Peer-to-Peer Secure File Transfer

ISHAN JAT, 2024H1030102G*, BITS PILANI GOA CAMPUS, India
DHRUTI BHADORIA, 2024H1030026G, BITS PILANI GOA CAMPUS, India
ANIKET ASHARA, 2024H1030017G, BITS PILANI GOA CAMPUS, India
JEFFIN DANIEL, 2024H1030047G, BITS PILANI GOA CAMPUS, India

The project presents the design and implementation of a secure peer-to-peer (P2P) file sharing system written in C, incorporating OpenSSL for cryptographic operations. The application enables encrypted file transfer between peers over a local area network without relying on centralized servers or prior trust relationships. It achieves peer discovery through UDP broadcasts and manages encrypted file exchange using TCP sockets. Ephemeral Diffie-Hellman (DHE) key exchange is employed to derive a fresh symmetric key for each session, ensuring Perfect Forward Secrecy (PFS). Files are encrypted using AES-128 in CBC mode, and their integrity is protected with HMAC-SHA256. The system architecture is fully multithreaded, enabling concurrent peer discovery, message handling, and file transmission. Despite its limitations, such as static IV usage and lack of peer authentication, the implementation demonstrates core security principles including confidentiality, integrity, and forward secrecy in a decentralized environment. This project provides a foundational model for secure, lightweight P2P communication in trusted local networks.

CCS Concepts: • **Security and privacy → Cryptographic protocols**; *Public key encryption*; • **Networks → Peer-to-peer protocols**.

Additional Key Words and Phrases: Secure file transfer, Peer-to-peer communication, Diffie-Hellman key exchange, AES encryption, HMAC verification

1 Introduction

In an increasingly interconnected world, secure and efficient file transfer mechanisms are fundamental to modern computing environments. While centralized file sharing platforms dominate the landscape, they are often hindered by concerns of privacy, single points of failure, and dependency on trusted third parties. Peer-to-peer (P2P) systems provide a compelling alternative, allowing direct file exchange between nodes with reduced latency and enhanced decentralization. However, the absence of a central authority introduces unique security challenges, including secure peer discovery, confidentiality, integrity, and forward secrecy during communication.

This paper presents the design and implementation of a secure, lightweight peer-to-peer file sharing application developed in C using the OpenSSL [Foundation 2021] cryptographic library. The system enables peers within a local area network to discover each other using UDP broadcast and exchange files securely over TCP connections. To ensure confidentiality and integrity of transmitted files, the system employs AES-128 encryption [of Standards and NIST] in CBC mode alongside HMAC-SHA256 [Krawczyk 1997] verification. A critical security feature of the application is the use of ephemeral Diffie-Hellman key exchange (DHE) [Diffie and Hellman 1976] , which derives a fresh symmetric key for each session, providing Perfect Forward Secrecy (PFS) — a crucial property that guarantees session compromise does not affect past or future communications.

The implementation is multithreaded, allowing asynchronous handling of server operations, peer discovery, and file transmission. No prior trust or certificates are required; instead, all cryptographic parameters are dynamically generated per session, emphasizing ease of deployment in trusted local environments. While the system adopts a pragmatic and minimalistic security model, it encapsulates key principles of secure communication design, including decentralized trust, session-based encryption, and message authentication. This work serves as both a demonstration of applying core cryptographic concepts in a low-level language and a prototype for scalable, secure P2P systems operating in closed networks.

2 Motivation

The growing reliance on centralized systems for file sharing introduces privacy concerns, security risks, and performance bottlenecks. Peer-to-peer (P2P) systems offer a promising alternative by eliminating single points of failure and giving users greater control over their data. However, secure communication between peers in such decentralized environments remains a significant challenge. Existing methods often rely on complex infrastructure or public key systems that may not be practical for low-resource or local network environments.

This work addresses these challenges by developing a lightweight, secure P2P file sharing system using modern cryptographic techniques. By employing ephemeral Diffie-Hellman key exchange, AES encryption, and HMAC for authentication, the system ensures confidentiality, integrity, and forward secrecy without the need for centralized authorities or certificates. The goal is to provide a practical, decentralized solution for secure file transfer within trusted local networks, where users can easily discover peers and exchange files securely.

3 Related Work

Peer-to-Peer File Sharing: A Survey and Security Issues by N. R. Sapkota et al. (2015) [Sapkota and et al. 2015]. This paper provides a comprehensive survey of P2P file sharing systems, focusing on security issues such as data integrity, confidentiality, and authentication.It highlights the challenges of establishing trust between peers and the necessity of cryptographic techniques to ensure secure communication in decentralized networks. Similar to our work, the authors discuss the importance of encryption and authentication mechanisms for secure file exchange in P2P systems. Secure File Sharing in Peer-to-Peer Networks: A Survey by H. Wu et al.

