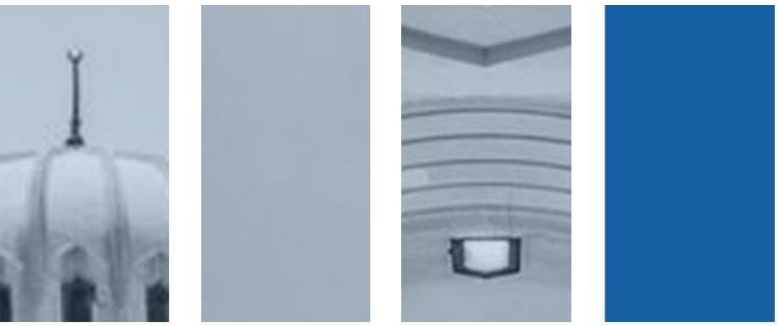


CipherShare: Basic P2P File Transfer & Unencrypted Sharing

Program: CESS

Course Code: CSE 451

Course Name: Computer and Network Security

Phase 1 Report

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

Table of Contents

1. Overview: 2
2. Implemented features: 2
2.1. Peer-to-Peer Architecture: 2
2.2. User Registration and Login: 2
2.3. File Upload: 3
2.4. File Download: 3
2.5. File Listing: 3
2.6. Stateless Peer-to-Peer Sharing: 4
2.7. Peer File Change Monitoring: 4
2.8. Interactive Command-Line Interface: 4
3. Architecture Diagram: 5
4. Challenges Encountered: 5
4.1. Stateless Session Handling: 5
4.2. Protocol Sipmlicity vs Robustness: 6
4.3. Manual Socket and Thread Management: 6
4.4. Peer Discovery Limitations: 6
4.5. Lack of Encryption & Integrity Checks: 6
5. Technologies Used: 7
6. Phase 2 Plan: 7
6.1. Improvements: 7
6.2. Tasks: 7
7. Basic User Manual: 8

1. Overview

Phase 1 establishes the core of **CipherShare**, a peer-to-peer (P2P) file sharing system. The current milestone enables users to connect to peer nodes, register/login, and share or retrieve files without encryption or fine-grained access control.

---

2. Implemented Features

2.1 Peer-to-Peer Architecture

Peers act as both servers and clients:

- A peer node listens on a port and handles multiple incoming connections using threads.
- Each peer maintains a catalog of users and shared files locally.

Code Reference: `FileSharePeer.start_peer()` and `_handle()` methods in `fileshare_peer.py`

2.2 User Registration and Login

The system supports registering and logging in users with a username and password. Passwords are hashed using **Argon2** if available, or **PBKDF2** as a fallback.

User credentials are stored in-memory using `{username: (hashed_pw, salt)}`.

No session enforcement yet—login is acknowledged but not required for file access (to be added in Phase 2).

Code Reference:

Server: `_register()` and `_login()` in `fileshare_peer.py`

Client: `register_user()` and `login_user()` in `fileshare_client.py`

Crypto Reference: `hash_password()` and `verify_password()` in `crypto_utils.py`

2.3 File Upload (Share a File)

A logged-in or connected user can upload a file to a peer. The peer:

- Saves the file under the `shared/` directory
- Tracks it in a `self.shared` dictionary for discovery

Code Reference:

- Peer: `_upload()` method in `fileshare_peer.py`
- Client: `upload_file()` and `share_to_peer()` in `fileshare_client.py`

2.4 File Download

Users can download files from connected peers or from remote peers using direct peer-to-peer connections.

Code Reference:

Peer: `_download()` and `_sendfile()` methods

Client: `download_file()` (session-based) and `download_from_peer()` (stateless)

2.5 File Listing

Two types of file listing are supported:

LIST: lists files shared by the currently connected peer

LISTFILES: queries any remote peer for its shared file list

Code Reference:

Peer: `_list()` and `_listfiles()` in `fileshare_peer.py`

Client: `list_shared_files()` and `list_from_peer()` in `fileshare_client.py`

2.6 Stateless Peer-to-Peer Sharing

Beyond the main session-based connection, the client supports stateless interactions with any peer:

Direct upload via `shareto`

Direct download via `peerdl`

File listing via `peerlist`

Code Reference: `list_from_peer()`, `download_from_peer()`, `share_to_peer()` in `fileshare_client.py`

2.7 Peer File Change Monitoring (Watch Mode)

The client can periodically poll a peer for file list changes to simulate discovery or monitoring.

