# Diffie-Hellman Key Exchange and PAKE Protocol Implementation

# Theoretical Support

# 1.1 Background

In today's digital landscape, the secure exchange of information has become increasingly vital. With the widespread use of the internet and significant advancements in computing power, traditional encryption methods struggle to meet the growing demands for security. Consequently, the study and implementation of novel cryptographic protocols are essential to safeguard user privacy and data security.

# 1.2 Diffie-Hellman Key Exchange Protocol

The security of the Diffie-Hellman key exchange protocol is based on the computational difficulty of the discrete logarithm problem. Here's how it works:

1. Both parties agree on a large prime number and its primitive root .
2. Party A selects a random private key and computes the public key mod p
3. Party B selects a random private key and computes the public ke​
4. The public keys and are exchanged.
5. Party A calculates the shared key K = mod p.
6. Party B calculates the shared key K = mod p.

Given the complexity of solving the discrete logarithm problem, even if an attacker intercepts the public keys and, they cannot easily derive the shared key K.

**1.3 OPAQUE Protocol**

The OPAQUE protocol is a type of PAKE protocol designed to prevent offline password dictionary attacks. Here's a simplified explanation of how it works:

1.User Registration:The client generates a random password and the server stores a blinded version of the password along with a public key.The server also stores an encrypted private key (envelope) for the client, which is protected by the password.

2.User Login:The client blinds the password using a random value and sends it to the server.The server signs the blinded password using its private key and returns the signature, envelope, and client's public key.

3.Authentication and Key Exchange:The client unblinds the server's signature to obtain an authentication value.Both the client and server use this value along with their respective keys to generate a shared session key.

**1.4 Basic Principles of the Two Attacks**

Man-in-the-Middle Attack: The attacker positions themselves between two communicating parties, intercepting and possibly altering the communication. They can relay and modify messages without the knowledge of the legitimate parties, leading to information leakage, data tampering, or session hijacking.

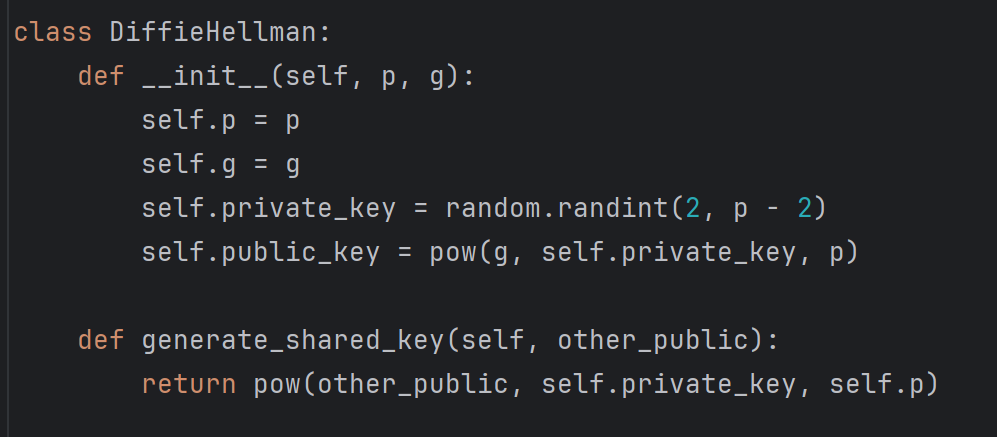
Offline Password Dictionary Attack: The attacker gains access to stored password hashes or encrypted password data. Using a dictionary of common passwords, they perform brute-force cracking offline to find a match for the hash. If a match is found, the attacker can obtain the original password, leading to potential account compromise.

**1.5 Security Analysis**

The Diffie-Hellman protocol's security relies on the discrete logarithm problem's complexity. In this implementation, choosing a sufficiently large prime p and an appropriate primitive root g ensures this complexity. The integration of PAKE enhances security by combining password hashes with key exchange, protecting against offline attacks. Additionally, the local file storage approach eliminates server involvement, reducing communication - related security risks.