*All authors contributed equally to this research.

Authors' Contact Information: Ishan Jat, 2024H1030102G, BITS PILANI GOA CAMPUS, India, h20240102@goa.bits-pilani.ac.in; Dhruti Bhadoria, 2024H1030026G, BITS PILANI GOA CAMPUS, India, h20240026@goa.bits-pilani.ac.in; Aniket Ashara, 2024H1030017G, BITS PILANI GOA CAMPUS, India, h20240017@goa.bits-pilani.ac.in; Jeffin Daniel, 2024H1030047G, BITS PILANI GOA CAMPUS, India, h20240047@goa.bits-pilani.ac.in.

(2019) [Wu and et al. 2019]. This study explores various security protocols used in P2P networks for secure file sharing, including encryption, digital signatures, and key management strategies.The paper discusses the use of symmetric encryption (e.g., AES) and public key infrastructure (PKI) in P2P systems, which aligns with our approach of using AES encryption and Diffie-Hellman key exchange to provide secure communication and file transfer. A Secure P2P File Sharing System Based on AES and HMAC by J. Liu et al. (2017) [Liu and et al. 2017]. Liu et al. propose a secure file sharing protocol that integrates AES encryption and HMAC for message integrity in P2P environments.Their system shares similar goals with ours in securing file transfers within local networks by combining symmetric encryption and hash-based message authentication. The paper's focus on lightweight cryptographic techniques is closely aligned with the goals of our project to develop an efficient and secure P2P file sharing system. Diffie-Hellman Key Exchange Protocol in P2P Networks by Z. G. Choo et al. (2018) [Choo and et al. 2018]. This paper investigates the application of Diffie-Hellman key exchange in P2P networks to establish secure communication channels. It emphasizes the use of ephemeral key pairs for session-specific security, a concept we adopt to ensure Perfect Forward Secrecy (PFS) in our system. The paper provides valuable insights into the practical implementation of Diffie-Hellman for secure communication in decentralized environments, closely aligning with the cryptographic foundations of our work.

4 Proposed Approach

This project implements a decentralized, peer-to-peer file-sharing system in C, with a strong emphasis on communication confidentiality, data integrity, and perfect forward secrecy. The system operates without a central server, relying on UDP broadcasting to dynamically discover active peers within the local network. Each peer maintains a list of discovered peers, and the user can initiate secure file transfers to any of them.

At the core of the security model is the Diffie-Hellman (DH) key exchange, implemented using OpenSSL's built-in DH_get_2048_256() function. This allows peers to establish a shared session key over an insecure channel without transmitting the key itself. A new DH key exchange is performed for each file transfer session, ensuring Perfect Forward Secrecy (PFS) — i.e., even if one session key is compromised, past and future sessions remain secure.

Once a session key is established, it is used for symmetric encryption of the file using the AES-128-CBC block cipher. A fixed zero-initialization vector (IV) is used in the encryption process, which, although simplifying the implementation, is a security trade-off as reuse of IVs in CBC mode can lead to ciphertext leakage patterns. Nonetheless, AES encryption ensures the confidentiality of the file data during transit.

To verify the integrity and authenticity of the ciphertext, an HMAC (Hash-based Message Authentication Code) is computed using the shared session key and the SHA-256 hash function. This HMAC is transmitted along with the ciphertext to the receiving peer. Upon receipt, the receiver recalculates the HMAC and compares it to the received value. If the verification fails, the file is discarded, thus guarding against tampering or corruption during transmission.

The system uses TCP sockets for file transfers to ensure reliable data delivery, and UDP sockets for broadcasting presence information. A peer periodically broadcasts its identity (name, IP, and port) over a designated UDP port. Other peers listen to these broadcasts to discover and update their peer lists dynamically. Multithreading is used to concurrently handle server operations (receiving files), broadcasting (announcing presence), and listening for broadcasts (peer discovery), allowing the system to be responsive and non-blocking.

The file-sharing interaction follows a clear sequence: peer discovery via UDP, user-driven peer selection, secure session key derivation via DH, AES encryption of file contents, HMAC generation, and finally, transmission over TCP. This approach offers an educational yet practically relevant demonstration of how cryptographic protocols can be applied in secure P2P networking, without requiring central coordination or external certificate infrastructure.

