#### COEN 146: Computer Networks

**Lab 9: Encryption (Optional for extra credit)**

**Objectives**

##### To use OpenSSL software for general-purpose cryptography and secure communication.

##### To demonstrate the use of encryption algorithms: symmetric, asymmetric and no key

##### 

##### **Guidelines**

[[1]](#footnote-1)OpenSSL is a software library for applications that secure communications over computer networks against eavesdropping or need to identify the party at the other end. It is widely used by Internet servers, including the majority of HTTPS websites.

OpenSSL contains an open-source implementation of the SSL and TLS protocols. The core library, written in the C programming language, implements basic cryptographic functions and provides various utility functions. Wrappers allowing the use of the OpenSSL library in a variety of computer languages are available.

The OpenSSL Software Foundation (OSF) represents the OpenSSL project in most legal capacities including contributor license agreements, managing donations, and so on. OpenSSL Software Services (OSS) also represents the OpenSSL project, for Support Contracts.

For more information visit: <https://www.openssl.org>

**Getting started with OpenSSL**

OpenSSL is available for most Unix-like operating systems (including Linux, macOS, and BSD) and Microsoft Windows.

Make sure you have OpenSSL installed by typing on the command line: openssl version -a

In this lab, you will follow the steps below and write down your observations for each of the steps.

**Secret key Encryption (symmetric key)[[2]](#footnote-2)**

Here we look at substitution cipher, DES, and AES. In general:

* Encryption is done by replacing *units* of plaintext with ciphertext, according to a fixed system.
* *Units* may be single letters, pairs of letters, triplets of letters, mixtures of the above, and so forth
* Decryption simply performs the inverse substitution.
* Two typical substitution ciphers:
  + monoalphabetic - fixed substitution over the entire message
  + Polyalphabetic - a number of substitutions at different positions in the message

Step 1. Monoalphabetic encryption: use the tr utility to encrypt a plain text file as follow:

* Generate a plain text file as follows:

% echo "This is a secret file that has important information which we do not want to reveal" > plaintext

* Generate a shift key for each letter, so let us encode ‘a – z’ letters with ‘qgvmftzyceolhsuwbjaxdnikpr’. Hint you can use the following python code to generate a random substitution key:

import random

s = "abcdefghijklmnopqrstuvwxyz"

list = random.sample(s, len(s))

key = ''.join(list)

print(key)

* For encryption and decryption use, then compare plaintext with plaintxt files:

% tr 'a - z' 'qgvmftzyceolhsuwbjaxdnikpr' < plaintext > ciphertext

% tr 'qgvmftzyceolhsuwbjaxdnikpr' 'a - z' < ciphertext > plaintext

* Frequency analysis is used to break Monoalphabetic Substitution Cipher. Try to break the ciphertext given in Lec24, slide 14. So

% echo "UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZVUEPHZHMDZSH ZOWSFPAPPDTSVPQUZWYMXUZUHSXEPYEPOPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ" > ciphertext

* Then use the tr utility by finding the matching patterns as the passing parameter.

Step 2. Data Encryption Standard (DES) and Advanced Encryption Standard (AES) are block ciphers. Here are the characteristics of DES and AES:

* DES can only encrypt a block of data.
* Block size for DES is 64 bits.
* DES uses 56-bit keys although a 64-bit key is fed into the algorithm.
* Theoretical attacks were identified. None was practical enough to cause major concerns.
* Triple DES can solve DES’s key size problem
* AES has 128-bit block size.
* AES uses three different key sizes: 128, 192, and 256 bits

Now we use the OpenSSL software for the encryption and decryption, let us try AES encryption and decryption for the plaintext file used in step 1, then compare plaintext with plaintxt files:

* We use openssl enc command
* We use the 128-bit (key size) AES algorithm
* The **-aes-128-ecb** option specifies ECB mode
* The **-e** option indicates encryption
* The **-d** option indicate decryption
* The **-K** option is used to specify the encryption/decryption key

% openssl enc -aes-128-ecb -e -in plaintext -out ciphertext -k 00112233445566778899AABBCCDDEEFF

% openssl enc -aes-128-ecb -d -in ciphertext -out plaintxt -k 00112233445566778899AABBCCDDEEFF

* We can also use an initialization vector (IV) by passing –v option with the IV key (128 bits). The main purpose of **IV** is to ensure that even if two plaintexts are identical, their ciphertexts are still different, because different IVs will be used.