1. **Code analysis**

**2.1 Implementing the Diffie-Hellman Key Exchange Protocol**



**2.1.1 Definition and Initialization of the Diffie-Hellman Class**

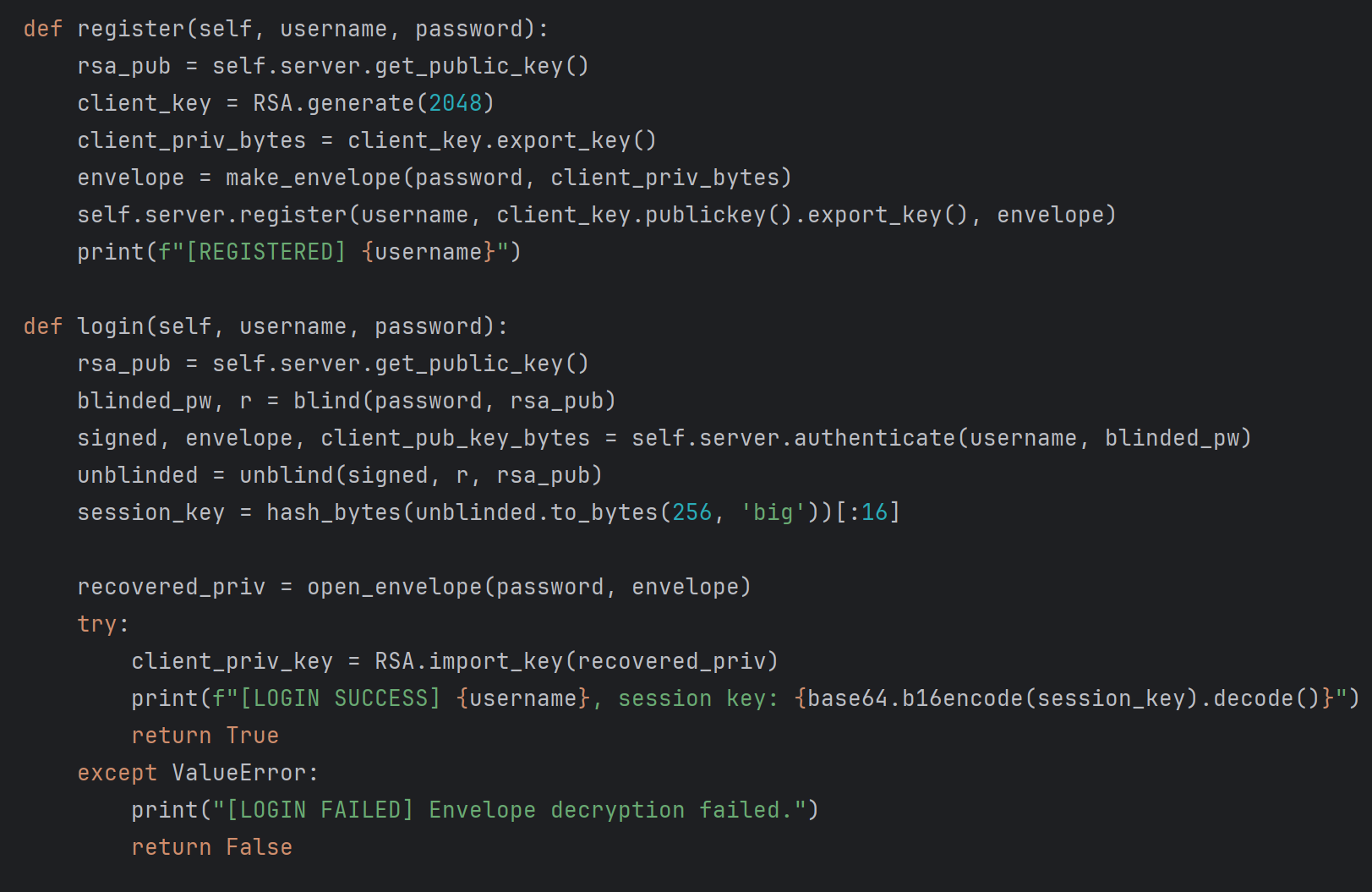
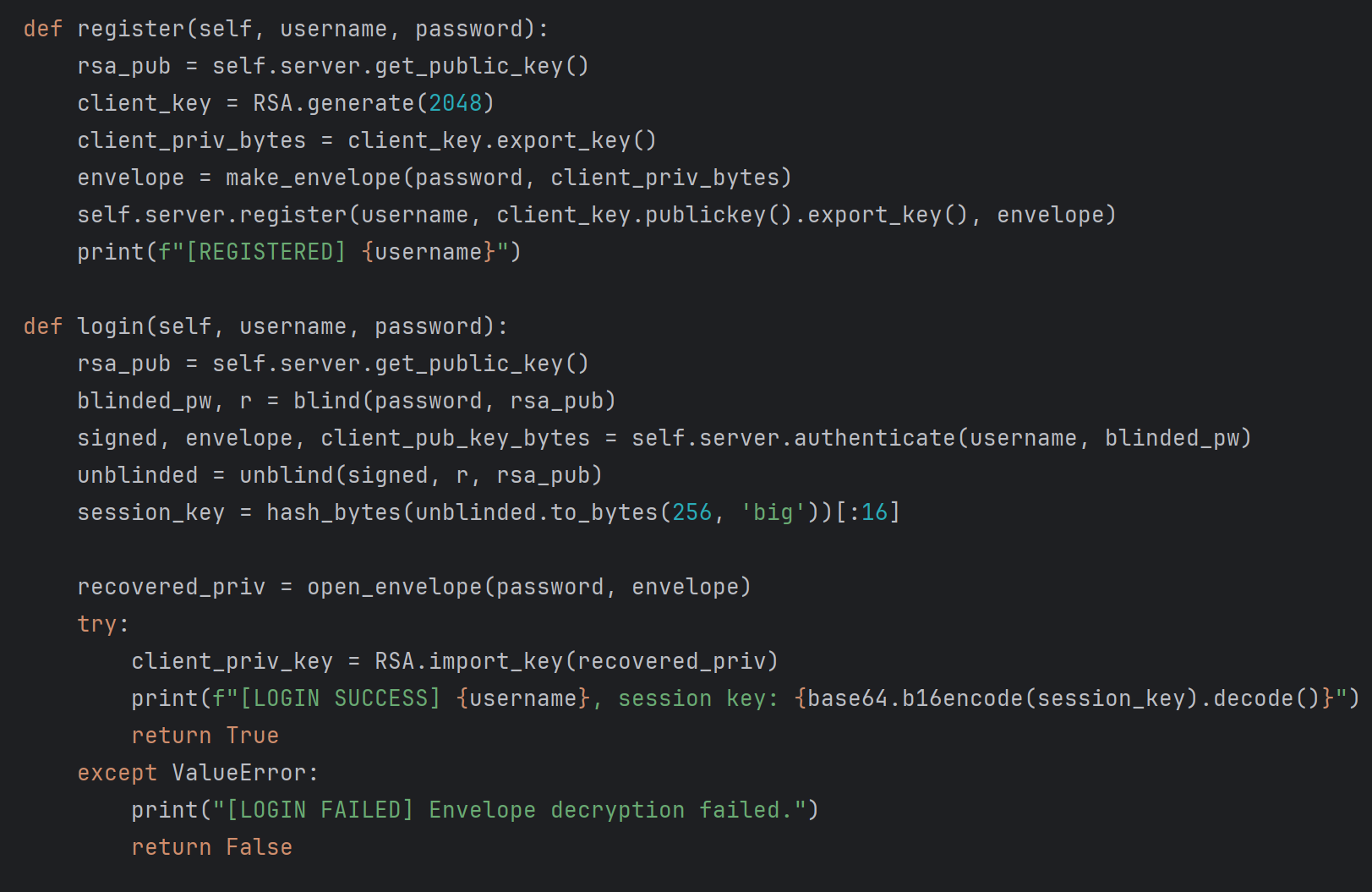
* The class Diffie-Hellman is defined, and its constructor \_\_init\_\_ takes two parameters, p and g, which represent the modulus and base in the Diffie-Hellman algorithm.
* Inside the constructor, a private key private\_key is randomly generated using random.randint(2, p - 2), ensuring its randomness and sufficient security within the range [2, p-2].The public key public\_key is generated using the formula pow(g, self.private\_key, p),

