## Host Identity Protocol Advanced Implementation Report

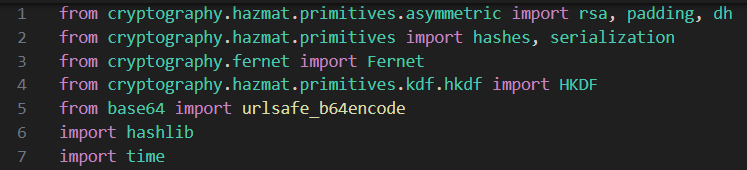
**Authentication, Authorization, Encryption & Replay Attack Protection**

## Team members

1. Farida Ahmed Salem 2206160
2. Hadir Amr 22010450
3. Martin Maher 22010445
4. Youssef Ahmed Ragaee 2202126
5. Youssef Tamer Mohamed Ahmed 2206172
6. Rewan Salah Mahmoud Shiha 20221447143

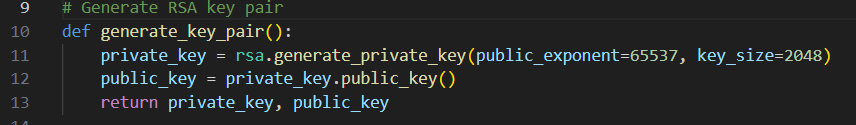
## Overview

In today’s digital world, sensitive information such as login credentials, financial data, or private messages constantly moves across networks. But what happens if someone intercepts this data, pretends to be someone else, or reuses an old message to deceive a system? These threats are real and demand robust solutions rooted in modern cybersecurity.  
To address them, we apply core security principles such as authentication, authorization, encryption, and replay attack protection. To go even further, our implementation explores the Host Identity Protocol (HIP) a network-layer solution that separates host identity from IP address using public-key cryptography.  
HIP introduces a unique identity for each device based on cryptographic keys rather than potentially dynamic IP addresses. This shift enables more secure, mobile, and private communication by supporting encrypted IPsec tunnels and robust identity management, even across IPv4 and IPv6 networks.  
In this project, we simulate HIP’s core ideas through Python. Our system establishes a secure communication channel between two users User A and User B by incorporating Cryptographic identities (HITs) instead of IP-based identifiers , a simulated HIP Base Exchange to establish trust and shared session keys ,RSA digital signatures for mutual authentication ,role-based access control (ACL) for enforcing permissions ,AES encryption using a session key derived via Diffie-Hellman key exchange and Timestamp-based replay protection to prevent the reuse of old messages  
This simulation offers a practical demonstration of how identity-based security, as proposed by HIP, can enhance the confidentiality, integrity, and resilience of modern communications.

**Dependencies :**  


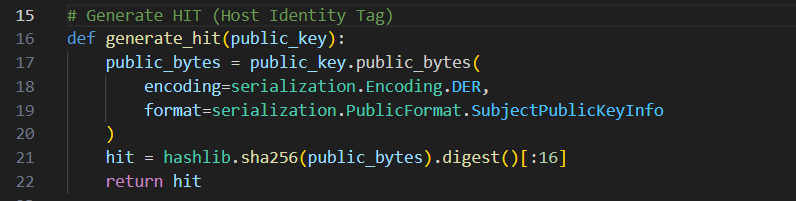
**1.Key Generation**

The generate\_key\_pair() function is responsible for creating a pair of RSA cryptographic keys — a private key and a public key. It uses a commonly accepted public exponent value of 65537, which is considered secure and efficient, and a key size of 2048 bits, which offers strong protection against brute-force attacks. The private key is used for operations like signing or decrypting data, while the corresponding public key is used to verify signatures or encrypt information. This function is essential for establishing secure identity and communication, as each entity in the system relies on its key pair to participate in authentication and encryption processes.

