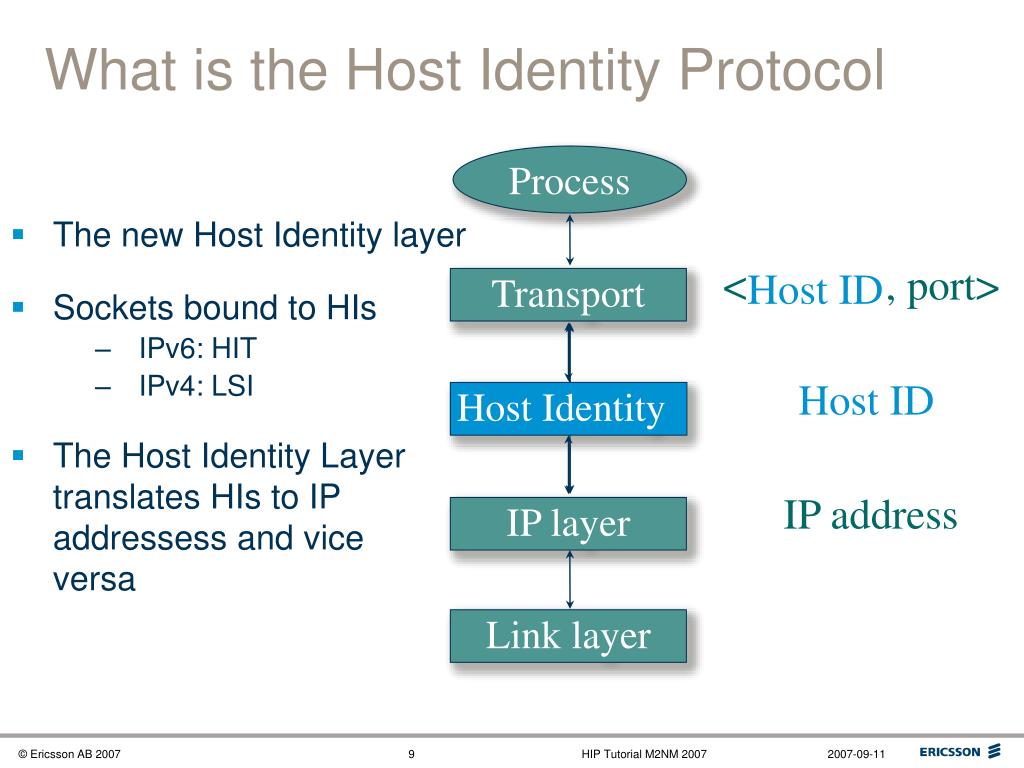
HIP Protocol Simulation

# 1. Introduction

The Host Identity Protocol (HIP) is a secure communication protocol that separates the role of IP addresses as identifiers and locators by introducing a new namespace based on public key cryptography. HIP enhances the security and mobility of network connections while preserving compatibility with existing applications and transport protocols.  
This project simulates a simplified HIP-based secure communication model using Python, where RSA is used for identity, Diffie-Hellman for key exchange, and Fernet (AES) for encryption. The system includes authentication, authorization, and secure session handling.



# 2. Project Objectives

- Understand the core principles of the Host Identity Protocol (HIP).  
- Implement a simplified HIP-based simulation using Python sockets and cryptography.  
- Demonstrate secure communication using RSA, DH, and AES (Fernet).  
- Enforce Role-Based Access Control (RBAC) and session management.

# 3. Technologies Used

- Cryptography Library (for RSA, DH, AES, HKDF)  
- Socket Programming  
- JSON for user and session storage  
- RSA for identity and signing  
- DH for key agreement  
- AES (Fernet) for secure message encryption

