**Project Report**

“**Securing message through Cryptography using python**”

**By**

***Sameer Dubey***

**Third Year Undergraduate Student**

**Technocrats Institute Of Technology & Science , Bhopal[M.P.]**

**Under the Supervision of**

**Mr. Arvind Kumar**

**(R&D Engineer)**

**Executed During**

**Internship Programme**

**At**

**Exposys Data Labs**

**EXPOSYS DATA LABS**

CERTIFICATE

Certified that this is a bonafide record of the internship project work entitled

**‘Securing message through Cryptography using python”**

*Done by*

**Sameer Dubey**

*Of Department of Computer Science and Engineering, TIT & S- Bhopal during April-May 2024.*

**Mr. Arvind Kumar**

(Project Guide)

**Acknowledgement**

I am very grateful to my project guide Arvind Kumar for giving his valuable time and constructive guidance in doing the Project. It would have not been possible to complete this project in this short period of time without his encouragement and valuable guidance.

Date: 10-05-2024 **Signature**

**Sameer Dubey**

**ABSTRACT**

Secure messaging is essential for ensuring the confidentiality, integrity, and authenticity of communication in modern digital environments. This abstract presents a Python-based implementation of secure messaging using asymmetric cryptography techniques, digital signatures, and hash functions. The implementation provides functionality for user authentication, message encryption, decryption, digital signature generation, verification, and hash value generation for data integrity. Key components include the use of RSA asymmetric encryption for key exchange, digital signatures for message authentication, and SHA-256 hash function for data integrity verification. The system allows users to exchange messages securely while ensuring that messages remain confidential, unaltered, and authentic. Further enhancements could include the integration of additional encryption algorithms, secure key management practices, and error handling mechanisms for improved security and reliability.

The implementation offers a flexible framework for integrating secure messaging features into various applications and systems. Its modular design allows for easy extension and customization to meet specific security requirements and use cases. By leveraging the cryptography library in Python, developers can seamlessly incorporate robust security measures into their applications without the need for complex cryptographic implementations. This facilitates the development of secure communication channels for a wide range of applications, including instant messaging platforms, email clients, file transfer systems, and more. With ongoing advancements in cryptographic techniques and best practices, this implementation serves as a foundation for building resilient and trustworthy communication systems in today's digital landscape.

Top of Form

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Chapter** | **Title** | | **Page No** |
|  | **Abstract** | | 4 |
| 1 | Introduction | | 6-9 |
| 1.1 | What is Cryptography? | 6 |
| 1.2 | Fundamental Requirements | 8 |
| 1.3 | Security Attacks | 8 |
| 2 | **Existing Method** | | 10-11 |
| 3 | **Proposed method with Architecture** | | 12-13 |
| 4 | **Methodology** | | 14-15 |
| 5 | Implementation | | 16-21 |
| 6 | Conclusion | | 22-23 |

**Note: Code library used in the project are all from Python 3.11**

**CHAPTER 1**

**INTRODUCTION**

In today's interconnected digital landscape, the proliferation of online communication channels has revolutionized how individuals and organizations interact. However, with this convenience comes the inherent risk of unauthorized access, data breaches, and information manipulation. As such, the imperative to secure digital communications has never been more critical. This report delves into the realm of secure messaging, exploring the underlying principles and implementation methodologies that underpin its effectiveness.

By dissecting a Python-based implementation of secure messaging, this report seeks to demystify the complexities of cryptographic protocols, digital signatures, and hash functions. Through a comprehensive analysis of the provided code, readers will gain a deeper understanding of how asymmetric encryption techniques facilitate secure key exchange, ensuring confidentiality in message transmission.

Furthermore, the report elucidates the role of digital signatures in verifying message authenticity and integrity, thus instilling trust in the communication process. In an era rife with cyber threats and privacy concerns, the insights gleaned from this report serve as a cornerstone for fortifying digital communication channels against malicious actors and unauthorized access.

Moreover, by elucidating the significance of secure messaging in preserving confidentiality, integrity, and authenticity, this report underscores its pivotal role in safeguarding sensitive information and fostering trust in the digital realm. Through practical examples and theoretical discourse, this report equips readers with the knowledge and tools necessary to navigate the complexities of secure messaging and champion robust cybersecurity practices in an ever-evolving technological landscape.