****

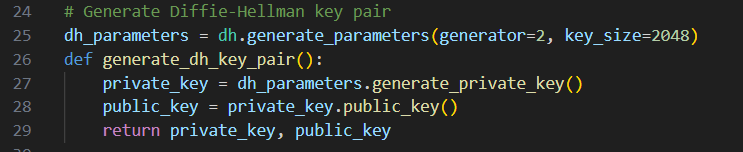
**2.Generate HIT**

To uniquely identify each host regardless of its IP address, we implemented Host Identity Tags (HITs), which are central to the HIP architecture. The generate\_hit() function creates a HIT by first converting a public RSA key into a standardized byte format using DER encoding. It then applies the SHA-256 hashing algorithm to the byte representation and extracts the first 16 bytes (128 bits) of the hash. This shortened hash serves as a stable and compact identity for the host — effectively replacing the traditional IP-based identity with a cryptographically bound, consistent identifier. This approach ensures that even if the device’s IP changes, its identity remains verifiable and secure.

****

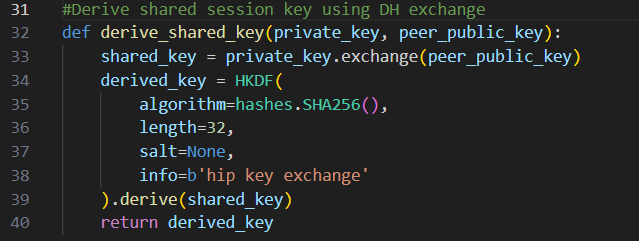
**3.Diffie-Hellman Key Pair Generation**

To establish a shared secret between two parties without directly transmitting it, the system uses the Diffie-Hellman (DH) key exchange protocol. First, a set of DH parameters is generated, which defines the mathematical foundation for secure key exchange. Each entity then creates its own DH key pair by generating a private key and deriving the corresponding public key from it. This setup ensures that both parties can independently compute the same shared secret after exchanging public keys.

****

**4.Session Key Derivation with HKDF**

Once each entity has the other's DH public key, they use it in combination with their own DH private key to compute a common shared secret. This shared secret is then passed through a key derivation function (HKDF), which standardizes and strengthens the output by using SHA-256 as the underlying hashing algorithm. The resulting 32-byte session key is used for symmetric encryption and ensures that the session remains secure even if intercepted.

****

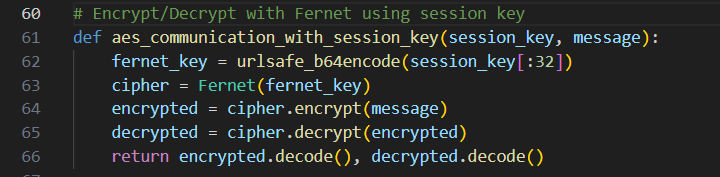
**5.** **RSA-Based Mutual Authentication**

Authentication in the system is achieved using RSA digital signatures. The sender signs the message using their private RSA key, and the recipient verifies the signature using the sender’s public key. If the signature is valid, it proves both the authenticity of the sender and the integrity of the message. This mutual authentication step is crucial before proceeding to encryption or authorization, as it establishes trust between the communicating parties.

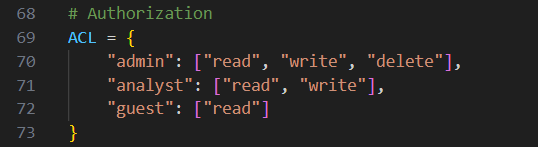
****

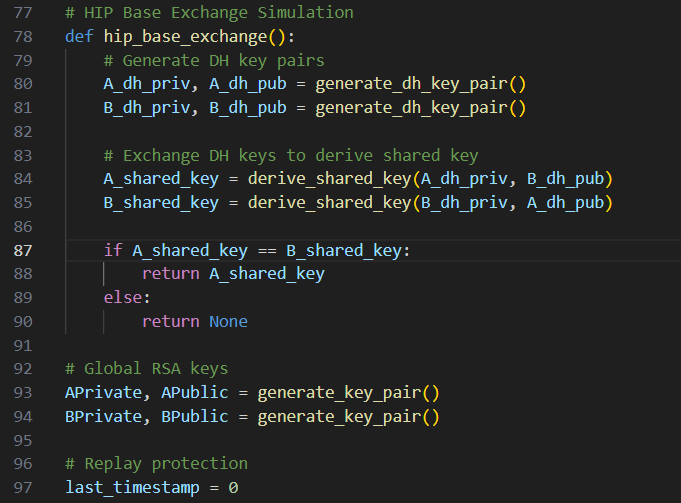
**6.** **AES Encryption Using Session Key**

