HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Apply Cryptography

GROUP 15
Bùi Duy Anh - 20225563
Chu Trung Anh - 20225564
Phạm Minh Tiến - 20225555

HUST

Table of contents

Project 1:
Password Manager

1. Project 1: Password Manager

1.1 Motivation:

- **Problem:** Users often use weak passwords and find it difficult to remember many strong ones.
- **Solution:** A Password Manager helps store and retrieve strong, unique passwords for each service.
- **Challenge:** Ensuring security when storing sensitive data.
    -

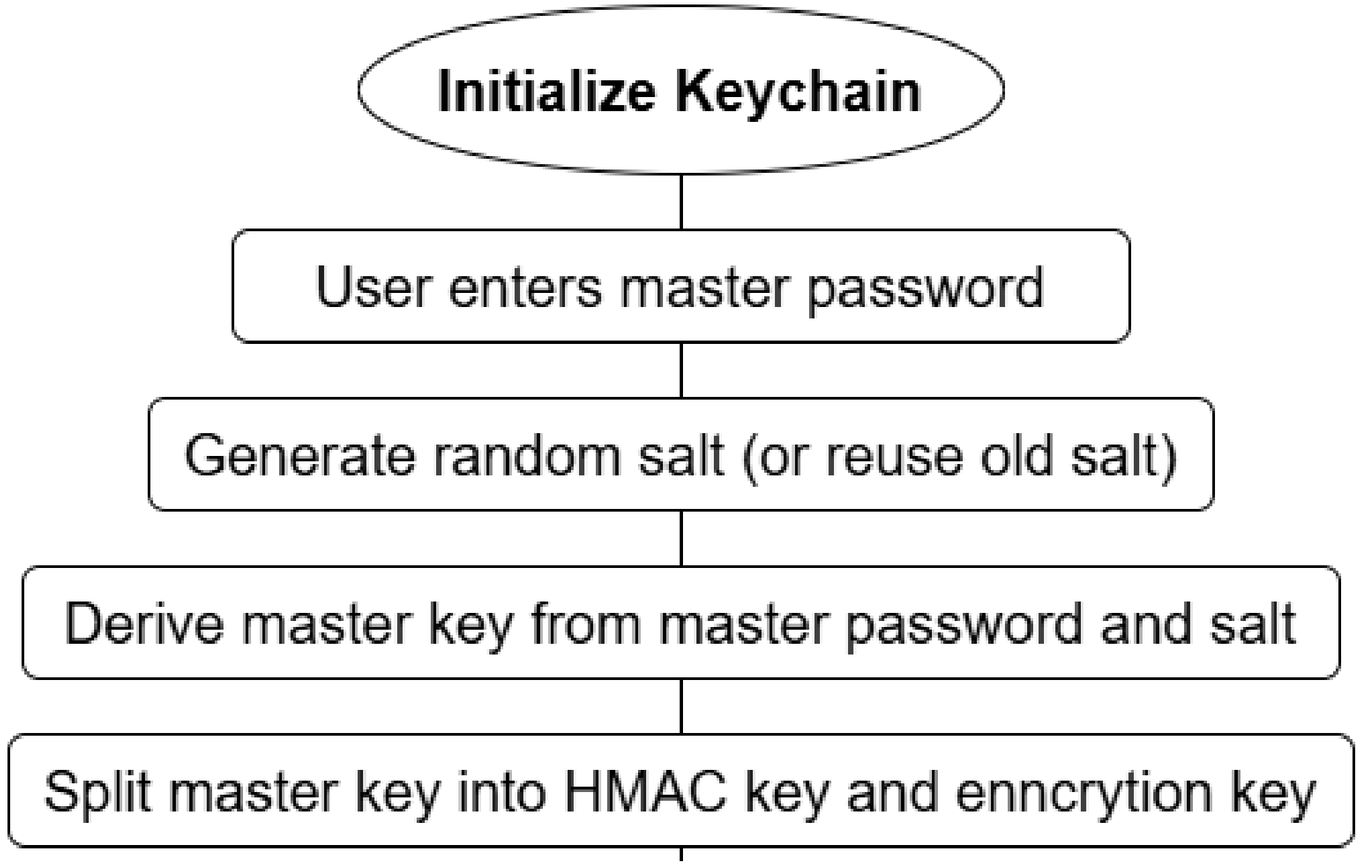
1. Project 1: Password Manager

1.1 Overview:

- **Object:** password manager
- **Motivation:**
- **Encryption and hashing methods:**
  - PBKDF2 + Salt: Anti brute-force, rainbow table.
  - HMAC domain: Hide real domain, anti guessing.
  - AES-GCM: Password encryption, anti modification.