**2.1.2 Public Key Signing and Verification (with RSA signature)**

This code implements secure public key authentication using RSA signatures. The sign\_public\_key method generates a digital signature to ensure key integrity, while verify\_signature validates that signature. Together, they protect against man-in-the-middle attacks by confirming the origin and authenticity of exchanged public keys.

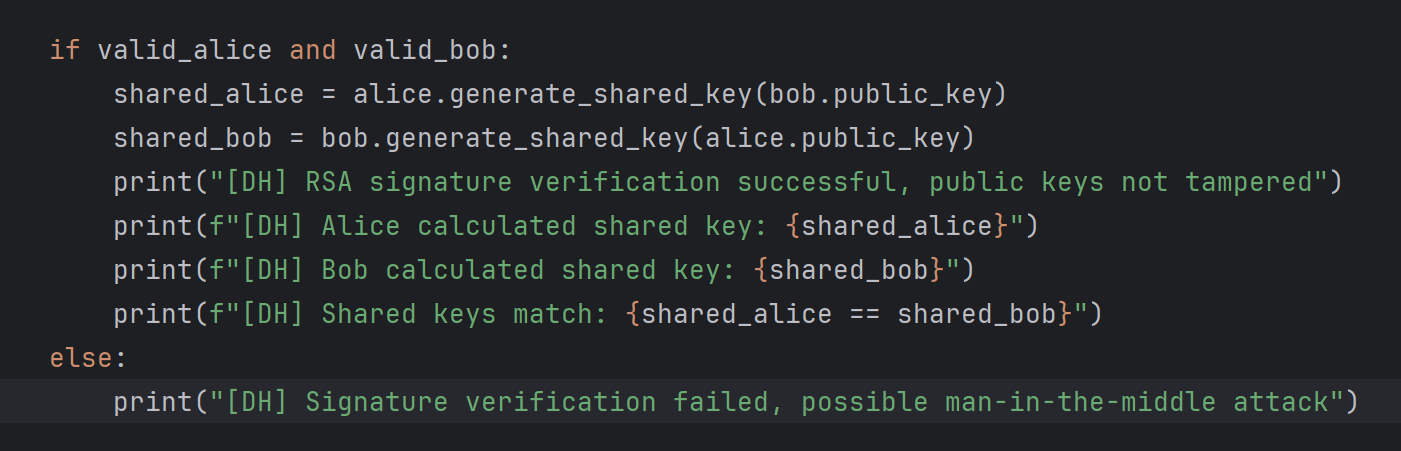
**2.2 Enabling Two Communicating Parties to Securely Establish a Shared Key over an Insecure Channel**