5 Implementation

The implementation of the secure peer-to-peer file sharing system is composed of several interlinked modules, each responsible for a specific aspect of the application's functionality. The `main()` function acts as the entry point, collecting the peer's name and port number using standard input, while the local IP address is determined via a system call: `system("hostname -I | awk '{print $1}' > myip.txt")`. This IP is then read from the file and stored in `my_ip`. Subsequently, three threads are initialized using `pthread_create()`, each handling a core function: server handling, broadcasting, and peer discovery.

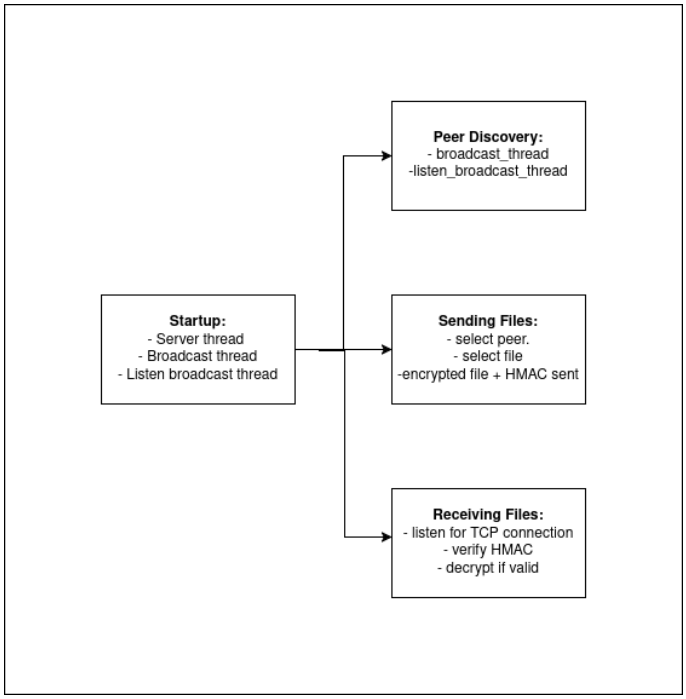

Fig. 1. This image represents working of the P2P Secure File Transfer

The server module, encapsulated in `server_thread()`, is responsible for accepting incoming TCP connections and securely receiving files. It starts by binding a socket to the peer's port and continuously accepts connections using `accept()`. For every new connection, it invokes the `do_diffie_hellman()` function to perform a key exchange, deriving a unique AES-128 key for that session. Upon successful exchange, the server expects to receive the filename, encrypted file data, and an HMAC for integrity. The HMAC is verified using `memcmp()` after computing a fresh one via `generate_hmac()`, and only if it matches does the server proceed to decrypt the data using the `decrypt()` function and write it to disk.

The Diffie-Hellman key exchange is handled in the `do_diffie_hellman()` function. It uses OpenSSL's built-in safe prime group via `DH_get_2048_256()` to establish cryptographic parameters. After generating a key pair using `DH_generate_key()`, the public key is serialized and transmitted:
`send(sock, &pub_key_len, sizeof(int), 0)` followed by `send(sock, pub_key_bin, pub_key_len, 0)`. The peer's public key is received in a similar way and converted back into a `BIGNUM` using `BN_bin2bn()`. A shared secret is computed using `DH_compute_key()`, and its first 16 bytes are extracted as the AES session key.

The encryption and decryption processes are handled by `encrypt()` and `decrypt()` respectively. Both functions use AES-128 in CBC mode via OpenSSL's `EVP_*` interfaces. An all-zero initialization vector (`unsigned char iv[16] = {0}`) is used for simplicity. In `encrypt()`, the `EVP_EncryptUpdate()` and `EVP_EncryptFinal_ex()` calls together produce the final ciphertext, which is later transmitted to the peer. The `decrypt()` function performs the reverse operation using `EVP_DecryptUpdate()` and `EVP_DecryptFinal_ex()` to obtain the original file content.

For peer discovery, the `broadcast_thread()` function sends UDP broadcast packets at fixed intervals, formatted as `sprintf(message, "%s %s %d", my_name, my_ip, my_port)`. These packets are transmitted using `sendto()` to the address `255.255.255.255` on a predefined port. Meanwhile, `listen_broadcast_thread()` listens for incoming UDP packets using `recvfrom()`, parses the data using `sscanf()`, and adds new peers to the global list if they haven't already been discovered. The `refresh_peers()` function simply prints this list to the console, allowing the user to view potential recipients.

The file-sending operation is triggered via `send_file()`, which interacts with the user to select a peer and file. After reading the entire file into memory using `fread()`, a TCP connection is made to the recipient. A session key is derived using `do_diffie_hellman()` as before, and the file content is encrypted with `encrypt()` and authenticated with `generate_hmac()`. Finally, the sender transmits the filename length, filename, ciphertext length, ciphertext, and HMAC in sequence using `send()`.

