SECURITY LAB
CRYPTOGRAPHY

Achraf FAYAD

ECE PARIS

Session Requirements:

- Each group (composed of 2 persons at most) shall submit a report. Same groups shall be maintained during the semester.
- The report shall be uploaded on the campus page.
- Make a Zip file contains all files generated in **Task 4**, keeping the **same file names**, and a PDF report just for **Task 4**, including all commands used. **You have to clearly write your names on a cover page**. **Only PDF format is accepted (3 points penalty for other formats)**
- The deadline for submitting the reports is 19 March 2015.
  Tuesday 12/02/2019 midnight for group Gr05.
  Thursday 14/02/2019 midnight for group Gr01, Gr02 and Gr03
- Late reports are penalized (2 points per day)

Security Services, Algorithms with toolkit OpenSSL

Activity objective:

The learning objective of this lab is to get familiar with the toolkit OpenSSL. In this lab, we will use ***openssl*** commands and libraries.

OpenSSL is an open-source implementation of the SSL and TLS protocols. It provides:

a. The core library, written in the C programming language, implements basic cryptographic functions and provides various utility functions, and produce a secured applications Client/Server using SSL/TLS
b. We use line command OpenSSL to:
    1. Generate RSA, DSA keys (signing)
    2. Generate X509 certificate
    3. Calculate fingerprint (MD5, SHA…..)
    4. Encrypt and decrypt files (RSA, DES, 3DES ...)
    5. Testing Client/Server SSL/TLS
    6. Signing and crypt emails (S/MIME)

Lab Environment

In this lab, we will use ***openssl*** commands and libraries. You should first install openssl package, you will use **apt-get install openssl**.

- To see the manuals, you can type ***man openssl***

The OpenSSL is an entry point for many functions. You call it following the pattern:

```
$ openssl  <command>  <option>
```

Task 1: Hash function

In this task, we will play with various one-way hash algorithms. You will use the following *openssl dgst* command to generate the hash value for a file.

a. Create a text file ***Plain.txt*** contain: ***security lab***
b. Generate the hash values H1 for this file using SHA1 hash algorithm.
c. Modify ***s*** into ***S (capital letter)*** for the word ***security*** in the file Plain.txt; generate the hash values H2 for the modified file using SHA1 hash algorithm.
d. Please observe whether H1 and H2 are similar or not.

Task 2: Symmetric Encryption

In this task, you will use the command line ***openssl enc*** for encrypt and decrypt messages, for more information about this command you can type:

openssl enc –h

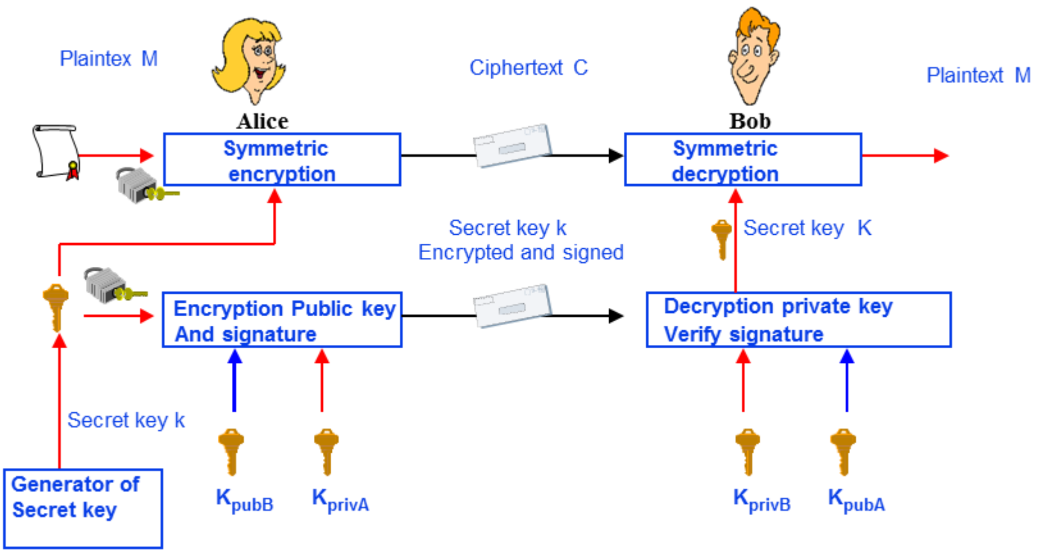
a. Encrypt Plain.txt was created in previous task with ***des-cbc*** algorithm, and save the file encrypted as Cipher.txt, use ***–k*** option to enter the password and you can use the ***–base64*** option to encode binary to text.
b. Decrypt the file Cipher.txt and save the new file as NewPlain.txt, compare the new file with Plain.txt, use : ***diff Plain.txt NewPlain.txt -q***

