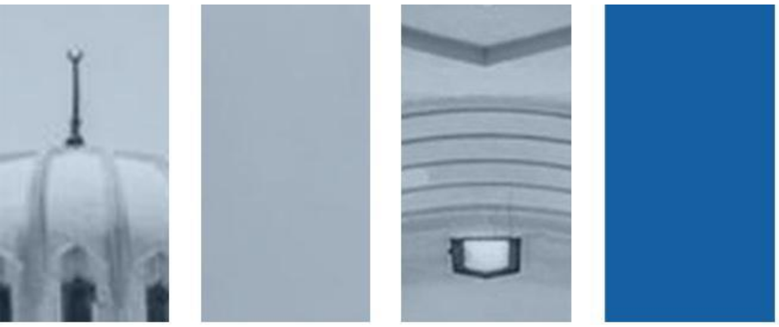


CipherShare: Basic P2P File Transfer & Unencrypted Sharing

Program: CESS

Course Code: CSE 451

Course Name: Computer and Network Security

Phase 2

Submitted by:

Team 19

Ahmad Sabry Issa	**17P8030**
Engy Ahmed Mostafa	**20P4230**
Omama Mohammed Alnajjar	**18P2797**
Mahmoud Maged Hafez	**18P2847**

Submitted to:

Dr. Ayman Bahaa

Ta. Rahma Hussein Ghouneim

Table of Contents

Table of figures:

1. Abstract:

CipherShare is a secure, modular, and decentralized peer-to-peer (P2P) file sharing platform built in Python. It enables users to register, authenticate, and securely share files without relying on a centralized server. CipherShare emphasizes strong cryptographic protections, including salted password hashing, session management, and plans for AES file encryption and integrity checks. This report thoroughly details all design decisions, architecture, and implementation progress up to Phase 2, including challenges, diagrams, detailed coding breakdowns, and a robust user manual.

2. Introduction:

File sharing systems have become an integral part of modern communication and collaboration, yet many suffer from weak security practices, centralization risks, and limited user control. CipherShare addresses these concerns by offering a distributed, secure platform where each peer operates independently and securely.

The project simulates a realistic file-sharing scenario where multiple users can share, upload, and download files over a network while preserving user privacy. It prioritizes secure user registration and login procedures, protecting passwords using modern cryptographic algorithms and defending against common attack vectors like brute-force, session hijacking, and file tampering.

CipherShare also promotes education by using modular Python code, clear separation of concerns, and transparent session management. It provides an environment where cryptography, network programming, and system design converge to build a real-world system.

This report documents the entire process: from the vision to code, detailing every feature implemented and decisions made throughout Phase 1 and Phase 2, while outlining the roadmap for Phase 3.

3. Project Scope

CipherShare aims to provide a secure, scalable, and decentralized peer-to-peer (P2P) platform for file sharing. The scope includes:

- Building a socket-based P2P architecture
- Implementing user registration and login with credential protection
- Enabling secure file sharing only for authenticated users
- Planning for future extensions with encrypted file transmission and access control policies
- Creating a modular codebase that is easy to extend, test, and maintain

CipherShare is implemented in four phases:

- **Phase 1:** Unsecured file sharing (basic networking)
- **Phase 2:** Secure login and session control
- **Phase 3:** Encryption and data integrity verification
- **Phase 4:** Access control, metadata security, peer discovery enhancements

4. Project Goals

The core objectives of CipherShare are:

- To enable secure file exchange among decentralized users
- To protect user credentials using secure, salted hashing algorithms
- To ensure that only authenticated users can perform file operations
- To minimize dependencies while using secure cryptographic standards
- To demonstrate a strong understanding of network security and P2P design

5. Security Threats and Considerations

Threat	Description	Mitigation
Credential theft	Attacker reads raw password data	Use strong hashing + random salt
Brute-force attack	Repeated login attempts	Use Argon2id/PBKDF2 and generic error replies
Session hijacking	Reuse of old or leaked sessions	Bind sessions to live sockets only
Unauthorized file access	Download/upload without auth	All file actions require session check
File tampering (future)	Files modified in transit	SHA-256 hashes (planned in Phase 3)

CipherShare uses Argon2id for hashing when available, with a fallback to PBKDF2. This ensures secure password handling even on systems lacking Argon2id support.

