# Password Manager Master

An Application of Number Theory

Ruotian Zhang

Zhiliang Yu

## Introduction

a context that will help to understand your chosen topic, with a clearly- defined question inspired by your issue with a rationale that explains what you are doing and why.

### 1.1. What is cryptography

Seems far but actually everywhere in our life

### 1.2. The Question of “How to manage passwords better”

For modern individuals navigating the digital era, passwords are an inescapable part of daily life, encompassing everything from email and online shopping to bank accounts, educational platforms, and social media sites like Facebook and Instagram. The use of passwords not only secures our personal digital space, but also poses challenges such as the risk of privacy breaches and the inconvenience of forgotten passwords.

For instance, we may encounter situations such as data loss, due to a browser crash or something we used to rely on. Many of us tend to rely on our browsers, like Chrome, to manage our passwords for convenience. However, such an unexpected crash can result in significant time and effort spent in recalling or resetting various passwords for email, social media, and other online platforms. Such experiences highlight the potential risks of depending solely on something like browser-based password storage and underscore the importance of considering more secure and stable alternatives, like local password management solutions.

In this case, a question arises: How can we develop a more reliable and secure method for managing and storing passwords that reduces the risks associated with browser-based systems?

Our aim is to help people ….

### 1.3. Rationale

Our project aims to develop a local password manager that effectively balances security and convenience. Therefore, we decided to develop a local password manager, a tool designed to securely store (super-duper highly encrypted) and manage your passwords directly on your computer. Unlike cloud-based solutions, this manager will keep all data locally, encrypting it to ensure maximum security. The encryption is solid, meaning that even if someone were to gain physical access to your device, they wouldn't be able to decipher your passwords. Additionally, the manager is designed to be user-friendly, making it easy for anyone to securely store and retrieve their password

## 2. Analysis

a description of the methods you used to gather your data and/or solve your problem. What did you do and why? Show clear steps throughout every step of your analysis, referencing specific topics/modules covered in the CS 5002 course. If you produced a Python program, make sure you submit the *.py* file as an Appendix to your .pdf report.

### 2.1. Analysis Process

In this section, we provide an overview of the methodologies employed to address our password management challenge. Our approach involved the creation of a password manager using Python, drawing extensively from the concepts covered in the CS 5002 course, particularly in the domains of number theory and cryptography. We utilized Python programming to implement the functionalities required for effective password management. The following steps outline our analysis process:

1. **Exploration of Encoding Methods**

We initiated the analysis by exploring fundamental encoding methods, including the Caesar cipher, Vigenère cipher, RSA, and AES. Each method was chosen based on its relevance to password management and its alignment with the principles covered in the CS 5002 course.

1. **Implementation of Different Versions**

Subsequently, we applied those methods above to implement different versions of the password manager, each as a distinct encoding method. This step allowed us to compare the strengths and weaknesses of each approach.

1. **Identification of Challenges**

Throughout the analysis, we encountered challenges specific to each encoding method. These challenges provided valuable insights into the practical aspects and limitations of each technique.

1. **Evaluation of Difficulty and Comparison of Versions**

We summarized the difficulty levels associated with implementing each method and provided a concise overview of different strengths and weaknesses of them. This involved assessing factors such as security, ease of implementation, and adaptability to different requirements.

By following these steps, we not only addressed our password management problem but also gained a comprehensive understanding of the theories and practices related to cryptographic techniques, as taught in the CS 5002 course. This integration of theoretical knowledge and practical application has been instrumental in achieving our project objectives.

### 2.2. Methodology Overview

basically概念+公式，优缺点

#### 2.2.1. 凯撒密码

#### 2.2.2. Vigenère 密码

#### 2.2.3. RSA

#### 2.2.4. AES

我们应用上述不同的方法编写不同版本，以适应不同需求

### 2.3. Step-by-Step Analysis

Within this section, we delve into the practical application of three distinct encoding methods, seeking a profound understanding through their real-world implementation in addressing our password management challenge. Through hands-on exploration and problem-solving, we aim to gain deeper insights into the intricacies of each encoding approach and assess their effectiveness within the context of our project.

