1.INTRODUCTION

The increasing reliance on Electronic Health Record (EHR) systems, coupled with the rise of resource-constrained IoT devices, demands robust solutions. In response, we propose a Publicly Verifiable Shared Updatable EHR Database Scheme. Unlike existing Verifiable Database (VDB) models, our approach prioritizes real-time proof generation, minimizing user overhead for storage integrity checks. By modifying the Functional Commitment (FC) scheme and incorporating a verifier-local revocation group signature, our scheme ensures privacy preservation, batch integrity checking, and dynamic group member operations. This innovative solution addresses the dual challenges of security and efficiency in EHR systems, contributing to the seamless and secure exchange of sensitive patient information in the digital healthcare landscape.

1.1 Motivation

The motivation behind developing these modules stems from the need to effectively manage large databases within dynamic group environments while ensuring data security, integrity, and access control. The Member module addresses the challenges faced by various entities such as patients, clinics, hospitals, and insurance providers in handling extensive databases. By allowing users to outsource databases to a server and locally generate authentication tags for enhanced security, the module streamlines database management and facilitates dynamic group collaboration. Key functionalities like file upload, audit proofing, and access management empower members to efficiently manage their data within the group ecosystem. The Group Manager module serves as the administrative backbone of the system, providing oversight and control over group activities and member statuses. With tools to view member details, manage their statuses, and revoke or reinstate members as necessary, the Group Manager ensures the integrity and security of the group. This module is essential for maintaining proper access control and enforcing security policies within the group environment. The TPA module introduces a mechanism for regular auditing to validate the integrity of data stored in the cloud. By enabling third-party auditors to efficiently audit databases using public key infrastructure, the module enhances data trustworthiness without imposing significant overhead. The presence of a third-party auditor adds an additional layer of security and trust, reassuring users about the integrity of their stored data. Utilizing cloud storage services through the Cloud module provides scalable and reliable storage solutions for group members. The module also serves as a data sharing platform, facilitating seamless collaboration and exchange of encrypted data among group members. Leveraging a specific cloud service provider, such as DriveHQ, underscores the flexibility of the solution, allowing users to choose cloud services based on their preferences or requirements. Overall, these

modules collectively address the complexities of managing large databases, ensuring data integrity, facilitating collaborative workflows, and empowering administrators within dynamic group environments. Whether in healthcare, insurance, or other professional networks, these modules offer robust solutions to meet the diverse needs of users while maintaining high standards of security and efficiency.

1.2 Problem Definition

In the context of Electronic Health Records (EHR) storage, existing verifiable database (VDB) approaches face challenges in proof reuse and server-driven updates, hindering effective data integrity checks. We propose an innovative updatable VDB scheme based on functional commitment, addressing privacy-preserving integrity auditing and group member operations. Our scheme prioritizes server response correctness and data storage integrity, aiming for security without excessive computational overhead. We design an updatable functional commitment scheme and present a practical VDB scheme under computational assumptions. Batch auditing enhances efficiency in multi-cloud, multi-user scenarios. Theoretical proofs affirm desired security properties, with our scheme outperforming alternative algorithms.

1.3 Objectives of the Project

The overarching objective of this project is to develop a sophisticated system tailored for the efficient management of large databases within dynamic group settings, all while prioritizing data security, integrity, and access control. This initiative aims to provide a comprehensive solution that caters to the diverse needs of users including patients, clinics, hospitals, and insurance providers. By enabling users to seamlessly outsource extensive databases to a centralized server, the project seeks to streamline database management processes. Integral to this objective is the implementation of robust security measures, including authentication mechanisms and encryption techniques, to safeguard data during transfer and storage, thus mitigating the risk of unauthorized access or tampering.

Furthermore, the project endeavors to foster dynamic collaboration among group members by facilitating secure database sharing while affording administrators, represented by the Group Manager module, the necessary tools and functionalities to oversee group activities and enforce security policies. This administrative oversight includes monitoring member statuses, managing access rights, and responding to user needs effectively. Additionally, the project places a significant emphasis on ensuring the integrity of stored data through regular auditing procedures conducted by third-party auditors (TPA). This auditing mechanism serves to validate the accuracy and reliability of the data stored in the cloud, thereby instilling trust and confidence in users regarding the integrity of their information.

In conjunction with these objectives, the project leverages cloud storage services to provide scalable and reliable storage solutions for group members, allowing them to store and access large volumes of data without constraints. User-centric functionalities are also paramount, with interfaces designed to empower users to manage their data efficiently, perform audits, and collaborate seamlessly with other group members. Finally, the system is engineered for flexibility and compatibility, accommodating different cloud service providers to cater to users' preferences and organizational requirements. Through the attainment of these objectives, the project aims to deliver a comprehensive and adaptable solution that meets the complex demands of dynamic group environments across various sectors, including healthcare networks, insurance providers, and professional associate

2.LITERATURE SURVEY

[1] An approach of membership revocation in group signatures is verifier-local revocation (VLR for short). In this approach, only verifiers are involved in the revocation mechanism, while signers have no involvement. Thus, since signers have no load, this approach is suitable for mobile environments. Although Boneh and Shacham recently proposed a VLR group signature scheme from bilinear maps, this scheme does not satisfy the backward unlikability. The backward unlikability means that even after a member is revoked, signatures produced by the member before the revocation remain anonymous. In this paper, we propose VLR group signature schemes with the backward unlinkability from bilinear maps.

[2] We present a Hierarchical Identity Based Encryption (HIBE) system where the ciphertext consists of just three group elements and decryption requires only two bilinear map computations, regardless of the hierarchy depth. Encryption is as efficient as in other HIBE systems. We prove that the scheme is selective-ID secure in the standard model and fully secure in the random oracle model. Our system has a number of applications: it gives very efficient forward secure public key and identity based cryptosystems (with short ciphertexts), it converts the NNL broadcast encryption system into an efficient public key broadcast system, and it provides an efficient mechanism for encrypting to the future. The system also supports limited delegation where users can be given restricted private keys that only allow delegation to bounded depth. The HIBE system can be modified to support sublinear size private keys at the cost of some ciphertext expansion.

[3] We introduce and formally define polynomial commitment schemes, and provide two efficient constructions. A polynomial commitment scheme allows a committer to commit to a polynomial with a short string that can be used by a verifier to confirm claimed evaluations of the committed polynomial. Although the homomorphic commitment schemes in the literature can be used to achieve this goal, the sizes of their commitments are linear in the degree of the committed polynomial. On the other hand, polynomial commitments in our schemes are of constant size (single elements). The overhead of opening a commitment is also constant; even opening multiple evaluations requires only a constant amount of communication overhead. Therefore, our schemes are useful tools to reduce the communication cost in cryptographic protocols. On that front, we apply our polynomial commitment schemes to four problems in cryptography: verifiable secret sharing, zero-knowledge sets, credentials and content extraction signatures.

