Lab - Symmetric Cryptography

João Paulo Barraca Vitor Cunha Pedro Cirne Catarina Silva

2024-10-10

In this guide, we will develop programs that use cryptographic methods, relying on the Python3 Cryptography module. The module can be installed using typical package management methods (e.g., `apt install python3-cryptography`) or the `pip3` tool (e.g., `pip3 install cryptography`).

It will help visualize and edit files in binary format. For that purpose, you may install `GHex` or `hexedit` from the repositories if you use Linux.

For educational purposes, we will be exploring the low-level interface of the `Python cryptography` library. Stay with the Fernet interface if you use this library in real-world applications.

As the documentation clearly states:

```
This is a "Hazardous Materials" module. You should ONLY use it if you're 100% absolutely
sure that you know what you're doing because this module is full of land mines, dragons,
and dinosaurs with laser guns.
```

1 Symmetric Cryptography

Symmetric cryptography is used by creating an object that represents a given cipher, with some parameters specifying the mode, as well as a key. The cipher object presents an `encryptor` method, that is applied (`update`) to the text in chunks (may require alignment with the cipher block size). After the text is ciphered, a `finalize` method may be used. Decryption is done similarly.

For more information, please check the documentation available at the Cryptography.io Hazmat section.

1.1 File encryption

Create a function to encrypt the contents of a file whose name should be provided by the user. The key should be random or supplied by the user. The user must provide (as program parameters or by request or any other suitable method): (i) the name of the file to encrypt, (ii) the name of the file to store the cryptogram, and (iii) the name of the encryption algorithm. Check the documentation and implement functions using multiple ciphers. We recommend `AES` and `ChaCha20`.

If you need to save multiple fields to the same file (e.g., salt, cryptogram), save these as fixed-length fields at the beginning of the file. A simpler alternative is to use Base64 to convert the objects to text (`base64` module) and then use a delimited such as `\n`.

Note 1: Consider that data may need to be encrypted in blocks, and the last block may require padding. Please check the next section.

Questions:

- What is the output of some encrypted data?
- Can you determine the structure of the text?
- What are the lengths of the text and the cryptogram?
- What is the impact of using different keys sizes (e.g., 16 vs 32 bytes)?

1.2 Padding

A block cipher requires input blocks of a fixed size that equals the algorithm block size. However, it is improbable that the file size to be encrypted is always multiple of the algorithm's block size to be used, i.e., frequently, the number of bytes that remain for the last block does not equal the block size. Extra bytes are added to have a block with the correct size and solve this problem. These extra bytes are then removed during the decryption operation.

There are several standards for padding. `PKCS#7` is one of them, please check the symmetric padding documentation. This exercise aims to demonstrate the existence of padding in an encryption operation and that it is according to the `PKCS#7` standard.

Using a binary file editor and the program you developed, idealize an experiment involving encryption and decryption operations with `ECB` cipher mode, `PKCS#5` padding, and an algorithm of your choice that shows the presence of padding and how `PKCS#7` padding is made.

Note: `PKCS#5` is very similar to `PKCS#7` (i.e., calculated the same way just with a fixed block size).

Questions:

- Is padding required for all cipher modes?
- What is the impact of padding when selecting the cryptographic primitive?

1.3 File decryption

Alter your program by adding a function that decrypts a user file. For this functionality, the user must provide the following (as program parameters or by request or any other suitable method): (i) the name of the file to decrypt and (ii) the name of the file to store the decryption result. The key must be requested.

Take care of removing the padding (if present!)

Questions:

- Is padding visible in the decrypted text?
- What happens if the cryptogram is truncated?
- What happens if the cryptogram lacks some bytes in the beginning?

1.4 Symmetric key generation

Most ciphers require keys to be of a certain fixed size. The key can be obtained from a random number generator, but human users would prefer using their password of any size. We can use a derivation function, such as the Password-Based Key Derivation Function 2 (PBKDF2), to generate a symmetric cipher key of the required size. The derivation algorithm will perform the hashing successively by first concatenating the salt and key, then iterating over the resulting digest for the specified number of iterations, and finally truncating the output to the required size.

Your task is to enable the user to provide a password that will be used to generate the symmetric keys:

- Construct a function that generates the secret key according to PBKDF2-HMAC-SHA256, i.e., with SHA256 as the hashing function used in the derivation.

2 Cipher modes

2.1 Initialization Vector

Some cipher modes use feedback to add more complexity to the cryptogram, namely `CBC`, `OFB`, and `CFB`. Feedback implies using an Initialization Vector (`IV`), which must be provided when initializing an object for one of such cipher modes. Note that the IV used to encrypt some data must also be provided when decrypting it. Therefore, the IV is usually sent in clear text.

Alter your program so the user, when requesting an encryption operation, also indicates the cipher mode to be used. This should be applied both to encryption and decryption.

Note that the `IV` is only used on cipher modes with feedback, which is not the case for `ECB`. Your program must be able to handle encryption using cipher modes both with and without feedback.

Hint: Use the `secrets` module to obtain securely random IVs.

