Timothy Jelinek

CIS313, Cryptography
Module 3 Lab – AES Key Generation, Encryption, and Decryption

In this third module, we will explore symmetric encryption algorithms, specifically AES. There are several technical issues related to AES that need to be understood for it to be used safely. These issues include securely generating keys, storing keys, and encrypting or decrypting data. We will be using a few different tools including the Linux Bash Shell to get random data from the /dev/random device (a pseudorandom number generator), OpenSSL to generate keys using a password based key derivation function and OpenSSL to encrypt and decrypt a file.

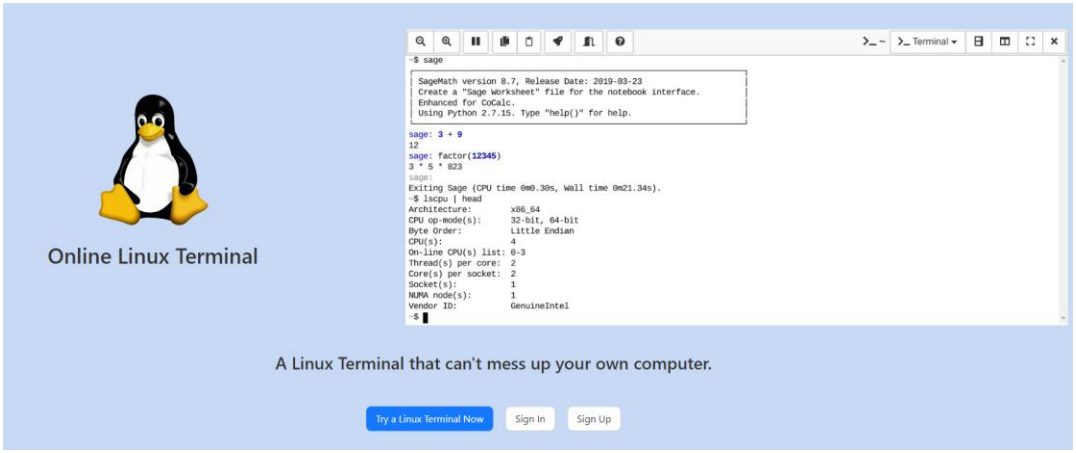
You will be required to submit the following graded items as part of this lab:

- A screenshot and text of your randomly generated AES key
- A screenshot and text of your randomly generated AES IV
- A screenshot and text of your password based key derivation function generated salt, key, and IV
- A screenshot of your encryption command and screenshot of the encrypted file
- A screenshot of your decryption command and screenshot of the decrypted file

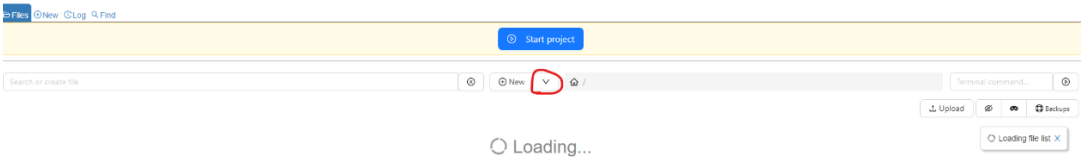
Lab 1 – Generating a key using random data

Generally, there are only two safe ways to generate keys for a symmetric encryption algorithm. The first of these is generating a key using a PRNG. The second of these is using a seed to generate a key. This can be done using a password with a salt or using a key derivation function with a seed from an asymmetric algorithm such as the Diffie-Hellman Key Exchange Protocol. In this lab we generate keys from a random source.

1. Open your web browser and surf to https://cocalc.com/doc/terminal.html. (Note: If you have access to a Linux OS you can also use that Linux terminal/command line to do this lab.)
2. Click on the Blue 'Try a Linux Terminal Now' button.