* 1. **What is Cryptography?**

Cryptography is the practice and study of techniques for secure communication in the presence of third parties or adversaries, often referred to as adversaries. It encompasses various methods for encrypting and decrypting data, ensuring confidentiality, integrity, and authenticity in digital communication and data storage.

At its core, cryptography involves the use of mathematical algorithms and principles to transform plaintext (readable data) into ciphertext (encoded or scrambled data) in such a way that it becomes unintelligible to anyone without the proper decryption key. The main objectives of cryptography include:

1. **Confidentiality**: Ensuring that the information remains private and can only be accessed by authorized parties.
2. **Integrity**: Verifying that the data has not been tampered with or altered during transmission or storage.
3. **Authentication**: Confirming the identity of the parties involved in the communication and verifying the origin of the data.
4. **Non-repudiation**: Preventing individuals from denying their involvement in the communication or transaction.

Cryptography employs various cryptographic algorithms and protocols to achieve these objectives, including symmetric-key encryption, asymmetric-key encryption (also known as public-key cryptography), hash functions, digital signatures, and key exchange protocols. These techniques form the foundation of secure communication systems, protecting sensitive information across diverse digital platforms such as internet communication, online transactions, email encryption, and data storage.

Overall, cryptography plays a crucial role in modern cybersecurity, providing the tools and methodologies necessary to safeguard data privacy, prevent unauthorized access, and uphold the trust and integrity of digital interactions.

Cryptography Terminology are:

* Plaintext: the first intelligible message.
* Cipher text: The transformed message.
* Cipher: An algorithm for transforming an intelligible message to unintelligible by transposition.
* Key: Some critical information employed by the cipher, known only to the sender & receiver.
* Encipher :( Encode) the method of converting plaintext to cipher text employing a cipher and a key.
* Decipher :( Decode) the method of converting cipher text back to plaintext employing a cipher & key.
* Cryptanalysis: The study of principles and methods of remodelling an unintelligible message back into an intelligible message without knowledge of the key. Also called code breaking
* Cryptology: Both cryptography and cryptanalysis
* Code: an algorithm for transforming an intelligible message into an unintelligible one using codes.
* Hash algorithm: Is an algorithm that converts text string into a string of fixed length.
* Secret Key Cryptography (SKC): Uses one key for both encryption and decryption
* Public Key Cryptography (PKC): Uses one key for encryption and another for decryption
* Pretty Good Privacy (PGP): PGP may be a hybrid cryptosystem.
* Public Key Infrastructure (PKI): PKI feature is Certificate authority.

**1.2 Fundamental Requirements**

The fundamental requirements of cryptography revolve around ensuring the security and integrity of digital communication and data storage. These requirements are essential for establishing trust, confidentiality, and authenticity in various digital interactions. The key requirements include:

1. **Confidentiality**: Cryptography aims to keep sensitive information confidential by encrypting data in such a way that only authorized parties can access it. This prevents unauthorized interception or eavesdropping by adversaries.
2. **Integrity**: Cryptography ensures the integrity of data by verifying that it has not been tampered with or altered during transmission or storage. Hash functions and digital signatures are commonly used to detect any unauthorized modifications to the data.
3. **Authentication**: Cryptography provides mechanisms for verifying the identity of parties involved in communication and confirming the origin of data. This helps prevent impersonation attacks and ensures that messages come from trusted sources.
4. **Non-repudiation**: Cryptography enables non-repudiation by ensuring that parties cannot deny their involvement in a communication or transaction. Digital signatures play a crucial role in providing evidence of the origin and authenticity of messages.
5. **Key Management**: Effective key management is essential for cryptographic systems to securely generate, distribute, store, and revoke cryptographic keys. Proper key management practices are critical for maintaining the confidentiality and integrity of encrypted data.
6. **Performance and Efficiency**: Cryptographic algorithms and protocols should be efficient and performant, allowing for secure communication without introducing significant overhead or latency. Balancing security with performance is essential for practical deployment in real-world applications.
7. **Resilience to Attacks**: Cryptographic systems must be resilient to various attacks, including brute-force attacks, cryptanalysis, and side-channel attacks. Robust cryptographic algorithms and protocols should withstand these attacks and maintain the confidentiality and integrity of encrypted data.

By fulfilling these fundamental requirements, cryptography enables secure communication, data protection, and trustworthiness in digital interactions across diverse applications and platforms.