**2.2.1 Signing and Verification of Public Keys**



* The sign\_public\_key method of the Diffie-Hellman class is called to generate signatures sig\_alice and sig\_bob for Alice's and Bob's public keys, respectively. The signing process uses the RSA signature algorithm along with the SHA-256 hash function. Signing the public key ensures its integrity and authenticity.The verify\_signature static method is then used to verify the signatures. If the verification is successful, it means the public key has not been tampered with during transmission, ensuring the communicating parties are using trustworthy public keys.

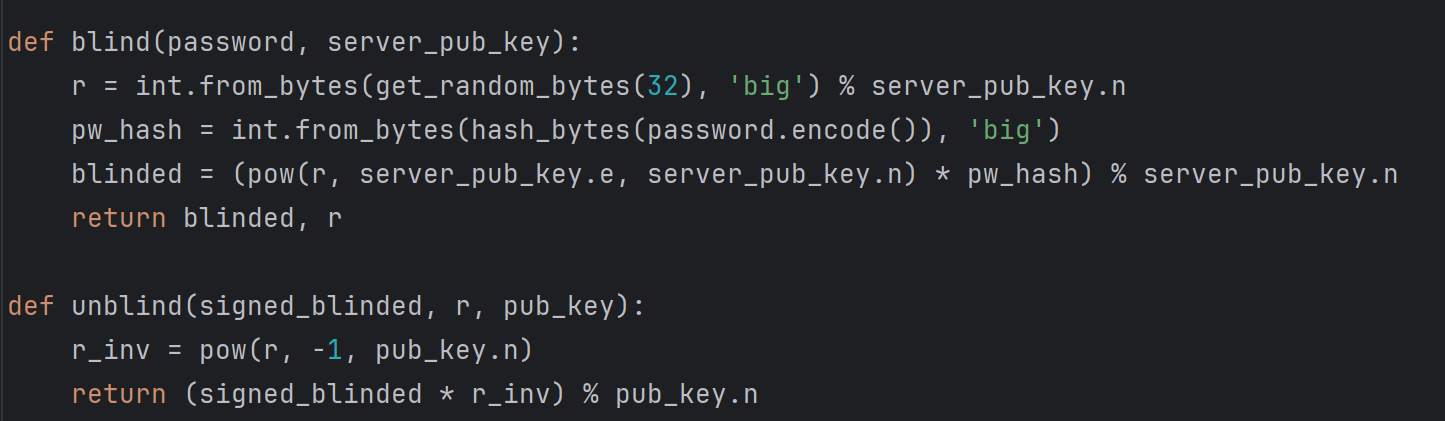
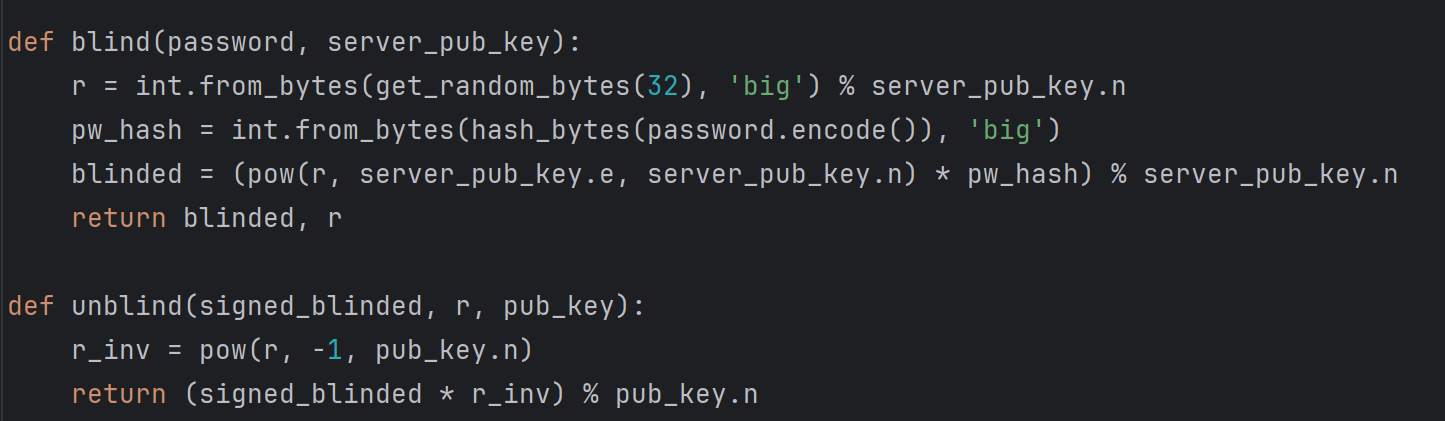
**2.2.2 Verification of Shared Key Consistency**