6. System Architecture Diagram and Explanation

The following diagram visually represents the overall system architecture of the CipherShare platform as of Phase 2:

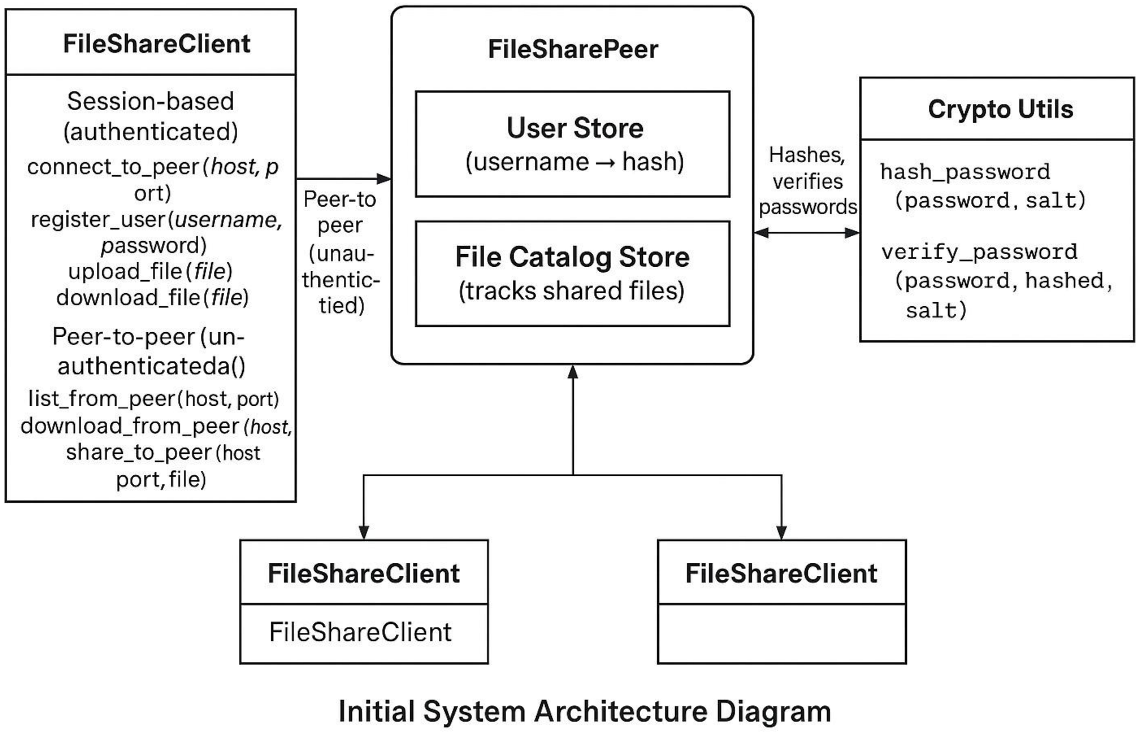

FileShareClient
Session-based (authenticated)
connect_to_peer(host, port)
register_user(username, password)
upload_file(file)
download_file(file)
Peer-to-peer (un-authenticateda()
list_from_peer(host, port)
download_from_peer(host,
share_to_peer(host
port, file)
Peer-to peer (unau-thentic-tied)
FileSharePeer
User Store (username → hash)
File Catalog Store (tracks shared files)
Hashes, verifies passwords
Crypto Utils
hash_password (password, salt)
verify_password (password, hashed, salt)
FileShareClient
FileShareClient
FileShareClient
Initial System Architecture Diagram

Figure 1 architecture digram

6.1. Components Description:

1. FileShareClient

- Provides a CLI-based interface for the user.
- Contains session-based operations (e.g., login, upload, download) and stateless peer-to-peer operations (e.g., shareto, peerdl).
- Establishes socket connections with one or more FileSharePeer nodes.

2. FileSharePeer

- Acts as the core node in the P2P network.

- Maintains two critical data stores:
    - **User Store**: Maps usernames to securely hashed passwords (and salts).
    - **File Catalog Store**: Tracks all shared files and makes them accessible to authenticated users.
- Performs authentication and access control through login sessions.
- Verifies credentials by consulting Crypto Utils.