#### 2.3.1. Implementing the Vigenère Cipher

In our initial testing phase, we began by implementing a simple version of the Vigenère Cipher as one of the encoding methods for our password manager.

To initiate the Vigenère Cipher implementation, we defined a basic key, "Hu5ky" (from Husky). Our first step involved handling the retrieval of an existing password from the ".pwd" file, establishing a foundation for subsequent encoding and decoding processes.

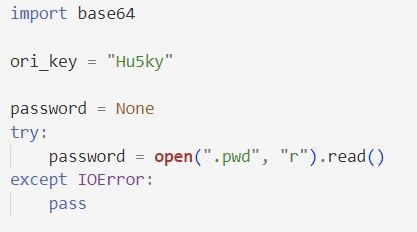


Figure 1 Vigenère Cipher (1)

We then crafted a preliminary encoding function, *vigenere\_encode(text)*, which aimed to encode text using the Vigenère Cipher. The algorithm cycled through characters of both the input text and the key to produce the encoded output.

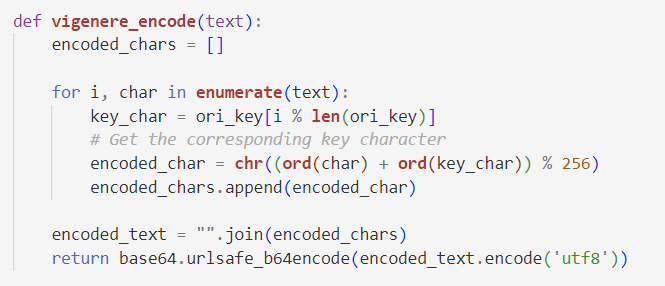


Figure 2 Vigenère Cipher (2)

Finally, we developed the *vigenere\_decode(encoded\_text)* function to decode the encoded text back to its original form using the Vigenère Cipher.

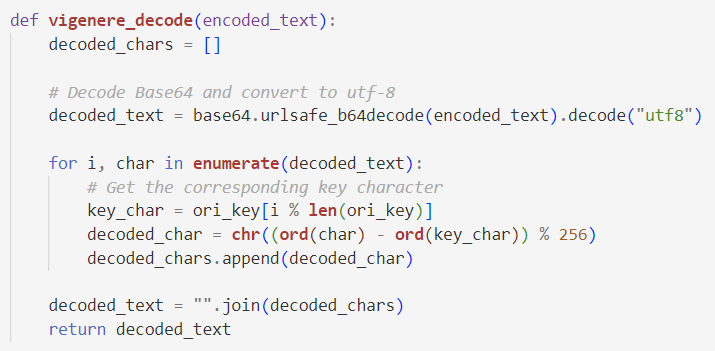


Figure 3 Vigenère Cipher (3)

Several challenges emerged during the implementation. Efficiently cycling through characters of the key during encoding and decoding required careful consideration. We addressed this challenge by utilizing the modulo operator to ensure proper cycling. Additionally, managing the retrieval of an existing password from the ".pwd" file presented complexities, particularly when the file might not exist. We implemented exception handling to gracefully address this scenario.

Beyond the realm of encryption and decryption, we faced additional challenges related to character format types, specifically in the conversion between UTF-8 and Unicode. This aspect became crucial when handling different character sets within our application. Additionally, the usage of *base64.urlsafe\_b64encode* introduced complexities. Its specific functionality involves converting input data to a URL-safe, base64-encoded string. These intricacies required a nuanced approach to ensure seamless integration and compatibility within our password manager.

The initial test of the Vigenère Cipher implementation in our password manager was successful. Both the encoding and decoding functions demonstrated effectiveness, providing a secure and practical method for handling passwords within our application.