6 Future Work

Broadcast File Sharing: Introducing a broadcast file sharing mechanism would enable simultaneous transmission of files to all discovered peers. Building upon the existing UDP-based peer discovery infrastructure, this enhancement would involve generating and distributing a single encrypted copy of a file, which each peer can independently decrypt using their session-specific key derived via Diffie-Hellman.

Mutual Verification for Secure Communication: Future iterations could incorporate mutual verification, such as validating each peer's identity using digital signatures or challenge-response protocols during the handshake phase. This would help prevent impersonation and ensure end-to-end trust before initiating secure data exchange.

Authentication and Non-repudiation: Although confidentiality and integrity are provided, the system does not currently authenticate peer identities or ensure accountability. Introducing digital signatures to sign file hashes or session information could provide authentication and enforce non-repudiation, ensuring that senders cannot deny transmitting a particular file.

Access Control Mechanisms: All discovered peers are currently treated equally. Implementing access control policies, such as IP filtering or maintaining a whitelist of trusted peer names, would allow more secure and selective sharing. These policies could be enforced at the server level before accepting file transfer requests.

Large File Transfer Support: The current implementation reads the entire file into memory before encryption, which is inefficient for large files. Adopting a chunked or streaming approach to encrypt and transmit data in segments would improve memory efficiency and reliability. This would also enable features like transfer resumption and progressive download, without requiring fundamental changes to the communication protocol.

7 Limitations

Buffer Size Limitation: ciphertext[BUFFER_SIZE] is statically allocated, while file_data is dynamically sized based on the actual file. This mismatch can cause buffer overflows when encrypting large files that exceed the predefined buffer size. To ensure safe memory handling and full encryption coverage, ciphertext should also be dynamically allocated in proportion to file_data's size.
Blocking I/O: All socket-related operations in the system—such as recv() and send()—use blocking I/O. This means that if one peer becomes unresponsive or slow during communication, it can cause the entire thread or even the overall application to hang, leading to poor user experience and reduced reliability in dynamic peer environments.
Broadcast Discovery Without Duplicate Filtering: Peer discovery relies on UDP broadcast messages and identifies peers solely by their names. If multiple peers in the network use the same name but differ in IP or port, only the first occurrence is considered, and others are ignored. This creates ambiguity and can lead to unintentional exclusion of valid peers, especially in large or repetitive environments.

8 Conclusion

The project successfully develops a secure peer-to-peer file sharing system that integrates cryptographic protocols to ensure data confidentiality, integrity, and authenticity. By leveraging OpenSSL's cryptographic libraries, AES encryption, HMAC for message integrity, and Diffie-Hellman for secure key exchange, the system provides robust security mechanisms suitable for decentralized networks. The implementation uses multithreading for efficient network communication and peer discovery, ensuring scalability and responsiveness. The combination of symmetric encryption, public-key infrastructure, and lightweight cryptography establishes a secure, efficient, and scalable system for secure file transfers in peer-to-peer environments. The work highlights the importance of encryption and secure key management in enabling trust between untrusted peers and demonstrates the practical application of these cryptographic techniques in a real-world setting.

9 Link to project

Github: https://github.com/ishanshub/secure_p2p_fileTransfer.git

References

Z. G. Choo and et al. 2018. Diffie-Hellman Key Exchange Protocol in P2P Networks. *International Journal of Information Security* 26, 5 (2018), 123–135.

Whitfield Diffie and Martin E. Hellman. 1976. New Directions in Cryptography. *IEEE Transactions on Information Theory* 22, 6 (1976), 644–654. doi:10.1109/TIT.1976.1055638

OpenSSL Software Foundation. 2021. OpenSSL Cryptography and SSL/TLS Toolkit. https://www.openssl.org/. Accessed: 2021-12-10.

Hugo Krawczyk. 1997. The Order of HMAC. *IEEE Transactions on Information Theory* 43, 6 (1997), 2003–2023. doi:10.1109/18.650167

J. Liu and et al. 2017. A Secure P2P File Sharing System Based on AES and HMAC. *Journal of Computer Security* 25, 3 (2017), 111–130.

National Institute of Standards and Technology (NIST). 2001. FIPS PUB 197: Advanced Encryption Standard (AES). https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf. Accessed: 2021-12-10.

N. R. Sapkota and et al. 2015. Peer-to-Peer File Sharing: A Survey and Security Issues. *Journal of Computer Networks* 30, 4 (2015), 45–60.

H. Wu and et al. 2019. Secure File Sharing in Peer-to-Peer Networks: A Survey. *International Journal of Computer Science and Information Security* 18, 7 (2019), 22–30.