3. Crypto Utils

- Implements secure password handling using Argon2id or PBKDF2.
- Responsible for:
    - hash_password(password, salt) – returns a hash and salt.
    - verify_password(password, hashed, salt) – authenticates a password by comparing a derived hash to a stored hash.

4. Peer-to-Peer Communication Paths:

- **Session-Based (Authenticated)**: These operations require login and include upload, download, and list.
- **Stateless (Unauthenticated)**: Includes shareto, peerdl, and peerlist, which involve direct file operations between clients and peers without requiring login.

This architecture enforces secure, direct, and modular communication between peers and clients. It ensures that user data and file sharing operations are protected via cryptographic methods, while also supporting flexible, decentralized file exchange.

6.2. Connection Flow Details:

- When a FileShareClient executes the connect command, it initiates a TCP connection to a specified FileSharePeer. This connection persists and becomes the channel through which commands like register, login, upload, download, and list are transmitted.
- If a user registers, the client sends the credentials, which are processed by the peer using hash_password from Crypto Utils. The resulting hash and salt are stored.
- During login, credentials are verified using verify_password. A session is then created by storing the client socket mapped to the username.
- Any authenticated file operation must originate from the same session socket. If a command is issued without an active session, the peer responds with ERR login_required.

- Stateless commands (shareto, peerlist, peerdl) create new short-lived TCP connections to peers. Since these operations do not authenticate or maintain a session, they are suitable for public file sharing scenarios where access control is relaxed.
- The watchpeer command periodically connects to a peer and lists any new files available, effectively polling the file catalog store without requiring session login.

This detailed connection model allows CipherShare to balance security and flexibility, using persistent authenticated sessions for secure operations, and transient connections for anonymous interactions.

7. Phase 1 Summary

Implemented Features:

- Socket-based server and client communication
- File upload and download operations between peers
- Basic peer discovery and manual IP/port connection
- Shared file directory creation and usage on peer side

Challenges:

- Maintaining consistent and reliable TCP connections
- Handling file I/O operations and stream interruptions
- Ensuring compatibility across different OS environments
- Creating a command-line interface that is user-friendly and robust

Phase 1 created a foundational P2P infrastructure that enabled file transfers in a controlled environment, preparing the ground for secure communication in Phase 2.

8. Phase 2

Phase 2 introduced critical security upgrades to the CipherShare platform by implementing secure user registration and login features, password hashing using modern cryptographic standards, and an in-memory session control system.

The authentication system in Phase 2 ensures that only verified users can access file-sharing functionality. It introduces the concept of stateful client sessions, tying each connected socket to a verified username. This prevents unauthorized access and enforces access control on all sensitive commands like upload, download, and list.

Users register with a username and password. The password is hashed with either Argon2id or PBKDF2 and stored with a salt in a persistent JSON file. The server then validates users at login by verifying the password hash using the stored salt.

Each successful login initializes a session. All file-sharing commands check this session using a lightweight validation function. When the user disconnects or the session is reset, access is denied until a new login is performed.

This authentication layer lays the foundation for the encryption and verification systems to come in Phase 3.

8.1. Hashing Algorithm Design

CipherShare supports two main password hashing approaches:

8.1.1. Argon2id (Preferred)

Argon2 is a memory-hard key derivation function recommended by OWASP. The following parameters are used:

- Time Cost: 4 iterations
- Memory Cost: 64 MB (2**16)
- Parallelism: 2
- Hash Length: 32 bytes (suitable for AES-256 keys if needed in future phases)

Advantages:

- Highly secure against GPU-based brute-force attacks
- Customizable memory/time tradeoffs

8.1.2. PBKDF2 (Fallback)

If Argon2id is not available on the executing environment, CipherShare uses PBKDF2 with the following:

- Hash Algorithm: SHA-256
- Iterations: 200,000
- Salt Size: 16 bytes (128 bits)
- Derived Key Length: 32 bytes

PBKDF2 is widely supported and offers secure derivation, albeit with less resistance to memory-bound attacks.

All password hashes are stored in users.json as hex-encoded strings. The system never stores or logs plain text passwords.

Password Storage Example

```
{
  "alice": [
    "a71cfcf183f7e23a909fa25c66...",
    "7f3a0b6a6f4b2ffcd89312e3a4..."  ]
}
```

First element: hashed password; second element: salt

8.1.3. Session Management Design

CipherShare enforces strict login-based access control. A temporary session is established in-memory per client socket.

