Cryptography Project Report

Merkle Tree Based File System

by

Chakradhar Makkena 22CSB0C05
Indresh Vikram Singh 22CSB0C04

Department of Computer Science and Engineering

National Institute of Technology, Warangal

March 2025

GitHub: https://github.com/indresh56/Cryptography-Project

1. Introduction:

Problem Addressed & Importance

Ensuring secure and efficient file transfer is a critical challenge in modern applications. Many existing systems lack strong encryption and reliable integrity verification, making them vulnerable to cyber threats. Additionally, traditional file transfer methods often require sending entire files even for minor updates, leading to inefficiencies and increased bandwidth consumption. As cyber threats continue to rise, it is essential to implement a solution that not only secures file transfers but also optimizes data transmission.

Existing Solutions & Limitations

While encryption protocols like TLS, cloud storage security measures, and blockchain-based integrity verification exist, they each have drawbacks. Some methods still involve full file transfers for updates, while others introduce high computational overhead, making them impractical for scalable applications. These limitations highlight the need for a more efficient and secure approach.

Key Contributions of Our Project

Our project addresses these challenges by developing a secure and optimized client-server file transfer system with the following features:

a. Secure Communication: Uses RSA key exchange for session setup, followed by AES-128 encryption to protect data transmission.

b. Merkle Tree-Based Integrity Verification: Ensures data integrity by computing and storing Merkle tree hashes for files.

c. Optimized File Updates: Detects modified file chunks using Merkle trees, reducing data transfer by only sending changed portions.

d. Two-Factor Authentication (2FA): Strengthens security by requiring an additional verification step during login.

e. Merkle Tree Visualization: Generates a graphical representation of the Merkle tree to enhance transparency and debugging.

f. By integrating these features, our system provides a secure, efficient, and reliable file transfer mechanism, effectively overcoming the limitations of existing solutions**.**

2. Objectives:

- To implement a secure client-server communication system using AES-128bit Encryption with RSA Key Exchange, Merkle tree-based integrity verification and Authentication with 2FA.
- To optimize file updates by transmitting only modified chunks using Merkle tree comparison, reducing bandwidth and storage overhead.

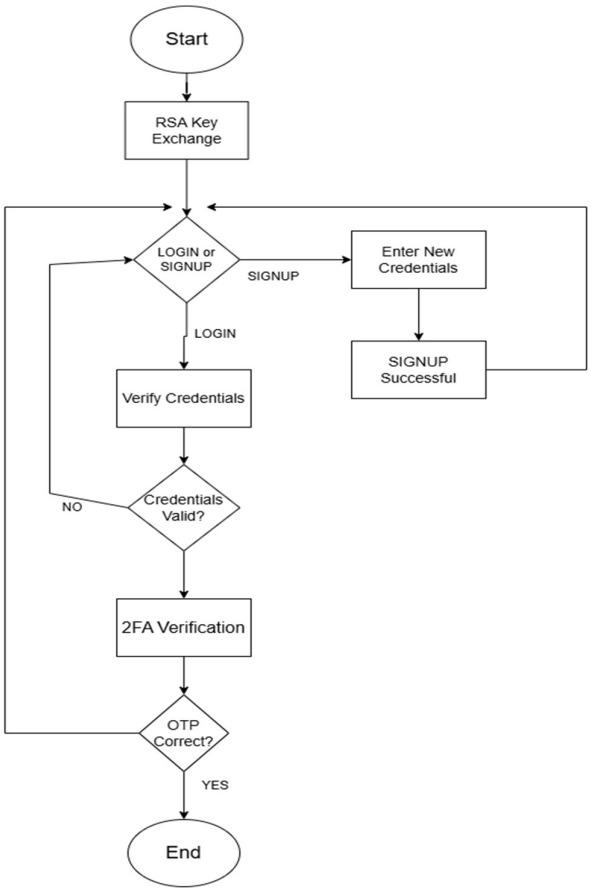
3. Implementation and Results Analysis

Implementation Details

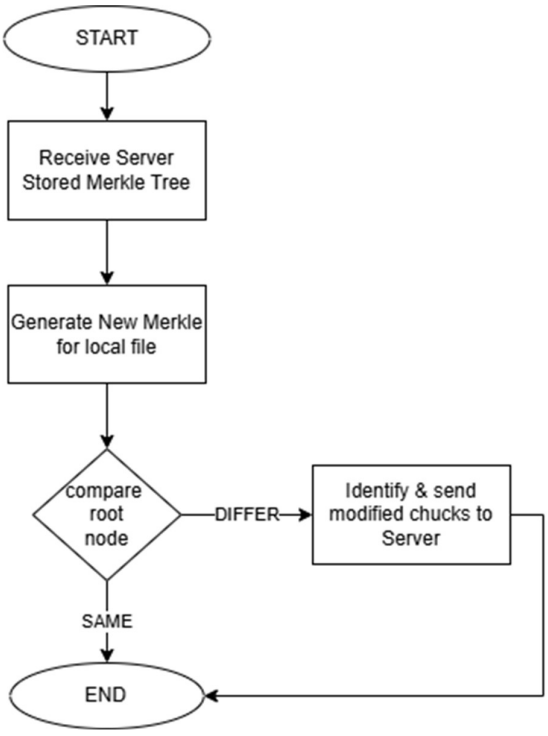
The project is implemented using a client-server architecture where the client interacts with the server over an encrypted channel. The key steps in the implementation are:

