Exercise

Cryptography

Download link:

https://drive.google.com/file/d/1Z9r_cA6Ly2ciIPF8U8hoFKdteKGvZPqK/view?usp=sharing

Question 1 (70 pt)

In this exercise you will focus on a simple stream cipher called **Repeated Key cipher**. A repeated key cipher works by XOR-ing the bytes of the plaintext with the bytes of the key similarly to a one-time pad:

- The first byte of the plaintext is XOR-ed with the first byte of the key
- The second byte of the plaintext is XOR-ed with the second byte of the key
- ...
- **If the plaintext is longer than the key, start using the key again from the beginning** (first it's first byte, then it's second byte, etc.)

Part A (10 pt)
Inside `q1.py`, implement the `encrypt` method in the `RepeatedKeyCipher` class.
Tip: To get the Unicode/ASCII code of a char, use `ord`. To convert a code back to a char, use `chr`.
We attached a tester to help you verify your implementation is correct. You can run your code on it with `python decrypt.py 1a`. **Your solution must pass this test**

Reach out an instructor after you finish this part.

Bonus A+ ε (5 pt)
Inside `q1.py`, implement the `encrypt_oneliner` method in the `RepeatedKeyCipher` class, using a single line of code only. **Skip this part for now if you are unsure how to do this.**

Part B (5 pt)
Inside `q1.py`, implement the `decrypt` method in the `RepeatedKeyCipher` class. **Add the smallest amount of code possible** on what you already implemented in the previous part.

Part C (20 pt)

Inside `q1.py`, implement the `plaintext_score` method in the `BreakerAssistant` class. The method should take a string (of any length) and return a numeric score such that a string containing a plausible text in English will receive (with a high probability) a higher score than a random string of the same length.

- For example, it should give this string `"I am a sentence!"` a higher score than `"\xc6\xc9u\xd0v\x14\xe2\xcf\xc5\xf5\x1eZ\x10Yd\xd3"`[1].
- As you will be using this method in the following parts, **we recommend implementing a simple method now and improving it if necessary as you proceed in the next parts**.

Reach out an instructor after you finish this part.

Part D (10 pt)

Inside `q1.py`, implement the `brute_force` method in the `BreakerAssistant` class. The function receives a key length, attempts all possible keys of the given length, evaluates the plaintext possibilities using the `plaintext_score` function you have written, and **returns the "correct"[2] plaintext**.

In `q1d.cipher` we attached a sample encrypted text you can try breaking (it has a short key - 2 bytes long). You can run your code on it with `python decrypt.py 1d`. **Your solution must decrypt the attached ciphertext to the correct plaintext[3]**.

Reach out an instructor after you finish this part.

Part E (25 pt)

The problem with the brute force method is, well, that it's brute force. This means we try many attempts without much thought. Trying to a break a key longer than a few bytes is going to be impractical (as we need $2^{8\cdot(number\ of\ bytes)}$ attempts).

In this part we urge you to try and break they key in a different way - find a way which is much faster, and for example, can break a key of length 10 or higher. For this part, **you may assume the ciphertext is long**.

Inside `q1.py`, implement the `smarter_break` method in the `BreakerAssistant` class using the technique you devised. As before, the method should receive the key length and return the "correct" plaintext.

---

[1] The notation `\x<2 hex digits>` is used in Python to denote a char by it's hexadecimal code. We mainly use this where writing the char directly would result in binary gibberish or chars we can't read.
[2] While you can't know what is the correct plaintext, we hope that the plaintext with the highest score is indeed the correct one.
[3] Some texts have typos in them, that's a mistake in the source we quoted and not necessarily a problem with your code. If everything looks like English, perhaps with typos, it's OK.

In `q1e.cipher` we attached a sample encrypted text you can try breaking (it has a looooong key - 16 bytes long). You can run your code on it with `python decrypt.py 1e`. **Your solution must work on this text**.

Reach out an instructor after you finish this part.