.

[4] The Health Insurance Portability and Accountability Act (HIPAA) has set privacy and security regulations for the US healthcare industry. HIPAA has also established principles for

security standards that global e-health industry tends to follow. In this paper, a hybrid public key infrastructure solution (HPKI) is proposed to comply with the HIPAA regulations. The main contribution is the new e-health security architecture that is contract oriented instead of session oriented which exists in most literatures. The proposed HPKI has delegated the trust and security management to the medical service provider during the contract period, which is more realistic. It is much an analogy to existing paper based health care systems in terms of functional structure. The cryptographically strong PKI scheme is deployed for the mutual authentication and the distribution of sensitive yet computational non-intensive data while efficient symmetric cryptographic technology is used for the storage and transmission of high volume of medical data such as medical images. One advantage is that the proposed HPKI can be constructed from existing cryptographic technologies where various relevant security standards, tools and products are available. Discussion has been provided to illustrate how proposed schemes can address the HIPAA privacy and security regulations.

3. ANALYSIS

3.1 Existing System

Benabbas et al. proposed the verifiable database (VDB) as a secure and efficient updatable cloud storage model for resource-limited users. In a VDB scheme, a client can outsource the storage of a collection of data items to an untrusted server. Later, the client can query the server for an item (a message) at position i, the server returns the stored message at this position along with a proof that it is the correct answer. However, the security of only verifying the server response correctness is far from enough for the EHR system, and it is not clear whether data that is not frequently accessed is still stored correctly. If these data are destroyed and not discovered in time, it can cause huge losses in the event of an emergency. Jiang et al.'s scheme proposed to use vector commitment scheme to construct the audit scheme. Although the reduction of tags has been achieved, their scheme fails to achieve the expected security due to the neglect of the real-time performance of proof generation. There is still no good way to minimize the communication for low performance users.

3.1.1 Disadvantages Of Existing System:

The primary problem faced by the EHR system is on how to verify that the server responses correctly each time.The existing VLR group signature scheme does not have backward unlinkability (BU), which means that even if a member is revoked at a certain time, the signature before that time remains anonymous. It poses a threat to user identity privacy.

In the existing system, due to the fact that their model did not consider a notion of "real-time" proof, the use of these techniques makes their audit scheme and other VDB schemes incapable of checking storage integrity. In this case, only the queried data is involved in the verification process. This leads to verification only on the data being queried, while storage integrity of other cloud data is not checked. If the cloud data which is not queried is damaged, it will not be detected in time. When the damaged data is needed, there will be varying degrees of loss.

3.2 Proposed System

Our research focuses on the security and efficiency of large database storage, such as EHR. According to the characteristics of EHR system, two aspects of security deserve our attention, namely, the server response correctness and the data storage integrity. In order to deal with above problems, we use a new tool called functional commitment (FC) and design a publicly verifiable updatable database scheme based on functional commitment supporting privacy-preserving integrity auditing and dynamic group operation. We modify the existing functional commitment scheme in order to use the function binding of functional commitment to design an auditable VDB scheme. We point out security problems with existing scheme and propose a publicly verifiable updatable VDB scheme based on the functional commitment and group signature without incurring too much computational overhead and storage cost. Moreover, our scheme is applicable for large-scale data storage with minimum user communication cost. Our proposed scheme not only preserves all the properties of the original VDB scheme, but also implements efficient privacy preserving integrity auditing, non-frameability and traceability.

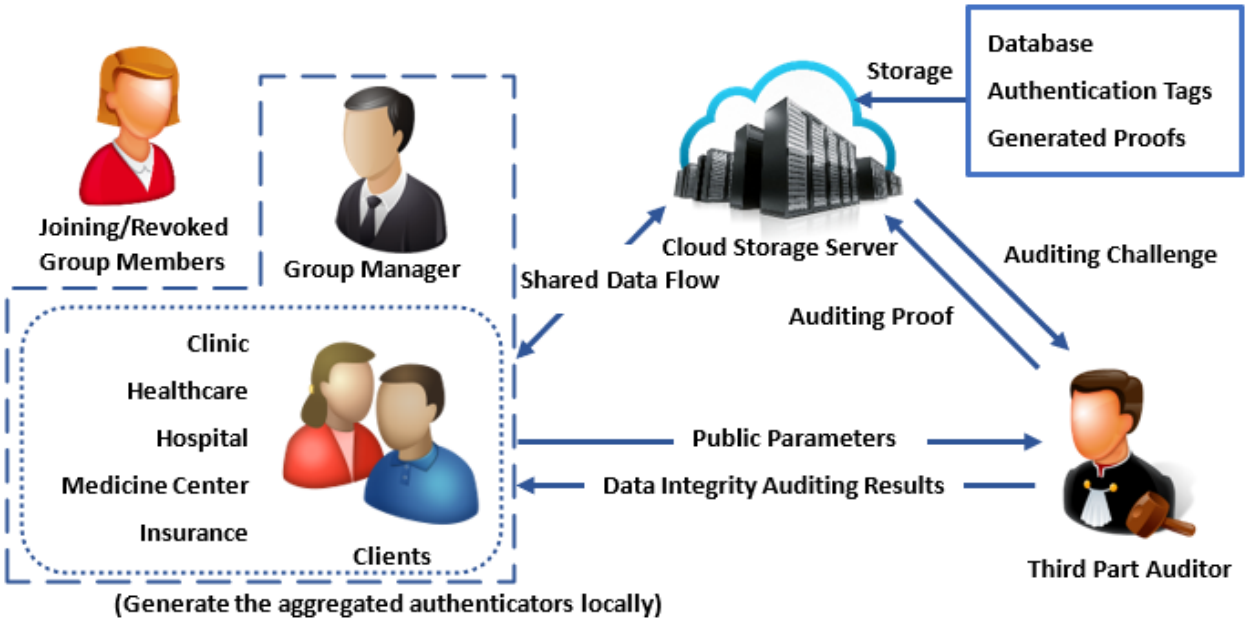

Figure 3.2

3.2.1 Advantages Of Proposed System:

The scheme preserves data privacy from the auditor by using a random masking technique and the sparse vector is used for sampling auditing. Our scheme supports dynamic group member operations which include join and revocation. In addition, our VDB supports batch

auditing and it supports multi-cloud server, multiuser and multi-storage vector scenarios. Security analysis and experimental comparison with existing schemes are provided and it shows that our VDB is secure and efficient. Our VDB scheme can securely and efficiently query and update database stored in the cloud and publicly audit data storage integrity

