1. Caesar cipher

The earliest known use of a substitution cipher, and the simplest, was by Julius Caesar. The Caesar cipher involves replacing each letter of the alphabet with the letter standing three places further down the alphabet. For example,

plain: meet me after the toga party
cipher: PHHW PH DIWHU WKH WRJD SDUWB

The algorithm to encrypt the plaintext to the ciphertext can be expressed as follows. First, number the letters from a to z as 1 to 26. Then, for each plaintext letter with value p, and a key k ($k = 3$ in the example above), substitute it in the ciphertext with the letter corresponding to value c:

$$c = Enc(k, p) = (p + k) \, mod \, 26$$

And to decrypt letter c with key k:

$$p = Dec(k, c) = (c - k) mod \, 26$$

TODO:
Write two functions, one to encrypt and one to decrypt Caesar cipher, with a given key k.

TODO:
Decrypt the ciphertext below, with your decryption function. Find the key by yourself.
H tlzzhnl myvt aol ltwlyvy

2. Affine Caesar cipher

A generalization of the Caesar cipher, knows as the affine Caesar cipher, has the following form: For each plaintext letter p, substitute the ciphertext letter c:

$$c = Enc(k = [a, b], p) = (ap + b) \, mod \, 26$$

A basic requirement of any encryption algorithm is that it be one-to-one. That is, if $p \neq q$, then $Enc(k, p) \neq Enc(k, q)$. Otherwise, decryption is impossible. The affine Caesar cipher is not one-to-one for all values of a. For example, for a = 2 and b = 3, then $Enc([a, b], 0) = Enc([a, b], 13) = 3$.
Q1: Are there any limitations on the value of b?
Q2: Determine which values of a are not allowed. (Hint: You may do this by coding the encryption and decryption functions, then test all possible values.)
Q3: Provide a general statement of which values of a are and are not allowed. Justify your statement.

3. Symmetric Cipher

We will simulate the process of a symmetric encryption.
You are writing functions for key generation, encryption and decryption. You will manage the keys for users, and help them protect the confidentiality of their conversation.

You are using AES algorithm built in PyCryptodome. You should use the PyCryptodome documentation to find out how to use the built-in algorithms.

- Step 1: create keys.

For two users, Alice and Bob, create a pair of symmetric keys. At this stage, simply generate a strong random value of 128 bits.

- Step 2: encrypt a message.

Alice wants to send to Bob a message: 'Hello Bob, what time should we meet?'
TODO: write an encryption function and use the symmetric key generated in step 1, and return the ciphertext.

- Step 3: decrypt the ciphertext.

Bob wants to read the message.
TODO: write a decryption function to use the symmetric key to decrypt the ciphertext, and return the plaintext.

- Extra activities:

If you have time, write the code to assess the time it takes to finish sending and reading one message with symmetric encryption. This may help you compare the efficiency between different encryption methods.

You can try other encryption algorithms as well.

It may help if you do the courses on cryptohack. The introduction course covers some useful type conversion/transformation exercises, which may help you get familiar with the variable types used in PyCryptodome.

You can also try to test the randomness of the key you generated, with the NIST entropy assessment test tool. There is an open-source implementation of the standard. You may also try to create your own random number generator and compare the performance (i.e., entropy) between your creation and the one built in PyCryptodome.

4. Mathematical background for asymmetric cryptography

It will be useful if you read Chapter 2 (Introduction to Number Theory), Chapter 5 (Finite Fields) and Section 10.3 (Elliptic Curve Arithmetic) of the book Cryptography and Network Security (7th edition) before the week 5 lecture. We will cover the very important math properties used in the cryptographic algorithms we will introduce, but we will not have time to go into the details. You can get more familiar with the maths foundation from the two chapters. If you are more interested, you can also read the relevant chapters of the Foundations of Cryptography books, listed in the Module Brief file.

You should get familiar with:

- Fermat's theorem, Euler Totient, Euler's theorem
- Greatest common divisor (GCD), extended GCD
- Group, finite field, and arithmetic in them
- Elliptic curve arithmetic

You can practice solving these problems by finishing the Modular Arithmetic course on cryptohack.