Feature: `watchpeer <host> <port>`
Code Reference: `watch_peer()` in `fileshare_client.py`

2.8 Interactive Command-Line Interface

The client exposes a clean and flexible CLI interface using `shlex` to parse commands across platforms (especially for Windows paths).

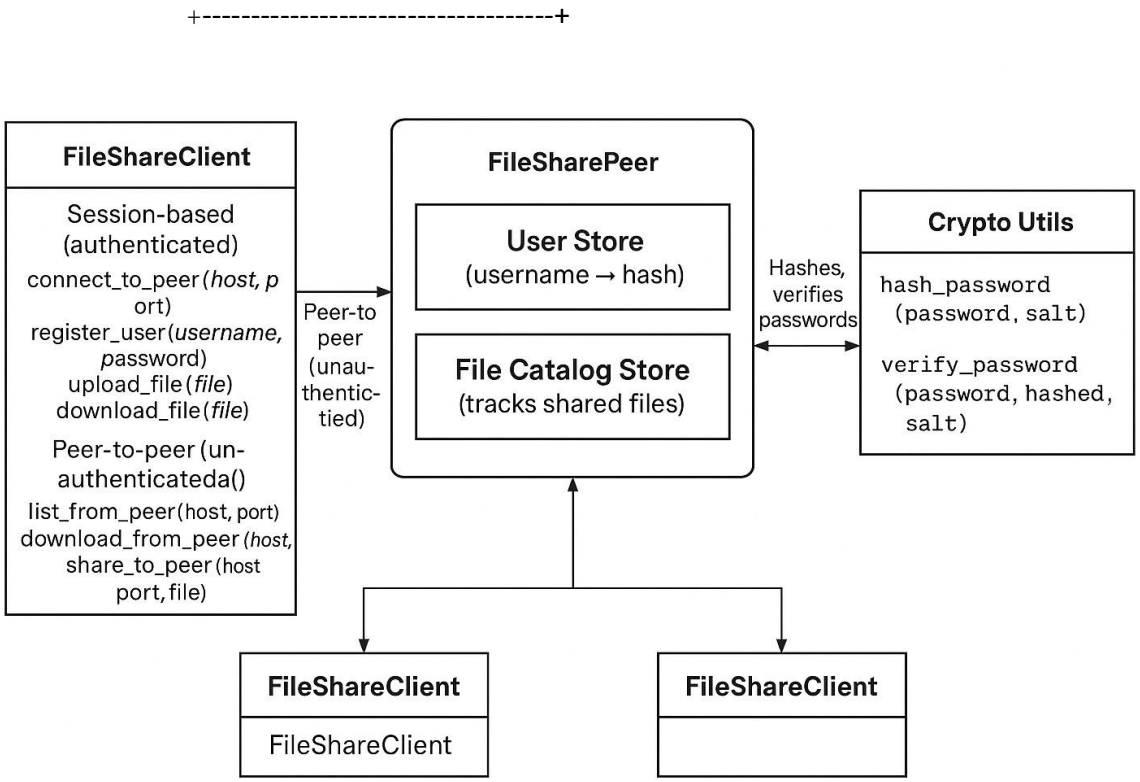
Commands Available:

```
connect, register, login, list, share, shareto,
download, peerlist, peerdl, watchpeer, quit
```

Code Reference: `main()` and `FileShareClient` in `fileshare_client.py`

---

3. Architecture Diagram



Initial System Architecture Diagram

4. Challenges Encountered

4.1 Stateless Session Handling

Although the system supports user registration and login, there's currently no state tracking to enforce authentication. Once a client connects, they can upload or download files without verifying if they're logged in. This makes it functionally open-access.