However, it's essential to note that this encryption method, while functional, proved to be relatively simple. Its simplicity raises concern about susceptibility to potential decryption attempts. Acknowledging this limitation, we recognized the need for a more robust and sophisticated encryption approach. Consequently, we embarked on further exploration and experimentation with more complex encoding methods to enhance the security of our password manager.

#### 2.3.2. Implementing the RSA Algorithm

In this section, we ventured into implementing the RSA Cipher as part of our password manager's encryption methods. The RSA algorithm, known for its robust security through public-key cryptography, offered a sophisticated alternative to our previous attempts.

To better understand the idea of RSA Algorithm, we wrote a simple demo by python first. We originally intended to write an algorithm by hand and use it for our program, while we realized that it would not be powerful enough after we run our demo. We need to give big enough *p* and *q* manually to handle complicated messages. But during this process, we did understand how RSA works step by step.

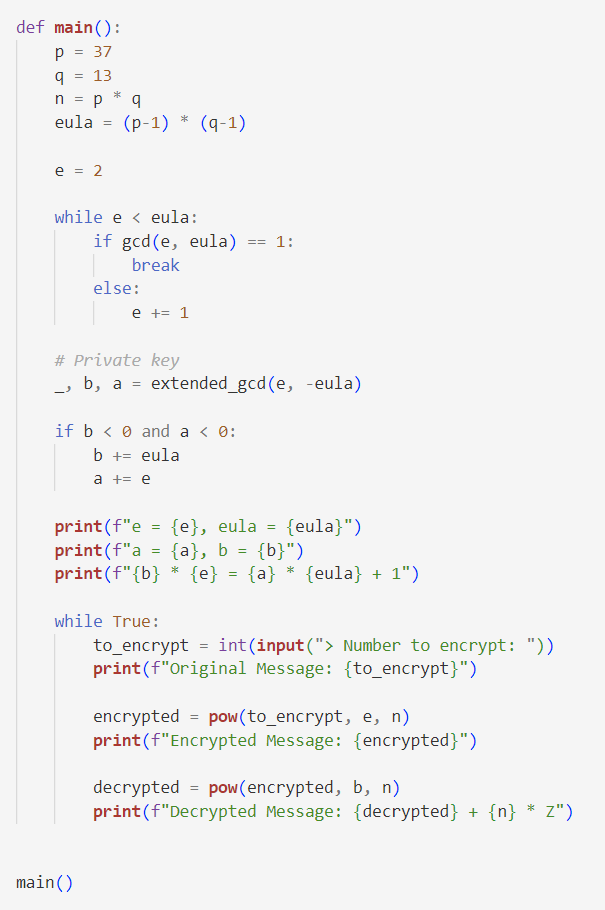


Figure 4 RSA Algorithm Demo

Therefore, our approach changed to integrate the RSA algorithm using the Crypto library. We generated RSA keys, saved them to files, and utilized these keys for encryption and decryption. The process aimed to enhance the security of our password manager by employing public-key cryptography.

If the password has not been set, we generated a new pair of RSA keys (2048 bits) – a private key and its corresponding public key. The keys were saved to files named "*private.pem*" and "*public.pem*" for later use.

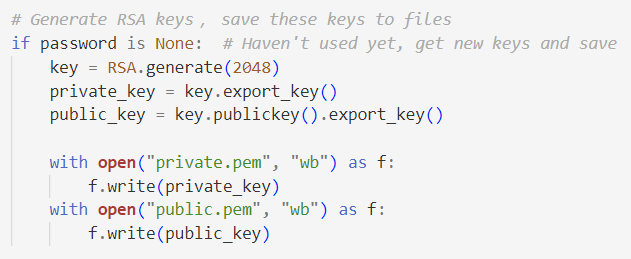


Figure 5 RSA Cipher (1)

To encrypt a message, the public key from "private.pem" was loaded. The RSA algorithm using PKCS1\_OAEP was applied to encrypt the message, and the result was Base64 encoded for secure transmission.