* Based on the successful signature verification, Alice and Bob respectively compute the shared key using the other party's public key and their own private key.
* The code checks whether the shared key calculated by Alice, shared\_alice, is equal to that calculated by Bob, shared\_bob. If they are the same, it indicates that both parties have successfully established the same shared key over an insecure channel, which can be used for subsequent secure communication

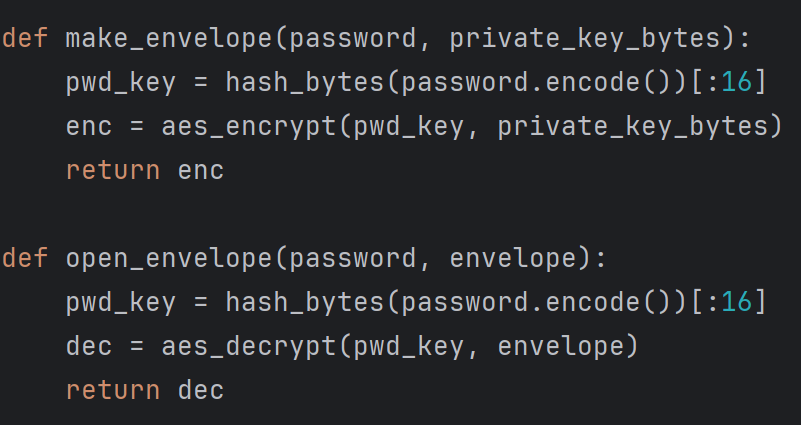
**2.3 Extending the Diffie-Hellman Protocol Implementation to the OPAQUE Protocol**

**2.3.1 Blinding and Unblinding Operations in the OPAQUE Protocol**



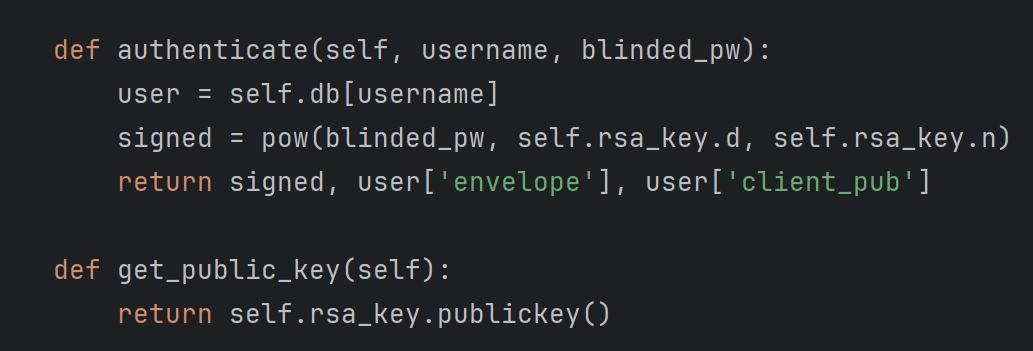
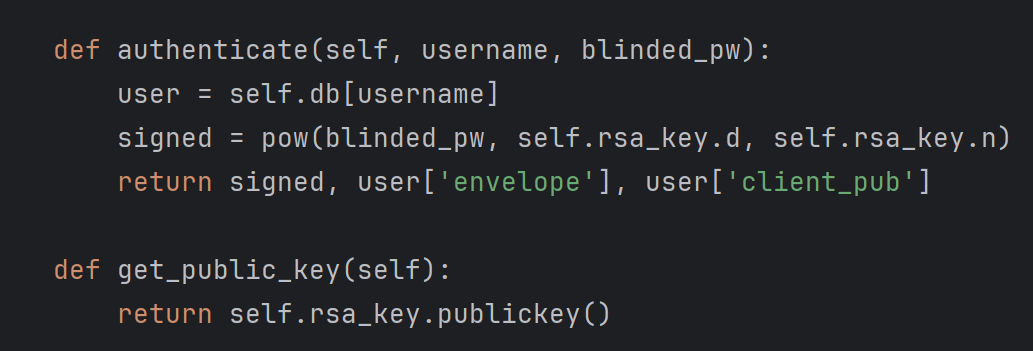
* The blind function takes the user's password and the server's public key as parameters. It first generates a random number r, then hashes the user's password and converts it into an integer pw\_hash. Finally, it computes the blinded password value blinded using the formula (pow(r, server\_pub\_key.e, server\_pub\_key.n) \* pw\_hash) % server\_pub\_key.n and returns the blinding factor r.
* The unblind function takes the signed blinded password, the blinding factor r, and the server's public key as parameters. It calculates the modular inverse of r, r\_inv, and then unblinds the blinded password using the formula (signed\_blinded \* r\_inv) % pub\_key.n to obtain the final authentication value.