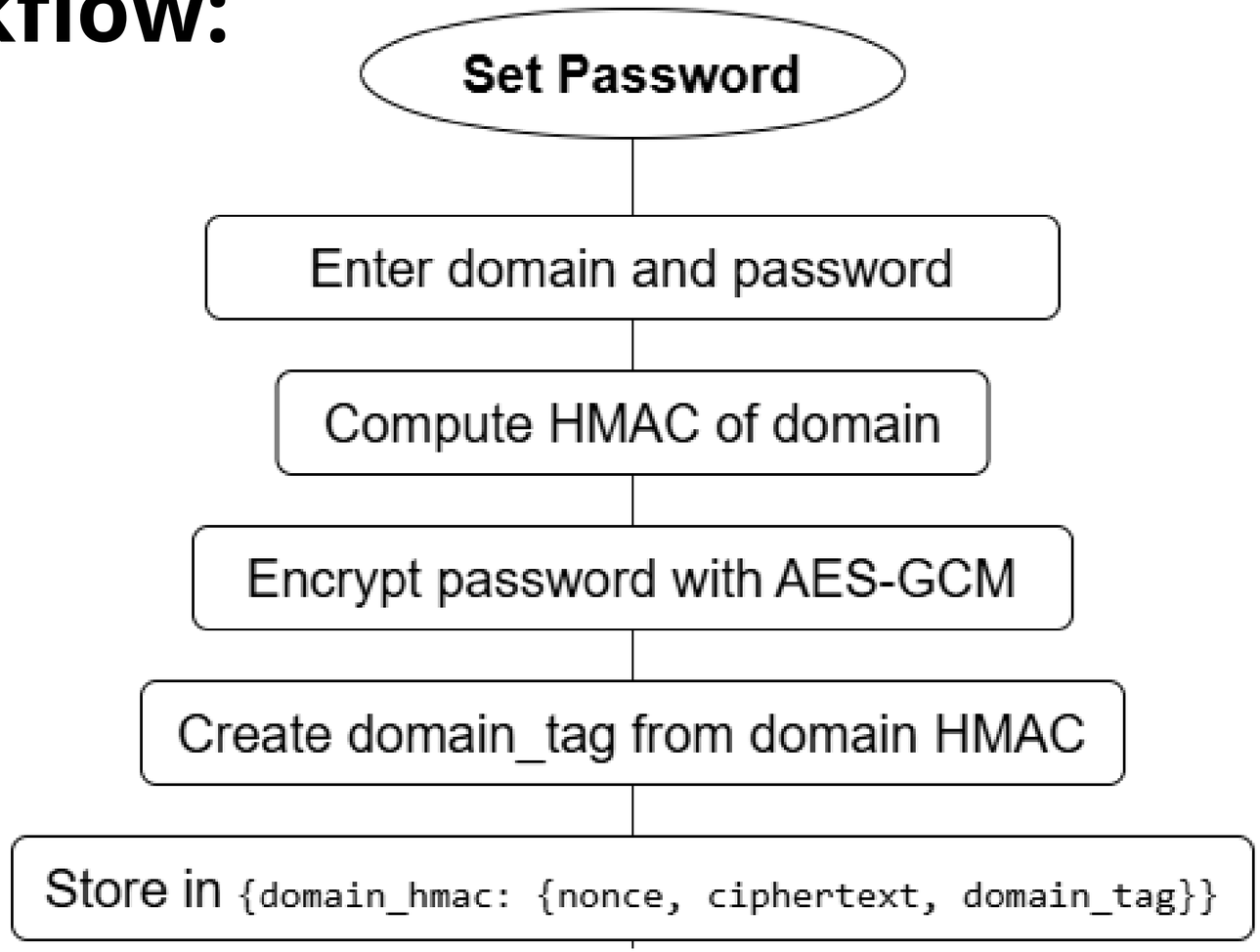
1. Project 1

1.2 Workflow:



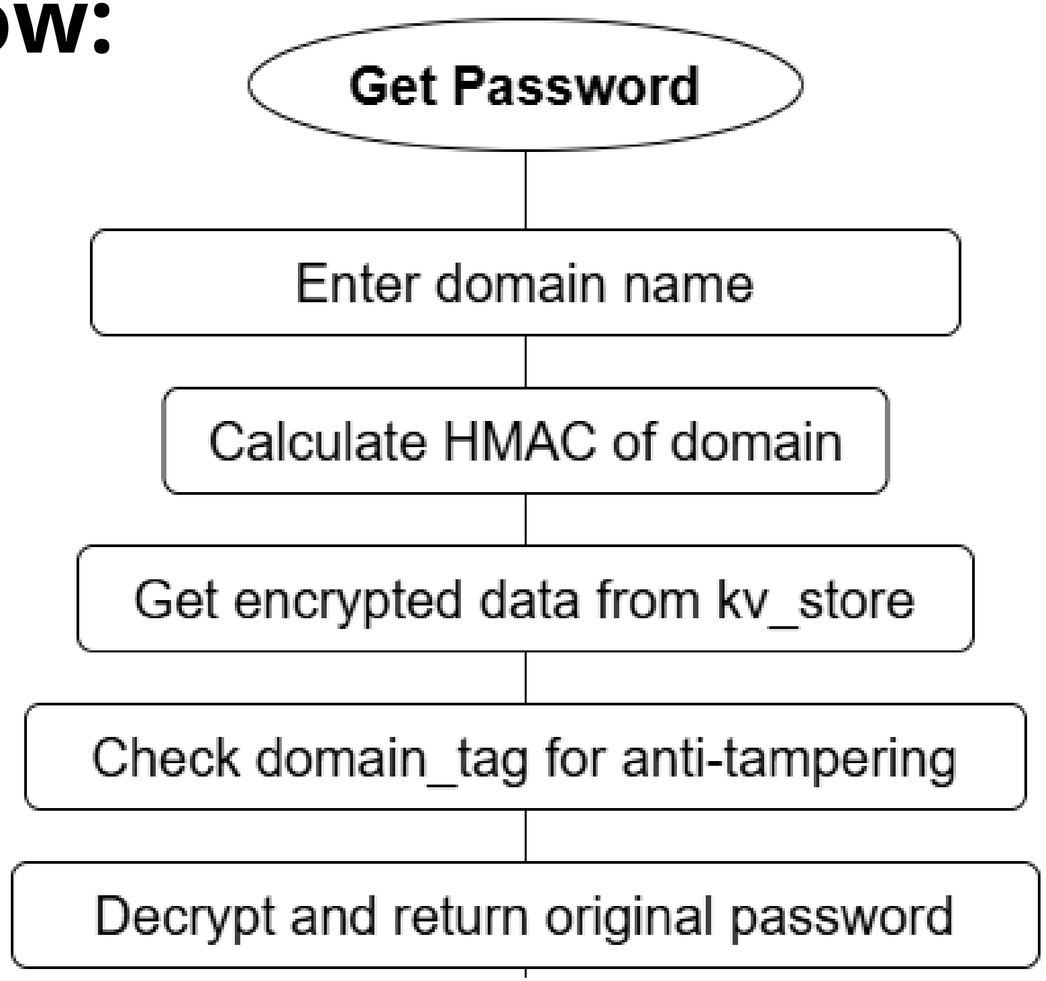
1. Project 1

1.2 Workflow:



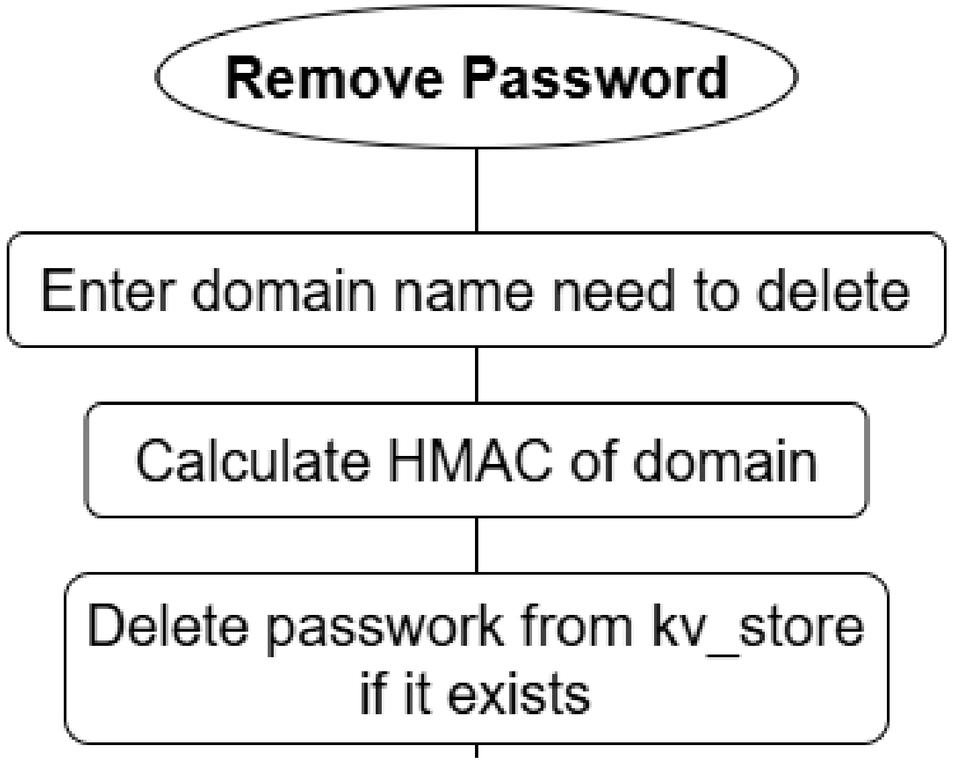
1. Project 1

1.2 Workflow:



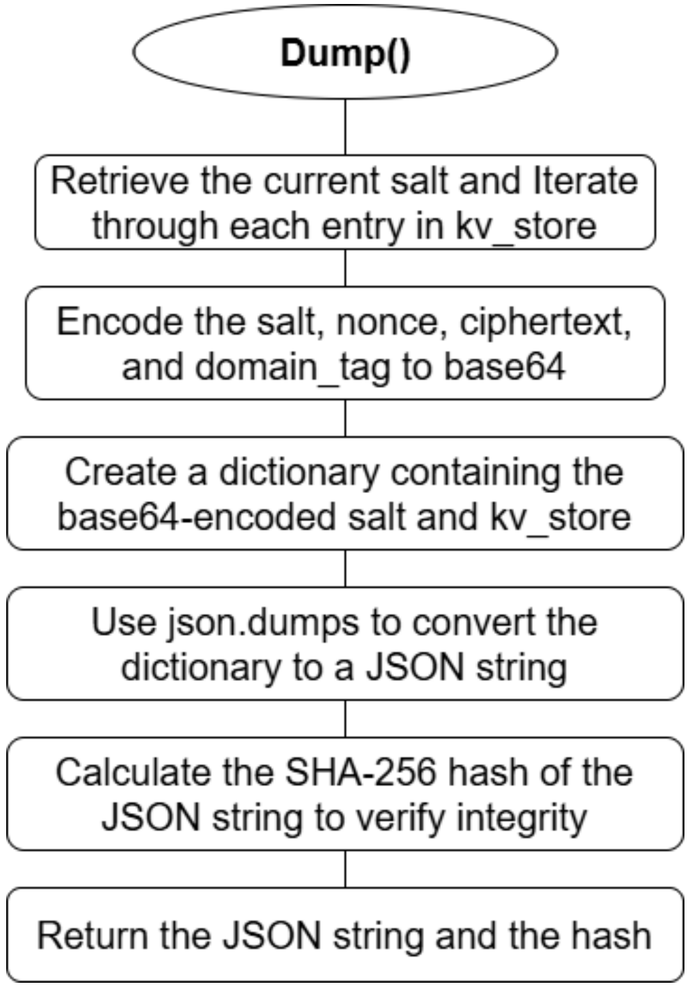
1. Project 1

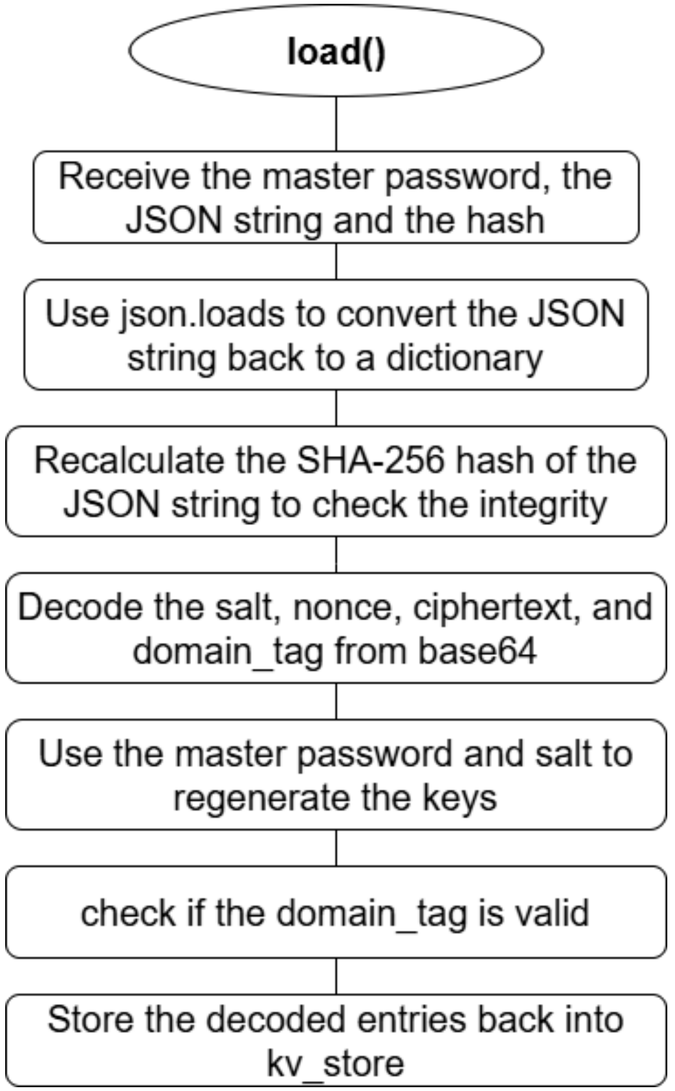
1.2 Workflow:



1. Project 1

1.2 Workflow:





1. Project 1: Password Manager

- **Store key- value**

```
self.kv_store = {} if kv_store is None else kv_store
```

```
{
    "Qk1vZ2F6d2VqR3p3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3": {
        "nonce": "...",
        "ciphertext": "...",
        "domain_tag": "..."
    },
    "Rk5vZ2F6d2VqR3p3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3b2t3": {
        "nonce": "...",
        "ciphertext": "...",
        "domain_tag": "..."
    }
}
```

1. Project 1: Password Manager

- **Domain name**

```
def _compute_domain_hmac(self, domain):
    ...
    domain_hmac = h.finalize()
    return base64.b64encode(domain_hmac).decode('utf-8')
