# Cryptography Homework 3a

## Required Reading

Cryptology 3 slides

## Optional Reading

Wikipedia has good articles on [stream](https://en.wikipedia.org/wiki/Stream_cipher) and [block](https://en.wikipedia.org/wiki/Block_cipher) ciphers, [block cipher modes of operation](https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation), [DES](https://en.wikipedia.org/wiki/Data_Encryption_Standard) and [AES](https://en.wikipedia.org/wiki/Advanced_Encryption_Standard).

## Install PyCryptodome

Please use the document “PyCryptodome Installation.docx” to install PyCryptodome on your computer or VM if you have not already done so.

## OS Hints

There are several environments you can use to do this lab.

In Windows:

* Command prompt (terminal) with Notepad++ [(download Notepad++)](https://notepad-plus-plus.org/downloads/)
* Command prompt plus Notepad
* IDLE (installed when you installed Python)
* Visual Studio Code ([download Code](https://code.visualstudio.com/docs/setup/linux))

For Linux:

* Terminal with text editor (gedit). Change preferences -> Editor to select “Insert spaces instead of tabs,” and Tab Width 4 spaces.
* Visual Studio Code ([download Code](https://code.visualstudio.com/Download))
* IDLE (sudo apt install idle3 for Ubuntu)

## Python and Strong Cryptography

There are several Python cryptography modules, but some of them are not suitable for use in production systems (see <https://theartofmachinery.com/2017/02/02/dont_use_pycrypto.html>.) A common module you will see in books and articles is PyCrypto. However, PyCrypto has exploitable bugs, and has not been supported since 2014. Do not use PyCrypto. A new project, PyCryptodome, is a fork of PyCrypto and is supported (<https://github.com/Legrandin/pycryptodome>.) However, even though PyCryptodome has all the encryption pieces we need, it still takes experience in cryptography to assemble those pieces in a secure manner. It is easy to create code based on standard functions like AES and accidentally implement flaws that severely weaken the cryptography.

Our goal in this lab is to look at the functions (primitives) that comprise current symmetric encryption, specifically AES. The Electronic Codebook mode of AES (AES-ECB) is not secure; we will use it to get started, but it should never be used in production.

# Symmetric Encryption with AES

## Overview

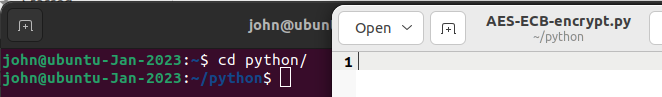
We will perform an AES encryption like that in slide 7 of the class notes. Instead of printing the results to the screen, we will save them to a file for practice.

We will create a key and plaintext, and then encrypt the plaintext with the simplest AES mode, Electronic Codebook (ECB). This is not secure, but it is an easy mode to use for our first attempt. The AES module will create ciphertext, which we will save to a file. Then we will open the file and decrypt it.

Note: This example will jump back and forth between Linux and Windows to demonstrate that it works in both places. We will use the terminal and a text editor, but you can use an IDE if you like.

## Encryption

Create a directory in a convenient location and use your text editor to create a file for our Python AES encryption. I am calling my file AES-ECB-encrypt.py. The terminal should be in the same directory as the file.  
Graphical user interface, application

Description automatically generated  
or  


Import the AES module from Crypto.Cipher. Note that Python is case sensitive, even on Windows. Run your script just to make sure Pycryptodome is installed. No errors (no output at all) is good.  
Graphical user interface, text, application, chat or text message

Description automatically generated

from Crypto.Cipher import AES

Create an AES object and give it the key you will use to encrypt. To start with, we will use ECB mode for simplicity and get more realistic later. Remember that the key must be exactly 128, 192, or 256 bits (16, 24, or 32 bytes) long. One byte is one ASCII character. If the length is not a multiple of 16 bytes, you will receive an error. Test your key to make sure it is 16 bytes (128 bits) long. This is an example of what happens when the key is too short.

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

Note: In Python, the most relevant error message is usually the one at the bottom.