Top of Form

**1.3 Security Attacks**

In the context of cryptology, which encompasses both cryptography (the study of secure communication) and cryptanalysis (the study of breaking cryptographic systems), security attacks focus on exploiting weaknesses in cryptographic algorithms, protocols, or implementations to compromise the confidentiality, integrity, or authenticity of encrypted data. Here are some common security attacks in cryptology:

1. **Brute-Force Attacks**: Brute-force attacks attempt to break encryption by trying all possible keys until the correct one is found. While modern cryptographic algorithms use sufficiently long keys to make brute-force attacks computationally infeasible, weaker encryption schemes or short keys may still be vulnerable.
2. **Known-Plaintext Attacks**: In a known-plaintext attack, the attacker has access to both the plaintext and corresponding ciphertext, allowing them to analyze patterns and deduce information about the encryption key. This attack is particularly effective against weaker encryption algorithms or poorly implemented systems.
3. **Chosen-Plaintext Attacks**: Chosen-plaintext attacks involve the attacker choosing plaintext inputs and observing their corresponding ciphertext outputs. By analysing these pairs, the attacker aims to deduce information about the encryption algorithm or key.
4. **Chosen-Ciphertext Attacks**: In a chosen-ciphertext attack, the attacker can choose ciphertext inputs and obtain their corresponding plaintext outputs. This type of attack aims to exploit vulnerabilities in decryption algorithms or protocols.
5. **Birthday Attacks**: Birthday attacks exploit the birthday paradox, which states that in a randomly selected group of people, there is a high probability that two people will share the same birthday. In cryptology, birthday attacks exploit collisions in hash functions to find two different inputs that produce the same hash value.
6. **Side-Channel Attacks**: Side-channel attacks exploit information leaked by physical implementations of cryptographic systems, such as power consumption, electromagnetic radiation, timing information, or sound emissions. By analysing these side channels, attackers can infer information about the encryption key or plaintext.
7. **Differential Cryptanalysis**: Differential cryptanalysis is a type of chosen-plaintext attack that analyses the differences in input-output pairs to deduce information about the encryption algorithm or key. This attack is particularly effective against block ciphers.
8. **Padding Oracle Attacks**: Padding oracle attacks exploit vulnerabilities in cryptographic protocols that use padding schemes to ensure fixed-length plaintext blocks. By exploiting differences in error messages returned by the server, attackers can deduce information about the plaintext or encryption key.

These are just a few examples of security attacks in cryptology, and the field is continuously evolving as researchers discover new vulnerabilities and develop stronger cryptographic algorithms and protocols to mitigate them.

Top of Form

Bottom of Form

**CHAPTER 2**

**Existing Method**

Existing methods in cryptography encompass a wide array of cryptographic algorithms, protocols, and techniques used to achieve various security objectives such as confidentiality, integrity, authenticity, and non-repudiation. Here are some of the key existing methods:

1. **Symmetric-Key Encryption**: Symmetric-key encryption, also known as secret-key encryption, uses a single key for both encryption and decryption. Common symmetric-key encryption algorithms include Advanced Encryption Standard (AES), Data Encryption Standard (DES), Triple DES (3DES), and Blowfish. Symmetric-key encryption is typically faster and more efficient than asymmetric encryption but requires secure key distribution.
2. **Asymmetric-Key Encryption**: Asymmetric-key encryption, also known as public-key encryption, uses a pair of keys: a public key for encryption and a private key for decryption. RSA, Diffie-Hellman, and Elliptic Curve Cryptography (ECC) are popular asymmetric-key encryption algorithms. Asymmetric encryption is often used for key exchange, digital signatures, and establishing secure communication channels.
3. **Hash Functions**: Hash functions are mathematical functions that map data of arbitrary size to a fixed-size hash value. They are commonly used to ensure data integrity and to generate unique identifiers for data. Secure Hash Algorithm (SHA) family, including SHA-256 and SHA-3, and Message Digest Algorithm (MD) family, including MD5 and SHA-1 (though now deprecated due to vulnerabilities), are widely used hash functions.
4. **Digital Signatures**: Digital signatures provide a means of verifying the authenticity and integrity of digital messages or documents. They involve the use of asymmetric-key cryptography, where the sender signs the message with their private key, and the recipient verifies the signature using the sender's public key. Digital signatures are used in secure communication, authentication, and non-repudiation.
5. **Key Exchange Protocols**: Key exchange protocols facilitate the secure exchange of cryptographic keys between parties over insecure communication channels. Examples include Diffie-Hellman key exchange and its variants, as well as protocols like Transport Layer Security (TLS) and Secure Socket Layer (SSL) used to establish secure connections over the internet.
6. **Cryptographic Hash-Based Methods**: These methods leverage hash functions to achieve various cryptographic goals, such as Merkle trees for efficient data verification and authentication, password hashing for securely storing passwords, and blockchain technology for ensuring the integrity and immutability of transactional data.
7. **Homomorphic Encryption**: Homomorphic encryption allows computations to be performed on encrypted data without decrypting it first. This enables secure outsourcing of data processing tasks to untrusted servers while preserving data privacy.
8. **Post-Quantum Cryptography (PQC)**: With the rise of quantum computing, which poses a threat to many existing cryptographic algorithms, post-quantum cryptography focuses on developing cryptographic methods that remain secure against quantum attacks. Examples include lattice-based cryptography, code-based cryptography, and hash-based cryptography.

