Secured Peer to Peer Distributed File System

Siddhesh Bhande Maheswar Reddy Gajjala Dharmesh Patibandla
Chaitanya Kumar Gummalla Naga Panindra Ganni

University of Maryland, Baltimore County
Maryland

{ sbhande1, g170, dharmep, zf16251, GP22733} @umbc.edu

Abstract

In order to enable users to store data on unreliable P2P file servers, the Encrypted File System project intends to develop a distributed file system implementation with security features. Users will be able to configure permissions on files and directories, create, delete, read, write, and restore files, and use the system's user key revocation capability. The system must also be capable of handling simultaneous write and read activities, guaranteeing the confidentiality of file and directory names, and preventing illegal changes to files and directories. Both the data held in each peer and the communication between peers should be encrypted. Finally, the documentation for the project should explain how each criterion is satisfied.

1 Introduction

This project describes the creation of a distributed file system that is peer-to-peer (P2P) that enables users to create, update, read, write, remove, and restore files on other peers. The Master Server, which houses all the information pertaining to the files, and the Peer, which serves as both client and server, make up the system's two essential components. The RMI library is used in the project's design and implementation for entity-to-entity communications, the scheduled executor service is used to look for malicious activity, multithreading is used to handle concurrent requests, and AES encryption is used for secure data transfer. The flow of file operations, such as creating,

reading, writing, updating, deleting, and restoring files, as well as managing permissions, are being implemented.

2 Architecture

A central master server, a key generator, and numerous peers connected in a peer-to-peer network make up the system. For user authentication and file access, each peer has a set of private and public keys. A hash table that contains details about the files kept by various users is kept up to date by the master server. New key pairs for peers are created by the key generator, which is embedded into the client peer.

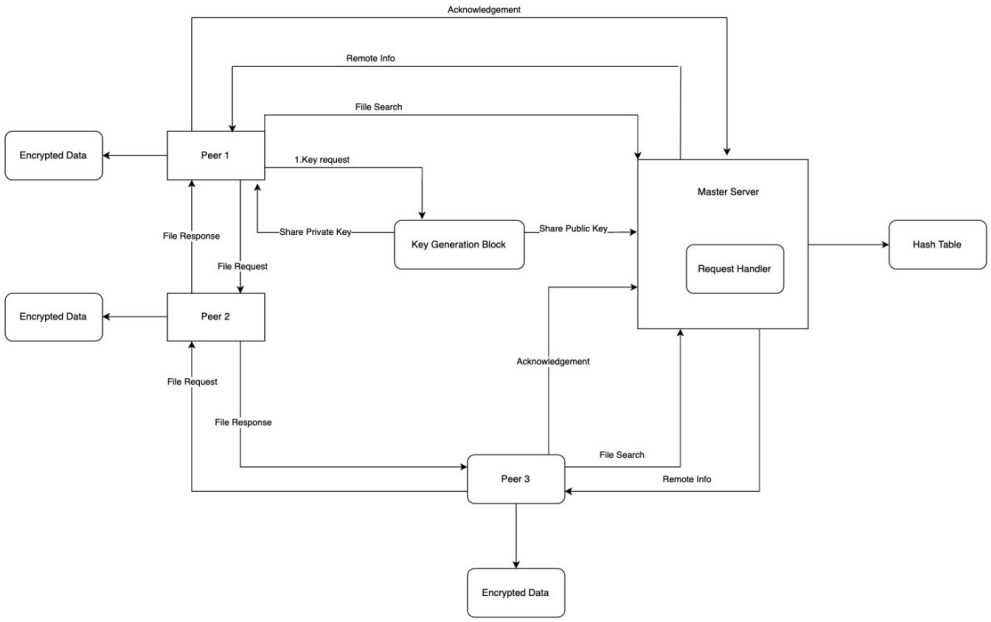

Fig 2.1. System Design and Workflow

Key Components

A. Peers

Peers are individual systems connected in the peer-to-peer network.

Responsibilities: -

- Generate a new private/public key pair for user authentication.
- Register with the master server using the generated key pair.
- Send file access requests to the master server.
- Receive file access permissions and instructions from the master server.

- Establish connections with other peers for file sharing.

B. Master Server

The central server that coordinates file access and manages user permissions.