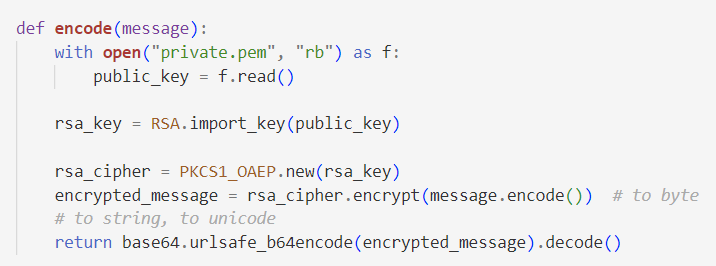


Figure 6 RSA Cipher (2)

To decrypt an encrypted message, the private key from "private.pem" was loaded. The Base64 encoded encrypted message was decoded, and the RSA algorithm was used for decryption.

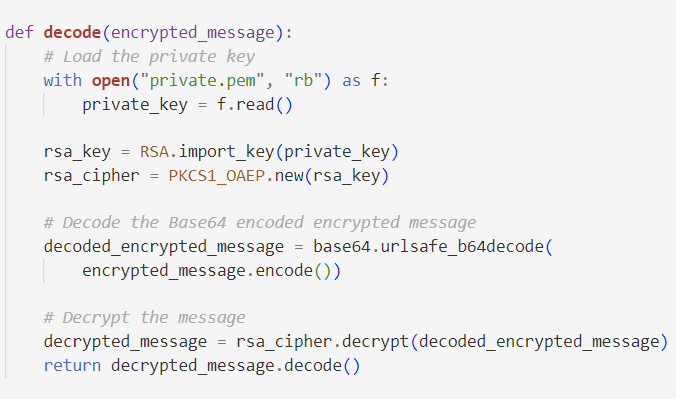


Figure 7 RSA Cipher (3)

While implementing the RSA Cipher, we faced several challenges that required careful consideration and solutions.

The first is about key management. To ensure consistent decryption, we needed to use the same set of public and private keys. However, generating random keys for each encryption posed difficulties. To solve this, we opted to store the keys in files ("*private.pem*" and "*public.pem*") to maintain consistency between encryption and decryption processes. But in the meantime, storing keys in files introduced a potential security risk, as unauthorized access to these files could compromise the integrity of the encryption.

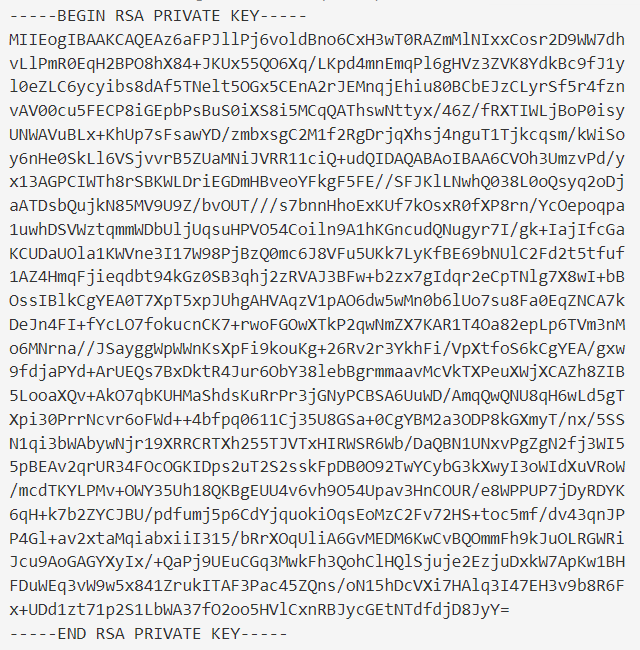


Figure 8 Private.pem

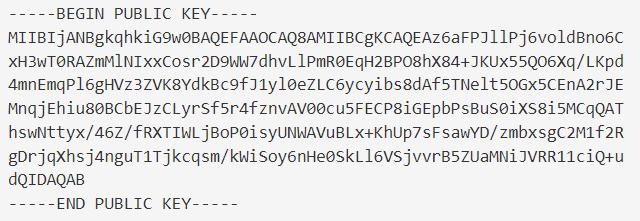


Figure 9 Public.pem

The second is about File-Based encryption. Implementing file-based encryption required careful handling of file formats, ensuring proper encoding and decoding during encryption and decryption. So we carefully managed file operations, encoding data for secure storage and decoding when retrieving information.