3. Click the blue “Use CoCalc Anonymously” button
4. Select ‘Projects’ from the menu bar under the ‘Signed in as’ title
5. Click the ‘Create Project’ button. You can leave the default project title as is.
6. Click the blue ‘Start Project’ button.
7. Click on the down arrow to the right of “New” and select Terminal



8. Accept defaults for name for Terminal file, then click ‘Create.’
9. At the ~$ prompt type the following command to generate a random key AES 256 bit key (32 bytes):

```
dd if=/dev/random bs=32 count=1 | xxd | cut -f 2-9 -d ' ' | tr
[:lower:] [:upper:] | tr '\n' ' ' | sed -e 's/ //g' -e 's/$/\n/'
```

The command has the following part:

| - this pipe operator is used to take the output from the previous command and direct it to the input of the next command.

dd if=/dev/random bs=32 count=1 – This command reads 32 bytes or 256 bits of data from the Linux PRNG.

- xxd – This command converts the random bytes to their hexadecimal value. Cryptographic values often include non-printable characters so things like keys, IVs, hashes, and ciphertext are often stored either as their hexadecimal values or encoded with Base64 so the text can be displayed on a screen.
- cut -f 2-9 -d ‘’ – This command takes the output from xxd and separates out the hexadecimal numbers from the rest of the output by taking only fields 2-9, separated by spaces.
- tr [:lower:] [:upper:] – This command converts all lowercase letters to uppercase.
- tr ‘\n’ ‘ ‘ – This command converts all newlines or carriage returns to spaces.
- sed -e ‘s/ //g’ -e s/$/\n/’ – This command removes all spaces and puts a newline or carriage return at the end of the key for nice formatting and display.

The animation linked to below shows this command being run:

Animated Example

Paste a screenshot of the command and the randomly generated key below:

Key:

5237E69C5DC634DD1939261B8D911DA5D9BE36F5FF5116016615165456141647

```
dd if=/dev/random bs=32 count=1 | xxd | cut -f 2-9 -d ' ' | tr [:lower:] [:upper:] | tr '\n' ' '
| sed -e 's/ //g' -e 's/$/\n/'
```

```
1+0 records in
1+0 records out
32 bytes copied, 2.8856e-05 s, 1.1 MB/s
5237E69C5DC634DD1939261B8D911DA5D9BE36F5FF5116016615165456141647
```

10. Next, we will generate a random IV. It is important that IVs be random for each message encrypted. The IV should be random, but unlike the key, it can be public and is often sent along with the ciphertext either appended to the beginning of the ciphertext or appended. The IV for AES 256 CBC is 128-bit (16 bytes) long, so there we only need to change the bs parameter from 32 to 16:

```
dd if=/dev/random bs=16 count=1 | xxd | cut -f 2-9 -d ' ' | tr
[:lower:] [:upper:] | tr '\n' ' ' | sed -e 's/ //g' -e 's/$/\n/'
```

Animated Example

Paste a screenshot of the random IV you generated below and the IV.

IV:

7A84F7F381AD779D262A3FBEDF5C0D66

Edit Copy Run Bash (Linux)

```
dd if=/dev/random bs=16 count=1 | xxd | cut -f 2-9 -d ' ' | tr [:lower:] [:upper:] | tr '\n' ' '
| sed -e 's/ //g' -e 's/$/\n/'
```

```
1+0 records in
1+0 records out
16 bytes copied, 3.6041e-05 s, 444 kB/s
7A84F7F381AD779D262A3FBEDF5C0D66
```

Lab 2 – Generate a salt, key, and IV using a password derivation function (PBKDF 2)

In this lab we use OpenSSL and a password derivation function to generate a key. You will notice that an additional random value is used. This salt is used to protect the password against precomputed table attacks. This lab can be done on a Linux machine. In your previously opened CoCalc Window, or on the command line of your Cybersecurity Desktop using VMWare Horizons.

1. If you are using the Cybersecurity Desktop, open the command line by typing cmd in the Type here to search box on the taskbar. The type the following command: set PATH=%PATH%;C:\Program Files\OpenSSL-Win64\bin

Animated Example

2. Next type the following command:

Note: This command works the same between Windows and Linux

CoCalc:

```
openssl enc -aes-256-cbc -salt -pbkdf2 -k YourPassword11 -P
```

Windows:

```
openssl enc -aes-256-cbc -salt -pbkdf2 -k YourPassword11 -P
```

This command does the following:

openssl – This is the name of the command we are using. It has many cryptographic functions.

enc – This tells OpenSSL that we are using the encryption function

-aes-256-cbc – This tells OpenSSL that we will be using the AES 256 symmetric algorithm in CBC mode

-salt – This tells OpenSSL that we want it to generate a random salt to store with the password

-pbkdf2 – This tells OpenSSL that we want it to use the PBKDF2 function to generate the key

-k – This is the password we are using as a seed to the PBKDF2 function to create a key. This same key will always be generated every time we use the same function but it is very hard to find the password if we only have the hexadecimal key.