Responsibilities: -

- Validate user keys and grant file access permissions.
- Maintain a hash table with information about files stored by users.
- Notify users about the presence of files on other peers.
- Handle file access requests and provide remote peer connection information.

C. Key Generator

Generates private/public key pairs for user authentication.

Responsibilities: -

- Generate a new private/public key pair when client connects to the system and share the public key with the master server.

D. Security and Access Control

- User authentication is enforced using private/public key pairs.
- Files are encrypted using a 128 bit-AES encryption algorithm to ensure data security.
- User permissions and access levels are stored in the user database on the master server.
- Path encryption is used to restrict unauthorized access to files.

3 Implementation

We would like to introduce some core algorithms and libraries used in our project to ensure secured communication and encryption.

- **RMI Registry**

  In our project, we utilized the RMI registry as a vital component to enable communication and interaction between different components of our distributed file system. We leveraged the registry to register and lookup remote objects, facilitating the seamless invocation of methods across multiple Java Virtual Machines (JVMs). Specifically, we used the RMI registry to register our remote objects such as the MasterQuery and PeerServer. When these objects were instantiated, we bound them to unique names within the registry, allowing clients to easily locate and obtain references to these objects.

- **AES Encryption Algorithm**

  AES (Advanced Encryption Standard) is a widely used symmetric encryption algorithm known for its security and efficiency. In our project, we employed AES encryption to

ensure the confidentiality and integrity of file data. By utilizing AES encryption, we ensured that the file content remains secure during transmission and storage. It adds an additional layer of protection to prevent unauthorized access to sensitive information.

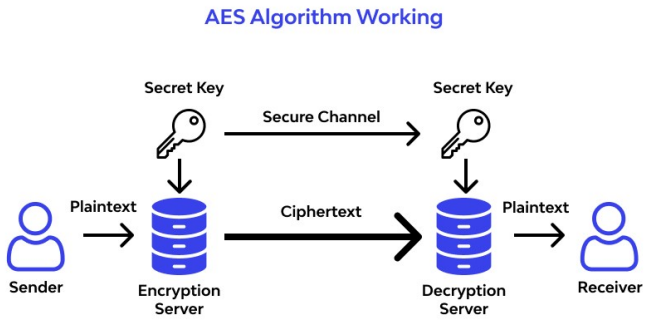

Fig 3.1 Symmetric key encryption
Image Credits: [https://www.wallarm.com/what/what-is-aes-advanced-encryption-standard]

- **RSA Encryption Algorithm**

  In our project, we utilized RSA (Rivest-Shamir-Adleman) encryption algorithm for secure key exchange and file encryption. Each peer generated an RSA key pair upon connecting to the DFS, registering the public key with the master server. When a peer needed an AES key for file encryption, the master server generated a new AES key, encrypted it with the peer's RSA public key, and sent it back. The peer then decrypted the AES key using their RSA private key to securely encrypt and decrypt file data. This implementation ensured secure key exchange and maintained the confidentiality and integrity of file encryption within our DFS project.

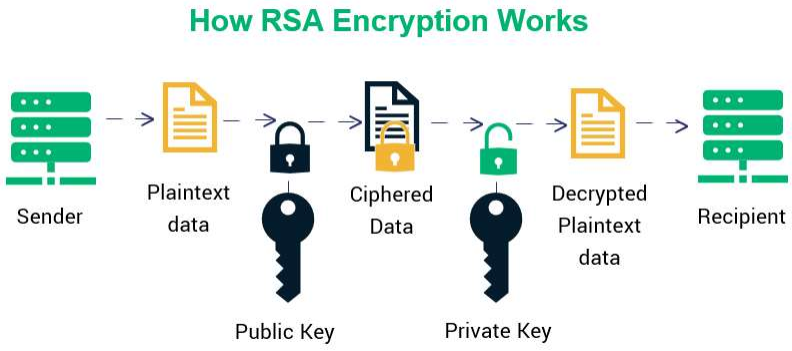

Fig 3.2 Visual breakdown of RSA encryption process.
Image Credits: [https://sectigostore.com/blog/ecdsa-vs-rsa-everything-you-need-to-know/]

This is our system implementation with various classes and their functions explained below:

3.1 MasterQuery:

Read:
This method will first check whether a file with the given file name exists. If the file does not exist, the method returns a Map. Entry object with a message indicating that the file doesn't exist. If the file exists, the method checks whether the peer with the given URI has permission to read the file. If the peer does not have permission to read the file, the method returns a Map. Entry object with a message indicating that the peer doesn't have permission to read.

