Cryptography
Encryption and decryption

This lab is based on Openssl.

The OpenSSL cryptographic API is widely recognised and utilised due to its open-source nature under the GNU General Public License (GPL). For comprehensive details, you can visit the official website at http://www.openssl.org.

OpenSSL encompasses functionalities like SSL/TLS protocols, which are also integrated into the Apache server for providing secure services.

OpenSSL offers a command-line interface, allowing users to interact directly with its API by simply invoking the 'openssl' command. However, one might find that the documentation for OpenSSL is not as exhaustive or user-friendly as desired. To explore the functionalities of various commands, users can append '-help' after any command for immediate assistance. Additionally, OpenSSL's official website hosts a manual detailing these commands.

The purpose of this lab exercise is to acquaint you with the core security services, including encryption and hashing, provided by OpenSSL, enhancing your understanding and skills in cryptographic operations.

Notation all commands and arguments involved will be noted in the following in ***bold italic***
example:
openssl> ***genrsa mykey-des3-out 1024***

Symmetric Encryption

The command that allows you to use symmetric encryption is the command ***enc***
Example:
Openssl> ***enc help***

The symmetric cipher commands allow data to be encrypted or decrypted using various block and stream ciphers using keys based on passwords or explicitly provided. Base64 encoding or decoding can also be performed either by itself or in addition to the encryption or decryption.

Question 1:
Select a file containing textual data for this exercise. Use the command to encrypt this file, resulting in an output named ***file.enc.*** This process renders the content of ***file.enc*** unintelligible, as verified by attempting to read ***file.enc*** directly.
Send this encrypted file to a colleague, who will decrypt it.
After decryption, verify that the original content has been accurately restored by comparing the decrypted file with the original. Throughout this process, you're encouraged to employ a variety of encryption algorithms.
Additionally, incorporate the ***-base64*** option in your encryption command to generate a base64-encoded file, which, while not making the file 'readable' in terms of revealing its original content, transforms it into a format that can be safely transmitted through text-based channels.
Experiment with different algorithms and options, like ***-a,*** to produce a base64-encoded, encrypted file, enhancing the security and transmission ease of your data.

Question 2:
Encrypt a file using a symmetric encryption algorithm of your choice, along with any mode of operation, and subsequently decrypt the resulting ciphertext.

Question 3:
Analyse the file sizes of the plaintext and its encrypted version. Provide a detailed explanation for the differences observed between these sizes.

Question 4:
Attempt to decrypt a ciphertext with an incorrect password. What results do you observe?

Question 5:
Encrypt a file using the AES-128 algorithm in CBC mode, specifying the key and initialization vector in hexadecimal format with the -***K*** and -***iv*** options.
Subsequently, decrypt the resulting ciphertext.

Question 6:

Repeat the process described in Question 5, but this time, incorporate the option -***des-ede*** for encryption.

Question 7:
Decrypt the file named ***secret.enc,*** which has been encrypted using the DES-CBC method.
The initialization vector used is "ABABABABABABABAB", and the key, encoded in Base64, is "QUJBQkFCQUJBQkFCQUJBQg==".
Discuss the implications of using a Base64 encoded key in this context.

Question 8:
To answer this question, you'll need to utilize the time command available in Linux to measure the duration of cryptographic operations. Compare the time taken for encryption versus decryption processes. Additionally, evaluate the performance differences between various encryption algorithms by selecting a file of consistent size for your tests. This approach will provide a fair basis for comparison.

Question 9:
Why, when you apply the ***enc*** command twice—first to the plaintext file and then to the resulting encrypted file—do you not obtain clear text at the end?

Question 10:
Encrypt a file using the AES-128 algorithm in CBC mode, then attempt to decrypt it with AES-128 in CFB mode using the same key. What differences or issues do you observe?

Question 11:
Create a Bourne Shell command to encrypt either a single file or an entire directory.
Example :

myciphercde arg key algorithm
    arg may be a file or directory
    key encryption key
    algorithm symmetric encryption algorithm to be used

Question 12:
How can symmetric encryption be utilized to authenticate users or processes using a password?