```

- **Encrypt**

```
# Encrypt the password
aesgcm = AESGCM(self.encryption_key)
# Use domain_hmac as associated data to bind the ciphertext to the domain
ciphertext = aesgcm.encrypt(nonce, padded_password, domain_hmac.encode('utf-8'))
```

1. Project 1: Password Manager

- **Key derive**

```
def _derive_master_key(self, master_password):
    kdf = PBKDF2HMAC(...)
    master_key = kdf.derive(master_password.encode('utf-8'))
    return master_key
def _derive_sub_keys(self, master_key):
    ...
    hmac_key = h1.finalize()
    encryption_key = h2.finalize()
    return hmac_key, encryption_key
```

1. Project 1: Password Manager

- **Test**

```
PS C:\Users\Chu Trung Anh\Desktop\project1_ChuTrungAnh> pytest .\test_password_manager.py
============================= test session starts =============================
platform win32 -- Python 3.13.2, pytest-8.3.4, pluggy-1.5.0
rootdir: C:\Users\Chu Trung Anh\Desktop\project1_ChuTrungAnh
collected 11 items

test_password_manager.py ...........                                    [100%]

============================= 11 passed in 0.95s ==============================
```

1. Project 1: Password Manager

- **Swap attack**

```
# Verify the domain tag to prevent swap attacks
domain_tag = encrypted_data['domain_tag']
# Fix: Use the correct way to create HMAC with built-in hmac module
h = hmac_module.new(key=self.hmac_key, msg=domain_hmac.encode('utf-8'), digestmod='sha256')
computed_domain_tag = h.digest()

if not hmac_module.compare_digest(domain_tag, computed_domain_tag):
    raise ValueError("Swap attack detected: domain hash verification failed")
```

1. Project 1: Password Manager

Rollback attack

```
def dump(self):
    ...
    hash_value = self._compute_hash(serialized)
    return serialized, hash_value
def load(...):
    ...
    if trusted_data_check is not None:
        current_hash = Keychain._compute_hash(repr_str)
        if current_hash != trusted_data_check:
            raise ValueError("Integrity check failed: data may have been tampered with")
```