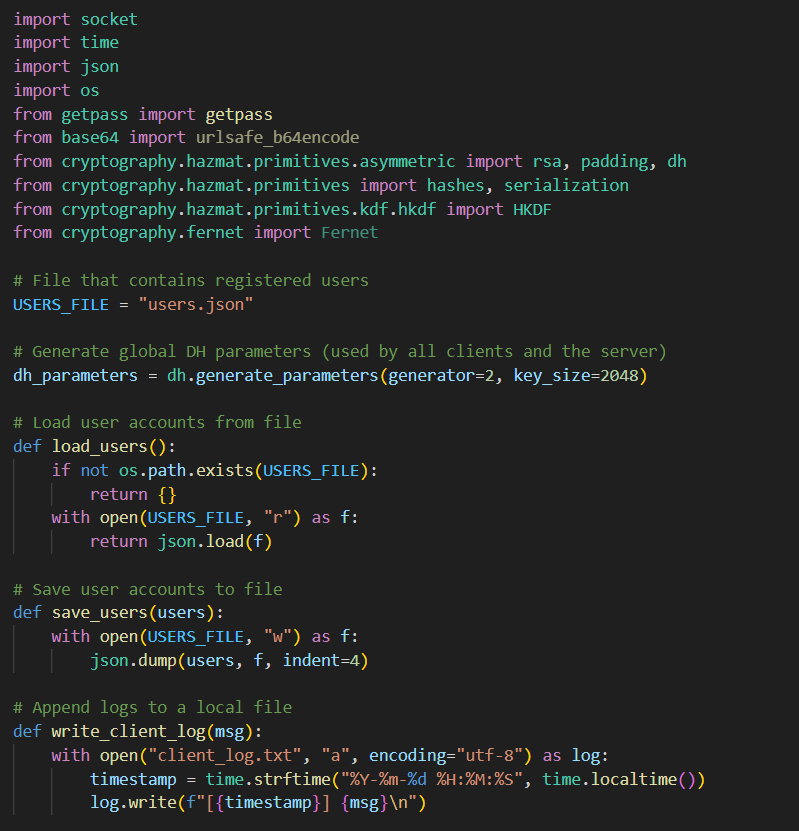
# 4. System Architecture

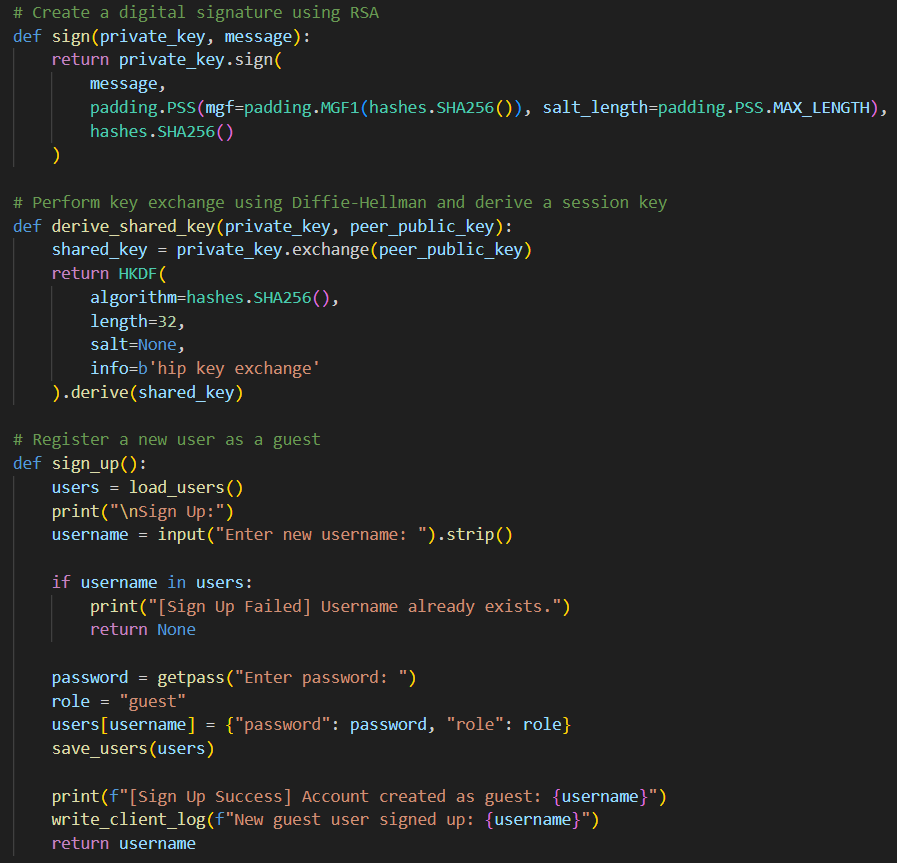
The system is divided into a client and a server. The server listens on a specific port and handles multiple client connections.  
Each client performs the following steps:  
1. Signs in or signs up as a user.  
2. Sends its RSA public key to the server.  
3. Receives the server’s public key and completes a Diffie-Hellman key exchange.  
4. Authenticates using digital signatures and timestamps.  
5. Sends encrypted messages based on its role (admin, analyst, guest).  
6. Each session has a unique key and lasts for 5 minutes.