Questions:

- What length should the `IV` be?
- What is the impact of repeating the `IV` while changing the Key for each cipher mode?
- What is the impact of repeating the Key while changing the `IV` for each cipher mode?

2.2 Patterns

In this exercise, we will analyze the impact of `ECB` and `CBC` cipher modes on reproducing patterns in the original document into the encrypted document. For this, we will encrypt an image in the bit map (BMP) file format, after which we will visualize the contents of the obtained encrypted file and compare it with the original image. For us to visualize the contents of the encrypted file, we must replace the first 54 bytes with the first 54 bytes from the original file (these bytes constitute the header of BMP formatted files, which is necessary so the file can be recognized as a BMP formatted file).

Use the program you developed in the previous sections to encrypt the file `p_ori.bmp` using the `ECB` cipher mode and a cryptographic algorithm of your choice. Using the `dd` application, copy the first 54 bytes from the original image file into the first 54 bytes of the encrypted file:

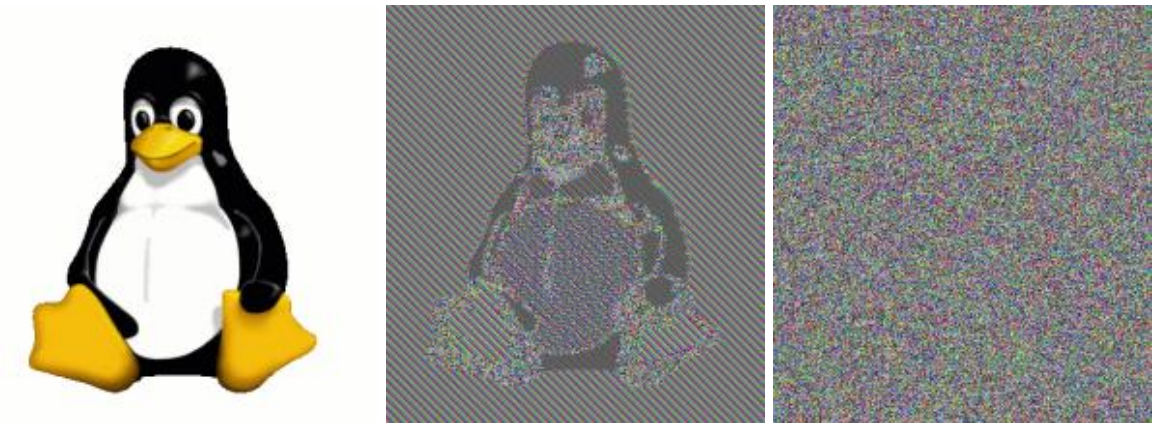
```
$ dd if=p_ori.bmp of=p_enc.bmp ibs=1 count=54 conv=notrunc
```

Using a program to visualize images, open the original and encrypted images and compare them. What do you observe?

Repeat all the above operations using the `CBC` cipher mode instead of the `ECB` cipher mode while still using the same algorithm. Then, compare the original image with the obtained encrypted image. What do you observe?

Repeat the experience using the same cipher modes but with a different algorithm.

A typical image (left) encrypted without feedback (center) and with feedback (right) will be:



(Images by Larry Ewing)

Questions:

- What do you conclude from the experiment?
- Can we have an insecure AES cryptogram?

2.3 Cryptogram corruption

In this exercise, we will analyze the impact in a decrypted text caused by errors in the cryptogram when using `ECB`, `CBC`, `OFB`, and `CFB` cipher modes.

Using the program developed in previous sections, encrypt the `p_ori.bmp` file using the `ECB` cipher mode and an algorithm of your choice. Using a binary file editor, change the value of a single bit in some byte of the encrypted image (notice that the first 54 bytes are not part of the image but rather part of the header of the file); for example, the byte in position `0x60`.

Decrypt the file with the corrupted bit using the same cipher mode and algorithm you used to encrypt. Using a binary file editor, open the original file and the decrypted file and compare them. What are your conclusions regarding the impact of the decrypted file produced by the corruption of a single bit in the cryptogram?

Repeat the experience for the remaining cipher modes and analyze the impact on the decrypted image produced by an error in a single bit of the cryptogram. Try to determine which cipher modes produce a bigger impact and which produce a smaller one in the decrypted file.

3 Additional Resources

While this guide primarily uses `Python`, the same processes can be executed in other languages. We include an example using `C` to observe what varies between your implementation and a similar implementation in another language. Check it here (Credits: Pedro Cirne).

4 (OPTIONAL) Human Readable Symmetric Encryption

Modern ciphers lie in their indistinguishability from random data by a computationally limited adversary. In the previous exercises, we used symmetric cryptography to produce indistinguishable results. In this exercise, we want to use ciphers to create a human "readable" text.

Write a program using AES encryption, for example, with CTR mode, and replace encrypted bytes with words from a dictionary, ensuring that your code encrypts and decrypts the input text while maintaining the ciphertext's human "readable" format.

Note you can use the markov-word-generator Python library to generate random credible words.

Experiment with different dictionaries and input strings to observe the effects on the ciphertext.