Encryption Algorithms Implementation using OpenSSL

Mary Alexis Solis

May 2017

1 Introduction

This document exhibits how AES, SHA, and RSA was implemented using OpenSSL. A Python script was written to run all the shell scripts in one compilation. However, the implementation of OpenSSL is purely on shell script.

2 Symmetric Encryption

Advanced Encryption Standard (AES) is based on a design principle known as a substitution-permutation network, a combination of both substitution and permutation, and is fast in both software and hardware. Unlike its predecessor DES, AES does not use a Feistel network. AES is a variant of Rijndael which has a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits. By contrast, the Rijndael specification per se is specified with block and key sizes that may be any multiple of 32 bits, both with a minimum of 128 and a maximum of 256 bits.[1]

AES operates on a 4 by 4 column-major order matrix of bytes, termed the state, although some versions of Rijndael have a larger block size and have additional columns in the state. Most AES calculations are done in a particular finite field.

2.1 Implementation

2.1.1 AES 128 ECB

To perform ECB mode on a AES-128, the following is run as a shell script.

```
openssl aes-128-ecb -a -salt -in
    resources/lena512color.tiff
  > results/aes/lena512-aes128-
  ecb.dat
```

The option *aes-128-ecb* is used to set the encryption to AES in ECB mode. Here is the output. Moreover, the result changes every encryption because the public key used is regenerated too.

2.1.2 AES 128 CBC

To perform CBC mode on a AES-128, the following is run as a shell script.

```
openssl aes-128-cbc -a -salt -in
    resources/lena512color.tiff
  > results/aes/lena512-aes128-
  cbc.dat
```

The option *aes-128-cbc* is utilized to direct OpenSSL use AES in CBC mode. Here is the output. Moreover, the resulting encrypted file is different every encryption even though the same key is used. This is because the public key also changes.

3 Hashing

The Secure Hash Algorithms (SHA) are a family of cryptographic hash functions published by the National Institute of Standards and Technology (NIST) as a U.S. Federal Information Processing Standard (FIPS) [2], including:

1. SHA-1: A 160-bit hash function which resembles the earlier MD5 algorithm. This was designed by the National Security

Agency (NSA) to be part of the Digital Signature Algorithm. Cryptographic weaknesses were discovered in SHA-1, and the standard was no longer approved for most cryptographic uses after 2010.

2. SHA-2: A family of two similar hash functions, with different block sizes, known as SHA-256 and SHA-512. They differ in the word size; SHA-256 uses 32-bit words where SHA-512 uses 64-bit words. There are also truncated versions of each standard, known as SHA-224, SHA-384, SHA-512/224 and SHA-512/256. These were also designed by the NSA.

3.1 Implementation

3.1.1 SHA1

To encrypt using SHA1, the following command is run through the shell script.

```
openssl dgst -sha1 resources/
    lena512color.tiff > results/
    sha/lena512-sha1.dat
```

The option *-sha1* directs OpenSSL to encrypt using SHA1. Here is the output.

3.1.2 SHA256

To encrypt using SHA256, the following command is run through the shell script.

```
openssl dgst -sha256 resources/
    lena512color.tiff > results/
    sha/lena512-sha256.dat
```

The option *-sha256* directs OpenSSL to encrypt using SHA256. Here is the output.

3.1.3 SHA512

To encrypt using SHA512, the following command is run through the shell script.

```
openssl dgst -sha512 resources/
    lena512color.tiff > results/
    sha/lena512-sha512.dat
```

The option *-sha512* directs OpenSSL to encrypt using SHA512. Here is the output.

4 Public Key Encryption

RSA, created by Ron Rivest, Adi Shamir, and Leonard Adleman, introduced the asymmetric encryption system. The system diverted from the traditional symmetric encryption which made a paradigm shift on how encryption is performed. A user who wishes to use RSA creates and publishes a public key based on two large prime numbers. The prime numbers were kept as a secret. Any wants to send a message to the user can use the public can encrypt the message using the public key. Even though the public key is known to the public, if the public key is large enough, only someone who knows the prime numbers can feasibly decode the message. Breaking RSA encryption is known as the RSA problem and its strength still lies of the hardness of the factoring problem[4].

4.1 Implementation

4.1.1 RSA-2048

```
echo resources/lena512color.
    tiff | openssl rsautl -
    encrypt -pubin -inkey
    resources/keys/rsa2048.pub.
    pem  > results/rsa/lena512-
    rsa2048.dat
```

The input is the same lena512color.tiff. The image file was encrypted using the *openssl rsautl -encrypt -pubin -inkey* using the rsa2048-public.pem that was created. Here is the output.

To create the public key, the rsa2048-private.pem was first generated by [3],

```
openssl genrsa -out rsa2048-
    private.pem 2048
```

Then, the rsa2048-public.pem file is generated by [3],

```
openssl rsa -in rsa2048-private.
   pem -outform PEM -pubout -out
    rsa2048-public.pem
```

4.1.2 ECDSA signature using SHA-256

```
openssl dgst -sha256 -sign
    resources/keys/ecdsa-private
    .pem -out results/rsa/
    lena512-ecdsa-sha256.dat
    resources/lena512color.tiff
```

The input is the same lena512color.tiff. The image file was encrypted using the *openssl dgst -sha256 -sign* using the ecdsa-private.pem that was created. This script was also run in python. Here is the output.

The file ecdsa-private.pem was generated by [5],

```
openssl ecparam -genkey -name
    secp384r1 -noout -out ecdsa-
    private.pem
```

References

[1] Advanced Encryption Standard. https://goo.gl/FFWZTJ.

[2] Secure Hash Algorithms. https://goo.gl/tpBLuO.

[3] OpenSSL: Generating an RSA Key From the Command Line. https://rietta.com/blog/2012/01/27/openssl-generating-rsa-key-from-command/.

[4] RSA algorithm (Rivest-Shamir-Adleman). https://goo.gl/5QGRL.

[5] OpenSSL ECDSA sign and verify file. https://goo.gl/K6xeWY.