Session Tracking Model

```
self.sessions = {conn: username}
```

- conn: The socket object representing a user connection
- username: The currently authenticated user

Session Lifecycle

- On login, a session is created and tied to the socket.
- On logout or disconnect, the session is destroyed.
- Each file operation checks for the presence of a valid session.

Security Considerations

- Sessions are not tokenized or stored across reboots for simplicity.
- No sensitive session data is persisted.
- File access is immediately blocked if the user is not logged in.

Session Validation

Each server-side operation calls this utility:

```
def _check_login(self, conn):
    if self.sessions.get(conn):
        return True
    conn.sendall(b"ERR login_required\n")
    return False
```

8.1.4. Benefits of this Design

- Simple but effective access control mechanism
- Easily scalable for socket-based peer interaction
- Modular and isolated from other logic

8.2. Code walkthrough

Below is a step-by-step breakdown of the core code segments involved in authentication and session control in Phase 2.

1. **Registration Command Handler (Server-side: fileshare_peer.py)**

```
def _register(self, conn, parts):
    if len(parts) != 3:
        conn.sendall(b"ERR format\n")
        return
    username, password = parts[1], parts[2]
    if username in self.users:
        conn.sendall(b"ERR exists\n")
        return
    hashed_pw, salt = hash_password(password)
    self.users[username] = [hashed_pw.hex(), salt.hex()]
    self._save_users()
    conn.sendall(b"OK registered\n")
```

- Validates registration format
- Prevents duplicate usernames
- Uses the hash_password() function to hash password and generate salt
- Saves to the persistent user file

2. Login Handler (Server-side)

```
def _login(self, conn, parts):
    if len(parts) != 3:
        conn.sendall(b"ERR format\n")
        return
    username, password = parts[1], parts[2]
    if username not in self.users:
        conn.sendall(b"ERR no_user\n")
        return
    stored_hash = bytes.fromhex(self.users[username][0])
    stored_salt = bytes.fromhex(self.users[username][1])
    if verify_password(password, stored_hash, stored_salt):
        self.sessions[conn] = username
        conn.sendall(b"OK welcome\n")
    else:
        conn.sendall(b"ERR bad_pwd\n")
```

3. Password Hashing Utility (Client/Server Shared - crypto_utils.py)

```
• def hash_password(password, salt=None):
      if salt is None:
          salt = secrets.token_bytes(16)
      pw = password.encode()
      kdf = _argon2_or_none(pw, salt)
      if kdf:
          return kdf.hash(pw), salt
      return hashlib.pbkdf2_hmac("sha256", pw, salt, 200_000, dklen=32),
  salt
```

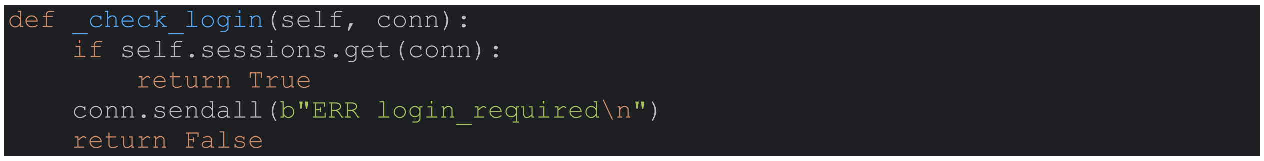
- Generates or accepts a salt
- Uses Argon2 if available, otherwise falls back to PBKDF2

4. Password Verification Function

```
def verify_password(password, hashed, salt):
    pw = password.encode()
    kdf = _argon2_or_none(pw, salt)
    if kdf:
        try:
            kdf.verify_hash(hashed, pw)
            return True
        except Exception:
            return False
    test = hashlib.pbkdf2_hmac("sha256", pw, salt, 200_000, dklen=32)
    return hmac.compare_digest(test, hashed)
```

- Compares entered password with stored credentials
- Uses constant-time comparison for security

5. Session Management Code (Peer-side)



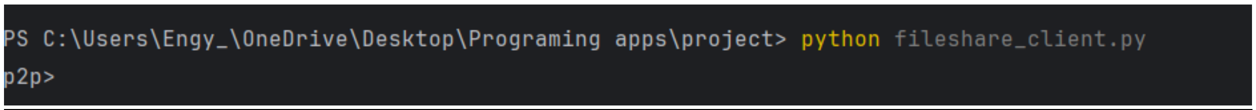
```
def _check_login(self, conn):
    if self.sessions.get(conn):
        return True
    conn.sendall(b"ERR login_required\n")
    return False
```