**2.3.2 Envelope Construction and Opening**



* The make\_envelope function takes the user's password and the client's private key bytes as parameters. It hashes the user's password, takes the first 16 bytes as the encryption key, and then uses the AES encryption algorithm to encrypt the client's private key, generating an envelope.
* The open\_envelope function takes the user's password and the envelope as parameters. It hashes the user's password to get the key and then uses the AES decryption algorithm to decrypt the envelope and recover the client's private key.

**2.3.3 Interaction Between Server and Client**



* The classes OpaqueServer and OpaqueClient are defined to simulate the behavior of a server and a client.
* The register method in the OpaqueServer class is used for user registration, storing the client's public key and envelope in the server's database. The authenticate method is used for user authentication. It takes the username and blinded password, signs the blinded password using the server's RSA private key, and returns the signature, the user's envelope, and the client's public key.
* The register method in the OpaqueClient class simulates the user registration process. It generates the client's RSA key pair, encrypts the private key (creating an envelope), and sends the public key and envelope to the server for registration.

**2.4 Abbreviated Analysis**

****

The code simulates two types of cyberattacks to test the robustness of the implemented cryptographic protocols:

**2.4.1 Man-in-the-Middle Attack**

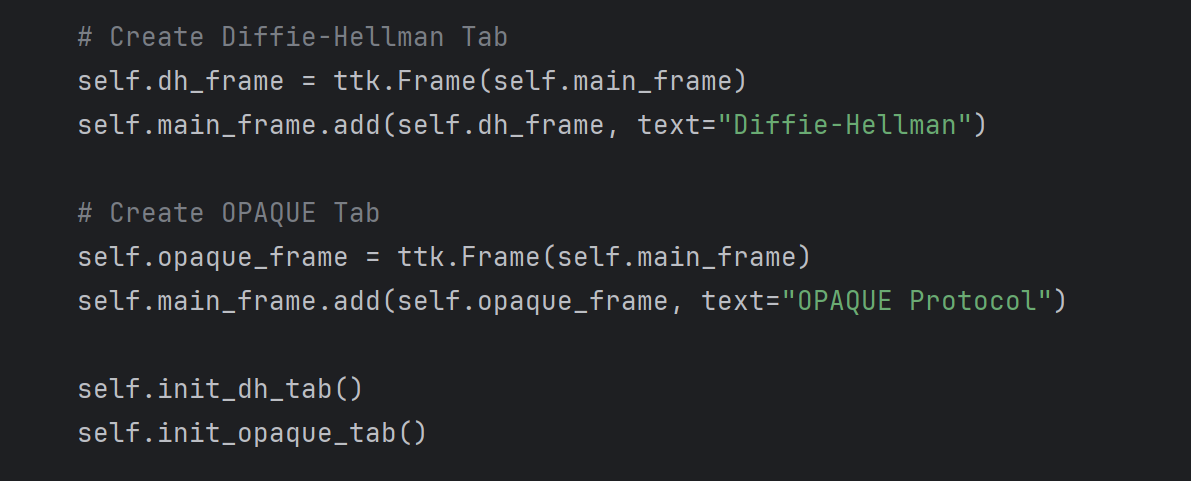
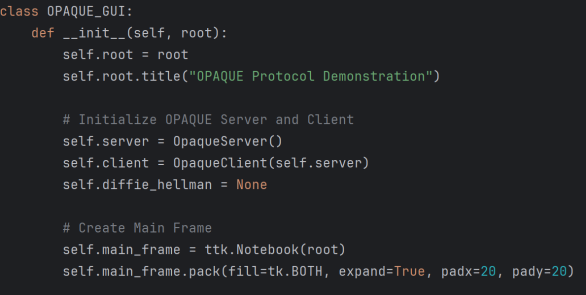
This attack is simulated by intercepting and altering the public keys exchanged between two parties (Alice and Bob) during the Diffie-Hellman key exchange. A middleman generates their own key pair and substitutes the original public keys with their own, leading to the establishment of different shared keys between the communicating parties and the attacker. This demonstrates how vulnerable the key exchange process can be if not properly secured.

**2.4.2 Offline Password Dictionary Attack:**

This attack targets the OPAQUE protocol's password storage. The simulation involves an attacker obtaining the server's password file and using a list of common passwords to brute-force decrypt the stored encrypted private key (envelope). The code checks each password in the list against the envelope, attempting to find a match that successfully decrypts it. If a match is found, the attack is considered successful, revealing the password. If not, the attack fails, showing the protocol's resistance to such attacks when strong passwords are used.