The implementation of the RSA Cipher marked a significant improvement in the complexity and security of our password manager. The adoption of public-key cryptography addressed vulnerabilities present in simpler encryption methods, offering enhanced protection for user passwords. The RSA algorithm's effectiveness in secure communication was evident through its successful integration into our system.

#### 2.3.3. Implementing the AES Algorithm

In this section, we ventured into the implementation of the AES Cipher, a symmetric encryption method chosen alongside the asymmetric encryption method RSA. Since AES was not covered in our coursework, we aimed to gain a basic understanding of its principles by incorporating it into our password manager.

We explored its implementation using the Crypto library in Python, employing modules such as AES, Random, and Util. The key steps in our implementation include steps below.

We generated a random 16-byte AES key using the *get\_random\_bytes* function. The generated key was saved to a file ("*aes\_key.pem*") to ensure consistent key usage during both encryption and decryption processes.

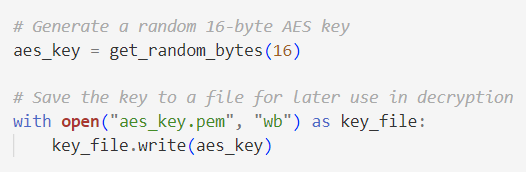


Figure 10 AES Cipher (1)

For encrypting user passwords, we employed AES in Cipher Block Chaining (CBC) mode. The *aes\_encrypt* function takes the user's password as input, generates a cipher text (ct) and initialization vector (iv), and returns the concatenation of iv and ct.

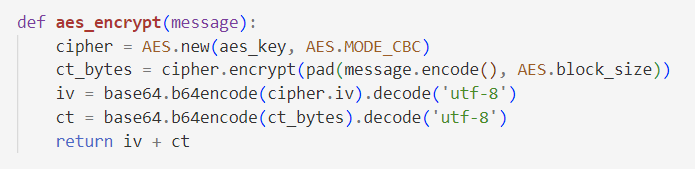


Figure 11 AES Cipher (2)

The *aes\_decrypt* function receives the concatenated iv and ct, decodes them, and performs AES decryption. It ensures proper unpadding to retrieve the original plaintext password.

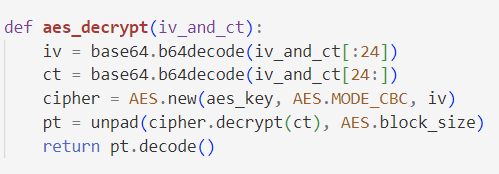


Figure 12 AES Cipher (3)

Amongst these three methods, the Vigenère Cipher proved to be relatively straightforward, characterized by a simple encoding and decoding process. The method's advantages include ease of implementation and understanding, making it suitable for scenarios with short texts. However, it has notable disadvantages, such as vulnerability to frequency analysis and known-plaintext attacks. The periodic repetition of the key limits its security for longer texts.

The RSA Algorithm implementation posed a moderate level of difficulty, involving tasks like key generation, encryption, and decryption with careful consideration for secure file handling. RSA offers strong security due to its use of asymmetric keys, making it suitable for secure communication and digital signatures. However, it comes with disadvantages, including relatively slower processing for large data and complexities in key management, especially with longer key lengths.

