CS2107 Self-Exploration Activity 3

Notes:

For Activity 3, you can perform the following:

1. To install **OpenSSL** on your Linux host so that you can run `openSSL` command.
2. To use the `openSSL` command to perform **encryptions using block ciphers**, such as AES, including with different **modes-of-operation**.
3. To check out **some Python library** that you can use to perform encryptions.
4. To get **pseudorandom numbers** from `/dev/random` and `/dev/urandom`.

Task 1: Installing OpenSSL on Your Linux Host

Last week, you were already asked to set up a Linux host. In this week, you are going to install **OpenSSL** (https://www.openssl.org/).

OpenSSL is a full-featured toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols, and is also a general-purpose cryptography library**.** The library additionally comes with the `openssl` ***command-line binary***, which allows you to handily perform a wide range of **cryptographic operations**.

To install the OpenSSL binary toolkit, install the OpenSSL package using the following command:

```
$ sudo apt-get install openssl
```

If needed, you may also refer to the following documentation on how you can install OpenSSL on Ubuntu: https://help.ubuntu.com/community/OpenSSL.

Once the package is installed, you can try running the following command to test the installed OpenSSL and check its **version**:

```
$ openssl version
```

To list all available OpenSSL **sub-commands**, you can run:

```
$ openssl help
```

Following this, run the following `openssl` command to benchmark your system's performance on all cryptographic algorithms:

```
$ openssl speed
```

To find out the details of various cryptographic-related `openssl` operations, you can read the following "*OpenSSL Command-Line HOWTO*": https://www.madboa.com/geek/openssl/. You can also refer to the following manual page of various `openssl`'s (sub) commands: https://www.openssl.org/docs/manmaster/man1/.

If you have any issues and need help with your Linux set-up and OpenSSL installation, your TAs will open an open consultation session after releasing Assignment 1 later. Please do your own self exploration first in setting up your Linux system and installing OpenSSL.

Task 2: Using the `openSSL` Command to Perform Encryptions

You can use the `openssl enc` command to perform **encryptions**. Do try using **AES-128** using both **ECB** (i.e. `-aes-128-ecb`) and **CBC** (i.e. `-aes-128-cbc`) modes-of-operation. Take note of the following necessary arguments to supply:

- `-in` <*file*>: input file
- `-out` <*file*>: output file

- `-e`: encrypt
- `-d`: decrypt
- `-K`: key (in hex)
- `-iv`: IV, if needed (in hex)

Task 3: Checking Out Python Library for Encryptions

Besides using a command like `openssl`, you can also utilize a **Python library** so that you can write your own script to perform encryptions. One available library is **PyCrypto (The Python Cryptography Toolkit)**, which is available from https://www.dlitz.net/software/pycrypto/.

To perform encryptions using the library, you can read: https://www.dlitz.net/software/pycrypto/api/current/. An AES encryption, for instance, can be done as follows:

```
from Crypto.Cipher import AES
from Crypto import Random

key = b'Sixteen byte key'
iv = Random.new().read(AES.block_size)
cipher = AES.new(key, AES.MODE_CFB, iv)
msg = iv + cipher.encrypt(b'Attack at dawn')
```

Do note about Python use of *bytes literals*, which are always prefixed with 'b' or 'B'. They produce an instance of the bytes type instead of the str type. You can read more about this at https://docs.python.org/3/reference/lexical_analysis.html#string-and-bytes-literals.

If necessary, you may also want to use Python's **`base64` module** to encode/decode/display a byte sequence. An example usage of the module is:

```
import base64
encoded = base64.b64encode(b'data to be encoded')
encoded

data = base64.b64decode(encoded)
data
```

Task 4: Getting Pseudorandom Numbers from /dev/random and /dev/urandom

In Linux and UNIX, `/dev/random` and `/dev/urandom` are special files (appearing as pseudo devices) that serve as **pseudorandom number generators**. `/dev/random` typically **blocks** if there is less entropy available than requested; `/dev/urandom` typically **never blocks**, even if the pseudorandom number generator seed was not fully initialized with entropy since boot. You can read about these two special files at: https://en.wikipedia.org/wiki//dev/random.

You can then use the following command to **get** 32 bytes of pseudo random numbers from `/dev/urandom`, and pipe the output to a **hexdump**:

```
$ head -c 32 /dev/urandom | xdd
```

If required, you can also pipe the output to the **base64 encoding** feature of `openssl` as follows:

```
$ head -c 32 /dev/urandom | openssl enc -base64
```

Do repeat the two commands above to get 32 bytes of pseudo random numbers from `/dev/random`, and notice the difference between the two special files.