-P – This tells OpenSSL not to encrypt but to only print out the salt, key, and IV values.

Animated Example

Note: If you want to generate the same key and IV you can include the -S option followed by the salt you previously used.

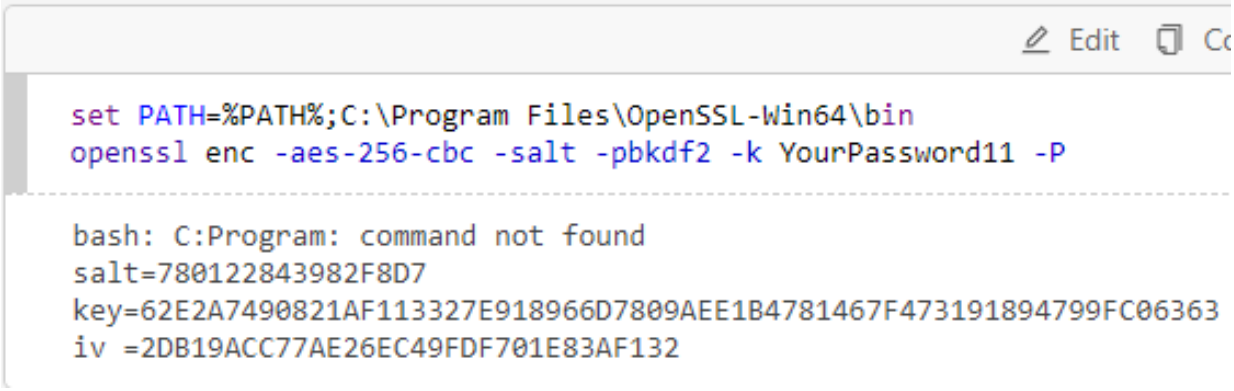
Record your salt, IV, and key values below

Salt: 780122843982F8D7

IV: 2DB19ACC77AE26EC49FDF701E83AF132

Key: 62E2A7490821AF113327E918966D7809AEE1B4781467F473191894799FC06363

Paste a screenshot of the output of your command below.



Lab 3 – Encrypting a File with OpenSSL and AES 256

Text to Encrypt:

Tomorrow, and tomorrow, and tomorrow,
Creeps in this petty pace from day to day,
To the last syllable of recorded time;
And all our yesterdays have lighted fools
The way to dusty death. Out, out, brief candle!
Life's but a walking shadow, a poor player,
That struts and frets his hour upon the stage,
And then is heard no more. It is a tale
Told by an idiot, full of sound and fury,
Signifying nothing.

- Create the text file we will be encrypting

If you are using CoCalc follow the instructions below to create the file to encrypt:

- Type vim tomorrow.txt at the command line ~$
- Type the letter i and look for the bottom of the command line to say --Insert--
- Right click the Text to Encrypt above and copy it
- Click on the CoCalc command line Window and press CTRL+v
- Press the ESC button and watch for the --Insert-- at the bottom to disappear
- Type :wq and then ENTER
- Type ls and ensure the file has been created and is named correctly

Animated Example

If you are using Windows follow the instructions below to create the file to encrypt:

- Right click the Desktop and then select New > Text Document
- Name the new text document tomorrow.txt

- Open the tomorrow.txt document
- Copy and paste the Test to Encrypt into the tomorrow.txt document
- Save and close the document
- If the command line windows isn't open type cmd in the Type Here to Search box on the taskbar and press ENTER
- Type cd Desktop
- Type dir to ensure the file has been correctly created

Animated Example

- Encrypt the file using the key and IV you have previously generated. The salt is used to ensure we get the same key using the same password and salt so it is not used in the encryption process. Run the command below substituting your key and iv in the appropriate place (Note: you can use the IV and Key from the first or second lab, either will work). This command will work the same on both Windows and Linux. You can substitute your key below and copy and paste the command into your command line.