You can check the length of your string at the interactive Python prompt ( >>> ) using len().  
len(b'mykey123')  
Text

Description automatically generated

Text

Description automatically generated

With standard encoding, one character is one byte, so that should help you to find a key that is 16 bytes long. Once you have a key that is 16 bytes long, use it to create an AES object. Use your own key!

aes\_obj = AES.new(b'This is the key!', AES.MODE\_ECB)

**Graphical user interface, text, application, Word, email

Description automatically generated**

Ah, no errors. Good.

**Important.** Note that there is a “b” at the beginning of the key.  
b'This is the key!'  
The “b” tells Python3 that this is a byte literal (like a byte array) and not a UTF-8 string object. That is necessary because the AES module wants bytes as input instead of strings.

Now we will put our plaintext message into the variable plaintext. The variable ciphertext will hold the encrypted version of the message. AES will only accept plaintext blocks of 128 bits or 16 bytes. You will have to add your own padding to make the length of your message a multiple of 16 bytes. Make your own plaintext!

Text

Description automatically generated

Once your plaintext is a multiple of 16 bytes long add it to your text editor and run it to check for errors.  
Graphical user interface, text, website

Description automatically generated

No errors, good.

Encrypt the plaintext with this line and print the result  
ciphertext = aes\_obj.encrypt(plaintext)  
print(ciphertext)

Graphical user interface, text, application

Description automatically generated

When you print a variable, Python will automatically convert it to a viewable string if it can. Since plaintext started as a string, it is readable. The ciphertext is binary data, so Python shows ASCII where it can and renders the rest in hex notation (i.e. \xe6). Not pretty, but it works.

Now that we have encrypted the message, let’s save it to a file. There are different formats for reading and writing to files. I prefer the with-open syntax. It automatically closes the file when the operation is complete, so I cannot forget to close the file.

Once the file is written, I can attempt to read it with the cat command. Note that the file is binary data. The print statement in the Python code shows the data as raw bytes. The cat command shows what happens when the OS tries to render binary data as ASCII. The xxd command shows binary data in hex format.

Graphical user interface, text

Description automatically generated

Here is the entire encryption script.

from Crypto.Cipher import AES

aes\_obj = AES.new(b'This is the key!', AES.MODE\_ECB)

plaintext = b'Attack at dawn, regardless of the weather. No excuses! 12345678'

ciphertext = aes\_obj.encrypt(plaintext)

print(ciphertext)

with open('aes-ecb.bin', 'wb') as myfile:

myfile.write(ciphertext)

## Decryption

The process is just about the reverse of what we did for encryption. I will put the decryption script in a new file called AES-ECB-decrypt.py

from Crypto.Cipher import AES  
Graphical user interface, application

Description automatically generated

Next, we need to read the binary file that holds our ciphertext. I’ll print the result just to be sure I read the correct file.

Graphical user interface, text, application, chat or text message

Description automatically generated

from Crypto.Cipher import AES

with open('aes-ecb.bin', 'rb') as myfile:

ciphertext = myfile.read()

print(ciphertext)

The rb in the open statement means read the file, and the file is binary.

Hopefully you encrypted your own message, so your ciphertext should be different from what is shown here.

Create an AES object using the same key that was used to encrypt the plaintext. Then call the decrypt method of the AES object and print the plaintext result.

Graphical user interface, text, chat or text message

Description automatically generated

Graphical user interface, text, application

Description automatically generated

from Crypto.Cipher import AES

with open('aes-ecb.bin', 'rb') as myfile:

ciphertext = myfile.read()

aes\_obj = AES.new(b'This is the key!', AES.MODE\_ECB)

plaintext = aes\_obj.decrypt(ciphertext)

print(plaintext)

Success!

# Hand in

Create your own plaintext and key. Remember that the plaintext must be an exact multiple of 16 characters (1 char = 1 byte) and the key must be exactly 16 characters.

Hand in a screenshot of your code encrypting and decrypting your message.