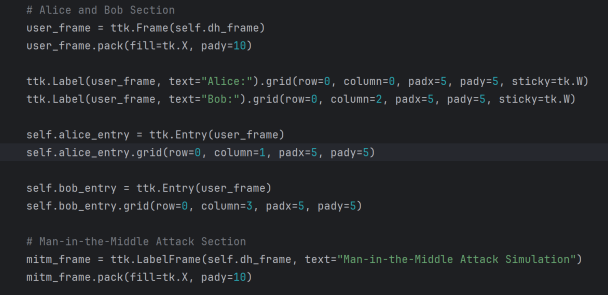
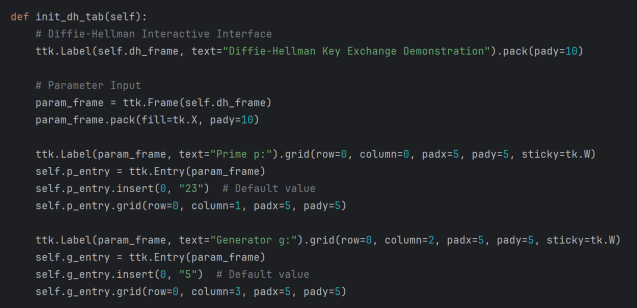
**2.5 Visualization Analysis Report**

**2.5.1 Overall GUI Framework**

****

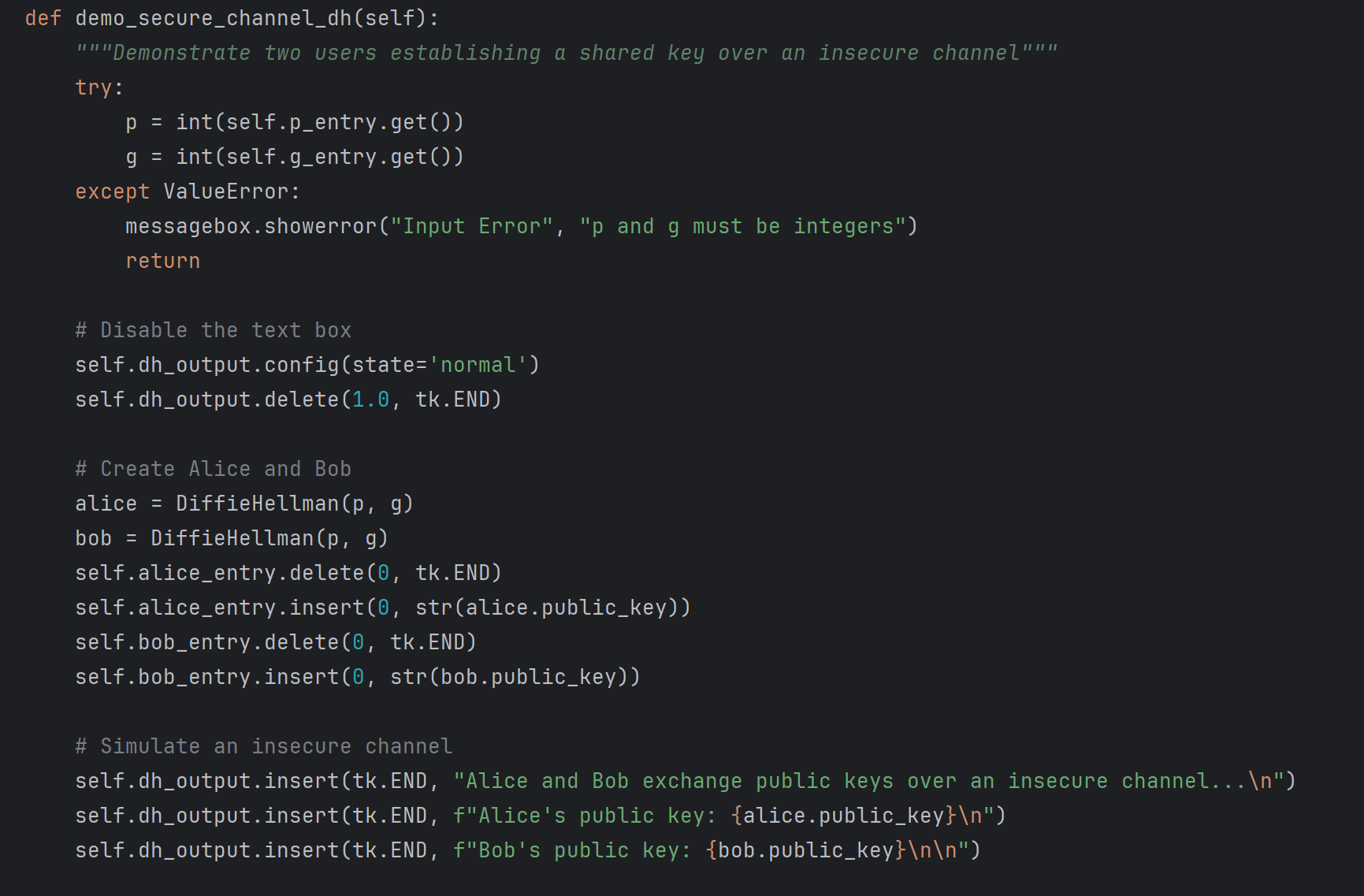
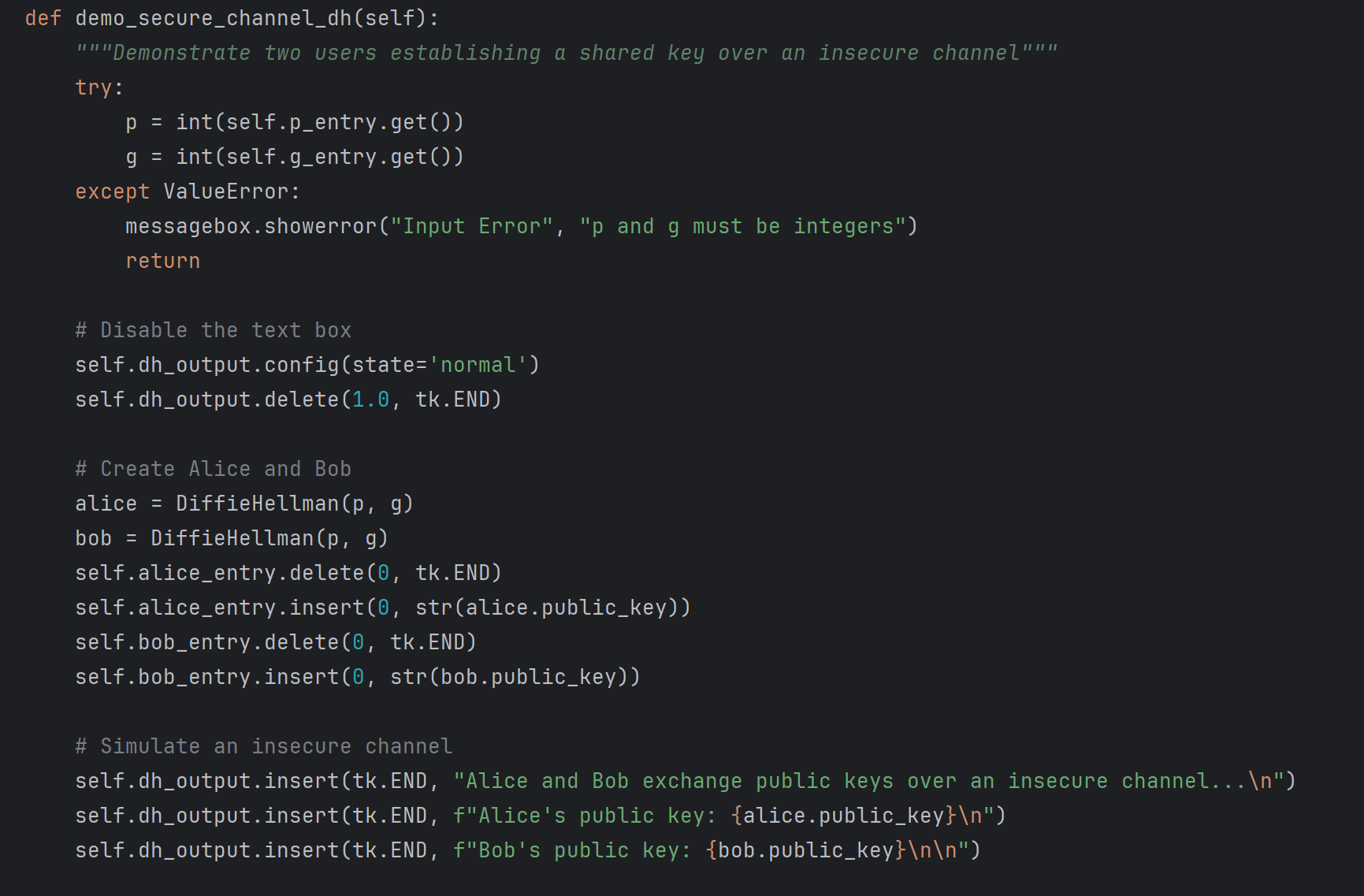
The GUI is built with tkinter, featuring a notebook with two tabs: one for the Diffie-Hellman key exchange and another for the OPAQUE protocol. This setup allows users to easily switch between views and demonstrates both protocols in an organized manner.

**2.5.2 Diffie-Hellman Tab**

****

The tab includes input fields for prime p and generator g, and displays Alice and Bob's public keys. Buttons enable key generation, signature operations, and MITM attack simulation, with results shown in a text widget.

**2.5.3 OPAQUE Tab**

****

This tab contains sections for user registration and login, with entry widgets for credentials and buttons for related actions. It also includes buttons to demonstrate secure key exchange and resistance to offline attacks, with results displayed in a text widget.

**3 Reflection and Takeaways**

This group assignment was quite a challenge for all of us. Firstly, cryptography itself is a difficult subject—understanding the Diffie-Hellman exchange and PAKE protocols required revisiting a substantial amount of mathematical foundations. Secondly, the time frame was tight; completing such a complex and concept-rich task within just one week was quite daunting. Fortunately, through clear communication and active collaboration among team members, we were able to successfully complete the project. We sincerely thank the course instructor and teaching assistants for their guidance and support!