Implementing the AES Cipher ranged from moderate to high difficulty, requiring attention to key generation, encryption, and decryption, including considerations for padding and mode of operation. AES is efficient for handling large amounts of data and provides strong security with proper key management. Nevertheless, challenges may arise in key distribution and management in certain scenarios, and there are potential vulnerabilities if not used securely, such as weak keys or improper modes.

In summary, the Vigenère Cipher, while simple, lacks robust security and is suitable for shorter texts. RSA, with its moderate complexity, offers strong security for various applications but may face challenges in processing large data and key management. AES, with moderate to high complexity, excels in efficiently securing large datasets but requires careful consideration in key distribution and management. We choose RSA Cipher for our final implementation, while other methods can still be valid choices for different use.

### 2.4. Tools and Technologies Used

In the development of our password manager, we harnessed a combination of tools and technologies to achieve a robust and user-friendly application. Python served as the programming language foundation, enabling us to implement core functionalities, cryptographic operations, and file handling. The Tkinter library played a pivotal role in crafting a graphical user interface (GUI), offering a seamless interaction experience. It's important to note that Tkinter is compatible primarily with Windows. Additionally, we enriched the user interface with self-designed icons, elevating the visual appeal and usability of the password manager.

#### 2.4.1. Tkinter Library

In order to enhance user experience and facilitate seamless interaction, we leveraged the Tkinter library in Python for designing the graphical user interface (GUI). Tkinter provides a versatile set of tools for creating intuitive and user-friendly interfaces, allowing us to organize distinct windows for functionalities such as log in, add, list, and search. Each of these windows is tailored to specific user actions, providing a clear and organized structure that contributes to an overall efficient and enjoyable user experience. The integration of Tkinter demonstrates our commitment to crafting a visually appealing and user-centric password management solution.

#### 2.4.2. Python

As introduced above, we use python as our main language to implement our program, and to make our developing clear, we split it into different parts to make it easier for implementing and debugging.

The primary workflow of our program follows a straightforward process. When the user executes manager.py, the initial interface prompts the user to either set up a password (if not done before) or enter the verification password (if already set). Upon successfully setting up or verifying the password, the main window displays three core functionalities: add, list, and search.

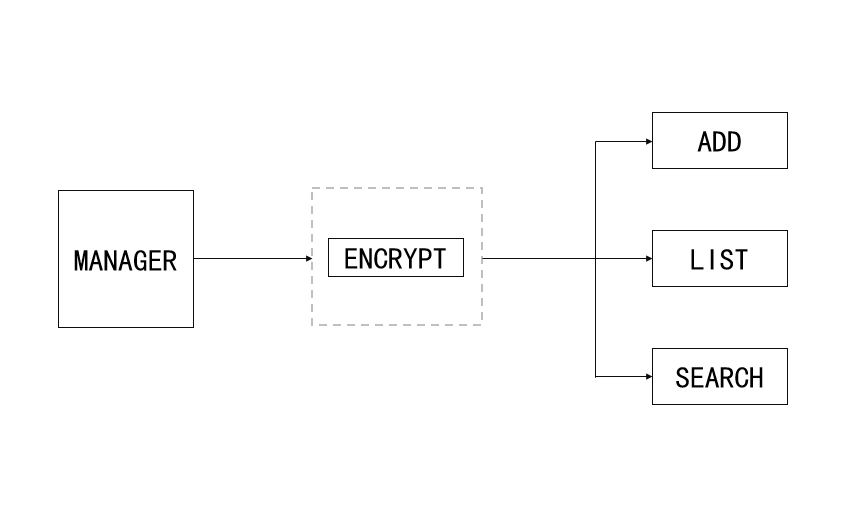


Figure 13 Structure of the Password Manager Master

Below are the modules of the project.

1. **Manager**

The purpose of this module is to serve as a main entry of the application, to initialize and manage the entire application work flow. This module contains the creation of the main interface, handling user login and registration, generating several main windows, and integrating other modules (such as List, Add, Search). It deals with user inputs, navigation, and state management of the entire application.