3.3 Software Requirement Specification

3.3.1 Purpose

The purpose of this project is to address the multifaceted challenges inherent in managing extensive databases within dynamic group settings, with a primary focus on ensuring data security, integrity, and accessibility. By providing a centralized platform for users such as patients, clinics, hospitals, and insurance providers to outsource and manage their databases, the project aims to streamline database management processes and enhance operational efficiency. A key objective is to implement robust security measures, including authentication mechanisms and encryption techniques, to safeguard data during transfer and storage, thereby protecting it from unauthorized access, tampering, or breaches. Additionally, the project seeks to foster dynamic collaboration among group members by facilitating secure database sharing and providing tools for effective group administration. Empowering administrators with oversight and control capabilities ensures proper access control and maintains the integrity of the group ecosystem. Regular auditing procedures conducted by third-party auditors further validate the integrity and reliability of stored data, instilling trust and confidence in users and ensuring compliance with regulatory requirements. Leveraging cloud storage services offers scalable and reliable storage solutions, accommodating the scalability requirements of growing databases within dynamic group environments. User-centric functionalities and interfaces empower users to efficiently manage their data, perform audits, and collaborate seamlessly, enhancing user satisfaction and engagement with the system. Lastly, flexibility and compatibility across various cloud service providers ensure adaptability to diverse preferences and organizational requirements, making the project comprehensive and adaptable to meet the complex demands of managing large databases in dynamic group environments.

3.3.2 Scope

The scope of this project is to develop a comprehensive system tailored for the efficient management of large databases within dynamic group environments, with a strong emphasis on data security, integrity, and accessibility. This entails the development and implementation of several key modules, including Member, Group Manager, TPA (Third-Party Auditor), and Cloud, each serving distinct functionalities essential for seamless database management and collaboration. User interface design plays a crucial role in ensuring user-friendly interactions, facilitating efficient access to system functionalities for members, group managers, TPAs, and cloud administrators. Robust security measures, including authentication mechanisms, encryption techniques, access controls, and audit trails, are integrated to safeguard data confidentiality and integrity throughout its lifecycle. Dynamic group collaboration is facilitated through tools for secure database sharing, access management, and group

administration, empowering users to collaborate effectively within their dynamic group environments. Administrative oversight tools empower group managers to monitor activities, enforce security policies, and manage access rights, ensuring proper governance and compliance. Integration of auditing mechanisms enables regular validation of data integrity by third-party auditors, ensuring adherence to regulatory requirements and industry standards. Leveraging cloud storage services provides scalable and reliable storage solutions, catering to the storage needs of large databases while facilitating seamless data access and retrieval. User training and support are provided to ensure effective utilization of the system, while flexibility and scalability are prioritized to accommodate diverse user needs and future expansion. Thorough testing and quality assurance procedures are conducted to ensure the reliability, stability, and security of the system, delivering a robust and adaptable solution to meet the complex demands of managing large databases within dynamic group environments.

3.3.3 Overall Description

H/W System Configuration:

- System : Pentium i3 Processor
- Hard Disk : 500 GB
- Ram : 2 GB

Software Requirements:

- Operating system : Windows 10.
- Coding Language : JAVA.
- Tool : Netbeans 8.2
- Database : MYSQL

4.DESIGN

4.1 Uml Diagrams

UML (Unified Modelling Language) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML comprises two major components: a Meta-model and a notation The Unified Modelling Language is used for business modelling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects

Various UML Diagrams are:

- ❖ Use Case Diagram
- ❖ Class Diagram
- ❖ Activity Diagram
- ❖ Sequence Diagram
- ❖ State Chart Diagram
- ❖ Collaboration Diagram
- ❖ Component Diagram
- ❖ Deployment Diagram

4.1.1 Use Case Diagram

A use case diagram in the Unified Modelling Language (UML) is a type of behavioural diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

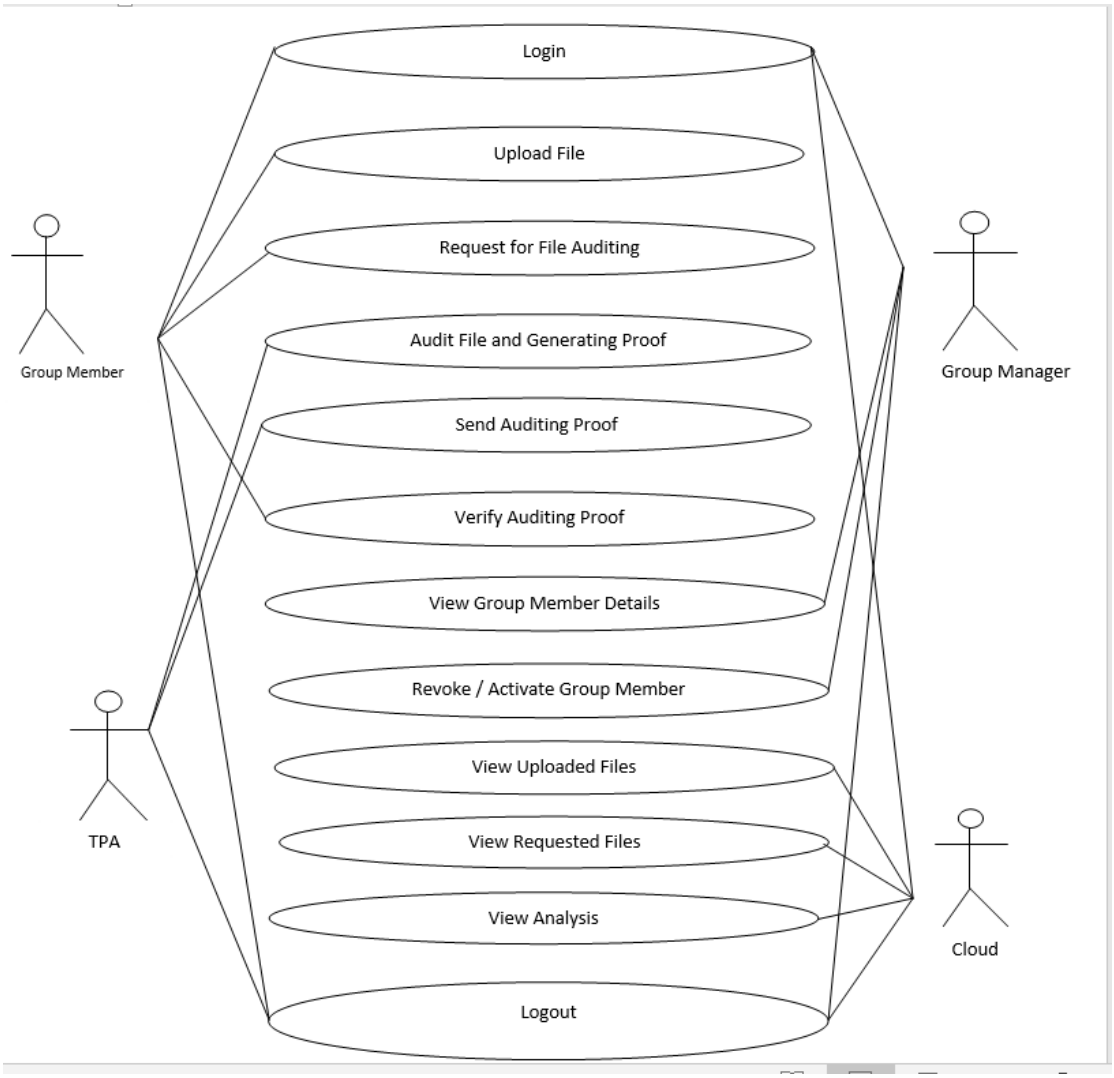
Login
Upload File
Request for File Auditing
Audit File and Generating Proof
Send Auditing Proof
Verify Auditing Proof
View Group Member Details
Revoke / Activate Group Member
View Uploaded Files
View Requested Files
View Analysis
Logout
Group Member
Group Manager
TPA
Cloud