After successful authentication and key exchange, the system encrypts the message using AES encryption through the Fernet module. The session key derived from Diffie-Hellman is formatted and passed to Fernet, which handles both encryption and decryption. This step ensures confidentiality, meaning only parties who participated in the key exchange can read the message. It also provides message integrity and prevents tampering.

****

**7. Access Control List**The system includes an Access Control List (ACL) to regulate which roles are allowed to perform specific actions. For instance, an 'admin' can read, write, and delete, while an 'analyst' can only read and write. After authentication, the user's role is checked against the ACL to determine if the requested action is permitted. If not, the system blocks the operation even if authentication was successful.

****



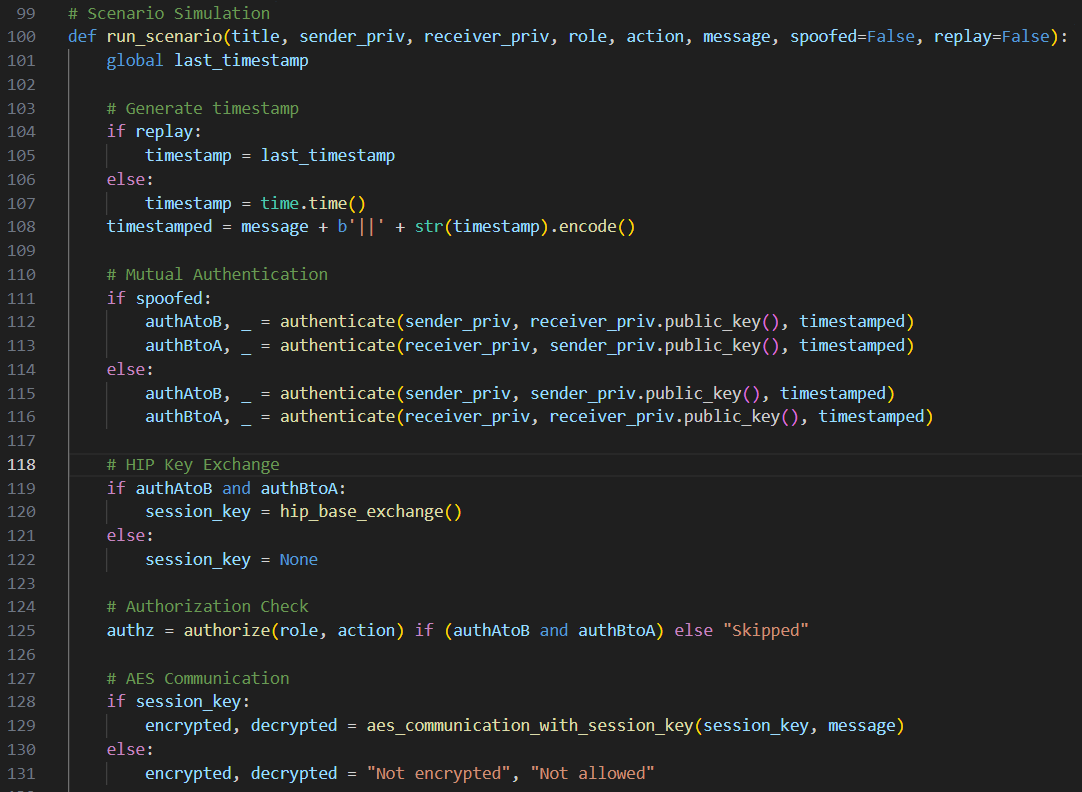
**8.HIP Base Exchange Simulation**

To simulate the HIP protocol’s initial negotiation process, the system implements a simplified base exchange. Each party generates a DH key pair, and then public keys are exchanged. Using each other's public key and their own private key, both parties derive the same session key. This step mirrors HIP’s identity-based key negotiation, ensuring the resulting communication is encrypted and tied to verified cryptographic identities.