- Used before allowing access to sensitive commands like UPLOAD, DOWNLOAD, LIST

9. User Manual and CLI Usage Guide

To launch the client interface:

python fileshare_client.py



```
PS C:\Users\Engy_\OneDrive\Desktop\Programing apps\project> python fileshare_client.py
p2p>
```

Figure 2client listen

9.1. Available Commands:

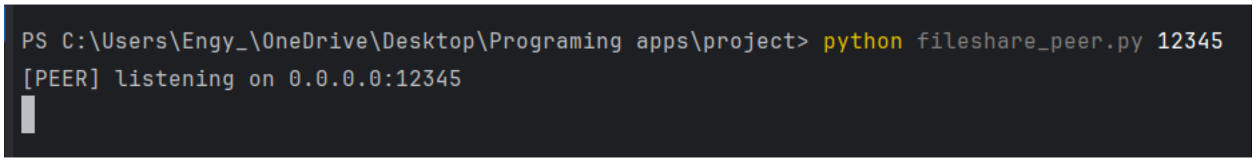
Command	Description
connect <host> <port>	Connect to a peer server
register <username> <password>	Create a new user account securely
login <username> <password>	Log in to an existing user account
upload <filepath>	Upload a file to the connected peer (must be logged in)
download <filename>	Download a file from the peer (must be logged in)
list	View the list of files shared by the peer
shareto <host> <port> <filepath>	Share a file directly to another peer (stateless)
peerlist <host> <port>	View shared files from a peer (stateless)
peerdl <host> <port> <filename>	Download a file statelessly from a peer

watchpeer <host> <port>	Monitor file changes on a peer
quit	Exit the client program

9.2. Walkthrough:

Launch the Peer Servers

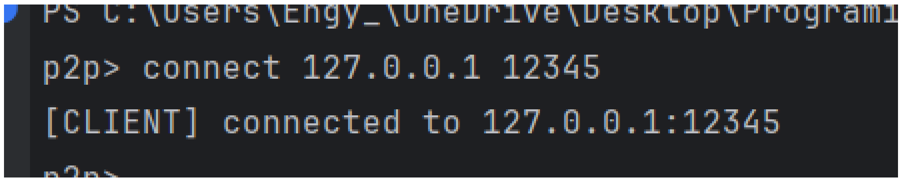
python fileshare_peer.py 12345



```
PS C:\Users\Engy_\OneDrive\Desktop\Programing apps\project> python fileshare_peer.py 12345
[PEER] listening on 0.0.0.0:12345
```

Figure 3 first peer connected

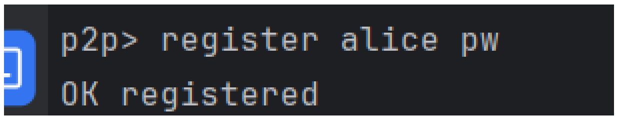
p2p> connect 127.0.0.1 12345



```
PS C:\Users\Engy_\OneDrive\Desktop\Programi
p2p> connect 127.0.0.1 12345
[CLIENT] connected to 127.0.0.1:12345
```

Figure 4 connected to client

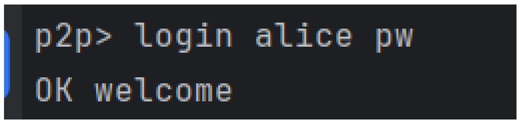
p2p> register alice secret123



```
p2p> register alice pw
OK registered
```

Figure 5 regiser new user and password

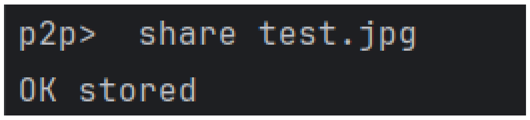
p2p> login alice secret123



```
p2p> login alice pw
OK welcome
```

