Lab 3: Code Cracking

Sending secret codes that adversaries cannot read has been a cornerstone of communication since forever. The science of protecting information through codes and ciphers is called **Cryptography** and is one of the largest areas of Computer Science today. The earliest known uses of cryptography dates back to around 2000BC! One of the most famous ciphers of antiquity is called **"The Caesar Cipher"** named after Julius Caesar who used it to encode battle strategies.

Today, Cryptography is the cornerstone of internet communication and commerce. We rely on encrypted messages to send payments, passwords, and data. We use the same principles for storing user information and even checking time.

In this lab we will learn some basics of cryptography and implement several famous ciphers. At the end of the course we may spend time diving deeper into a few more Cryptographic concepts..

In this lab, we will develop:

- Implement an encoding/decoding suite for the Caesar Cipher
- Import and interpret data to create a heuristic
- Demonstrate how to break the Cipher
- Learn and implement a more complex cipher
- Learn and implement a truly secure encryption scheme
- Explore why it is secure

Outcomes:

We will explore and understand both historic and modern encryption methods and see how they work. We will try to attack encryption schemes and see why some of the complexity and randomness is necessary in modern cryptography. To do all of this we will practice working with data files and writing clean and reusable code.

Understand the skeleton code

We have provided several files and functions to help you get started.

cipher.py is the skeleton code for the assignment so you should develop your code on this. Do not change the python file name as the autograder is expecting this specific file name.

Every function you will need is provided with the method signature we will use to test the different parts of the assignment (feel free to add more, but the autograder will be calling these methods).

You are given a test suite at the bottom of the file that shows you how to run the different parts. No changes need to be made here, but you can and should add more testing and play around with it. **Make sure to comment out this section or delete it before turning in your code to the autograder**.

The **read_message()** function can be used to read message.txt which is a good message to use for testing. It is called in the test cases at the bottom.

The four binary functions provided can be used for Phase 4 of the lab. They can be used to convert between related types.

To clarify the different types, ascii/string is a normal message, binary string is a string consisting of 1s and 0s. Binary is an integer that has the same value as the binary representation of a string. You should use these methods as described in their comments.

You are given 3 other files that are useful:

- **message.txt**: has a message you can use for testing
- **letter_frequencies.csv:** the english alphabet and the number of occurrences of each letter in online books
- **wordlist.txt**: the 10,000 most common English words from MIT
  - This file is for the extra credit portion

Make sure you add these files to your project directory in PyCharm. If you forget to do this, your code will fail as it'll be calling a file that's not in the same folder.

Part 0: Design (5 points)

In this lab, we will be building, using, and breaking ciphers. Before we die into building the lab, we must understand how everything works.

Draw or illustrate how you would go about encrypting and decrypting a message. Make sure to modularize your thinking and show the different inputs and outputs for each part.

Add a diagram showing the modularized logic for each cipher!

Part 1: Caesar Cipher (30 points)

The Caesar Cipher is a classic fun cipher that some children use to pass notes. The way the cipher works is by rotating every letter by some rotation amount.
For example, if the rotation amount is 2 then "a" becomes "c" and "b" becomes "d" etc.
Once every letter is rotated the "cipher text" or encrypted message can be sent. The receiver of the cipher text must then decrypt the message by using the same rotation amount.
For people of antiquity, most of whom were illiterate and untrained in math, this was sufficient for sending communications.

For another explanation of the Caesar Cipher (and how we will break it) go to the following link: Caesar Cipher

1.1 Init and running the first test

Start by examining the initialization function. I have provided the necessary variables and initialized self.letter_dict. Your task for this section is to write self.inverse_dict which inverses the letter dictionary that I provided (the number value as keys and the letter as values).

Run the file and examine the Part 1 test cases.

1.2 Encode Caesar and Rotate Letter

encode_caesar() is fairly simple. We will rotate every letter of the input message by the given rotation amount. Use rotate_letter as a helper method.

rotate_letter() will use the dictionaries made in part 1 (inside init) to map letters to numbers, add the rotation amount, and map numbers back to letters (remember to wrap around).

When you run the program with the given test cases, the first 10 characters should be: mnxeufqrxe...

1.3 Decode Caesar

Decoding the Caesar cipher works almost identically to encoding, except you rotate backwards. Make sure to use the rotate_letter() helper function to prevent code duplication.

When you finish this function, it should give you the original message.

