Lab 3: Asymmetric Encryption Algorithms in OpenSSL

1 Overview

The key objective of this lab is to provide a practical introduction to public key encryption, with a focus on RSA and Elliptic Curve methods. This includes the creation of key pairs and the signing process. Therefore, the learning objective of this lab is for students to get familiar with the concept of asymmetric cryptographic algorithms. After finishing the lab, students should be able to gain first-hand experience with asymmetric cryptographic algorithms. Moreover, students will be able to use OpenSSL commands to encrypt/decrypt short messages by using asymmetric encryption algorithms such as RSA.

2 Lab Environment

In this lab, we will use `openssl` commands such as rand, passwd, enc, genrsa, rsautl, dgst, req, and verif.

The systanx of openssl command is:

```
openssl command [options] [arguments]
```

3 Asymmetric Cryptographic Algorithms

In this section, asymmetric cryptographic algorithms with OpenSSL are presented such as RSA and elliptic curves.

3.1 Encryption with RSA Key generation

The key generation of RSA with OpenSSL is performed using:

```
openssl genrsa -out key.pem size
```

To obtain information on a key, we can use:

```
openssl rsa -in key.pem -text -noout
```

To retrieve the public key associated with a private key, we use:

```
openssl rsa -in private-key -pubout -out public-key
```

In order to save the key in a safe way, a symmetric encryption algorithm (such as triple-DES) can be used:

```
openssl genrsa -des3 -out private-key size
```

1. Create a private RSA key of 50 bits and save it into file “mySmall.pem”.
2. Display information on the private key you generated.

3. Should the private key be stored as plaintext? Is your key safely stored in the file “mySmall.pem”?
4. Create a private RSA key “private1.pem” of 1024 bits.
5. Create a private RSA key“ private2.pem” of 1024 bits.
6. Retrieve the information from keys “private1.pem” and “private2.pem”.
7. Create the corresponding public key associated to each private keys (“private1.pem” and “private2.pem”).
8. Should we store the public key in plaintext? Why?
9. To display information on a public key, we have to use option “-pubin” in order to inform OpenSSL that the input file only contains public data. Retrieve the information of the obtained two public keys.
10. Create a 4096 bits RSA public-private key pair.
11. Get the public key of the public-private key pair file and store it in another file.
12. Display the detailed private key information.
13. Encrypt the public-private key pair using AES-256 (“-aes256”) algorithm.

3.2 Working with Elliptic Curve Cryptography (ECC)

ECC is now used extensively within public-key encryption, including with Bitcoin, Ethereum, Tor, and many other applications. In this part of the lab, we will use OpenSSL to create a ECC key pair. For this, we generate a random 256-bit private key ($priv$), and then generate its corresponding public key point ($priv$ multiplied by G), where G is the generator point on the selected elliptic curve. Elliptic curves are also implemented in OpenSSL. To list the available elliptic curves available:

```
openssl ecparam -list\_curves
```

In this task, you should try at least 3 different variants of elliptic curves. You can find the meaning of the command-line options and all the supported options by typing "`openssl ecparam -help`". We include some common options for the `openssl ecparam -help` command in the following:

```
ecparam [options]
 -list_curves        Prints a list of all curve 'short names'
 -inform PEM|DER     Input format - default PEM (DER or PEM)
 -outform PEM|DER    Output format - default PEM
 -in infile          Input file  - default stdin
 -out outfile        Output file - default stdout
 -text               Print the ec parameters in text form
 -check              Validate the ec parameters
 -noout              Do not print the ec parameter
 -name val           Use the ec parameters with specified 'short name'
 -genkey             Generate ec key
 -writerand outfile  Write random data to the specified file
```

14. Create a private key using three types of EC such as the P-224 (“secp224k1”) elliptic curve.

15. Now, use “cat priv.pem” to view your key. Can you view your key? Why?

16. Encrypt private key using 3DES algorithm

To generate your public key based on your private key, we have to use the ”ec” OpenSSL command. You can find the meaning of the command-line options and all the supported options by typing "`openssl ec -help`". We include some common options for the `openssl ec -help` command in the following:

```
ec [options]
Valid options are:
 -in infile           Input file
 -inform format    Input format - DER or PEM
 -out outfile      Output file
 -outform PEM|DER  Output format - DER or PEM
 -noout            Don't print key out
 -text             Print the key
 -param_out        Print the elliptic curve parameters
 -pubin            Expect a public key in an input file
 -pubout           Output public key, not private
 -no_public        exclude public key from a private key
 -check            check key consistency
 -*                Any supported cipher
```

17. Create the corresponding public key using the previously produced EC private keys.

4 Encryption and decryption with RSA

With OpenSSL, the ”rsautl” command can be used to sign, verify, encrypt and decrypt data using the RSA algorithm.

You can find the meaning of the command-line options and all the supported dgst types and algorithms by typing "`openssl rsautl -help`". We include some common options for the `openssl rsautl` command in the following:

```
 rsautl [options]
     -in infile              Input file
     -out outfile            Output file
     -inkey val              Input key
     -keyform PEM|DER|ENGINE  Private key format - default PEM
     -pubin                  Input is an RSA public
     -certin                 Input is a cert carrying an RSA public key
     -sign                   Sign with a private key
     -verify                 Verify with a public key
     -encrypt                Encrypt with a public key
     -decrypt                Decrypt with a private key
```

Therefore, to encrypt a file or message using a public key, we use:

```
openssl rsautl -encrypt -in plaintext -inkey public-key -pubin -out ciphertext.
```

To decrypt it, we use the option -decrypt.

In the following, you should:

18. Encrypt and decrypt a small file using RSA private and public keys, respectively.
19. Encrypt a large file using RSA private. What happens? Why?