Figure 4.1.1

4.1.2 Sequence Diagram

A sequence diagram represents the interaction between different objects in the system. The important aspect of a sequence diagram is that it is time-ordered. This means that the exact sequence of the interactions between the objects is represented step by step. Different objects in the sequence diagram interact with each other by passing "messages".

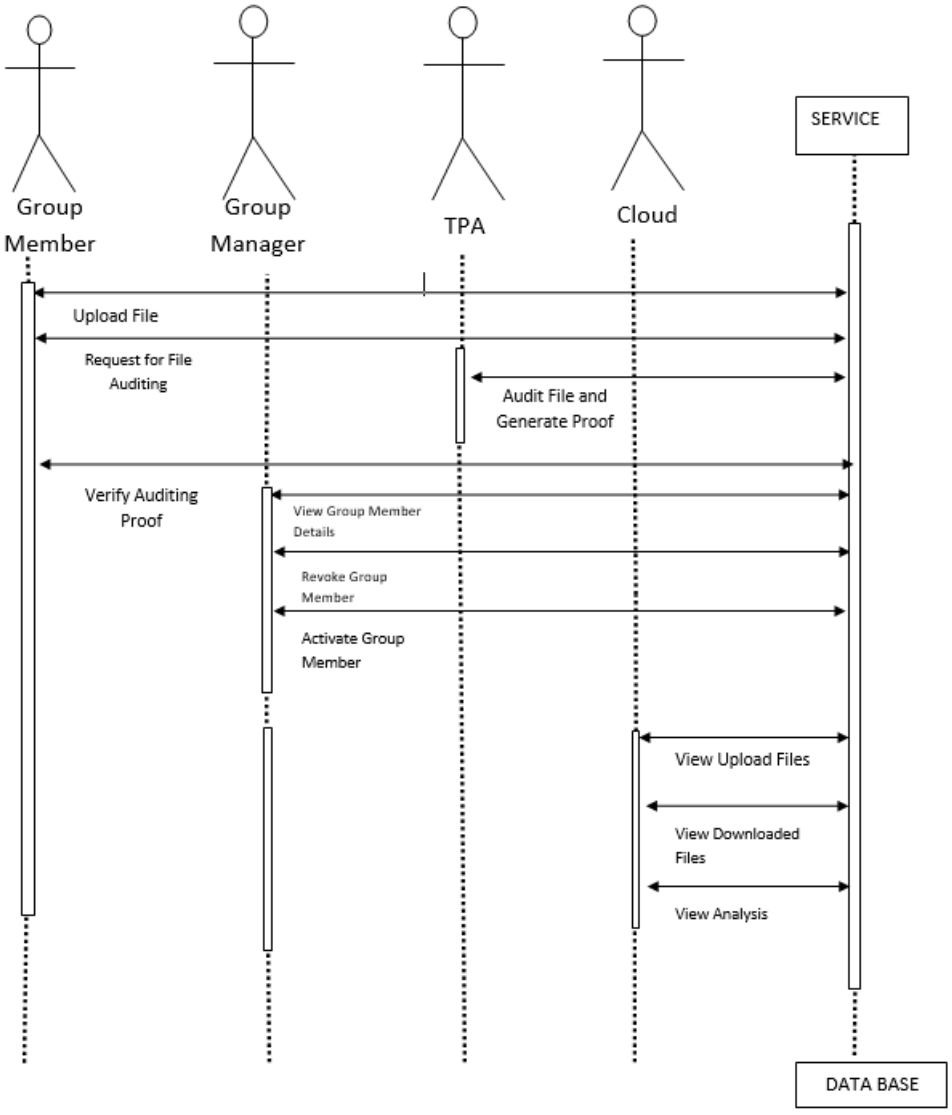

Figure 4.1.2

5. IMPLEMENTATION

5.1 Modules:

- ❖ Member
- ❖ Group Manager
- ❖ TPA
- ❖ Cloud

5.2 Modules Descsription:

Member

The member, including patients, clinic, hospital, medicine center, insurance, etc., are able to outsource large databases to the server. Unlike most auditing schemes, the client generates the aggregated authentication tags locally and sends them to the cloud. Then, the user could query and update the database and check the data storage integrity. In our dynamic group member scenario, any group user can upload own database to the cloud and share them with other group members. And a trusted group manager is responsible for joining or revoking on a client. In the member module, we develop the module, such that when the member logged into, then the member has the options of Upload file, My files, Audit proof, Access File, Requested file, Download file and Leave Group functionalities.