If the peer has permission to read the file, the method retrieves the paths of the file and selects the first path. The method then retrieves the secret key for the file, encodes it using Base64 encoding, and encrypts it using RSA encryption with the public key of the peer specified by the URI. The method returns a Map. Entry object with the peer path and the encrypted key.

Create:
Here a file is created in the shared memory. The RMI library encrypts the file name and content before storing the file in the shared memory.

Delete:
The method checks if the file exists. If it doesn't, it returns a message and exits. If the file exists, it checks if the peer has permission to delete or restore it. If not, it returns a message and exits. If the peer has permission, it retrieves the file paths, sets "isDeleted" to true, indicating deletion, and returns the URI of the owning peer.

Update:
In this method the master server will check if the requested file is present. If yes then it will check if peer has the permission to update the file. If the permission is satisfied the file can be updated with the required content. The RMI library ensures that the file is encrypted before storing it into the file system.

Restore:
When this operation is performed It will first check whether the file is present in the database or not, if the file is present in the database, then it will return a message saying “file already exists”, if it cannot find the file then peer will use URI method to restore the deleted file. The method returns the path of the first peer in the list of peers that possessed the file before it was deleted if the peer has permission to restore the file and the file does not already exist in the system. The file has been restored, as indicated by the setting of the isDeleted property to false for the given file name. The procedure finally returns the location of the peer where the file is recovered.

Has file:
This file method checks whether the specified file is present in the lookup table or not, if it is not present in the lookup table and If it’s marked as deleted or not. It returns true, indicating that the file is accessible for read/write operations, if the file is present and hasn't been removed. It returns false, indicating that the file is not available, if the file is either marked as deleted or does not exist.

Get path:

The method chooses a peer at random from a list of peers and then returns the path of that peer. The technique produces a random integer between 0 and the number of peers after first calculating the number of peers in the list. Following an iterative search through the list of peers, the method returns the path of the peer whose index in the list matches the randomly generated integer.

Register peer:
We can add a new peer to our system using this method. The method takes a String peerData as input which contains information about the peer. This method checks that the input data is not a null string or not and if it is not a null string then it will look for the input peer data. If it is a null string then it will add the input peer data and registers the peer with our system.

Malicious check:
This function is implemented in the master server. Master server checks the peers file system every 5 milliseconds whether all of the files are present in the file data, and if it cannot find a file then it will report as a malicious activity has been detected.

3.2 Master server

The master server is the most important server because it does all the request handling and file handling tasks, If a peer has to perform CRUD operations then it has to take permission from the master server, Moreover it maintains information of files like directories of the files and it maintains a hash table which contains all the file created by different users.

FileAccessQuery:

- Read: Retrieves the file data from memory (at the current path).
- Create: Generates a new file with the given name and data.
- Update: Appends the new data to the existing file with the provided name.
- Restore: Restores the previously deleted file.
- Delete: Erases the file permanently.

3.3 Peer Client:

To ensure that the file name and data is encrypted after each operation we have used the AES algorithm described above.

Create File:
By using peer client we can create and name the file. It fetches the IP and port of a random peer, checks whether the file name is null, and throws an error if the latest peer already has the file. The file name and data are then encrypted before being sent to the peer server to be constructed, where it employs a lookup mechanism to locate the reference to a distant object. If the file is successfully created, it then prints a success message.

Read file:

This operation allows the peer to read a file, but before giving the access to read the file it will check whether the peer was given the permission from master server or not, if it has the access to read then it retrieves the encryption key using RSA decryption and decodes the key.

Update file:
To perform this operation we need to fetch the necessary information from master server and checks for the permissions, Once master server gives the access then it will update the content of the file in different servers. Once it is updated it will print "successful update of the file" message.