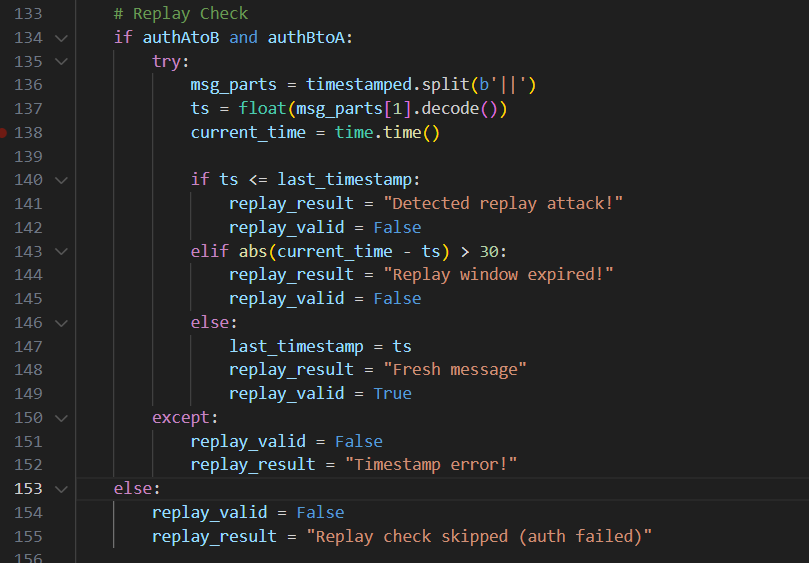
**9. Replay Protection Using Timestamps**

To prevent replay attacks, every message includes a timestamp indicating when it was created. The system checks whether the timestamp is recent (within a 30-second window) and whether it is newer than the last recorded message. If the timestamp is too old or reused, the system identifies it as a replay attack and blocks the message. This mechanism ensures that messages are processed only once and remain timely.

**Scenario Execution Function**The run\_scenario function orchestrates a complete simulation of secure communication between two entities under defined conditions. It begins by generating a timestamp to later assess message freshness. The function then performs mutual authentication using RSA signatures, with an optional spoofing parameter to simulate identity forgery. If both authentication checks succeed, a HIP-inspired Diffie-Hellman key exchange is conducted to derive a shared session key. This session key is used for AES-based encryption and decryption of the message. The function also checks whether the sender’s role is authorized to perform the requested action using a role-based Access Control List (ACL). If authentication fails, encryption and authorization are bypassed, and the message is blocked. This function integrates multiple security mechanisms to emulate realistic communication scenarios and evaluate system resilience.