Question 13:
It quantifies a machine's computational capability in terms of MIPS (Millions of Instructions Per Second). The average computational power of a standard PC is approximately 3000 MIPS. Assuming that testing one key requires executing 1000 instructions, consider the following scenarios for key sizes of 40 bits and 128 bits:

a) Calculate the time required for a brute force attack to test all possible keys, given that we have a plaintext/ciphertext pair encrypted with the same key.

b) Determine how many PCs would be needed in parallel to break the key in just 3 minutes.

c) Considering Moore's Law, which states that the computational power of machines doubles approximately every 18 months, estimate the number of years until a single PC could break a 128-bit key within 3 minutes.

Hash Functions

Question 1:
List several well-known hash functions. Which algorithms are you familiar with?

Question 2:
What term refers to the output produced by a hash function?

Question 3:
How do you invoke hash functions using OpenSSL? Provide the command syntax.

Question 4:
Identify all the hash functions that OpenSSL currently supports.

Question 5:
Apply each supported hash function to a specific file. Compare and discuss the sizes of the resulting hash fingerprints, particularly between MD5 and SHA-1.

Question 6:
What happens to the size of the hash if you apply the hash function a second time to its own output (the hash of the hash)? Explain the outcome.

Question 7:
If you hash a file with MD5 that is exactly 14 bytes in size, what characteristics would you expect in the resulting hash?

Question 8:
Perform the same analysis as in Question 7, but using SHA-1 instead of MD5. What differences, if any, do you observe?

Question 9:
How can hash functions be utilized in password authentication processes?

Question 10:
Describe the functionality and use of HMAC (Hash-based Message Authentication Code) functions.

Question 11 :
Demonstrate how to apply an HMAC function to a file using OpenSSL.

Question 12:
Define "One-Time Password" (OTP). How are OTPs generated, and what role do hash functions play in their creation?

Asymmetric Encryption

Generation of key private / public RSA
The ***genrsa*** command in OpenSSL is used for generating RSA private and public key pairs, with the key size (modulus) specified as an argument. The default output format for these keys is PEM (Privacy Enhanced Mail), but you can opt for DER format or revert to PEM using the ***-inform*** and ***-outform*** options, which support PEM and DER formats.

The resulting key file includes all components related to the RSA key pair, such as the primes p and q, the modulus n, the public exponent e, and the private exponent d. To inspect or decode the contents of this file, or to extract the public key into a separate file, you can employ the rsa command. This tool not only displays or decodes key information but also enables the extraction of the public key for standalone use.

In practice, using these commands and their options allows for flexible management of RSA keys. Additionally, OpenSSL provides a similar command, ***gendsa,*** for generating keys based on the ElGamal asymmetric algorithm.

For syntax and thus command options genrsa, use the option ***–help***

```
openssl> genrsa -help
openssl> genrsa 1024
```

The key pair is generated and displayed on the screen to save it in a file using the option ***–out***

```
openssl> genrsa -out mykey 1024
```

You have requested to view the content of the file mykey

```
openssl> genrsa -des3 -out mykey 1024
```

the private key is encrypted with Triple DES by

Verification of the RSA private key:

```
Openssl> rsa -in key
Openssl> rsa –in key -check
Openssl> rsa -in key –check -modulus
Openssl> rsa -in key –modulus –check -text
```

RSA public key extraction:

```
Openssl> rsa -in key -pubout -out mypubkey
```

Verification of RSA public key:

```
Openssl> rsa -pubin -in mypubkey -text
```

Question 1:
Using OpenSSL, generate an RSA key pair with a modulus size of 1024 bits. Provide the command used for this operation.

Question 2:
Generate an RSA key pair with a size of 1024 bit key and encrypt it with aes256.

Question 3:
Analyse the RSA key pair you generated earlier. Explain how to verify the integrity and attributes of the key pair using OpenSSL commands.

Question 4:
From the RSA key pair, extract and save the public key into a separate file. Provide the command to accomplish this task.

Question 5:
Intentionally introduce an error into the modulus of your RSA key pair. Then, use the rsa command with an appropriate check option to identify and describe the error detected.

Question 6:
Search for and locate examples of public keys on your operating system. Explain where you found them and what type of keys they are (e.g., SSH, SSL certificates).

Question 7:
Describe the operational principles of either the ElGamal or DSA asymmetric encryption algorithms. Include how they differ from RSA in terms of key generation, encryption, and decryption processes.