ing4/netsec/lab6$ cat newplain.txt
Security lab
```

3 Asymmetric Encryption

a. The genrsa command generates an RSA private key. To generate key with size 2048, we use:

```
openssl genrsa -out privMyName.key 2048
```

b. The rsa command processes RSA keys. They can be converted between various forms and their components printed out. To generate a public key from a private key, use:

```
openssl rsa -in privMyName.key -out pubMyName.key -pubout
```

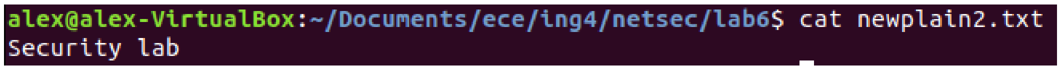
c. The rsautl command can be used to sign, verify, encrypt and decrypt data using the RSA algorithm. To encrypt our `plain.txt` file, we use:

```
openssl rsautl -encrypt -pubin -inkey pubMyName.key -in plain.txt -out cipherrsa.t
```

Now let's decrypt the file we just created using our private key.

```
openssl rsautl -decrypt -inkey privMyName.key -in cipherrsa.txt -out newplain2.txt
```

If we use `diff` to print the differences between both files, we will see that they are the same. This means that our encryption has worked correctly.



```
alex@alex-VirtualBox:~/Documents/ece/ing4/netsec/lab6$ cat newplain2.txt
Security lab
```

4 Asymmetric/Symmetric Encryption, Digital signature

a. Alice wants to send a private message to Bob but she doesn't want anyone to be able to read the content of the message. She decides to find a way to send messages to Bob without anyone else being able to read them: symmetric-key encryption.
Alice and Bob agree on a key K in private. Now they can talk aloud, encoding messages with the key.
Alice and Bob can't communicate a key in advance in private. This is a job for public-key cryptography. Now Bob has two keys, one (P) published, one (K) kept secret. A message encrypted with the public key P can only be decrypted with the private key K.
Alice gets P from Bob, encrypts a message, and sends it to Bob. Bob uses K to decrypt the message. A listener in the middle can easily get P, but cannot decrypt the message!
Alice and Bob agree on a key K in private. Now they can talk aloud, encoding messages with the key.

b. The rand command outputs n pseudo-random bytes after seeding the random number generator once. To generate a symmetric key, use:

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

c. Using `vim`, we create a new file called `plaintext.txt` containing the following sentence ”My Security LAB: My Name is Alex”.
Next, we generate two private RSA keys and export the public keys corresponding using the following commands:

```
openssl genrsa -out privA.key
openssl rsa -in privA.key -out pubA.key -pubout
```

For key A and

```
openssl genrsa -out privB.key
openssl rsa -in privB.key -out pubB.key -pubout
```

for key B.
d. Encrypt our plain text file using

```
openssl enc -aes-128-cbc -base64 -pbkdf2 -kfile sym.key -in plaintext.txt -out cip
```

e. Encrypt the symmetric key using

```
openssl rsautl -encrypt -pubin -inkey pubB.key -in sym.key -out secret.key
```

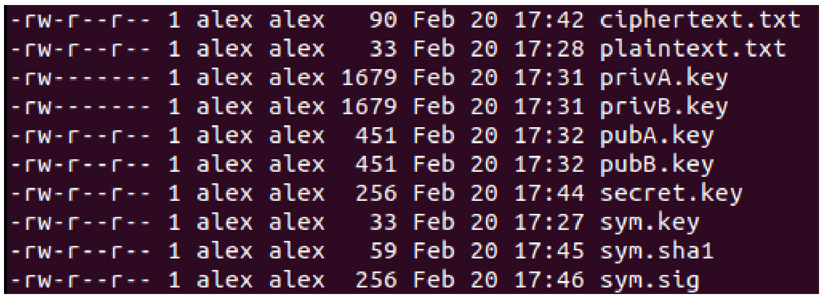
f. Generate the hash value of the key

```
openssl dgst -sha1 secret.key >> sym.sha1
```

g. Sign the hashed value file

```
openssl rsautl -sign -inkey privA.key -in sym.sha1 -out sym.sig
```

After all of this is set and done, we have all the necessary files to have a secret conversation between Alice and Bob. Here is what the folder looks like.



```
-rw-r--r-- 1 alex alex   90 Feb 20 17:42 ciphertext.txt
-rw-r--r-- 1 alex alex   33 Feb 20 17:28 plaintext.txt
-rw------- 1 alex alex 1679 Feb 20 17:31 privA.key
-rw------- 1 alex alex 1679 Feb 20 17:31 privB.key
-rw-r--r-- 1 alex alex  451 Feb 20 17:32 pubA.key
-rw-r--r-- 1 alex alex  451 Feb 20 17:32 pubB.key
-rw-r--r-- 1 alex alex  256 Feb 20 17:44 secret.key
-rw-r--r-- 1 alex alex   33 Feb 20 17:27 sym.key
-rw-r--r-- 1 alex alex   59 Feb 20 17:45 sym.sha1
-rw-r--r-- 1 alex alex  256 Feb 20 17:46 sym.sig
```