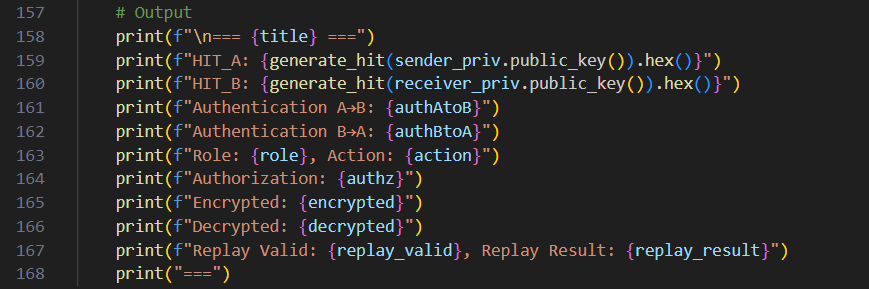


**Replay Attack Detection Using Timestamps**

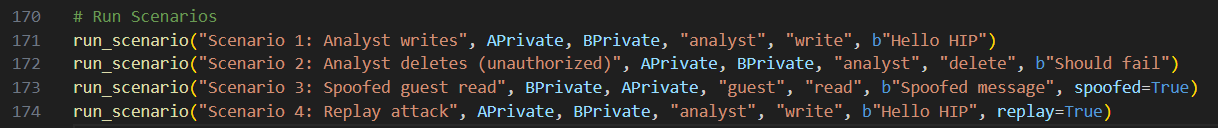
To protect against replay attacks, the system   
implements a timestamp-based verification mechanism. If both sender and receiver pass the mutual authentication step, the system extracts the timestamp embedded in the message and compares it against the last valid timestamp recorded. If the extracted timestamp is older or identical to the previous one, the message is flagged as a replay attempt. Additionally, if the timestamp is too far in the past exceeding a 30-second freshness window the message is also rejected. If the timestamp is recent and valid, the system updates the reference timestamp and accepts the message as fresh. This method ensures that each message is unique and timely, effectively preventing attackers from reusing old messages to gain unauthorized access. In the event authentication fails, the replay check is skipped to prevent unnecessary processing.

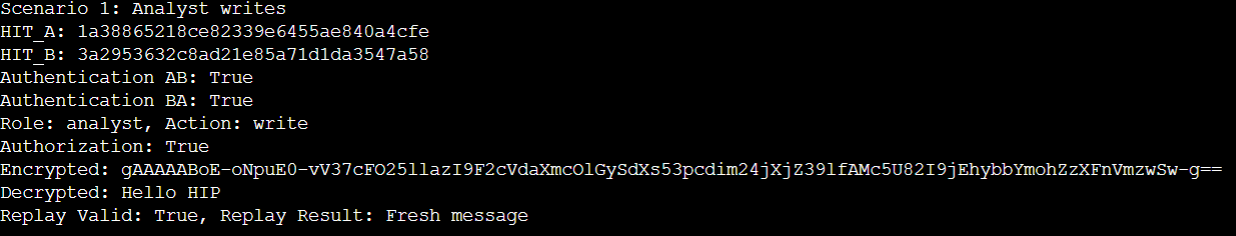
**Output Logging and Debugging Information**

At the end of each scenario execution, the system prints a detailed summary of all critical security operations and their outcomes. This includes the Host Identity Tags (HITs) of both communicating entities, the results of mutual authentication, the role and action attempted by the user, and whether authorization was granted. It also displays the encrypted and decrypted versions of the message, along with the outcome of the replay protection check. These outputs are essential for debugging, monitoring, and verifying the behavior of the system during each test scenario. They provide a transparent view of how each security mechanism functions and help ensure that every step of the secure communication process is working as intended.

