

Information Security IA-1

Report

Topic: Fernet Module of the Cryptography Library

Contents

1. Introduction
2. Features / Characteristics
3. Methodology
4. Results
5. Conclusion

Team Members:

- Yakshit Poojary: 16010121149
- Zaidali Merchant: 16010121108
- Shubraja Lalith: 16010121100

GitHub Repository: <https://github.com/YakshitPoojary/Text-file-encryption>

YouTube Demonstration: <https://youtu.be/a3L6UOeJiFg>

Introduction:

In this report, we explore the implementation of a text file encryption tool using the Python cryptography library. Encryption is essential for safeguarding sensitive information in today's digital landscape. Leveraging the Fernet module, known for its simplicity and security, we encrypt text files using symmetric encryption algorithms. This approach ensures data confidentiality by transforming plaintext into unintelligible ciphertext, mitigating the risk of unauthorized access. We provide an overview of cryptographic principles, elucidate the steps for encryption and decryption, and evaluate the efficacy of our approach. Text file encryption holds relevance across various domains, enhancing information security frameworks. Through this exploration, we aim to underscore the significance of encryption in preserving data integrity and confidentiality, equipping readers with insights into modern cybersecurity practices.

Features/Characteristics:

- **Symmetric Encryption:** Utilizes symmetric encryption algorithms where the same key is used for both encryption and decryption processes, simplifying the implementation and management of cryptographic operations.
- **Data Confidentiality:** Ensures the confidentiality of sensitive information by transforming plaintext data into ciphertext, rendering it unintelligible to unauthorized users.
- **Key Generation:** Provides mechanisms for the generation of encryption keys, ensuring randomness and robustness to enhance the security of the encryption process.
- **Fernet Module Integration:** Incorporates the Fernet module from the Python cryptography library, renowned for its simplicity and security in handling encryption and decryption tasks.
- **Data Integrity:** Enhances data integrity by preventing unauthorized modifications or tampering with encrypted files, ensuring that the decrypted data remains unchanged from its original state.
- **Secure Transmission:** Facilitates secure transmission of sensitive information over networks by encrypting text files before transmission and decrypting them upon receipt, thereby mitigating the risk of interception or eavesdropping.
- **User Authentication:** Supports user authentication mechanisms to control access to encrypted files, ensuring that only authorized users with the requisite decryption keys can access the protected information.
- **Cross-Platform Compatibility:** Offers cross-platform compatibility, enabling encrypted files to be seamlessly decrypted on different operating systems without loss of data integrity or security.
- **Performance Optimization:** Implements optimizations to enhance the performance of encryption and decryption processes, ensuring efficient handling of large text files while maintaining security standards.
- **Compliance with Standards:** Adheres to industry-standard encryption protocols and best practices, ensuring compatibility with existing security frameworks and regulatory requirements.

Methodology:

1. Encrypt() Function:
 - Read Key from the File: Opens the “SecretKey.py” file in binary mode and reads the symmetric key.
 - Read Data from File: Prompts the user to enter the name of the file to be encrypted (‘filename’). Reads the contents of the specified file in binary mode.
 - Encrypt Data: Initializes a “Fernet” object with the read symmetric key. Encrypts the read data using the ‘encrypt()’ method of the ‘Fernet’ object.
 - Save Encrypted Data to File: Writes the encrypted data into a file named ‘encrypted_file.txt’ in binary mode(‘wb’).
2. Decryp() Function:
 - Read Key from File: Opens the ‘SecretKey.key’ file in binary mode and reads the symmetric key.
 - Read Encrypted Data from File: Reads the contents of the ‘encrypted_file.txt’ file in binary mode.
 - Decrypt Data: Initializes a ‘Fernet’ object with the read symmetric key. Decrypts the encrypted data using the ‘decrypt()’ method of the ‘Fernet’ object.
 - Print Encrypted and Decrypted Data: Prints the decrypted data and the original encrypted data for comparison.
3. Gen_key() Function:
 - Generate a Symmetric Key: Utilizes the ‘generate_key()’ method from the ‘Fernet’ class to generate a symmetric encryption key.
 - Save the Key to a File: Writes the generated key into a file named ‘SecretKey.key’ in binary mode(‘wb’).

About the **Fernet** class which we have made use of:

Fernet (symmetric encryption) guarantees that a message encrypted using it cannot be manipulated or read without the key. Fernet is an implementation of symmetric (also known as “secret key”) authenticated cryptography. Fernet also has support for implementing key rotation via “MultiFernet”.