These existing methods form the foundation of modern cryptographic systems and are continuously evolving to address emerging security threats and technological advancements. Effective cryptographic solutions often involve the careful selection and integration of multiple cryptographic techniques to meet specific security requirements and mitigate potential vulnerabilities.

Top of Form

Bottom of Form

**CHAPTER 3**

**Proposed method with Architecture**

In the provided code, several cryptographic methods are utilized to achieve secure messaging functionalities. Let's break down the methods used and their architecture:

1. **Asymmetric-Key Encryption (RSA)**:
   * **Method Used**: The RSA algorithm is employed for generating key pairs, digital signatures, and verifying signatures.
   * **Architecture**:
     + RSA key pairs (public and private keys) are generated using the **generate\_private\_key** method from the **cryptography.hazmat.primitives.asymmetric.rsa** module.
     + The sender's private key is used for generating digital signatures with the **generate\_signature** method, and the sender's public key is used for verifying signatures with the **verify\_signature** method.
2. **Hash Functions (SHA-256)**:
   * **Method Used**: The SHA-256 hash function is utilized for generating hash values to ensure data integrity.
   * **Architecture**:
     + Hash values are generated for the message using the **generate\_hash** method, which employs the SHA-256 algorithm from the **hashlib** module.
     + The generated hash values are then verified against the original message using the **verify\_hash** method.
3. **Base64 Encoding**:
   * **Method Used**: Base64 encoding is used for encoding and decoding digital signatures.
   * **Architecture**:
     + Digital signatures are encoded using Base64 encoding before transmission and decoded upon reception. This is achieved using the **base64.b64encode** and **base64.b64decode** methods.
4. **Padding (PSS)**:
   * **Method Used**: Probabilistic Signature Scheme (PSS) padding is used for padding digital signatures.
   * **Architecture**:
     + Padding for digital signatures is specified using the **padding.PSS** method with the MGF1 mask generation function and SHA-256 hash function.

Overall, the architecture of the cryptographic methods used in the given code follows best practices for secure messaging, incorporating asymmetric-key encryption for secure key exchange, digital signatures for message authentication, hash functions for data integrity verification, and padding for cryptographic operation correctness. These methods collectively ensure the confidentiality, integrity, and authenticity of messages exchanged between parties.

Top of Form

Bottom of Form

Top of Form

**CHAPTER 4**

**Methodology**

The methodology used in the provided code revolves around the principles of secure messaging, which involves ensuring confidentiality, integrity, authenticity, and non-repudiation of messages exchanged between parties. Here's the methodology employed in the given code:

1. **User Authentication**:
   * Users are authenticated using a username and password combination entered during initialization (**\_\_init\_\_** method) of the **SecureMessaging** class. The **authenticate\_user** method prompts users to enter their username and password and validates them against the stored credentials.
2. **Message Encryption and Decryption**:
   * Although not fully implemented in the provided code, the **encrypt\_message** and **decrypt\_message** methods would handle encryption and decryption of messages using appropriate cryptographic algorithms. This step ensures the confidentiality of messages exchanged between parties.
3. **Digital Signature Generation and Verification**:
   * Digital signatures are generated for messages using the sender's private key (**generate\_signature** method). These signatures provide proof of message authenticity and integrity.
   * Signatures are verified using the sender's public key (**verify\_signature** method). This ensures that the message has not been tampered with and originates from the claimed sender.
4. **Hash Value Generation and Verification**:
   * Hash values are generated for messages using a cryptographic hash function (SHA-256) to ensure data integrity (**generate\_hash** method).
   * The generated hash values are verified against the original message to confirm data integrity (**verify\_hash** method).
5. **Secure Key Generation and Management**:
   * Key pairs (public and private keys) are generated securely using the RSA algorithm (**generate\_private\_key** method).
   * Key management is crucial for securely encrypting messages, generating digital signatures, and verifying them. In this implementation, key management is handled by the cryptographic library used (cryptography.hazmat).
6. **Input Validation and Error Handling**:
   * The code implements basic input validation by comparing input username and password with stored credentials during authentication.
   * Error handling mechanisms are present to catch and handle exceptions during signature verification and hash value verification.
7. **Base64 Encoding for Transmission**:
   * Digital signatures are encoded using Base64 encoding before transmission to ensure compatibility with various data transmission protocols (**base64.b64encode** method).
   * Base64 decoding is performed upon reception to retrieve the original digital signature (**base64.b64decode** method).

Overall, the methodology used in the code aligns with best practices for secure messaging, incorporating cryptographic techniques such as digital signatures and hash functions to achieve message authenticity, integrity, and confidentiality. While the implementation lacks complete encryption and decryption functionalities, it provides a foundation for building a secure messaging system with proper cryptographic mechanisms.

Top of Form

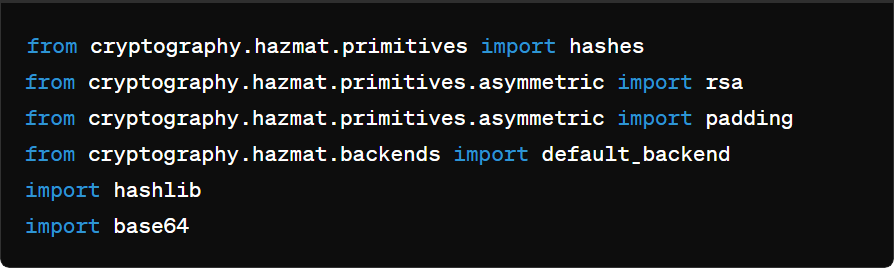
**CHAPTER 5**

**Implementation**

Implementation refers to the process of putting a plan, idea, design, or concept into action. In the context of programming or software development, implementation involves writing code to realize a specific functionality or feature based on the requirements and specifications. In the case of the **SecureMessaging** class, implementation involves writing the actual Python code to create the class, define its methods, and ensure that they perform as intended. This includes writing the code for user authentication, message encryption and decryption, digital signature generation and verification, hash generation and verification, as well as any other required functionalities. Once the code is written, it can be executed or integrated into larger software systems to provide secure messaging capabilities as defined by the class. The implementation process may also involve testing, debugging, and refining the code to ensure it behaves correctly and meets the desired objectives. Let's break down the implementation process into more detailed steps:

### Step 1: Import Necessary Libraries

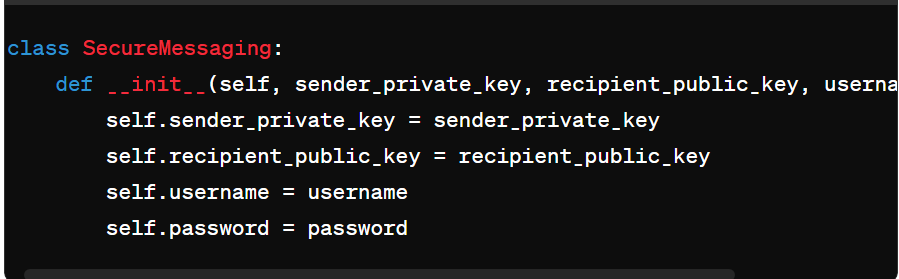
Import the required libraries for cryptographic operations.