Figure 6 login new user

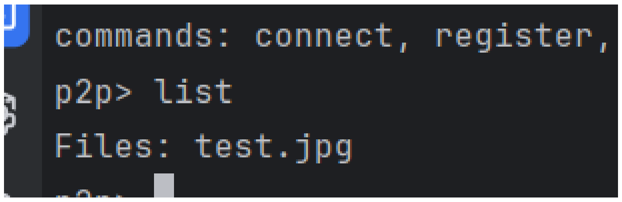
p2p>share test .jpg



```
p2p>  share test.jpg
OK stored
```

Figure 7 share file

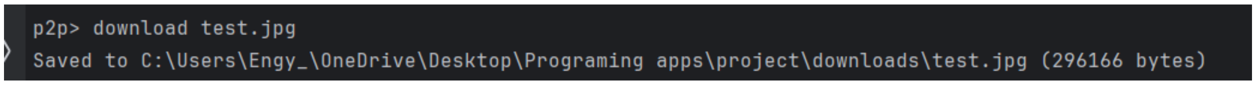
p2p> list



```
commands: connect, register,
p2p> list
Files: test.jpg
```

Figure 8 file in lists

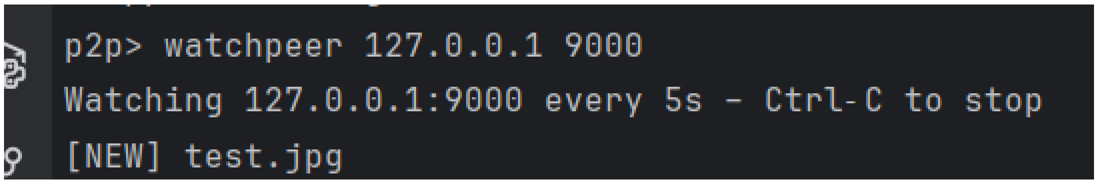
p2p> download test.jpg



```
p2p> download test.jpg
Saved to C:\Users\Engy_\OneDrive\Desktop\Programing apps\project\downloads\test.jpg (296166 bytes)
```

Figure 9 download file

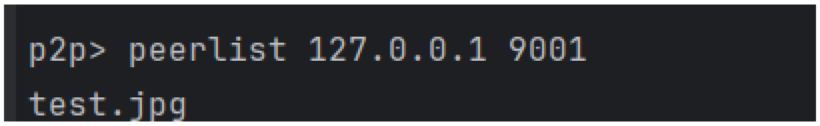
p2p> watchpeer 127.0.0.1 9000



```
p2p> watchpeer 127.0.0.1 9000
Watching 127.0.0.1:9000 every 5s - Ctrl-C to stop
[NEW] test.jpg
```

Figure 10view live share

p2p> peerlist 127.0.0.1 9001



```
p2p> peerlist 127.0.0.1 9001
test.jpg
```

Figure 11peer lists shared

10. Testing Strategy

- **Authentication:**
    - Attempt login with invalid credentials
    - Reuse a session after disconnect
    - Register duplicate usernames
- **Credential Security:**

- Check hash and salt values in users.json
- Confirm passwords are not stored in plaintext

- **Session Management:**
  - Try uploading/downloading before login
  - Check session destruction on disconnect
- **Peer File Operations:**
  - Test upload/download with large files
  - Use shareto and peerdl from another terminal to simulate multiple nodes

11. Phase 3 Plan: Encryption and File Integrity

Phase 3 will build on the authentication system by introducing file encryption and integrity verification.

- **Encryption:**
  - AES-256 symmetric encryption of file contents
  - Encrypt file client-side before upload
  - Generate per-file IVs and symmetric keys
- **Integrity:**
  - Compute SHA-256 hash during upload
  - Validate hash after download
  - Store hash metadata alongside files
- **Key Management:**
  - Derive keys from user password or use random key per session
  - Encrypt key with user password or public key (if implemented)

Commands will include:

- encryptupload <file>
- verifydownload <file>

12. GitHub Respiratory

https://github.com/AhmadSabryIssa/CipherShare

13. Conclusion

CipherShare is now a fully operational secure P2P file sharing system through Phase 2. It provides robust authentication, credential protection, and access control via session enforcement. The groundwork for file encryption and hash verification is complete, setting the stage for Phase 3.

This project not only teaches secure software design but also illustrates how cryptography and distributed systems can work together in practice. With continued development, CipherShare can evolve into a complete, secure, and educational open-source platform for modern file sharing.