Project 2:
Chat Client

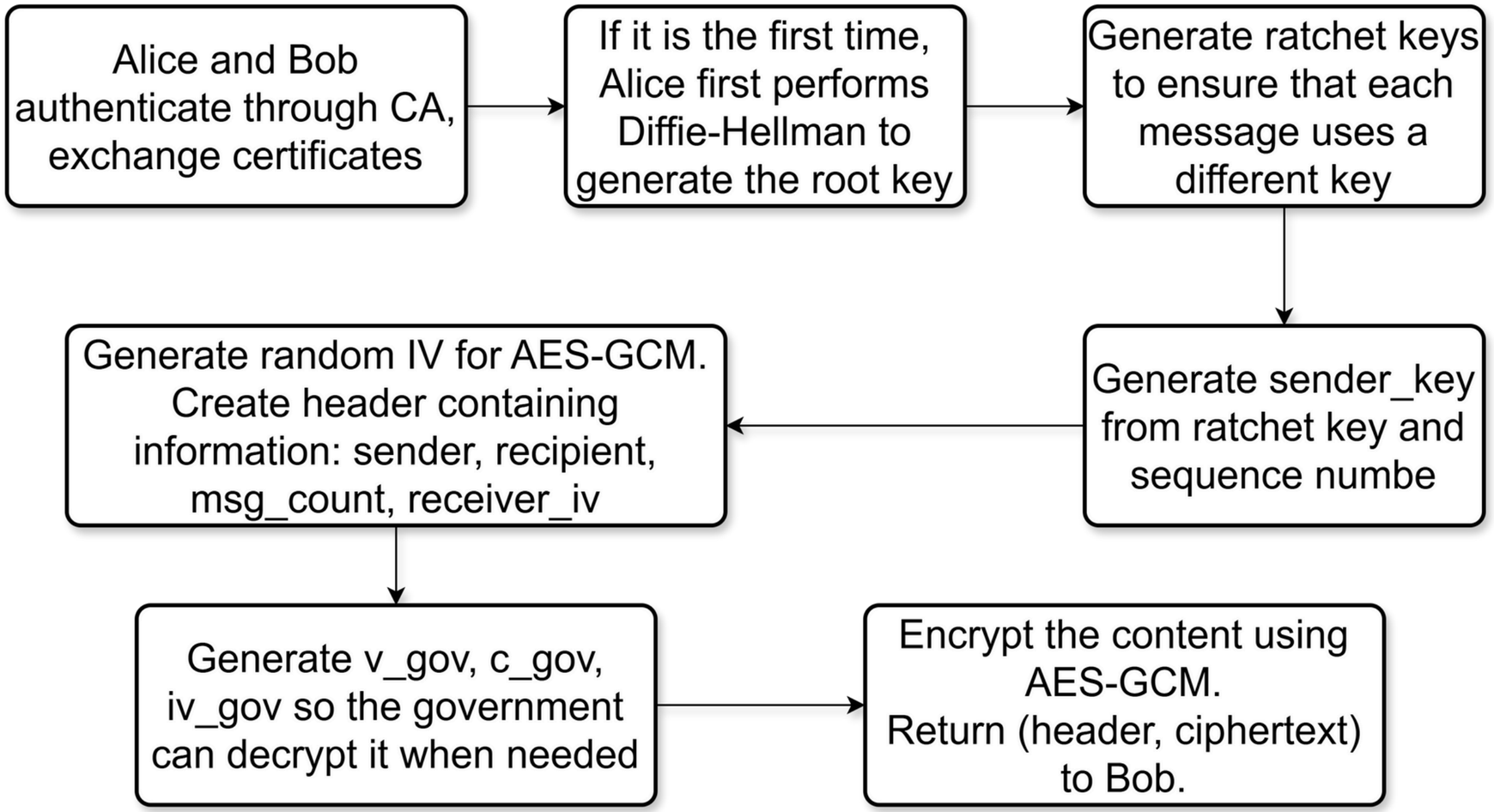
2. Project 2: Chat Client

2.1. Overview:

- **Object:** a secure messaging system between multiple users
- **Components:**
  - User (Messenger Client)
  - Certificate Authority (CA)
  - Government (Gov)
  - Cryptography library (lib.py)