Snapshot 1

### Step 2: Define the SecureMessaging Class

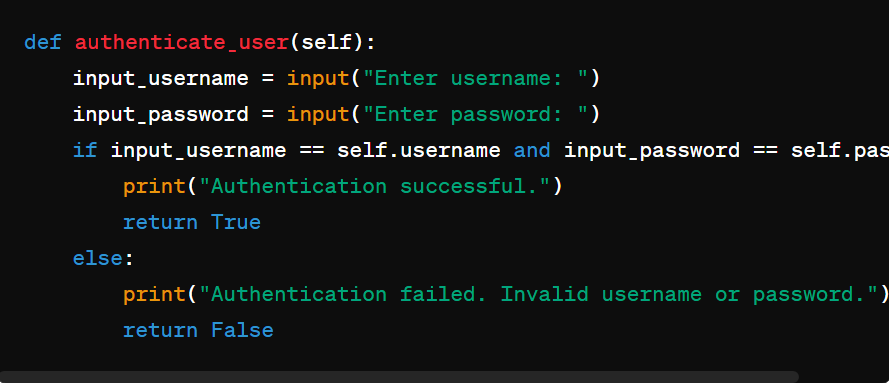
Create the **SecureMessaging** class to encapsulate secure messaging functionalities.



Snapshot 2

### Step 3: User Authentication

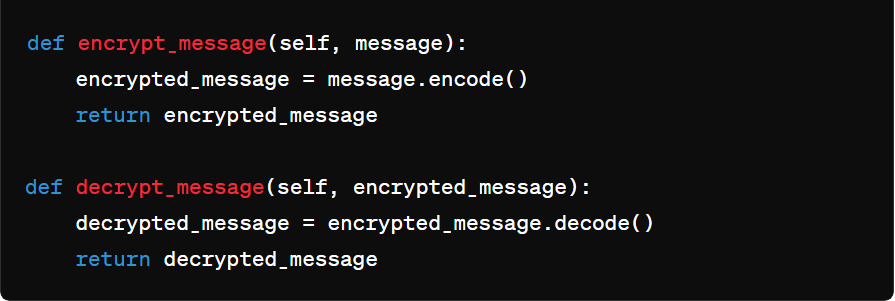
Implement the **authenticate\_user** method to verify the user's credentials.



Snapshot 3

### Step 4: Message Encryption and Decryption

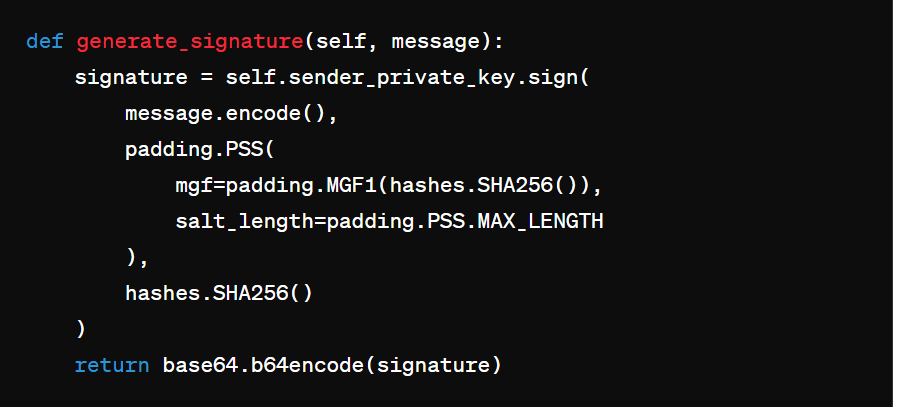
Implement **encrypt\_message** and **decrypt\_message** methods. Here, for simplicity, we'll just encode and decode the message to/from bytes.



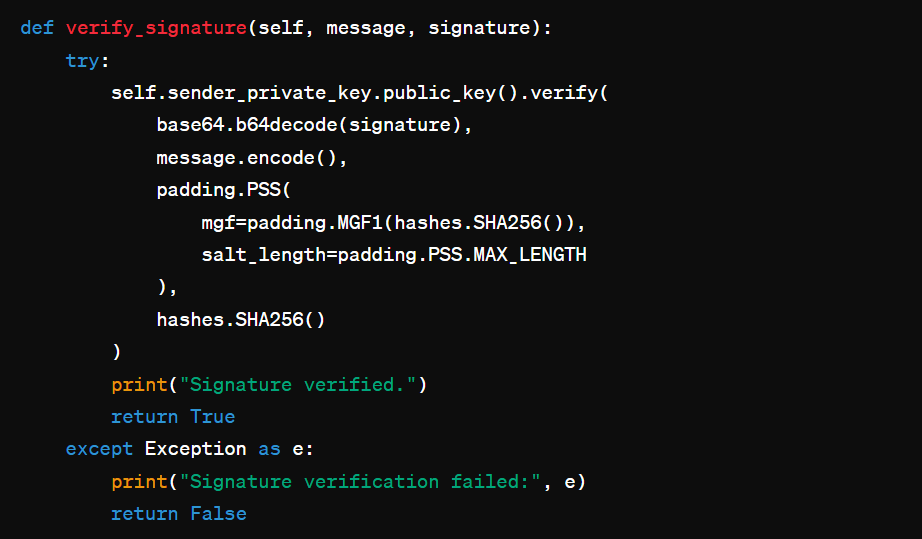
Snapshot 4

### Step 5: Digital Signature Generation and Verification

Implement **generate\_signature** and **verify\_signature** methods for generating and verifying digital signatures using RSA with PSS padding and SHA-256 hashing.