1. **Secure Key Exchange:** The Server generates a pair of keys using RSA, then it shares public key with connected clients and client generates a secret key AES 126bit key send it by RSA key exchange.
2. **User Authentication:** The client first logs in or signs up using a username-password system. Additionally, two-factor authentication (2FA) is implemented for enhanced security.
3. **File Operations:** Once authenticated, the client can perform various file operations, including:
    - Listing all files
    - Uploading a file
    - Updating a file
    - Deleting a file
    - Downloading a file
    - Exiting the system
4. **Merkle Tree for Integrity Verification:** During file uploads, the client generates a Merkle tree hash and sends it to the server along with the file. The server verifies the integrity of the received file using this hash.
5. **Optimized File Updates:** When a file is updated, the client calculates a new Merkle tree and compares it with the previously stored Merkle tree. Only the modified chunks of the file are sent to the server, reducing data transfer overhead.
6. **Server-Side Storage and Verification:** The server securely stores the files and maintains a record of Merkle tree hashes to detect unauthorized modifications.

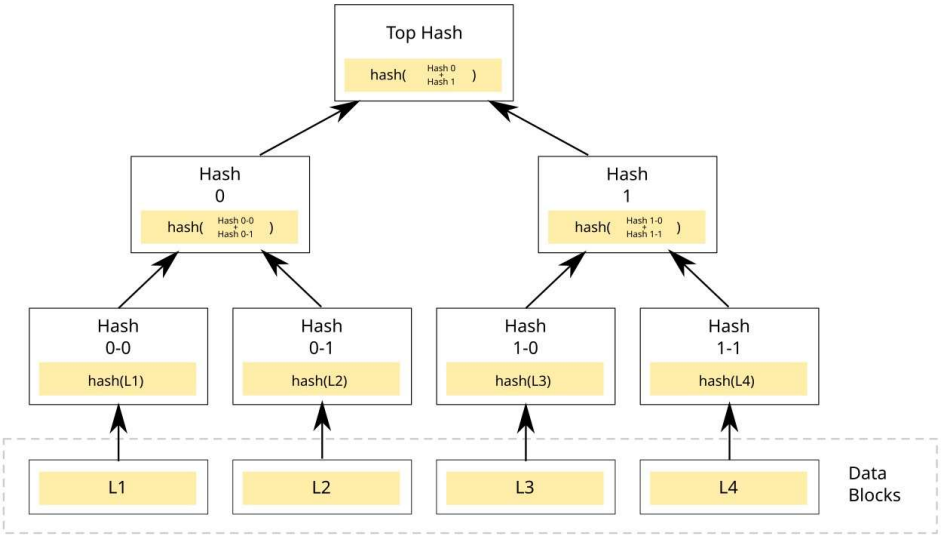
Client Authorization flow:



Update Feature:



Merkle tree construction:



Merkle tree build Algorithm:

Step 1: Generate Leaf Hashes
Divide file into fixed-size chunks
for each chunk **do**
    Compute $SHA256(chunk)$
    Append hash to leaf list
end for
if number of leaves is odd **then**
    Duplicate the last hash
end if
Initialize tree with leaves

Step 2: Build the Merkle Tree
while length of last tree level > 1 **do**
    Initialize next level as empty
    for each pair $(left, right)$ in current level **do**
        Compute parent hash: $SHA256(left||right)$
        Append parent hash to next level
    end for
    if number of nodes in next level is odd **then**
        Duplicate the last hash
    end if
    Update current level
end while
Return: Merkle Root (last remaining node)

Results:

```
(venv) chakradhar@ChakradharM:~/Crypto_LAB_Project$ python3 Client.py
AES Key generated and sent: 05f744c96171cadf2352a3a736c61027
Connected to server successfully.

1. Sign up
2. Log in
Enter choice: 1
Enter username: Indresh
Enter password: pass1234
Enter phone number (e.g., +1234567890): +919935806126
Signup successful!

1. Sign up
2. Log in
Enter choice: 2
Enter username: Indresh
Enter password: pass1234
Enter the OTP sent to your phone: 587182
Login successful with 2FA!
Files in your directory:
merkle_trees

Options:
1. List Uploads
2. Upload a file
3. Download a file
4. Update a file
5. Delete a file
6. Exit
Enter choice: 2
Enter file path to upload: testfile1.txt
```

```
(venv) chakradhar@ChakradharM:~/Crypto_LAB_Project$ python3 Server.py
Server listening on 127.0.0.1:12345
Connected: ('127.0.0.1', 46214)
AES Key received and decrypted: 05f744c96171cadf2352a3a736c61027
Received command: SIGNUP Indresh bd94dcda26fccb4e68d6a31f9b5aac0b571ae266d822620e901ef7ebe3a11d4f +919935806126
Received command: LOGIN Indresh bd94dcda26fccb4e68d6a31f9b5aac0b571ae266d822620e901ef7ebe3a11d4f
OTP sent to +919935806126: 587182
Received OTP 587182
Received command: LIST_FILES
Received command: UPLOAD testfile1.txt
File 'testfile1.txt' uploaded successfully.
Integrity Check Passed.

Server-side Merkle Tree for uploaded file:
506d35...01d97c
├──     e350e5...696f80
└──     38cc6b...18421d

Merkle tree visualization saved as 'tree.png'.
```