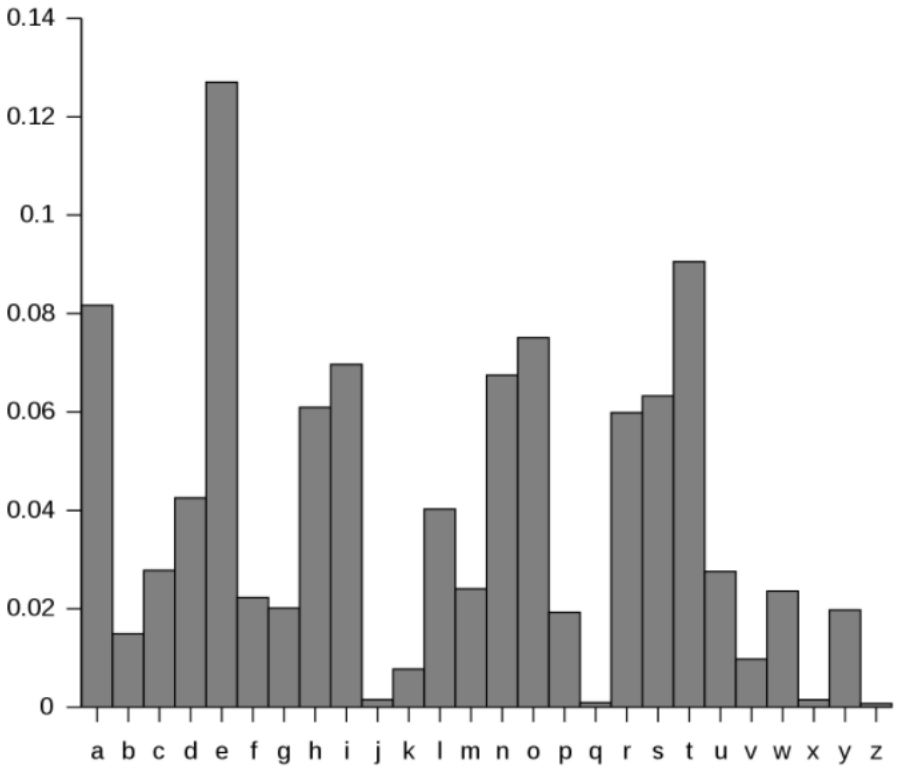
Part 2: Hacking Caesar (20 points)

The Caesar cipher is fundamentally flawed, since we rotate each letter by the same amount and we only have 26 distinct rotations, there are only 26 cipher texts that can exist for a given message.

We will attack the Caesar cipher by using a brute force attack and test all of these. One of these must produce the decrypted message; to choose the correct one we will employ a simple yet effective cryptographic tool - frequency analysis.
This attack works by examining what we expect the message to look like. We can look at the most

common letters of the potential messages and see how well it matches with English. In English different characters appear very differently, for example, the letter “e” appears far more often than “z”.

Below are the relative frequencies of letters in the English language.



If the message was random gibberish, then frequency analysis would never work, but in reality messages contain information (in English) so by seeing how the different messages compare to the “English distribution” we can make a good guess. The longer the message the more effective frequency analysis is.

For another explanation of the Caesar Cipher (and how we will break it) go to the following link: Caesar Cipher

2.1 Reading Letter Frequency

We have provided a frequency analysis of English letters that was done by Google (and made a few minor changes) in letter_frequencies.csv. In the file, each row is the character and then the number of billions of occurrences of that letter in Google’s online book archive.

You should start by adding a letter heuristics dictionary to the init method.
You can then read the csv and store each character/score pair as a key/value pair in the dictionary (remember that there is a header row in the csv). You can either score using the number of occurrences or change it to your own heuristic.

2.2 Scoring a String

Once you have the letter heuristics ready, you can quickly score a given string by adding up the scores for each letter.

2.3 Cracking Caesar

Now you can crack a Caesar Cipher ciphertext. You will be scoring all the different variations of the text with every letter rotation. The string with the highest score is most likely to be in English and to be the decoded message.

Part 3: Vigenere Cipher (20 points)

The Vigenere Cipher is an example of a more complex rotating cipher (like the Enigma Machine).

The biggest weakness of substitution ciphers (like the Caesar cipher) is the relatively small number of combinations and ease of frequency analysis which makes computers exceptionally good at breaking them.

The Vigenere Cipher solves this by using a changing rotation parameter.

To do this, instead of a rotation amount, it takes a password string; the password string is replicated until it is the same length or longer than the message and each letter (when converted to a numerical value) is used as a rotation amount.