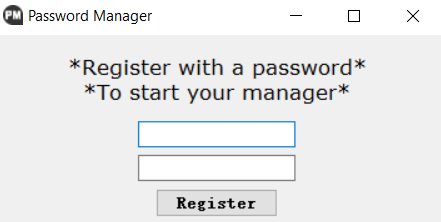


Figure 14 Start Window

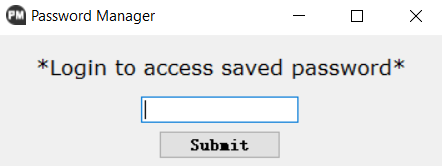


Figure 15 Log in Window

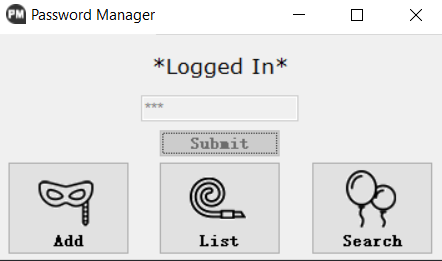


Figure 16 Function Window

1. **Add**

This module is to enable user to add new password records, including new keywords, usernames and passwords. Then the passwords and related information will be stored in a local database after encrypted.

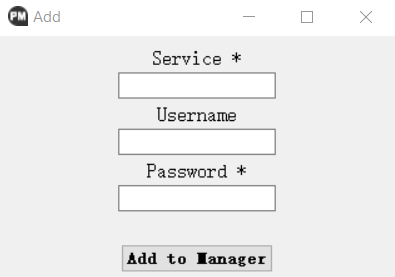


Figure 17 Add Window

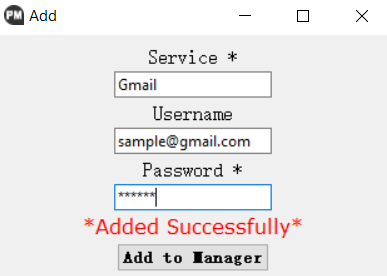


Figure 18 Added successfully Window

1. **List**

The purpose of this module is to store, manage and display the stored passwords. It is responsible to read the user’s input password, pass it to the Encode and Add module and parse the stored data. It also provides the user with an interface to view the saved password list and browse and copy it. The user can also double click to copy the password, which can avoid the password to be shown apparently.

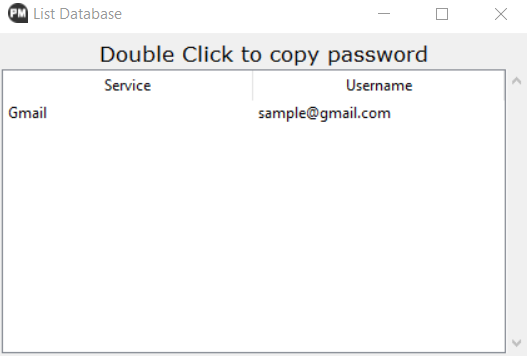


Figure 19 List Window

1. **Encode**

This part manages the encryption and decryption of passwords, and ensure the stored data is not easily accessible by unauthorized parties. This process includes not only the encryption but also the string converting into binary numbers, as well as encoding and decoding strings in formats like UTF-8 or UTF-16. We also use hash function to check whether the password for the manager is correct. This part has been detailed introduced in section 2.3.

1. **Search**

This module is designed for user to quickly find specific password in the stored records. The user can enter the keyword to search the particular result, and the search result will be displayed.

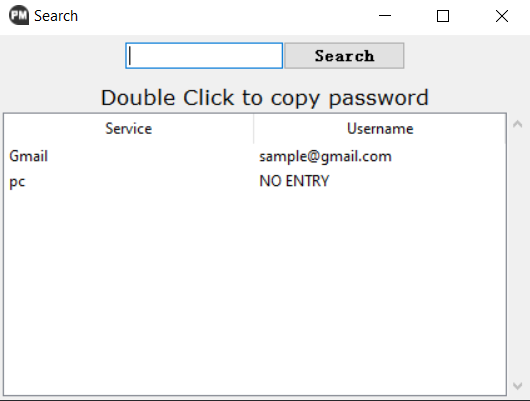


Figure 20 Search Window

#### 2.4.3. Icons

In addition, we have designed some small and refined icons for the user interface, including the top navigation bar and unique icons for each functionality button. This subtle detail injects vitality and charm into the entire program, making the user interface more dynamic, reader-friendly, and interactive.