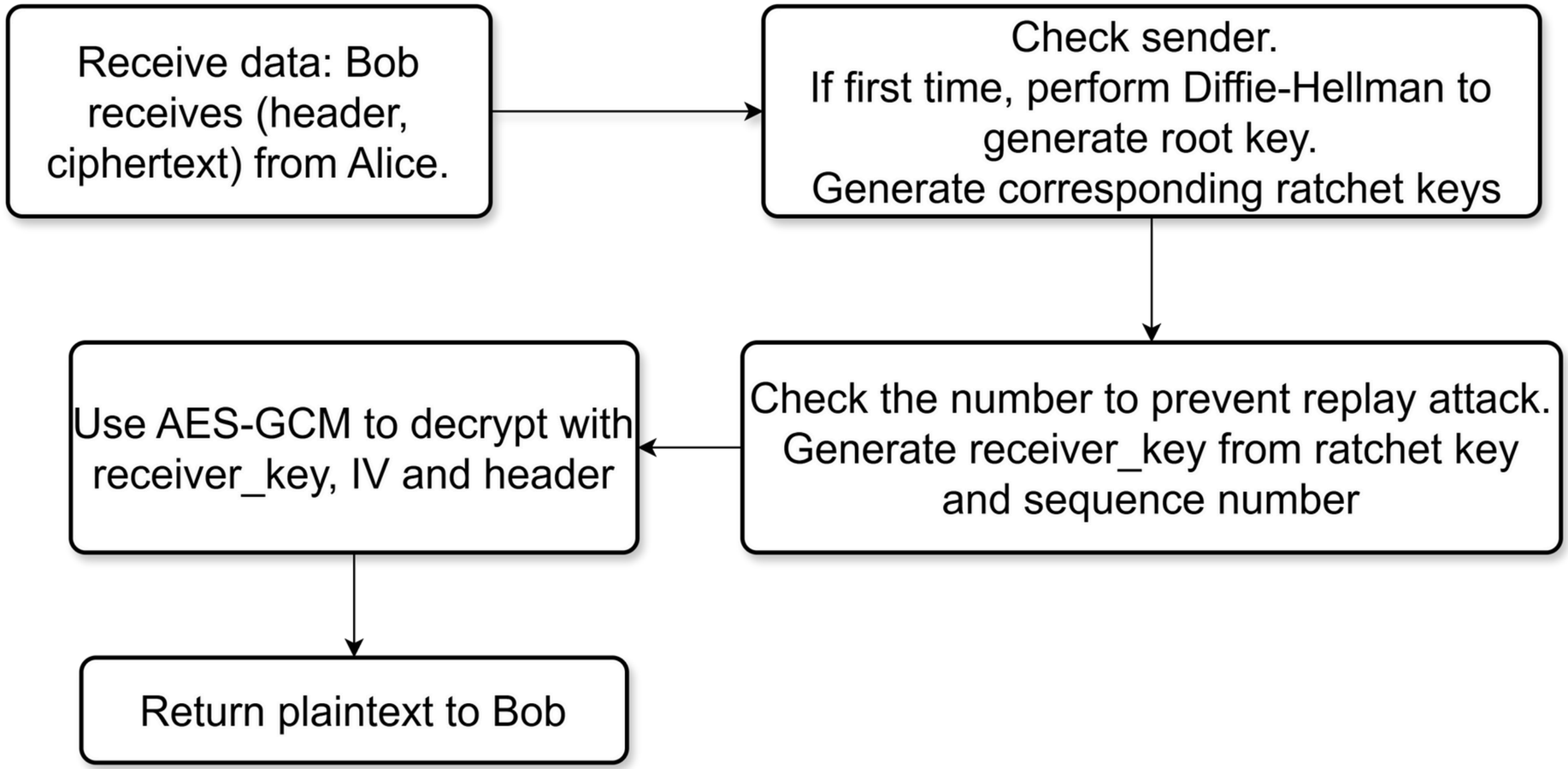
2. Project 2

2.2. Message Sending Process:



2. Project 2

2.3 Message Receiving Process:



2. Project 2

2.4. Cryptographic Techniques Used:

- **ECDSA:** Certificate authentication, ensuring no tampering.
- **Elliptic Curve DH:** Generate secure session keys, support forward secrecy.
- **HMAC & HKDF:** Derive session keys and ratchet keys from root key.
- **AES-GCM:** Message encryption and authentication (confidentiality + integrity).
- **Gov Escrow:** Header contains v_gov, c_gov, iv_gov allowing government to decrypt when needed.

Project 3:
P2P File Sharing

3. Project 3: P2P File Sharing

3.1. Overview:

- **Object:** Build a application allows users to log in, encrypt data, digitally sign files, and securely transfer files between computers over a network (can use ngrok to connect over the internet).
- **Fearture:**
  - File encryption to protect data.
  - Digital signature to authenticate file origin.
  - Supports multiple connection types: LAN, direct, ngrok tunnel.
  - User interface using tkinter.

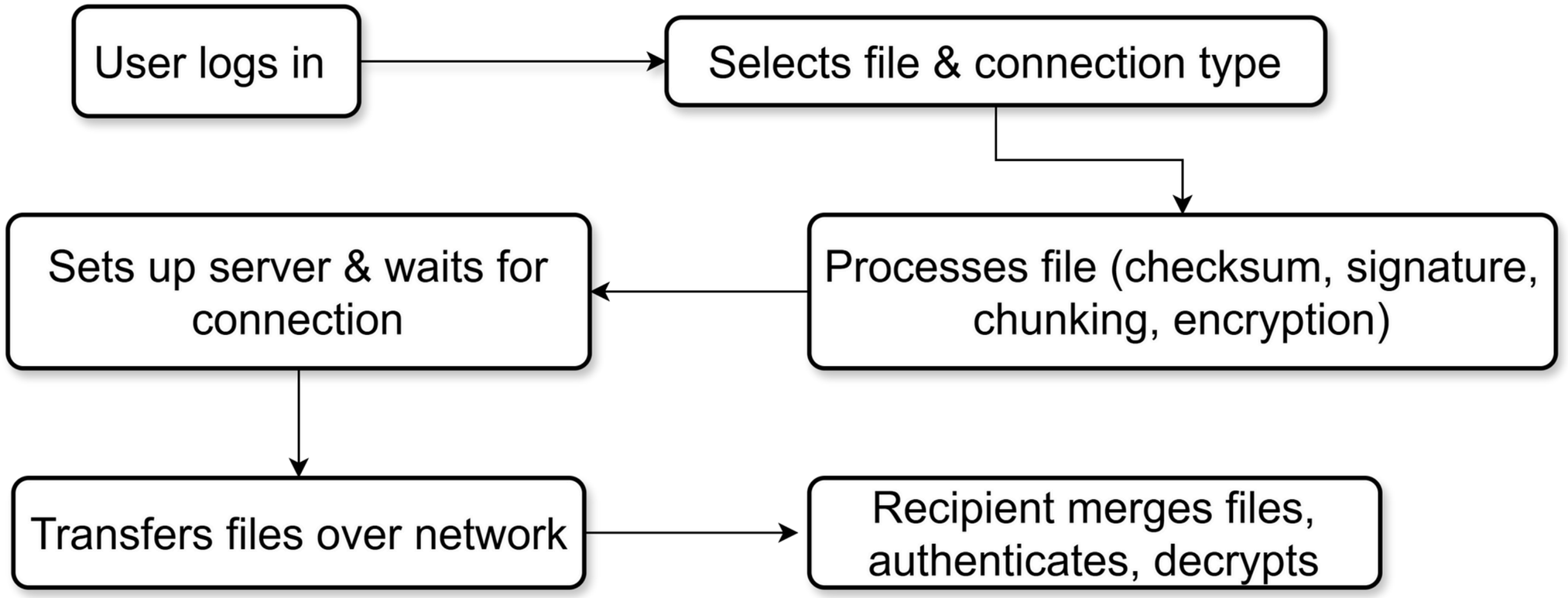
3. Project 3

3.2. Cryptographic Techniques Used:

- **AES (symmetric):** Encrypts file contents.
- **RSA (asymmetric):** Encrypts AES keys for secure transmission.
- **Elliptic Curve (digital signature):** Sign and authenticate files, ensuring integrity and origin authentication.