Group Manager

In this module, we develop the Group Manager. Group Manager is like an admin of the system. Once after logged into the system, the Group Manager has the options of seeing the Members details and access Revoked Users. In the Member details, the Group Manager can see the details of a particular member is active state or leave state and also the Group Manager can able to revoke the particular member. In the Revoked users part, the group manager can able to see the list of revoked users and also the group manager has the access of activating the particular member again. The group manager is a powerful entity. It can be viewed as an administrator of the group. When a user leaves the group, the manager is in charge of revoking this user. The revoked user cannot upload data to the cloud any more.

TPA

In this module, we develop TPA (Third Party Auditor). TPA, which can be anyone in the system, checks the data storage integrity of client outsourced database. The TPA could check the data storage integrity of the frequently updated database using public key in an efficient way. In the TPA module, it has the options of Auditing Request and

Auditing Details. The TPA is responsible for auditing the integrity of cloud data on behalf of group users. When the TPA wants to audit the data integrity, it will send an auditing challenge to the cloud. After receiving the auditing challenge, the cloud will respond to the TPA with a proof of data possession. Finally, the TPA will verify the data integrity by checking the correctness of the proof. The TPA is a powerful party and it is honest.

Cloud

The cloud storage server provides remote data storage services for the client. The cloud provides enormous storage space and computing resources for group users. Through the cloud storage, group users can enjoy the data sharing service. Cloud Server provides a public platform for data owners to store and share their encrypted data. The cloud provider doesn’t conduct data access control for owners. The encrypted data can be downloaded freely by any data users. We have used DriveHQ cloud service provider for the storage of files in the cloud part.

Front End and User Interface Design:

The user interface, a critical component, is designed within a browser-specific environment with an emphasis on an Intranet-Based Architecture for distributed concepts. HTML standards lay the foundation for browser-specific components, while Java Server Pages (JSP) add dynamism to the design. Communication architecture hinges on Servlets and Enterprise Java Beans, with Java Database Connectivity (JDBC) acting as the bridge for database connectivity. The three-tier architecture prioritizes higher cohesion and limited coupling for operational effectiveness.

Java Language Choice:

Java, selected as the language for the project, stands out due to its platform independence, cohesiveness, and consistency. Originally named "Oak," Java emerged in 1995 with a primary motivation for a platform-independent, architecture-neutral language suitable for embedded software in various consumer electronic devices. Java offers full control to programmers within the constraints of the Internet environment. Its significance in Internet programming is likened to C's role in system programming.

Java's Impact on the Internet:

Java has left an indelible mark on the Internet, expanding the universe of objects that can move freely in cyberspace. In networked environments, two categories of objects—passive information and dynamic active programs—are transmitted between servers and personal computers. Java applets, small programs dynamically downloaded across the network, introduced a new form of executable programs in cyberspace. Java addresses

security concerns associated with downloading executable programs, ushering in a new era of program development known as Applets.

Java Architecture:

The architecture of Java provides a portable, robust, and high-performance environment for development. Java's uniqueness lies in its bytecode, an optimized set of instructions executed by the Java Virtual Machine (JVM). The JVM, integral to Java technology, acts as an interpreter for bytecode, ensuring portability across various platforms. The development process involves writing Java source code, compilation into bytecode using the Java compiler (javac), and execution in the JVM.

Java Database Connectivity (JDBC):

JDBC, short for Java Database Connectivity, represents a critical Java API designed for executing SQL statements. More than just a set of classes and interfaces, JDBC provides a standard API for tool and database developers, enabling the creation of database applications using pure Java. The three primary functions of JDBC include establishing a connection with a database, sending SQL statements, and processing the results.

Two-Tier and Three-Tier Models:

JDBC supports both the two-tier and three-tier models for database access. In the two-tier model, a Java applet or application communicates directly with the database. This requires a JDBC driver capable of interfacing with the specific database management system. The three-tier model introduces a middle tier, where commands are sent to services that communicate with the database. The three-tier architecture is favored for its enhanced control over access and updates to corporate data, offering performance advantages and a simplified API.

JDBC Driver Types:

Several JDBC driver types exist, categorized into four groups: JDBC-ODBC bridge plus ODBC driver, Native-API partly-Java driver, JDBC-Net pure Java driver, and Native-protocol pure Java driver. Each type caters to specific requirements, with considerations for security, robustness, and ease of use. The choice of JDBC driver depends on the project's needs and the target database.

Java Server Pages (JSP):

Java Server Pages, a technology for creating dynamic-content web pages, emerges as a simple yet powerful tool based on the Java programming language. This architecture facilitates the separation of content generation from presentation, allowing different team members to focus on their expertise. Java Server Pages offer proven portability, open standards, and a mature re-usable component model, making them a preferred choice for building dynamic web applications.

Tomcat 6.0 web server

Tomcat is an open source web server developed by Apache Group. Apache Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlet and Java Server Pages technologies. The Java Servlet and Java Server Pages specifications are developed by Sun under the Java Community Process. Web Servers like Apache Tomcat support only web components while an application server supports web components as well as business components (BEAs Weblogic, is one of the popular

5.3 List of program files

Dbconnection:

This Java code defines a class called `SQLconnection` inside the `DBconnection` package, which is responsible for establishing a connection to a MySQL database. Overall, this class provides a static method `getconnection()` that can be called to obtain a connection to the specified MySQL database. However, note that this code should ideally handle exceptions more gracefully, such as by logging them or propagating them to the caller for handling. Additionally, it's recommended to close the database connection when it's no longer needed to prevent resource leaks.

Cloud.java:

Overall, this servlet handles user authentication for a cloud application by checking the provided username and password against expected values and redirecting the user accordingly. However, it should ideally handle exceptions more gracefully and provide appropriate error messages or logging.

Decription.java:

Overall, this class provides a method `decrypt()` to decrypt text using the AES algorithm with a provided secret key. It's worth noting that using `sun.misc.BASE64Decoder` and `sun.misc.BASE64Encoder` is not recommended because they are internal APIs and might not be available on all Java implementations. It's better to use standard libraries like `java.util.Base64`. Additionally, handling

exceptions more gracefully and securely managing the secret key would improve the robustness and security of this code.

Download.java:

Overall, this servlet facilitates file downloads by retrieving encrypted files from the database, decrypting them, and serving them to users for download. It also records download information in the database for auditing purposes.

GroupMember.java:

Overall, this servlet facilitates login authentication for a Group Manager role. It checks if the provided username and password match the expected values and redirects the user accordingly. However, it should ideally handle exceptions more gracefully and provide appropriate error messages or logging. Additionally, it's recommended to use more secure authentication mechanisms, such as hashing passwords, to enhance security.