Figure 21 Icon for top bar (representing Password Master)



Figure 22 Icons for Add, List, Search

## 3. Conclusion

based on your analysis, answer your question. Then discuss the weaknesses and limitations of your project and suggest avenues for future research. And finally, conclude with a paragraph (one separate paragraph per group member) describing what you learned from this project, and whether this report will be of any value to you – either for a future Northeastern course, or for some other project or endeavor you wish to pursue upon your graduation from Northeastern.

### 3.1. The result of our program

The outcome of our program is indeed successful, fulfilling our initial goal of enhancing password management through effective local storage, browsing, and searching functionalities. The program not only meets but surpasses our expectations, providing a seamless and efficient solution for users to manage their passwords with ease and security.

Throughout the process, we gained insights into various encryption methods. Upon careful consideration, we conclude that RSA or AES stands out as superior encryption methods, contributing to the overall security and robustness of our password management solution.

### 3.2. Project Weaknesses and Limitations

Our program currently functions as a basic password manager, and there are several areas that can be enhanced and refined. These include implementing multi-user password management, exploring methods to encrypt key files for increased security, and establishing connections with websites or applications to enable automatic password filling. These avenues present opportunities for future improvements and developments in our password management system.

### 3.3. What we have learned from this project

**Ruotian Zhang:** This project has addressed a fundamental issue for me—providing a secure storage solution for the passwords I lost after my Google Chrome browser crashed a month ago. Beyond that, it marks my first attempt at developing a project from scratch, albeit a relatively simple one. Personally, aspiring to become a software engineer and dreaming of developing my own independent game someday, I and my teammates share a greater interest in software development than data processing. We chose a distinct approach, primarily relying on programming to bring our ideas to life.

Throughout the process, I gained valuable experience, including breaking down a large project into manageable and executable parts, possibly incorporating some object-oriented programming concepts, and systematically completing tasks within a framework. I learned how to flexibly utilize Python libraries to achieve our goals, and, drawing on my past design knowledge, I established interactive interfaces to enhance visual effects. These insights not only provided me with a clearer and deeper understanding of program development but also sparked more interest. Perhaps, they will guide and assist me in the upcoming 5004 Object-Oriented Programming course next semester, instilling greater confidence in realizing my dream of developing an independent game.

**Zhiliang Yu:**

## 4. Appendix

### 4.1. Reference and Citation

1. Vigenère Cipher Concept: <https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher>
2. Vigenère Cipher: https://www.geeksforgeeks.org/vigenere-cipher/
3. RSA Algorithm Cryptography Concept: https://en.wikipedia.org/wiki/RSA\_(cryptosystem)
4. RSA Algorithm Cryptography: <https://www.geeksforgeeks.org/rsa-algorithm-cryptography/>
5. AES Algorithm Cryptography Concept: https://en.wikipedia.org/wiki/Advanced\_Encryption\_Standard
6. AES Algorithm Cryptography: <https://www.geeksforgeeks.org/advanced-encryption-standard-aes/>
7. Password Manager structure:

<https://github.com/ashutoshkrris/Password-Manager-using-Tkinter>

https://github.com/mukkachaitanya/Password-Manager/tree/master

1. Tkinter Usage: <http://stupidpythonideas.blogspot.in/2013/12/tkinter-validation.html>
2. Tkinter Usage, binding a key to a button: http://stackoverflow.com/questions/11456631/how-to-capture-events-on-tkinter-child-widgets

### 4.2. Source

代码