# 5. Code Walkthrough

## 5.1 Client-Side Logic

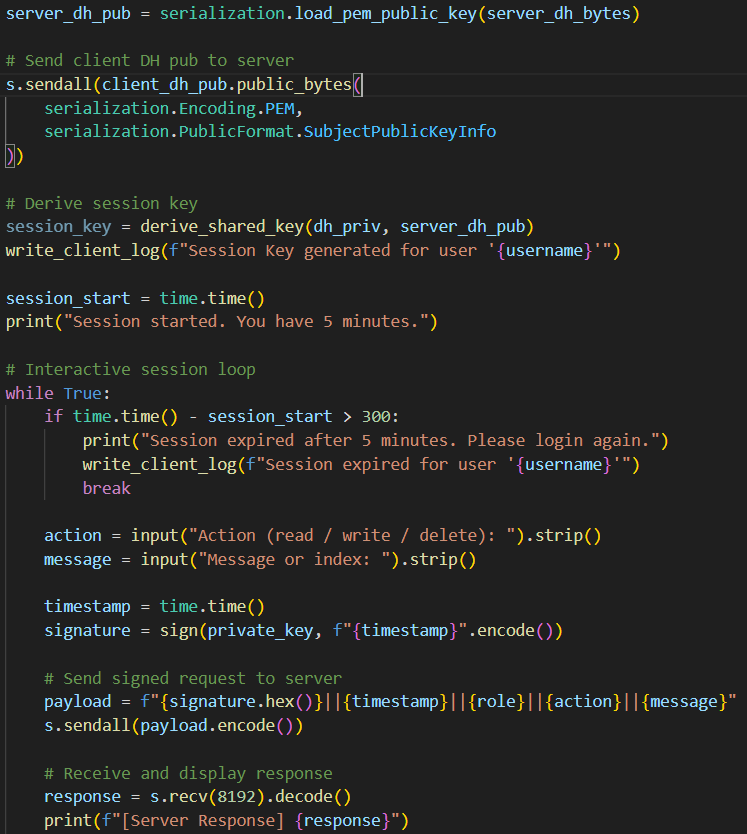
- Loads user credentials from a JSON file.  
- Signs up new users with default guest role.  
- Establishes a socket connection with the server.  
- Performs RSA key exchange and DH session key derivation.  
- Allows users to perform read/write/delete operations if authorized.  
- Logs each action in a local log file.





A computer screen shot of a program

AI-generated content may be incorrect.



A screen shot of a computer program

AI-generated content may be incorrect.