4.2 Protocol Simplicity vs. Robustness

The custom line-based protocol (e.g., `COMMAND param1 param2\n`) is intentionally minimal for simplicity, but lacks:

Error codes or structured responses

Command validation on the client side

Message integrity or framing for larger protocols

4.3 Manual Socket & Thread Management

The current architecture uses basic Python sockets and threads. While functional, it doesn't scale well for many peers or large file transfers. There's no timeout or graceful failure handling, so a slow or broken peer can hang a connection.

4.4 Peer Discovery Limitations

Peers need to be manually connected by IP and port. There's no automated discovery mechanism like broadcast, a central rendezvous server, or even a shared registry.

4.5 Lack of Encryption & Integrity Checks

Files are currently transferred and stored in plaintext without encryption or integrity verification. This leaves the system vulnerable to:

Eavesdropping

Tampering

Malicious file injection

---

5. Technologies Used

- **Python**
- **TCP Sockets** for network communication
- **Threading** for concurrent client handling on peers
- **Argon2 / PBKDF2** hashing in `crypto_utils.py` for secure password storage
- **Interactive CLI** using `shlex` and match-case for simplicity and cross-platform support

---

6. Phase 2 Plan

The main goal of phase 2 is Authentication & Credential Handling.

6.1. Planned Improvements

- **Client-side credential management**: store username/password securely (or use derived keys)
- **Server-side login enforcement**: require session login before file access
- **Session Restrictions**: Integrate per-connection login state tracking and restrict file actions to authenticated users.
- **Password hashing**: already handled with fallback to PBKDF2 if Argon2 unavailable

6.2. Tasks

- Add session state to `FileShareClient`
- Modify server handlers to enforce login
- Securely store credentials for reuse

---

7. User Manual

Start a Peer Node

On **Terminal 1**, run:

```
python fileshare_peer.py 9000
```

This peer will now listen for incoming connections on port 9000.

Step 2: Start the Client Shell

On **Terminal 2**, run:

```
python fileshare_client.py
```

This opens an interactive CLI where you can control the client.

Connect to a peer.

```
"connect <host> <port>"
```

Example:

```
p2p> connect 127.0.0.1 9000
[CLIENT] connected to 127.0.0.1:9000
```

Register a new user with the connected peer.

```
"register <username> <password>"
```

Example:

```
p2p> register alice strongpass
OK registered
```

Login to a peer using existing credentials.

```
"login <username> <password>"
```

Example:

```
p2p> login alice strongpass
OK welcome
```

Upload a file to the peer. The file will be stored in `shared/` on the peer.

"upload <path-to-file>"

Example:

```
p2p> upload mydoc.txt
OK stored
```

Download a file from the connected peer. The file will be saved to the `downloads/` directory.

"download <filename>"

Example:

```
p2p> download mydoc.txt
Saved to /full/path/to/downloads/mydoc.txt (2048 bytes)
```

List all files shared by the connected peer.

"list"

Example:

```
p2p> list
Files: mydoc.txt example.pdf
```

List files from another peer without connecting.

"peerlist <host> <port>"

Example:

```
p2p> peerlist 127.0.0.1 9001
shared1.txt image.jpg
```

Upload a file directly to a remote peer.

"shareto <host> <port> <path>"

Example:

```
p2p> shareto 127.0.0.1 9001 notes.txt
OK stored
```

Download a file directly from a remote peer.

"peerdl <host> <port> <filename>"

Example:

```
p2p> peerdl 127.0.0.1 9001 shared1.txt
Saved to /downloads/shared1.txt
```

Continuously monitor a peer for newly shared files. Polls every 5 seconds.

"watchpeer <host> <port>"

Example:

```
p2p> watchpeer 127.0.0.1 9001
Watching 127.0.0.1:9001 every 5s – Ctrl-C to stop
[NEW] file_added_later.txt
```

Exit the client shell.

"quit"

Example:

```
p2p> quit
```