Project 1: Encryption using the One-Time Pad

Aaron Huang | Last Updated: September 15, 2024

Contents

1. Introduction

This document describes the implementation of a one-time-pad for encrypting and decrypting text files. Basic understanding about Unicode characters, UTF-8 formatting, and binary strings will be covered in order to establish a framework for the encryption and decryption processes. Afterwards, the XOR operation and the inner functionalities and systems of the one-time-pad will be explained. Finally, a discussion of the one-time-pad's limitations will be considered.

2. Preliminaries

2.1. Basic Definitions

Byte: a collection of 8 bits (0's or 1's); a fundamental building block for larger data structures

Encryption/Decryption: encryption is the converting of text into an unreadable format (to prevent unauthorized access of the text); decryption is the reverse process.

Encoding: encoding transforms data from one format to another for purposes such as storage and compatibility.

Plaintext: readable text that is unencrypted

Ciphertext: unreadable text that is encrypted

Key: a mechanism that transforms plaintext into ciphertext and vice versa

2.2. Unicode and UTF-8

Unicode is a character encoding standard that assigns a unique code to every character in every language, ensuring a consistent representation of characters across different systems.

UTF-8 is a widely used encoding scheme for Unicode. It uses 1 to 4 bytes to represent each character (where common characters such as 'a' use just 1 byte, while less common characters such as '' use 2, 3, or 4 bytes).

2.3. Text to Binary (Binary Strings)

Using the embedded python functions of .encode("utf-8") and .decode("utf-8"), we may easily convert all Unicode characters into their respective UTF-8 encoding. This UTF-8 encoding is stored as a binary string, which is simply a sequence of bits.

3. Full Encryption and Decryption System Breakdown

3.1. XOR and One-Time-Pad

The XOR function (^) takes two integers, converts them into binary, and using the properties of Table 1, adds those two integers together, returning that value. For example, 87_{10} ^ $88_{10} = 15_{10}$. Since 87_{10} is 1010111_2 and 88_{10} is 1011000_2, we may XOR bit by bit and obtain the result 0001111_2, or 15_{10}.

Table 1.

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

A One-Time-Pad uses the XOR function to encrypt and decrypt messages. The fact that if A ^ B = C, then C ^ B = A, (if plaintext ^ key = ciphertext, then ciphertext ^ key = plaintext) reveals the basic process behind the encryption and decryption of a one-time-pad. First, using UTF-8 encoding, we may encode the plaintext. Then, we may also encode, using UTF-8, the key. Key length must match the message length to XOR however, but this phenomenon will be explored in the section below. XORing the plaintext and the key will result in ciphertext, and returning to the plaintext only requires the XORing of the ciphertext and the key. This covers the essential functionality of the one-time-pad's encryption and decryption.

3.2. Extending the Key

To XOR two integers together, they must have the same length. Therefore, if the provided key is too long or too short, we must write some code that accounts for that issue. However, this is easily accomplished using a conditional. If the length of the key is longer than the length of the message, truncate the end of the key until the lengths match. Otherwise, append the key to itself.

3.3. Other Considerations

3.3.1. File I/O

File Input/Output allows for easy encryption and decryption for the user. Therefore, it is vital to implement the “open()” function. It may be noted that the modes of ‘wb’ and ‘rb’, or write binary and read binary, respectively, should be considered when storing an encrypted message into a separate file and reading an encrypted binary message. In addition, a try/except method should be used to cover for an user-based error when typing in the name of a file.

3.3.2 Encrypted Text Format

The format of binary may not be so reasonable for the user. Instead, we may convert the ciphertext into base64, allowing for a more “reasonable format” of the text. This requires an extra step of encoding to base64, but uses minimal additional storage/memory.

4. Limitations of the One-Time-Pad

Simply stated, the key represents the major issue with the one-time-pad, because the length of the key must equal the length of the message. Such restriction puts a high risk on the safe-keeping of the key, which is often difficult to accomplish. Moreover, keys must be truly random and cannot be reused for multiple messages. Reusing a key even once or not using a random key compromises the security, making it easier for attackers to deduce the plaintext.

5. Acknowledgements

Thank you Dr. Zufelt for the help in the actual creation of the Python code that mimics the encryption and decryption processes of a one-time-pad.

6. References

Check out the GitHub repository for the actual Python code.