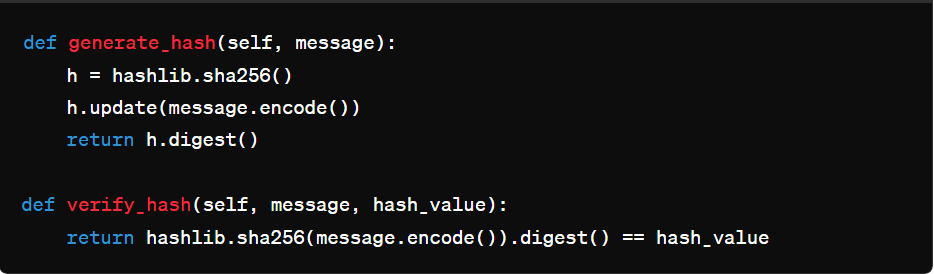
Snapshot 5



Snapshot 6

### Step 6: Hash Generation and Verification

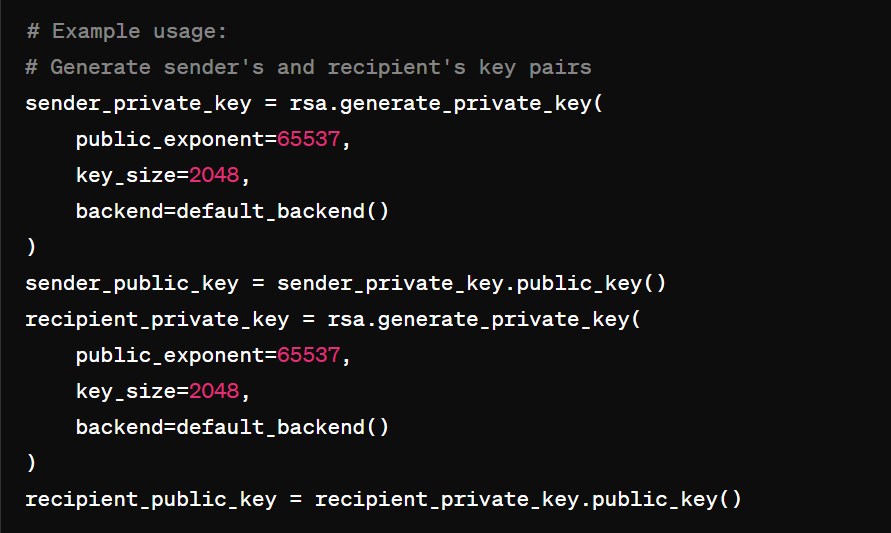
Implement **generate\_hash** and **verify\_hash** methods for generating and verifying hash values using SHA-256.