Web.Xml:

Overall, this `web.xml` file provides essential configuration information for servlet mappings and session management in the Java web application. It specifies which servlets handle which URL patterns and sets session timeout values.

Cloud.jsp:

This files is about login to the cloud by the credentials.

Member login.jsp,TPA.jsp,GroupMember.jsp:

This all files is about login to the portal through the given roles.

5.4 List of Libraries

Java Database Connectivity (JDBC):

JDBC is used to connect to a MySQL database. It provides a standard interface for Java applications to interact with relational databases.

Apache Tomcat:

Tomcat is a servlet container and web server that implements the Java Servlet and JavaServer Pages (JSP) technologies. It provides a runtime environment for Java web applications.

Java Servlet API:

The Java Servlet API provides the means to define HTTP-specific servlet classes. Servlets are used to extend the capabilities of servers, such as Tomcat, to respond to network requests.

6.EXPERIMENTAL RESULTS

6.1 Experiment setup

Setting up the experimental environment for a project involves preparing the necessary software, databases, and dependencies.

Prerequisites:

Java Development Kit (JDK):

Install the latest version of JDK on the server where your web application will run. You can download it from the official Oracle or OpenJDK website.

Apache Tomcat:

Download and install Apache Tomcat, which will serve as the servlet container for your web application. Configure Tomcat to work with the installed JDK.

MySQL Database:

Install and set up MySQL database server. Create a database named 'EHR' and a user with appropriate privileges. Update the database connection details in the project's JDBC code.

DriveHQ:

Download DriveHQ and create the own account in driveHQ with the perfect credentials https://www.drivehq.com/secure/FreeSignup.aspx.Then go to the source code location: “\src\java\Networks”There you can see “DRIVE_Network.java” file

Web Application Files:

Deploy your web application files (JSP pages, HTML, CSS, JavaScript) into the appropriate directories in the Tomcat webapps folder.

Configuration:

Database Connection Configuration:

Update the database connection details in the JSP files where JDBC connections are established. This includes the URL, username, and password.

Tomcat Configuration:

Adjust Tomcat configuration files if necessary. Specifically, you may need to configure the context for your web application in the server.xml file.

Security Configuration:

Implement necessary security measures such as HTTPS, user authentication, and authorization based on the requirements of your project.

Running the Experiment:

Start MySQL Database:

Ensure that the MySQL database server is running.

Deploy Web Application:

Start the Tomcat server and deploy your web application by placing the WAR file or the project directory in the webapps folder.

Access the Application:

Open a web browser and navigate to the URL where Tomcat is running (e.g., http://localhost:8090/DataTrustFramework/). Access the different roles in the application (Owner, End User, Cloud Service Provider) and perform actions as specified in your project.

Monitoring and Logging:

Implement logging mechanisms to capture relevant events and errors. Monitor Tomcat logs, database logs, and application-specific logs to track the behavior of the system.

Performance Testing (Optional):

If performance evaluation is part of your experiment, consider using tools like Apache JMeter to simulate concurrent users and analyze the system's performance under different loads.

Security Auditing:

Perform security audits using tools like AES or other security scanners to identify vulnerabilities in your application.

6.2 Parameters with Formulas

The Advanced Encryption Standard (AES) is a symmetric key encryption algorithm that is widely used to secure electronic data. It was established as a standard by the U.S. National Institute of Standards and Technology (NIST) in 2001. AES operates on blocks of data, each consisting of 128 bits, and uses keys of 128, 192, or 256 bits in length.

Key Expansion:

KeyExpansion(Key, RoundKeys) // Input: Key (128, 192, or 256 bits),

Output: RoundKeys (derived from Key)

RoundKeys[0] = Key

For i from 1 to Nr (Nr is the total number of rounds):

RoundKeys[i] = KeySchedule(RoundKeys[i-1], i)

Initial Round:

AddRoundKey(State, RoundKeys[0])

Rounds:

For round = 1 to Nr-1:

SubBytes(State)

ShiftRows(State)

MixColumns(State)

AddRoundKey(State, RoundKeys[round])

Final Round:

SubBytes(State)

ShiftRows(State)

AddRoundKey(State, RoundKeys[Nr])

Where the operations are defined as:

SubBytes(State):

Replace each byte of the State matrix with the corresponding byte from a fixed substitution table, S-box.

State[i][j] = S-box[State[i][j]]

ShiftRows(State):

Shift each row of the State matrix cyclically to the left.

Row i is shifted left by i bytes.

This operation ensures that each byte of the input affects multiple bytes of the output.

MixColumns(State):

Treat each column of the State matrix as a polynomial over the finite field GF(2^8).

Multiply each column with a fixed polynomial modulo an irreducible polynomial.

This operation provides diffusion throughout the block.

AddRoundKey(State, RoundKey):

Perform a bitwise XOR operation between each byte of the State matrix and the corresponding byte of the RoundKey. The encryption process takes the plaintext as input and generates the ciphertext, while the decryption process takes the ciphertext as input and generates the plaintext. The decryption process is essentially the same as encryption, but with the key schedule used in reverse order, and the MixColumns operation omitted in the final round.

6.3 Sample Code

SQL Connection:

```
package DBconnection;

import java.sql.Connection;

import java.sql.DriverManager;

public class SQLconnection {

   static Connection con;

   public static Connection getconnection() {

      try {

         Class.forName("com.mysql.jdbc.Driver");

         con = DriverManager.getConnection("jdbc:mysql://localhost:3306/ehr", "root", "root");

      } catch (Exception e) {

      }

      return con;

   }

}
```

Cloud.java:

```
package EHR;

import java.io.IOException;
```

```
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class Cloud extends HttpServlet {
   protected void processRequest(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {
       response.setContentType("text/html;charset=UTF-8");
       try (PrintWriter out = response.getWriter()) {
           String name = request.getParameter("name");
           String pass = request.getParameter("pass");
           System.out.println("======================================="
+name +pass);
           if (name.equalsIgnoreCase("cloud") && pass.equalsIgnoreCase("cloud")) {
               response.sendRedirect("Cloud_Home.jsp?Success");
           } else {
               response.sendRedirect("Cloud_login.jsp?Failed");
           }
       } catch (Exception ex) {
       }
   }

   // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the +
sign on the left to edit the code.">
   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
        throws ServletException, IOException {
    processRequest(request, response);
}


@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    processRequest(request, response);
}
@Override
public String getServletInfo() {
    return "Short description";
}// </editor-fold>

}
```