Task 3: Asymmetric Encryption

The objective of this task is to encrypt a message with a public key, and just the creator of this key will can decrypt the message.

a. Generate RSA keys with length of 2048 bits; ***privMyName.key*** is a name of the private key
b. Extract public key from ***privMyName.key***; ***pubMyName.key*** is a name of the public key.
c. Encrypt the file Plain.txt with the public key, save the file encrypted as CipherRSA.txt.
d. Decrypt the file CipherRSA.txt and save the new file as New2Plain.txt, compare the new file with Plain.txt, use : ***diff Plain.txt NewPlain.txt -q***

Task 4: Asymmetric/Symmetric Encryption, Digital signature



a. Explain the above scenario.
b. Generate the symmetric key ***sym.key*** with length of ***128 bits***; use the rand command and encode the key in ***hex***.
c. Realize the above scenario by exchanging a safe way the symmetric key with your colleague.
    a. Create ***Plaintext.txt*** contain : My Security LAB: My Name is *<your_name>*
    b. Generate RSA keys with length of 2048 bits: ***privA.key, pubA.key, privB.key*** and ***pubB.key*** (you can use keys generated in Task 3 but you have to modify the file names).
    c. Exchange the public keys (use sftp or USB Flash)
    d. Encrypt PlaintextM.txt using symmetric algorithm (-***aes-128-cbc***) : ***Ciphertext.txt*** (use the key generated in this task), you will use the options ***–kfile*** for password and ***-base64***
    e. Encrypt the symmetric key using asymmetric algorithm (generate an encrypted file: ***secret.key***)
    f. Generate the hash value of symmetric key using SHA1 (***sym.sha1***)
    g. Sign ***sym.sha1***, generate a file ***sym.sig***
    h. Send the necessary files to your colleague that allows decrypting your message, and verify your signature ***(you have to use the same options for decrypting as encrypting),***

explain the different steps to decrypt and verify the digital signature.

d. What are the different security services assured by this scenario?

Job Aids

$diff file1.txt file2.txt –q: compares the contents of the two files

```
$sftp username@remote_hostname_or_IP
        ls (list remote files)
        lls ( list local files)
        cd (change directory / remote)
        lcd (change directory / local)
        get remoteFile
        put localfile
        exit
```

$openssl genrsa -out <file_rsa.priv> <size>
$ openssl rsa -in <file_rsa.priv> -des3 -out <file.pem>
$ openssl rsa -in <_rsa.priv> -pubout -out <file_rsa.pub> :
$ openssl enc <-algo> -in <Plain.txt> -out <Cipher.enc> : encrypt
$ openssl enc <-algo> -in <Cipher> -d -out <Palin> : decrypt
$ openssl dgst <-algo> -out <out_file> <in_file>
$ openssl rand -out <key> <**number_bytes**> -hex
$ openssl rsautl -encrypt -pubin -inkey <rsa.pub> -in <Plain.txt> -out <Cipher.enc>
$ openssl rsautl -decrypt -inkey <rsa.priv> -in <Cipher.enc> -out <file.txt>
$ openssl rsautl -sign -inkey <ras.priv> -in <file.txt> -out <file.sig>
$ openssl rsautl -verify -pubin -inkey <rsa.pub> -in file.sig