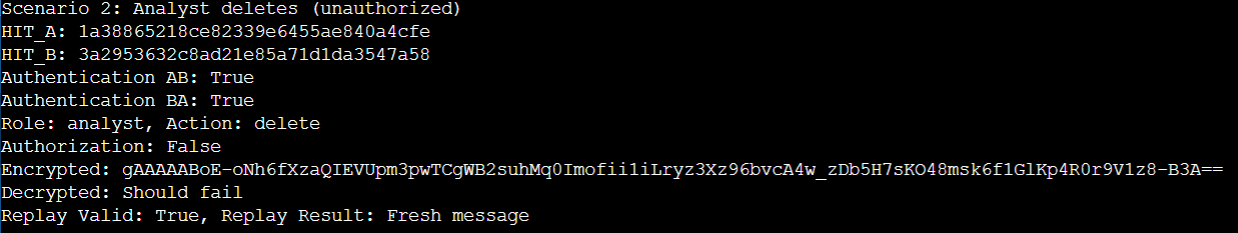


**Scenarios We Tested**

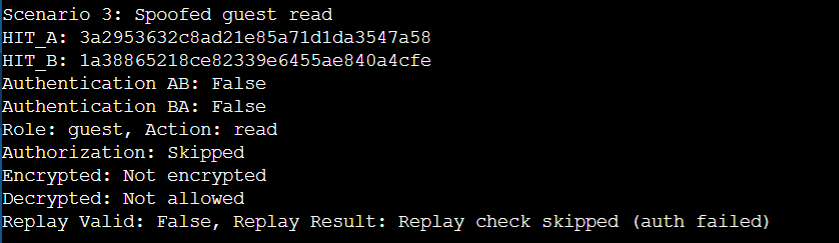


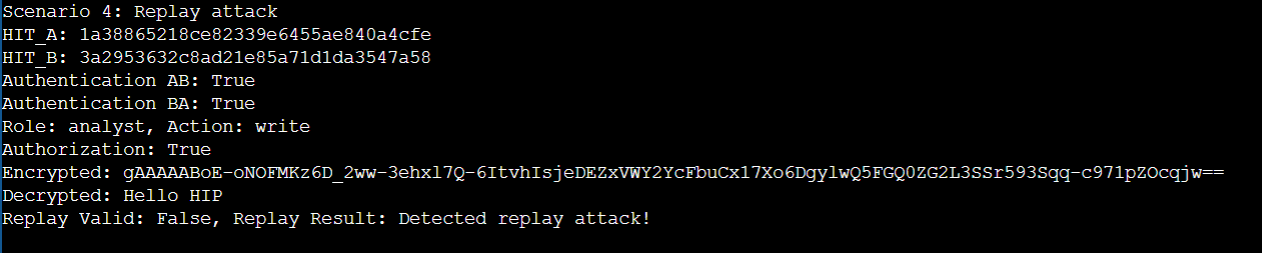
**Scenario 1 Analyst Writes Successful Case**In this scenario, a user with the role of “Analyst” attempts to perform a “write” operation. The system successfully verifies the sender’s identity using mutual RSA authentication between User A and User B. Both Host Identity Tags (HITs) are correctly generated, and the Diffie-Hellman key exchange completes, resulting in a secure session key. The user is authorized to perform the action based on the Access Control List (ACL). The message is securely encrypted using the session key, decrypted correctly on the other side, and marked as a fresh message, confirming that no replay attack occurred. All security mechanisms — authentication, authorization, encryption, and replay protection — functioned as expected.

**Scenario 2 Analyst Deletes Unauthorized Action**The same Analyst tries to perform a “delete” operation, which is not permitted for this role according to the ACL. While the RSA-based mutual authentication and HIT generation are still successful, and the message is properly encrypted/decrypted using the session key, the authorization check fails. The system correctly blocks the unauthorized request, demonstrating how authorization enforces role-specific restrictions, even after a successful identity verification.

****

**Scenario 3 Spoofed Guest Identity Failed Authentication**In this test, an attacker tries to spoof the identity of a Guest user by forging a message. However, the authentication fails in both directions — the system detects mismatched digital signatures, indicating the identity has been forged. As a result, authorization and encryption are skipped, and the replay protection is not performed. This scenario effectively showcases the strength of digital signatures in preventing identity spoofing.

****

**Scenario 4 Replay Attack Blocked**This scenario reuses a previously valid and authenticated message, simulating a replay attack. The system correctly authenticates the sender and decrypts the message, but upon inspecting the timestamp, it detects that the message has already been processed. As a result, the system blocks it, labeling it as a replay attack. This confirms the replay protection feature is working as intended and can detect duplicate messages even when the sender and action are valid.

**Conclusion:**This enhanced implementation of the HIP protocol not only ensures secure communication through RSA-based authentication and Diffie-Hellman key exchange but also introduces robust encryption and access control mechanisms. Most importantly, it successfully detects and blocks replay attacks by validating message freshness based on timestamps. Each scenario was carefully designed to simulate real-world threats and verify that the system behaves as expected granting access only to authorized roles, rejecting unauthorized actions, and preventing message reuse. This makes the code a strong foundation for secure identity-based communication in modern distributed systems**.**