Decription.java:

```
package EHR;
import com.sun.org.apache.xerces.internal.impl.dv.util.Base64;
import java.io.ByteArrayOutputStream;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.util.Scanner;

import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
```

```
import javax.crypto.spec.SecretKeySpec;
import javax.swing.JOptionPane;
import sun.misc.BASE64Decoder;
import sun.misc.BASE64Encoder;



public class Decryption
{

public String decrypt(String txt,String skey)
{
    String decryptedtext = null;
  try
  {

        //converting string to secretkey
        byte[] bs=Base64.decode(skey);
        SecretKey sec=new SecretKeySpec(bs, "AES");
        System.out.println("converted string to seretkey:"+sec);


      System.out.println("secret key:"+sec);


      Cipher aesCipher = Cipher.getInstance("AES");//getting AES instance
  aesCipher.init(Cipher.ENCRYPT_MODE,sec);//initiating ciper encryption using secretkey
```

```
        byte[] byteCipherText =new BASE64Decoder().decodeBuffer(txt); //encrypting data


         // System.out.println("ciper text:"+byteCipherText);

       aesCipher.init(Cipher.DECRYPT_MODE,sec,aesCipher.getParameters());//initiating ciper decryption


      byte[] byteDecryptedText = aesCipher.doFinal(byteCipherText);

  decryptedtext = new String(byteDecryptedText);


        System.out.println("Decrypted Text:"+decryptedtext);

       }



  catch(Exception e)

  {

            System.out.println(e);

  }

   return decryptedtext;

}


}
```

Download.java:

```
package EHR;

import DBconnection.SQLconnection;

import                                                                    static
com.sun.org.apache.xerces.internal.xinclude.XIncludeHandler.BUFFER_SIZE;

import java.io.IOException;
```

```
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.sql.Blob;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class Download extends HttpServlet {
   protected void processRequest(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {
       response.setContentType("text/html;charset=UTF-8");
       try (PrintWriter out = response.getWriter()) {

           String fileid = request.getParameter("fid");
           String dkey = request.getParameter("dkey");
           System.out.println("Fileid===" + fileid);
```

```
HttpSession user = request.getSession();
String mid = user.getAttribute("mid").toString();
String mname = user.getAttribute("mname").toString();
String mmail = user.getAttribute("mmail").toString();
String mrole = user.getAttribute("mrole").toString();

Connection conn = SQLconnection.getconnection();
Statement st = conn.createStatement();
Statement st1 = conn.createStatement();
Statement st2 = conn.createStatement();

ResultSet rs = st2.executeQuery(" Select * from request where fid ='" + fileid + "' AND status='Approved' AND dkey='" + dkey + "'");
if (rs.next()) {
   String filename = rs.getString("filename");
   ResultSet rs1 = st.executeQuery(" Select * from data_files where id ='" + fileid + "' AND filename ='" + filename + "'  AND dkey='" + dkey + "'");
   if (rs1.next()) {
      String doid = rs1.getString("mid");
      String doname = rs1.getString("mname");
      String file = rs1.getString("enc_data");
      String file1 = rs1.getString("data");

      System.out.println("dkey-- " + dkey);
      long aTime = System.nanoTime();
      Decryption d1 = new Decryption();
      String decrypted = d1.decrypt(file, dkey);
```

```
            long bTime = System.nanoTime();
            float decryptTime = (bTime - aTime) / 1000;

            System.out.print("\nPlain Text            --------  " + file);
            System.out.print("\nDecrypted Text            --------  " + decrypted);
            System.out.println("Time taken to Decrypt File: " + decryptTime + " microseconds.");
            System.out.println("filename,fileid==" + filename + fileid);

            String is = decrypted;

            response.setHeader("Content-Disposition", "attachment;filename=\"" + filename + "\"");
            out.write(file1);
            out.close();
            int i = st1.executeUpdate("insert into download (mid, mname, filename, time , fileid , doname ,doid, decrypt_time, mrole)values('" + mid + "','" + mname + "','" + filename + "',now(),'" + fileid + "','" + doname + "','" + doid + "','" + decryptTime + "','"+ mrole +"')");

            if (i != 0) {
               System.out.println("Download success...");
            } else {
               System.out.println("error  while updating information...");
            }
         } else {
            System.out.println("file not found...");
         }
      } else {
```

```
            response.sendRedirect("Download_file.jsp?failed");
        }

    } catch (SQLException ex) {
        ex.printStackTrace();
        response.sendRedirect("Download_file.jsp?download_failed");
    } catch (IOException ex) {
        ex.printStackTrace();
        response.sendRedirect("Download_file.jsp?download_failed");
    }
}

// <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on the left to edit the code.">
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    processRequest(request, response);
}
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    processRequest(request, response);
}
@Override
public String getServletInfo() {
    return "Short description";
```

```
    }// </editor-fold>
}
```

Encription.java:

```
package EHR;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import sun.misc.BASE64Encoder;
public class Encryption
{
public String encrypt(String text,SecretKey secretkey)
{
    String plainData=null,cipherText=null;
  try
  {
            plainData=text;


      Cipher aesCipher = Cipher.getInstance("AES");//getting AES instance
  aesCipher.init(Cipher.ENCRYPT_MODE,secretkey);//initiating ciper encryption using secretkey


      byte[] byteDataToEncrypt = plainData.getBytes();
  byte[] byteCipherText = aesCipher.doFinal(byteDataToEncrypt);//encrypting data


       //  System.out.println("ciper text:"+byteCipherText);


      cipherText = new BASE64Encoder().encode(byteCipherText);//converting encrypted data to string
```

```
        System.out.println("\n Given text : "+plainData+" \n Cipher Data : "+cipherText);


        }
   catch(Exception e)
   {
            System.out.println(e);
   }
    return cipherText;
}
}
```

GroupMember.java:

```
package EHR;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class GM_Login extends HttpServlet {
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        /* TODO output your page here. You may use following sample code. */
        String name = request.getParameter("name");
```

```
        String pass = request.getParameter("pass");
        System.out.println("======================================="
+name +pass);
        if (name.equalsIgnoreCase("admin") && pass.equalsIgnoreCase("admin")) {
            response.sendRedirect("GM_Home.jsp?Success");
        } else {
            response.sendRedirect("GM_login.jsp?Failed");
        }
    } catch (Exception ex) {
    }
}

// <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the +
sign on the left to edit the code.">
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    processRequest(request, response);
}
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    processRequest(request, response);
}
@Override
public String getServletInfo() {
    return "Short description";
}// </editor-fold>
```