  CoCalc:

  ```
  openssl enc -aes-256-cbc -e -in tomorrow.txt -out tomorrow.enc -K <your key> -iv <your iv>
  ```

  Windows:

  ```
  openssl enc -aes-256-cbc -e -in tomorrow.txt -out tomorrow.enc -K <your key> -iv <your iv>
  ```

  - openssl – This is the name of the command we are using. It has many cryptographic functions.
  - enc – This tells OpenSSL that we are using the encryption function
  - -aes-256-cbc – This tells OpenSSL that we will be using the AES 256 symmetric algorithm in CBC mode
  - -e - Specifies that we will be encrypting data
  - -in - Specifies the file we will be encrypting
  - -out - Specifies the destination for the encrypted output file
  - -K - Allows you to specify a hexadecimal key
  - -iv - Allows you to specify a hexadecimal iv

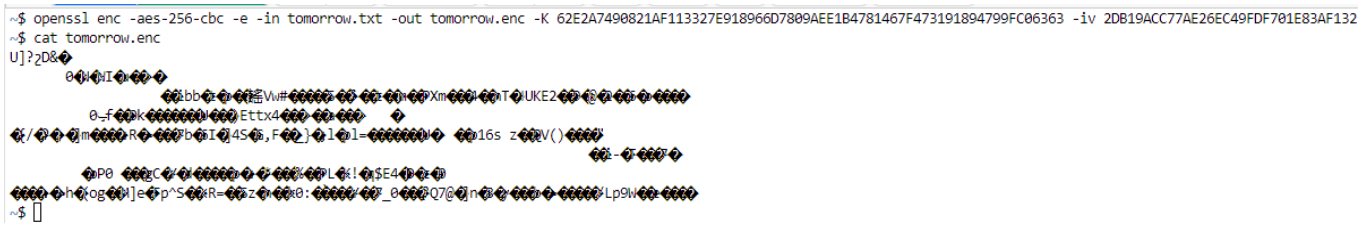
- CoCalc Example:

  Animated Example

- Windows Example

  Animated Example

Paste a screenshot of the successful encryption command and a screenshot of the gibberish inside the encrypted file



Lab 4 – Decrypting a File with OpenSSL and AES 256

Now we will decrypt the file we just encrypted using a similar command.

Note: Make sure you are using the exact same IV and Key that you used in the previous lab for encryption.

1. Decrypt the tomorrow.enc file using the following command:

CoCalc:

```
openssl enc -aes-256-cbc -d -in tomorrow.enc -out tomorrow.dec.txt -K
<your key> -iv <your iv>
```

Windows:

```
openssl enc -aes-256-cbc -d -in tomorrow.enc -out tomorrow.dec.txt -K
<your key> -iv <your iv>
```

- openssl – This is the name of the command we are using. It has many cryptographic functions.
- enc – This tells OpenSSL that we are using the encryption function
- -aes-256-cbc – This tells OpenSSL that we will be using the AES 256 symmetric algorithm in CBC mode
- -e - Specifies that we will be encrypting data
- -in - Specifies the file we will be encrypting
- -out - Specifies the destination for the encrypted output file
- -K - Allows you to specify a hexadecimal key
- -iv - Allows you to specify a hexadecimal iv

- CoCalc Example:

Animated Example

- Windows Example

Animated Example

Paste a screenshot of the successful decryption command and a screenshot of the inside of the decrypted file.

```
~$ openssl enc -aes-256-cbc -d -in tomorrow.enc -out tomorrow.dec.txt -K 62E2A7490821AF113327E918966D7809AEE1B4781467F47319189
4799FC06363 -iv 2DB19ACC77AE26EC49FDF701E83AF132
~$ cat tomorrow.dec.txt
Tomorrow, and tomorrow, and tomorrow,
Creeps in this petty pace from day to day,
To the last syllable of recorded time;
And all our yesterdays have lighted fools
The way to dusty death. Out, out, brief candle!
Life's but a walking shadow, a poor player,
That struts and frets his hour upon the stage,
And then is heard no more. It is a tale
Told by an idiot, full of sound and fury,
Signifying nothing.

~$
```