What that means is if we have a string like “dog”, every first letter will be rotated by the number value of ’d’ (rotated by 3), every second letter will be rotated by ’o’ (rotated by 14), and every third letter will be rotated by ’g’ (rotated by 6).

This results in a changing relationship between input and output letters and is virtually impossible to break by hand.

For another explanation of the Vigenere Cipher go to the following link (remember that we are also encrypting spaces to make it even more secure): Vigenere Cipher

3.1 Encoding and reusing code

Encoding will work similar to the Caesar Cipher, but now instead of having a single rotation amount you will have to calculate it for each letter. The process is simple, you will use the first letter of the key as the rotation amount for the first letter of the message. All subsequent letters will act the same. Once you run out of letters in the key, start back at the beginning of it.

The first word of the output for the encoded message should be: kwyccgo

3.2 Decoding

Much like with the Caesar Cipher, the Vigenere Cipher decoding procedure is just encoding but with rotations in reverse.

Part 4: One-Time-Pad (OTP) (20 points)

More complex ciphers were still insufficient to defeat computers because they were predictable and brute-forcible.

As you can see with the Vigenere Cipher, short or simple keys quickly degrade the quality of the encryption and relying on people to create passwords is problematic.

Computer Scientists and mathematicians came up with a principle that any encryption scheme that relies on adversaries not knowing how it works (like the case of The Engima Machine) is not secure because it is too strong of an assumption, at some point someone will either leak the method or figure it out. This is known as Kerckhoffs's principle.

The One-Time-Pad is the most fundamental truly secure encryption scheme and is extensively used in the modern world. It works on the bit level.

By viewing the message as a string of bits and randomly either flipping or leaving each one, the resulting cipher_text is equally likely to be any string of bits and cannot be distinguished from random (proving this is one of the first lessons of a Cryptography class). OTP is secure because it generates a random key that completely masks the message. Be aware that using the same key more than once can start making it vulnerable to frequency analysis; think about why.

For another explanation of the One Time Pad go to the following link: OTP

4.1 Encoding and understanding helpers

Our encoding scheme will take the message and start by converting it to a binary numeric (an integer) and binary string (string). We will then create another binary string of random bits that is the same length, the key. We will then convert the key to a binary numeric. To create the cipher text we will XOR the numeric key and numeric message to produce a random number. We then have to return both the key and cipher text; realize that we need the key because otherwise the cipher text is unreadable.

4.2 Decoding

Decoding is much easier. We must just XOR the key and cipher text to undo the first XOR (this is a very important concept). Now we have the binary numeric from Part 1 and simply need to convert it back to a string.
You can try to reuse the attacks on the other encryption schemes or try your own to explore why it is secure.

Extra Credit

You have four opportunities for extra credit. You can choose to do all opportunities or pick a subset, your choice.

1. **Breaking Vignere**: Write a function to attack the Vigenere Cipher (15 points)
    - Assume that people will choose a word as the password.
    - We have provided a txt file with the 10,000 most common words. You can use a similar approaches breaking a Caesar cipher, going through all the possible words as password. This approach is also widely used for brute-forcing passwords.
    - Return the decoded message and also the password used for the encryption. Use the given test cases to see if you can decode them.
2. **Do some external research and write a paragraph explaining why and how OTP works and briefly illustrate (5-10 sentences) why it's secure (3 points)**
3. **Do some external research and write a paragraph explaining why reusing keys makes OTP insecure (2 points)**
4. **Write a function to attack the OTP**
   Take only the cipher text and retrieve the original message
    - If you attempt a non-trival attack (like how we attacked Caesar Cipher) (5 points)
    - If you break OTP (invitation to teach the class next semester)

Submission Instructions

Submit 2 files:
cipher.py
README.pdf with any notes on the lab and your explanation from Part 0: Design

Submit both parts on Gradescope

At the top of the README and python file, include the following information:

- Your name
- The name of any classmates you discussed the assignment with, or the words "no collaborators"
- A list of sources you used (textbooks, wikipedia, research papers, etc.) to solve the assignment, or the words "no sources"
- Whether or not you're using the extension

Grading Methodology

Each part of the lab will be graded by an automatic grading program. It will use the method signatures specified in this lab and will use multiple test cases. Each phase of the lab will be graded independently.

The remaining 10 points are allocated for proper comments and styling:

- Descriptive variable names
- Comments for what functions do
- Comments for complex parts of code
- Proper naming conventions for any variables, functions, and classes