4. Conclusion:

Our project successfully implements a secure client-server communication system using AES -128bit Encryption with RSA key exchange and Merkle tree-based integrity verification. By encrypting all communication after an RSA key exchange, we ensure confidentiality, while the Merkle tree mechanism guarantees data integrity. The optimized file update process reduces bandwidth usage by transmitting only modified chunks. Additionally, two-factor authentication enhances security by preventing unauthorized access. The results demonstrate that our approach provides a robust and efficient file management system. Future enhancements could include support for additional encryption techniques and a graphical user interface to improve usability and accessibility for a broader range of users.

5. Learning Outcomes

- **Understanding Cryptographic Techniques:** Gained knowledge of RSA and AES encryption and Merkle tree-based integrity verification for secure communication.
- **Implementation of Two-Factor Authentication:** Understood the importance of multi-factor authentication in enhancing security.
- **Efficient File Management:** Learned how to optimize file updates by transmitting only modified chunks using Merkle trees.
- **Data Integrity Verification:** Gained experience in implementing and verifying file integrity through cryptographic hashing.
- **Problem-Solving and Debugging:** Enhanced skills in troubleshooting and optimizing security protocols in a client-server system.

6. Source Code:

Merkle tree Construction:

```python
def _build_merkle_tree(self, leaves):
    if not leaves:
        return [[self._hash(b"")]]

    tree = [leaves]
    while len(tree[-1]) > 1:
        level = []
        prev_level = tree[-1]

        for i in range(0, len(prev_level), 2):
            left = prev_level[i]
            right = prev_level[i + 1] if i + 1 < len(prev_level) else left
            combined_hash = self._hash(left + right)
            level.append(combined_hash)
        tree.append(level)
    return tree
```

Client's UPDATE Feature:

```python
def update_file(self, filename):
    """Update a file on the server with efficient delta transmission."""
    if not os.path.exists(filename):
        print("Local file not found!")
        return
    try:
        command = f"UPDATE {os.path.basename(filename)}"
        self.send_with_size(command.encode())
        server_merkle_encrypted = self.recv_with_size()
        if not server_merkle_encrypted:
            print("Failed to receive server's Merkle tree.")
            return
        if server_merkle_encrypted == b"FILE_NOT_FOUND":
            print("File not found on server. Please upload it first.")
            return
        server_merkle_data = self.decrypt_data(server_merkle_encrypted).decode()
        server_tree = json.loads(server_merkle_data)
        local_merkle_tree = MerkleTree(filename)
        changed_chunks = []
        local_leaves = local_merkle_tree.tree[0]
        server_leaves = server_tree[0]

        for i in range(max(len(local_leaves), len(server_leaves))):
            if i >= len(server_leaves) or (i < len(local_leaves) and local_leaves[i] != server_leaves[i]):
                changed_chunks.append(i)

        if not changed_chunks:
            print("File is already up to date on server.")
            self.send_with_size(b"NO_CHANGES")
            response = self.recv_with_size()
            if response:
                print(f"Server: {response.decode()}")
            return

        print(f"Found {len(changed_chunks)} changed chunks out of {len(local_leaves)} total chunks.")
```

```
            with open(filename, "rb") as f:
                file_data = f.read()

            chunk_info = {
                "chunk_size": local_merkle_tree.chunk_size,
                "changed_indices": changed_chunks
            }

            info_json = json.dumps(chunk_info)
            encrypted_info = self.encrypt_data(info_json.encode())
            self.send_with_size(encrypted_info)

            for chunk_idx in changed_chunks:
                start_pos = chunk_idx * local_merkle_tree.chunk_size
                end_pos = min(start_pos + local_merkle_tree.chunk_size, len(file_data))

                if start_pos < len(file_data):
                    chunk_data = file_data[start_pos:end_pos]
                    encrypted_chunk = self.encrypt_data(chunk_data)
                    self.send_with_size(encrypted_chunk)

            merkle_tree_data = json.dumps(local_merkle_tree.tree)
            encrypted_merkle = self.encrypt_data(merkle_tree_data.encode())
            self.send_with_size(encrypted_merkle)

            print("\nClient-side Merkle Tree after update:")
            print(merkle_tree_to_ascii(local_merkle_tree.tree))

            response = self.recv_with_size()
            if response:
                print(f"Server: {response.decode()}")
            else:
                print("No response from server.")

        except Exception as e:
            print(f"Update error: {e}")
```

References:

1. Merkle Tree | Brilliant Math & Science Wiki
2. https://www.geeksforgeeks.org/socket-programming-python/
3. https://onboardbase.com/blog/aes-encryption-decryption/
4. https://www.twilio.com/docs