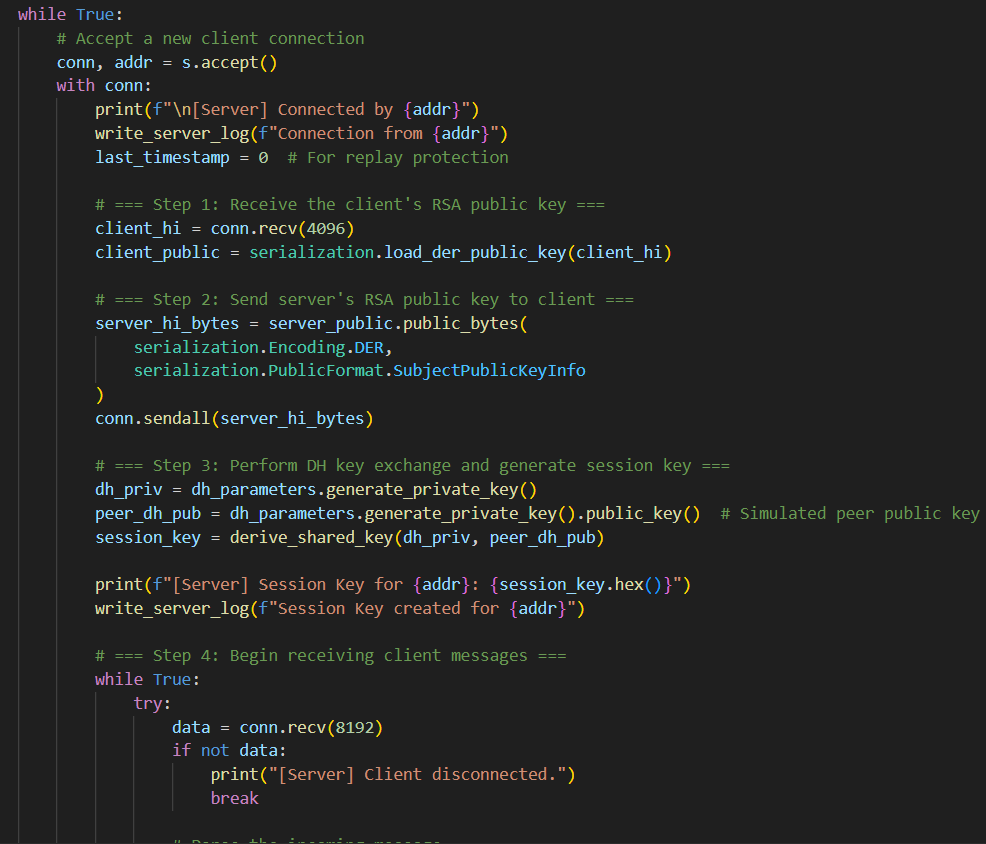
## 5.2 Server-Side Logic

- Generates RSA and DH keys.  
- Verifies digital signatures for authentication.  
- Checks timestamps to prevent replay attacks.  
- Assigns a session key for each connection.  
- Supports multiple operations (read, write, delete) depending on role.  
- Logs each action and session data.  
A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.



A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

# 6. Security Features

- Digital Signatures (RSA) for user authentication.  
- Diffie-Hellman for session key generation.  
- AES encryption (via Fernet) for message confidentiality.  
- Timestamp validation to prevent replay attacks.  
- Role-Based Access Control (RBAC) to enforce permissions.  
- Logs for audit and traceability.

# 7. Sample Logs

Client Log: Records user logins, actions, and session timeouts.  
Server Log: Records connections, session key generation, and all performed actions.

# 8. Limitations & Future Improvements

- No persistent session management across restarts.  
- Messages are stored in memory, not in a database.  
- Multi-client concurrency can be improved.  
- Replay attack simulation can be extended.  
- Better error handling and user interface can be implemented.

# 9. Conclusion

This project successfully simulates the key components of the Host Identity Protocol in a simplified yet functional manner. It demonstrates secure identity exchange, encrypted communication, and controlled access using Python. The modular design allows future extension into real-world applications, such as secure messaging platforms and IoT security frameworks.

# 10. References & Resources

1. RFC 5201 - Host Identity Protocol: https://datatracker.ietf.org/doc/html/rfc5201

2. RFC 7401 - Host Identity Protocol Version 2: https://datatracker.ietf.org/doc/html/rfc7401

3. Python Cryptography Library: https://cryptography.io/

4. Fernet Symmetric Encryption (Fernet - Cryptography Docs): https://cryptography.io/en/latest/fernet/

5. HKDF - HMAC-based Extract-and-Expand Key Derivation Function: https://en.wikipedia.org/wiki/HKDF

6. Diffie-Hellman Key Exchange - Wikipedia: https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman\_key\_exchange

7. RSA Public Key Cryptography - Wikipedia: https://en.wikipedia.org/wiki/RSA\_(cryptosystem)

8. Socket Programming in Python - RealPython: https://realpython.com/python-sockets/

9. Role-Based Access Control - NIST: https://csrc.nist.gov/projects/role-based-access-control

10. Replay Attack - OWASP: https://owasp.org/www-community/attacks/Replay\_attack