**Public key Encryption (asymmetric key)[[3]](#footnote-3)**

Historically same key was used for encryption and decryption, but the challenge is exchanging the secret key (e.g. face-to-face meeting). In 1976, Whitfield Diffie and Martin Hellman proposed a key exchange protocol laying the foundation of a new public-key cryptosystem. In 1978, Ron Rivest, Adi Shamir, and Leonard Adleman (all from MIT) created RSA algorithm, which is the foundation of today’s secure communication over the Internet. Here are the characteristics:

* It allows communicating parties to obtain a shared secret key
* Public key (for encryption) and Private key (for decryption)
* Private key (for digital signature) and Public key (to verify signature)

Now we use the OpenSSL software for the RSA encryption and decryption

Step 3. Using OpenSSL RSA tool:

* Generate a 1024-bit public/private key pair: % openssl genrsa -aes128 -out privatekey 1024
* You can view the actual content of the private key as: %openssl rsa -in privatekey -noout -text
* Extract public key: % openssl rsa -in privatekey -pubout > publickey
* You can view the actual content of the public key as: % openssl rsa -in publickey -pubin -noout -text
* For RSA Encryption and Decryption (of the plaintext in Step 1), we use:

% openssl rsautl -encrypt -inkey publickey -pubin -in plaintext -out ciphertext

% openssl rsautl -decrypt -inkey privatekey -in ciphertext

Step 4. You can also generate a Digital Signature using RSA as follow:

* Generate message hash

% openssl sha256 -binary plaintext > plaintext.sha256

% xxd plaintext.sha256

* Sign the hash

% openssl rsautl -sign -inkey privatekey -in plaintext.sha256 -out plaintext.sig

* Verify the signature

% openssl rsautl -verify -inkey publickey -in plaintext.sig -pubin -raw | xxd

**Hash function (no key)[[4]](#footnote-4)**

Hash function maps arbitrary size data to data of fixed size. Example: f(x) = x mod 1000. It has one-way Hash Properties:

* One-way: hash(m) = h, difficult to find m
* Collision resistant: Difficult to find m1 and m2 s.t. hash(m1) = hash(m2)

The Common One-way Hash Functions are MD (Message Digest) and SHA series. The MD was developed by Ron Rivest and includes MD2, MD4, MD5, and MD6 with the following characteristics:

* MD2, MD4 - severely broken (obsolete)
* MD5 - collision resistance property broken, one-way property not broken
* MD6 - developed in response to proposal by NIST

The SHA was published by NIST and includes SHA-0, SHA-1, SHA-2, and SHA-3 with the following characteristics:

* SHA-0: withdrawn due to flaw
* SHA-1: Designed by NSA; Collision attack found in 2017
* SHA-2: Designed by NSA; Includes SHA-256 and SHA-512; Other truncated versions; No significant attack found yet
* SHA-3: Released in 2015; Has different construction structure (compared to SHA-1 and SHA-2)

Now we use the OpenSSL software for the MD and SHA hash function

Step 5. Using OpenSSL dgst tool to calculate the hash

% openssl dgst -sha256 plaintext

% openssl dgst -md5 plaintext

**Requirements to complete the lab**

1. Show the TA your encryption (symmetric, asymmetric, no key).
2. Write up a description of your steps. Imagine you are writing a guide for a class-mate unfamiliar with OpenSSL and provide instructions guiding them to the point where they can simulate OpenSSL.

1. Source: https://en.wikipedia.org/wiki/OpenSSL [↑](#footnote-ref-1)
2. https://www.handsonsecurity.net/ [↑](#footnote-ref-2)
3. https://www.handsonsecurity.net/ [↑](#footnote-ref-3)
4. https://www.handsonsecurity.net/ [↑](#footnote-ref-4)