Here is some description from the Docs

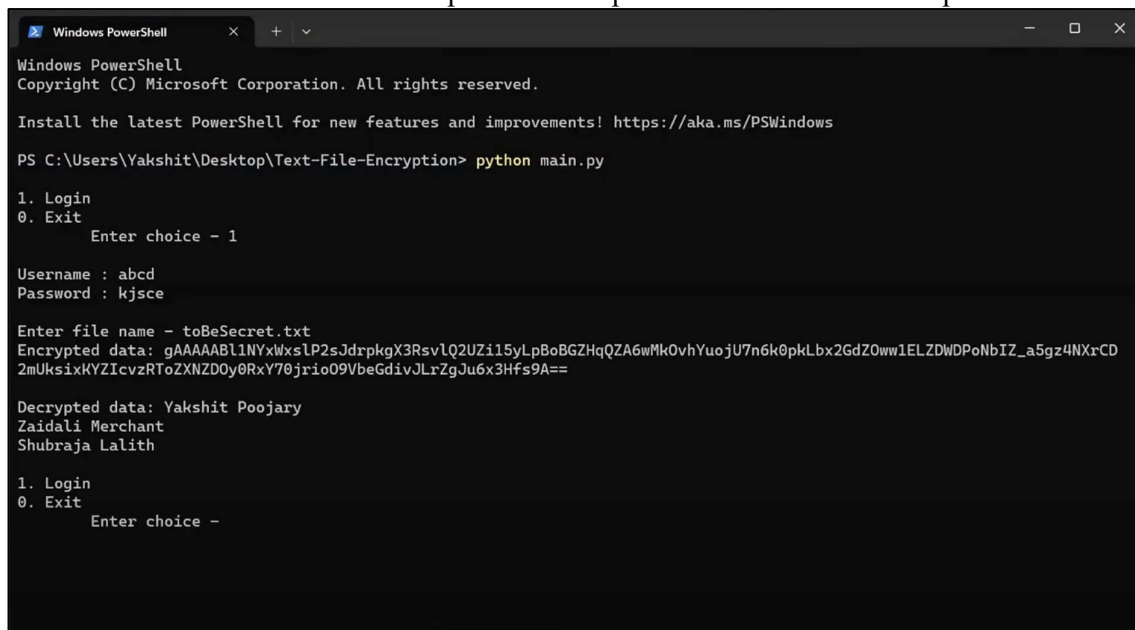
Reference: <https://cryptography.io/en/latest/fernet/>

- Classmethod generate_key()
Generates a fresh fernet key. Keep this some place safe! If you lose it you’ll no longer be able to decrypt messages; if anyone else gains access to it, they’ll be able to decrypt all of your messages, and they’ll also be able to forge arbitrary messages that will be authenticated and decrypted.
- encrypt(data)
Encrypts data passed. The result of this encryption is known as a “Fernet token” and has strong privacy and authenticity guarantees.
Parameters: data (bytes) – The message you would like to encrypt.
Returns bytes: A secure message that cannot be read or altered without the key. It is URL-safe base64-encoded. This is referred to as a “Fernet token”.
Raises: TypeError – This exception is raised if data is not bytes.

- `decrypt(token, ttl=None)`
 Decrypts a Fernet token. If successfully decrypted you will receive the original plaintext as the result, otherwise an exception will be raised. It is safe to use this data immediately as Fernet verifies that the data has not been tampered with prior to returning it.
 Parameters: token (bytes or str) – The Fernet token. This is the result of calling `encrypt()`.
 ttl (int) – Optionally, the number of seconds old a message may be for it to be valid. If the message is older than ttl seconds (from the time it was originally created) an exception will be raised. If ttl is not provided (or is None), the age of the message is not considered.
 Returns bytes: The original plaintext.
 Raises: `cryptography.fernet.InvalidToken` – If the token is in any way invalid, this exception is raised. A token may be invalid for a number of reasons: it is older than the ttl, it is malformed, or it does not have a valid signature.
`TypeError` – This exception is raised if token is not bytes or str.

Results:

Here are the screenshots from the implementation phase of the tool that we explored



```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Yakshit\Desktop\Text-File-Encryption> python main.py

1. Login
0. Exit
Enter choice - 1

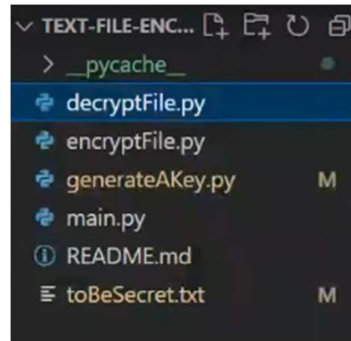
Username : abcd
Password : kjsce

Enter file name - toBeSecret.txt
Encrypted data: gAAAAABl1NYxWxsLP2sJdrpkgX3RsvlQ2UZi15yLpBoBGZHqQZA6wMkOvhYuojU7n6k0pkLbx2GdZ0ww1ELZDWPoNbIZ_a5gz4NXrCD
2mUksixKYZIcvzRT0ZXNZD0y0RxY70jr1o09VbeGdivJLrZgJu6x3Hfs9A==

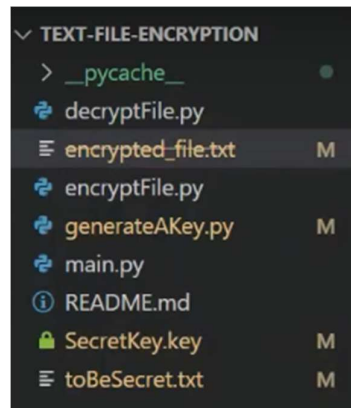
Decrypted data: Yakshit Poojary
Zaidali Merchant
Shubraja Lalith

1. Login
0. Exit
Enter choice -
  
```

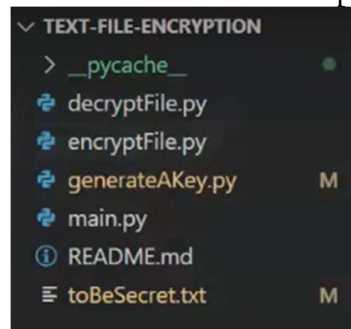
FS before execution:



FS after execution but before termination:



FS after termination of the program:



Conclusion:

In summary, our implementation of text file encryption and decryption using Python's cryptography library, particularly the Fernet module, demonstrates the crucial role of cryptographic techniques in enhancing information security. Through systematic processes, from key generation to encryption and decryption, we ensure data confidentiality and integrity.

This approach emphasizes the importance of secure key management and adherence to encryption best practices. By separating the symmetric key from the encrypted data, we bolster security and mitigate unauthorized access risks.

Looking forward, the methodologies presented here offer practical solutions for protecting sensitive information in various contexts. Encryption remains a cornerstone of cybersecurity, enabling us to safeguard data, uphold privacy, and adapt to evolving threats in today's digital landscape.