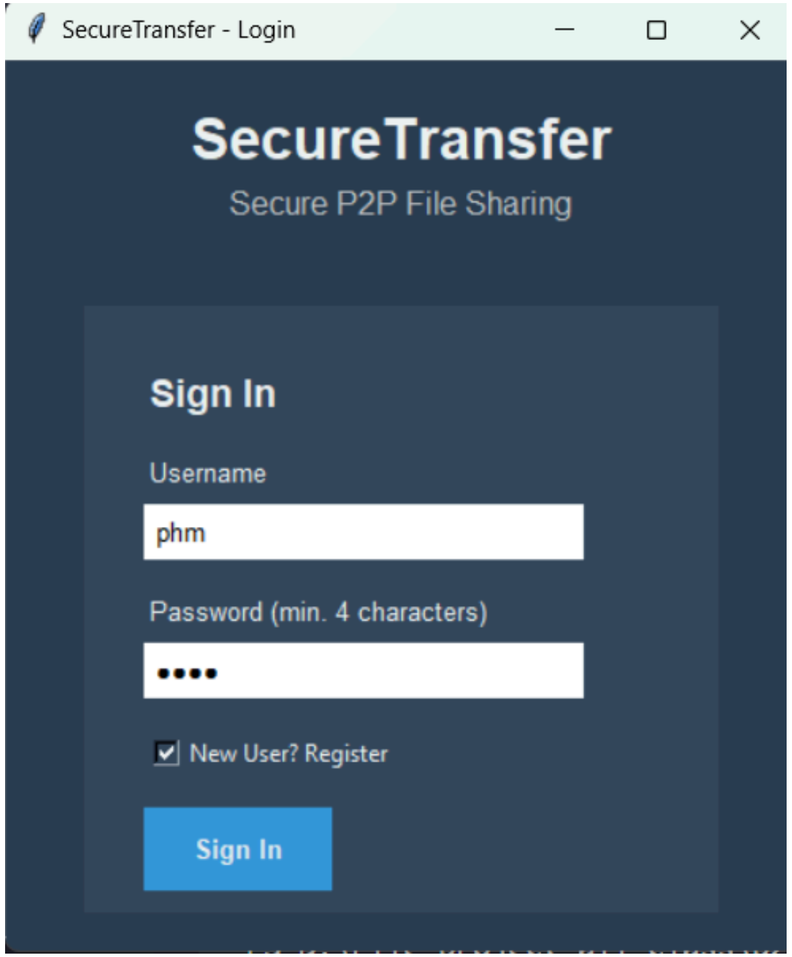
3. Project 3

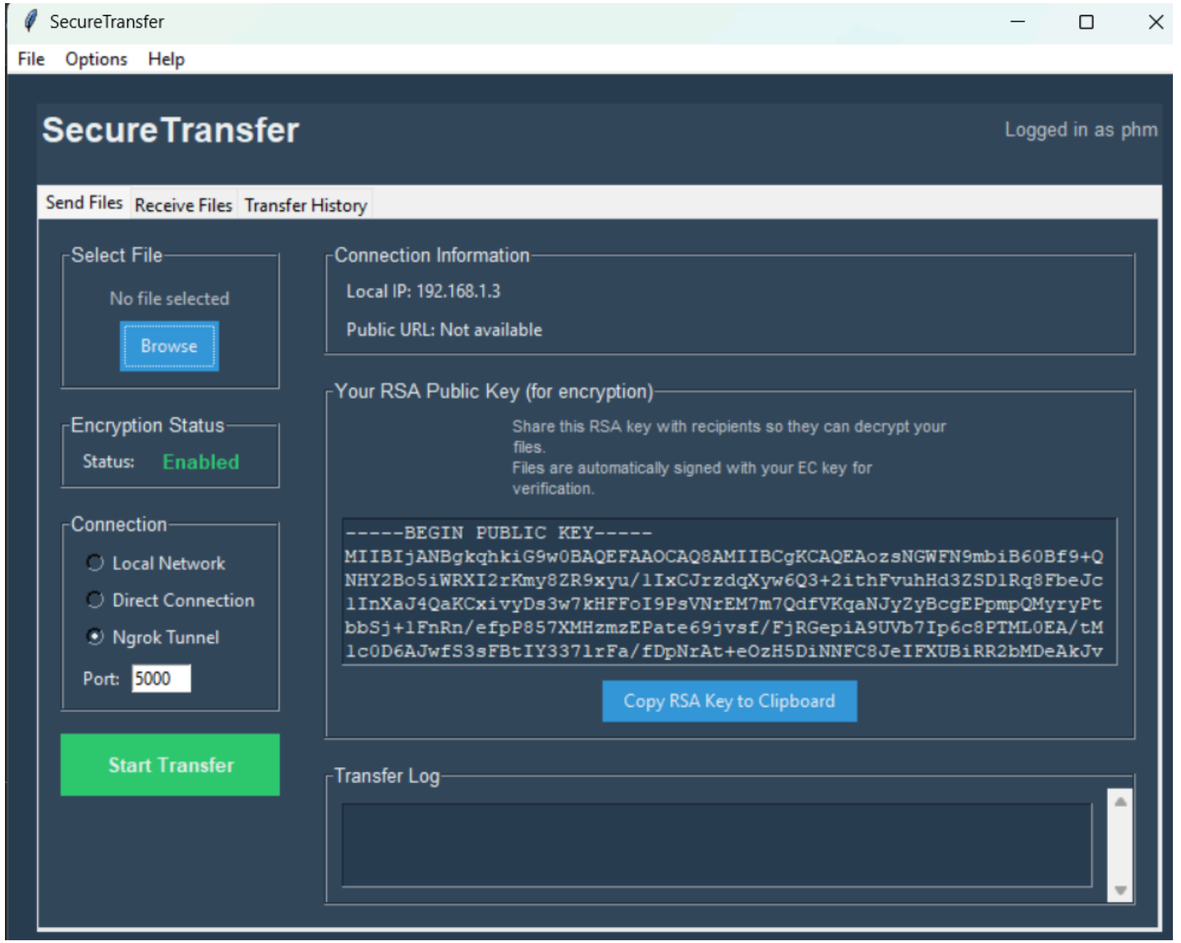
3.3. Workflow:



3. Project 3

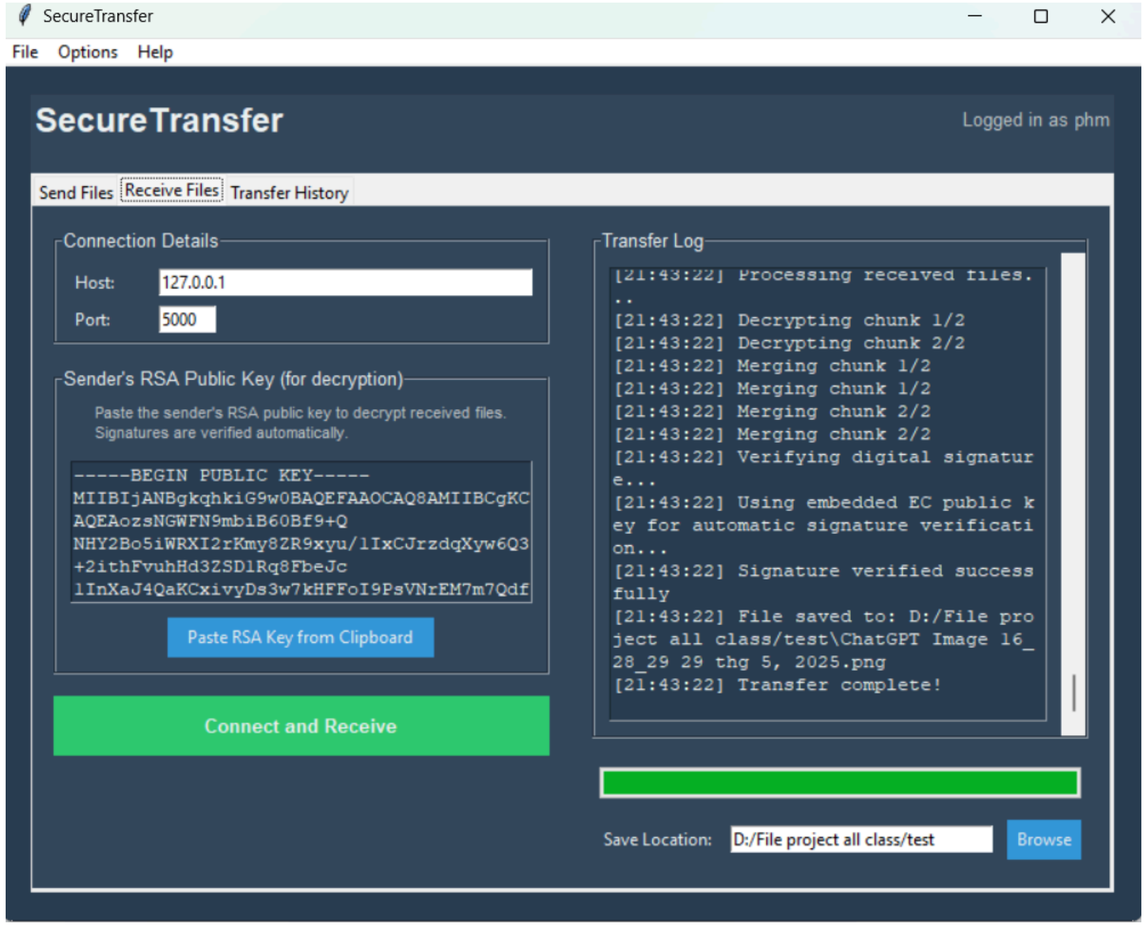
3.4. UI:

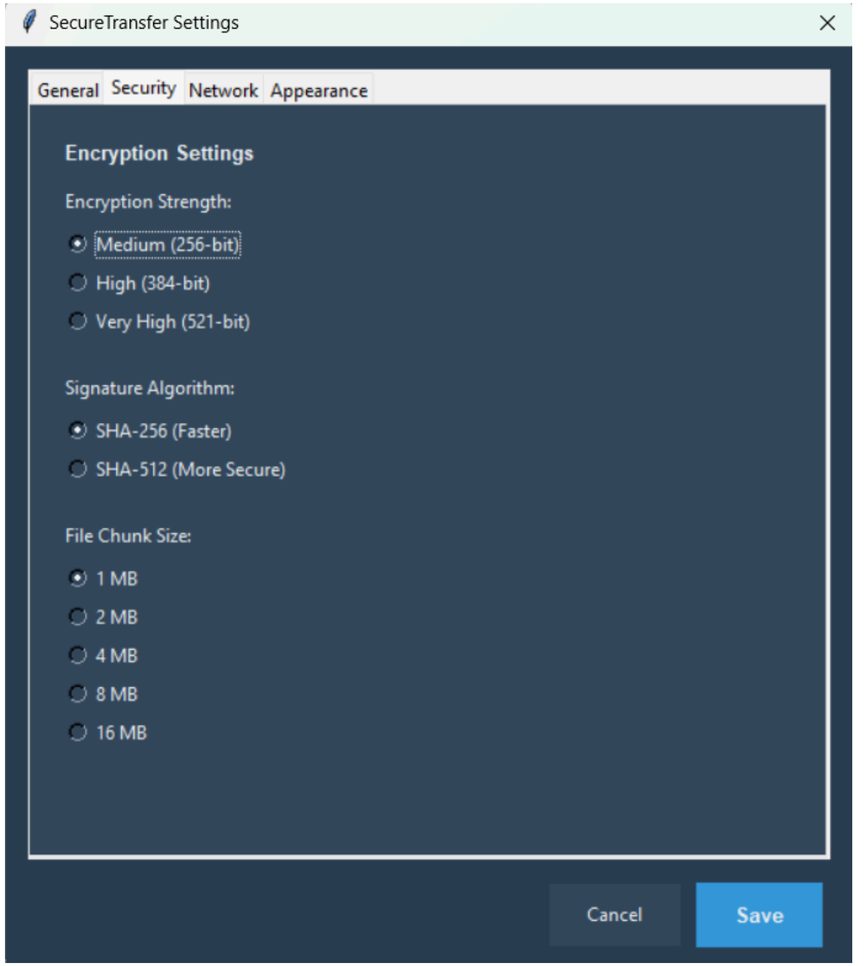




3. Project 3

3.4. UI:





HUST

THANK YOU !