3.4 Key Revocation Functionality

We have decided to use the RSA algorithm to implement the user key revocation. Each time a peer connects to the DFS, he generates a new public and private key pair and register the public key with the master server. Whenever the peer wants to perform a CRUD operation, the peer needs a AES key from the master server to encrypt the files. The master provides the user with the required AES key by encrypting it with the public key shared by the peer and sending it to the peer. Then the peer uses his private key to decrypt the AES key sent by the master server and uses it to encrypt the files

4 Benchmark

Operation Type	Operations	No of Clients	Total Operations	Time Taken
Create	10000	3	30000	8.533 seconds
Read	10000	3	30000	5.902 seconds
Delete	10000	3	30000	5.799 seconds

Fig 4.1. Benchmark results

The benchmark results show the performance of Create, Read, and Delete operations. With a total of 10,000 operations per type distributed among three clients, the Create operations took 8.533 seconds to complete, Read operations took 5.902 seconds, and Delete operations took 5.799 seconds. These results indicate efficient handling of a large number of file creations, quick retrieval of files, and responsive deletion of files. Overall, the benchmark showcases the system's ability to handle a significant workload and perform file management operations effectively within reasonable timeframes.

5 Future Scope

1. **Distributed Storage and Replication:** This system can be extended to include distributed storage techniques to improve scalability and fault tolerance. Implementing data distribution strategies, and improved compression algorithms can ensure high availability and performance.
2. **File Versioning and History:** In our system we have introduced a technique that can restore a illegally deleted file from another peer. This system can be enhanced to ensure previous versions of a updated file can be restored. This can be particularly useful in collaborative environments or situations where file revisions need to be tracked and managed.
3. **User-Friendly Interfaces:** Enhancing the user interfaces, both command-line and graphical, can improve the usability and user experience of the system. Intuitive interfaces with clear instructions, error handling, and informative feedback can make the system more user-friendly and accessible to a wider range of users.
4. **Cloud Systems integration:** Integrating the DFS with cloud services allows users to access and manage files across platforms like Dropbox, Google Drive, and Amazon S3. This integration combines the benefits of cloud services with the functionality of the DFS, expanding user options for flexible and convenient file storage and management.

6 Project setup and execution

1. Install Java Version 11 and IntelliJ IDE. Open the source code in IntelliJ.
2. For Mac users, install iTerm to open multiple terminals side by side.
3. Update the IP and PORT number of the Master and Client in the properties files located in the resources folder.
4. Open 4 terminals and execute the following commands:
    a. In the 1st terminal:
        i. Run the command '**javac *.java'**
        ii. Run the command '**rmic fileAccessQuery'**
        iii. Run the command '**rmic MasterQuery'**
        iv. Run the command '**rmiregistry &'**
    b. In the 2nd terminal, run the command '**java MasterServer'**
    c. In the 3rd terminal, run the command '**java PeerServer'**
    d. In the 4th terminal:
        i. Run the command '**java PeerClient'**
        - When prompted for a username and password, use the following credentials:
            - Username: naga
            - Password: 321

7 Libraires:

We have used many libraries in our project. Some of them are described below:

1. **Java RMI library**: We are using this library to create connection between Peers and Master. It allows to use the method objects that are in a different location.

2. **Javax.crypto**: Java Cryptography (Java Crypto) is a package in the Java standard library that provides tools for encryption, decryption, and hashing. It includes various classes and interfaces that can be used to implement cryptographic algorithms and protocols, as well as to manage keys and digital certificates.

3. **Java.rmi.Naming**: Java RMI (Remote Method Invocation) is a framework that allows you to invoke methods on objects that Distributed Storage and Replication: To improve scalability and fault tolerance, the system can be extended to include distributed storage techniques. Data replication across multiple servers can enhance availability and performance. Implementing data distribution strategies, such as consistent hashing or partitioning, can ensure efficient and balanced storage across multiple nodes.

4. **Java.util.concurent:** The java.util.concurrent package in Java provides classes and interfaces for writing concurrent and multithreaded programs.

8 References

[1] A Measurement Study of Peer-to-Peer File Sharing Systems, Stefan Saroiu, P. Krishna Gummadi, Steven D. Gribble , https://people.mpi-sws.org/~gummadi/papers/p2ptechreport.pdf

[2] https://www.wallarm.com/what/what-is-aes-advanced-encryption-standard

[3] https://sectigostore.com/blog/ecdsa-vs-rsa-everything-you-need-to-know/

[4] https://docs.oracle.com/javase/8/docs/technotes/guides/rmi/hello/hello-world.html#define

[5] https://www.andrew.cmu.edu/course/14-736-s19/applications/labs/lab3/project3.pdf

[6] https://www.youtube.com/watch?v=NmGytKDhCx4&t=42s&ab_channel=tutorialplus