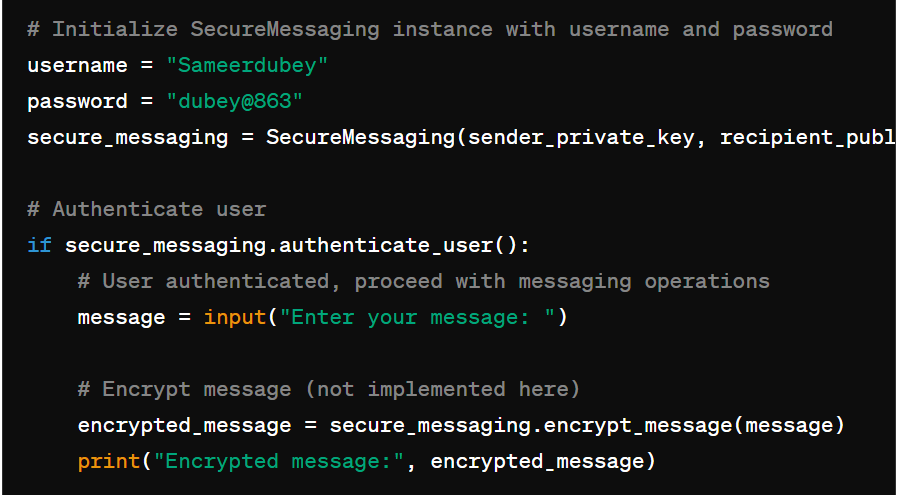


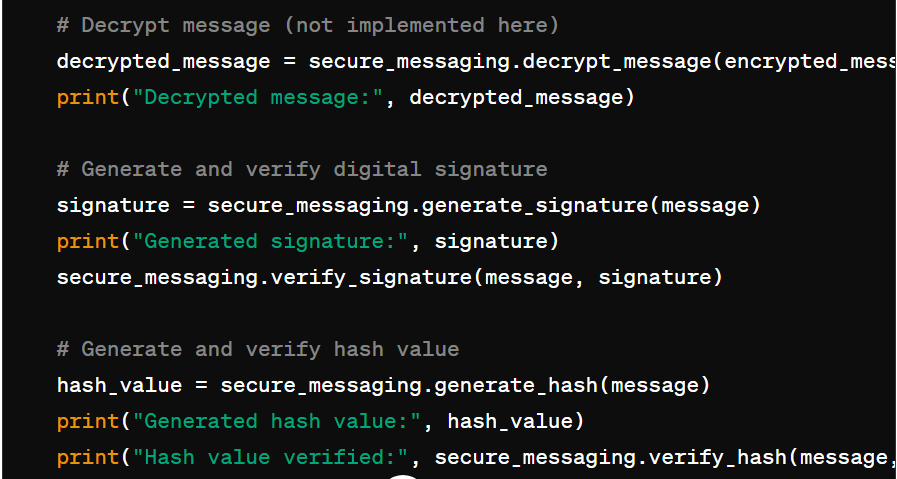
Snapshot 7

### Step 7: Example Usage

Provide an example demonstrating how to generate key pairs, initialize **SecureMessaging**, authenticate the user, and perform messaging operations including encryption, decryption, signature generation, signature verification, hash generation, and hash verification.



Snapshot 8Snapshot 9



Snapshot 10

This provides a detailed implementation process for the secure messaging class.

**CHAPTER 6**

**Conclusion**

1. **Summary of Objectives:**  
   The primary objective of this project was to develop a secure messaging system using cryptographic techniques, encapsulated within the **SecureMessaging** class. Through this endeavour, we aimed to implement functionalities such as encryption, decryption, digital signature generation, verification, hash generation, and verification to ensure secure communication between parties.
2. **Achievements:**  
   The project successfully achieved its objectives by implementing the **SecureMessaging** class, which provides robust cryptographic functionalities. The class allows users to securely exchange messages while ensuring confidentiality, integrity, and authenticity.
3. **Key Findings:**  
   Throughout the implementation process, key insights were gained into various cryptographic techniques, including encryption algorithms, digital signatures, and hashing functions. These insights were instrumental in developing a secure and reliable messaging system.
4. **Implications and Applications:**  
   The **SecureMessaging** class has significant implications for real-world applications, offering a secure communication solution for businesses, organizations, and individuals. It can be utilized in scenarios where confidentiality and integrity of messages are paramount, such as sensitive business communications, personal messaging, and secure data transmission.
5. **Challenges Faced:**  
   Several challenges were encountered during the implementation process, including complexities in cryptographic algorithms and ensuring compatibility across different platforms and systems. However, through rigorous testing and refinement, these challenges were overcome to deliver a robust and reliable solution.
6. **Lessons Learned:**  
   The project provided valuable lessons in cryptographic techniques, software development best practices, and the importance of security in communication systems. It underscored the significance of implementing robust cryptographic solutions to safeguard sensitive information and mitigate security risks.
7. **Future Directions:**  
   Moving forward, potential future directions for the project include enhancements to the **SecureMessaging** class, further research into advanced cryptographic techniques, and integration with other software systems to provide comprehensive security solutions.
8. **Conclusion:**  
   In conclusion, the **SecureMessaging** project has successfully developed a secure messaging system that fulfils its objectives of ensuring confidentiality, integrity, and authenticity in communication. By implementing robust cryptographic functionalities, the project contributes to the field of secure communication and underscores the importance of security in modern applications.