}

Member Registration.java:

```
package EHR;

import DBconnection.SQLconnection;

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class Member_register extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)

        throws ServletException, IOException {

      response.setContentType("text/html;charset=UTF-8");

      try (PrintWriter out = response.getWriter()) {

        String name = request.getParameter("name");

        String mail = request.getParameter("email");

        String pass = request.getParameter("pass");

        String role = request.getParameter("role");

      System.out.println("name" + name + "password" + pass + "Role"+ role +  "mail" + mail);
```

```
Connection conn = SQLconnection.getconnection();
        String message = null;

        try {
            Statement st = conn.createStatement();

            ResultSet rs = st.executeQuery("Select * from member_reg where email ='" + mail + "'");
            if (rs.next()) {

                response.sendRedirect("Member_login.jsp?msg=Mail_Id_Exists");
            } else {

                String sql = "insert into member_reg(name, email, role, password) values (?, ?, ?, ?)";
                PreparedStatement statement = conn.prepareStatement(sql);
                statement.setString(1, name);
                statement.setString(2, mail);
                statement.setString(3, role);
                statement.setString(4, pass);

                int row = statement.executeUpdate();
                if (row > 0) {

                   response.sendRedirect("Member_login.jsp?Register_Success");
                } else {
                   response.sendRedirect("Member_login.jsp?Register_Failed");
```

```
                }
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
        }

    }
}

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on the left to edit the code.">
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
    @Override
    public String getServletInfo() {
        return "Short description";
    }
}
```

Drive Network.java:

```
package Networks;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import org.apache.commons.net.ftp.FTPClient;
import org.apache.commons.net.ftp.FTPConnectionClosedException;
public class DRIVE_Network {

private FTPClient client;
    private FileInputStream fis;
    private boolean status;

    public DRIVE_Network() {
        client = new FTPClient();
    }

    public boolean upload(File file) {
        int maxRetries = 3;
        int retries = 0;

        while (retries < maxRetries) {
            try {
                System.out.println("Connecting to the FTP server...");
                client.connect("ftp.drivehq.com");
                System.out.println("Connected.");

                System.out.println("Logging in...");
```

```
            client.login("GanapuramRamya", "Ramya@123");

            System.out.println("Login successful.");

            client.enterLocalPassiveMode();

            // Check if the file exists and is a regular file
            if (!file.exists() || !file.isFile()) {
                System.out.println("File does not exist or is not a regular file.");
                return false;
            }

            System.out.println("Uploading file: " + file.getAbsolutePath());
            fis = new FileInputStream(file);
            status = client.storeFile("/" + file.getName(), fis);

            if (status) {
                System.out.println("File uploaded successfully.");
            } else {
                System.out.println("File upload failed. Server response: " + client.getReplyString());
            }

            break; // Break out of the loop if upload is successful

        } catch (FTPConnectionClosedException e) {
            System.out.println("FTP Connection Closed Exception. Server response: " + client.getReplyString());
```

```
            // Retry connecting after a delay
            try {
                Thread.sleep(5000); // 5 seconds delay
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }

            retries++;

        } catch (IOException e) {
            e.printStackTrace();
            break; // Break out of the loop if another IOException occurs
        } finally {
            try {
                if (fis != null) {
                    fis.close();
                }
                if (client.isConnected()) {
                    System.out.println("Logging out and disconnecting from the FTP server...");
                    client.logout();
                    client.disconnect();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
```

```
            }
        }

        return status;
    }
}
```

7.TESTCASES

Test case ID	INPUT	Expected Output	Actual Output	Rate
1.	Member Registration	Member Registered successfully.	Member Registered successfully.	success
2.	Member login	Login success	Login success.	success
3.	Member login.	Login success.	Login Failes due to invalid member details.	Failure
4.	TPA	TPA Login success.	TPA Login success and display the functions of TPA.	success
5.	TPA Login.	Login Success.	Login Failed.	Failure

8.SCREENSHOTS

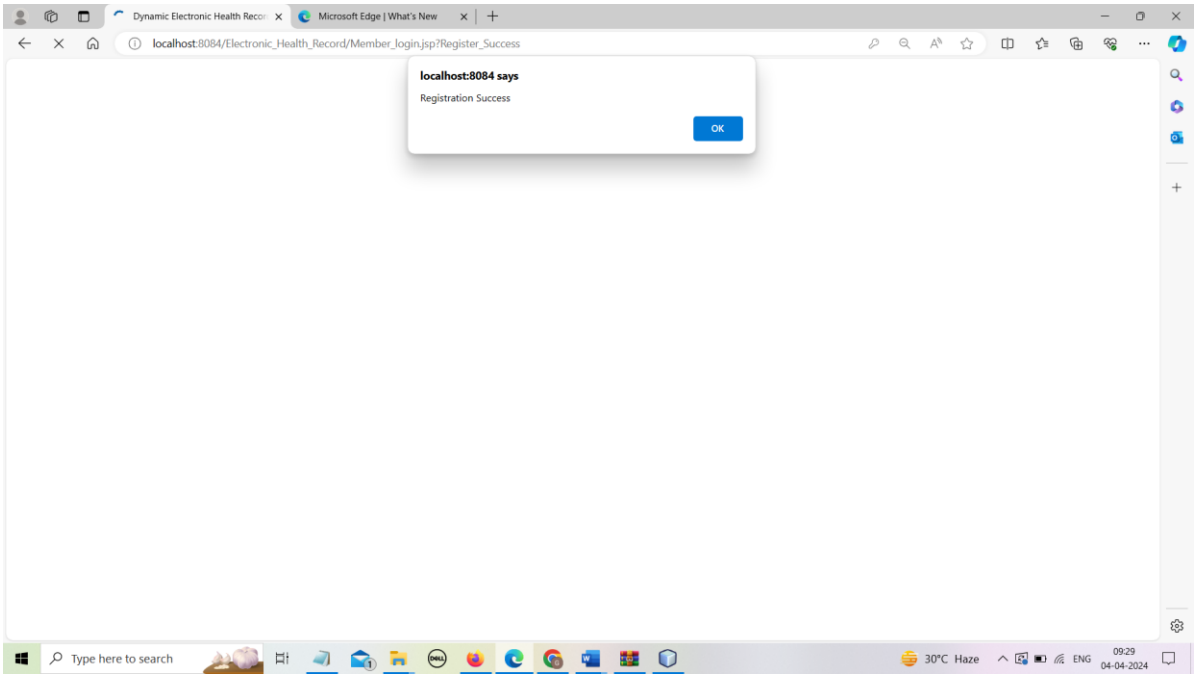

Figure 8.1

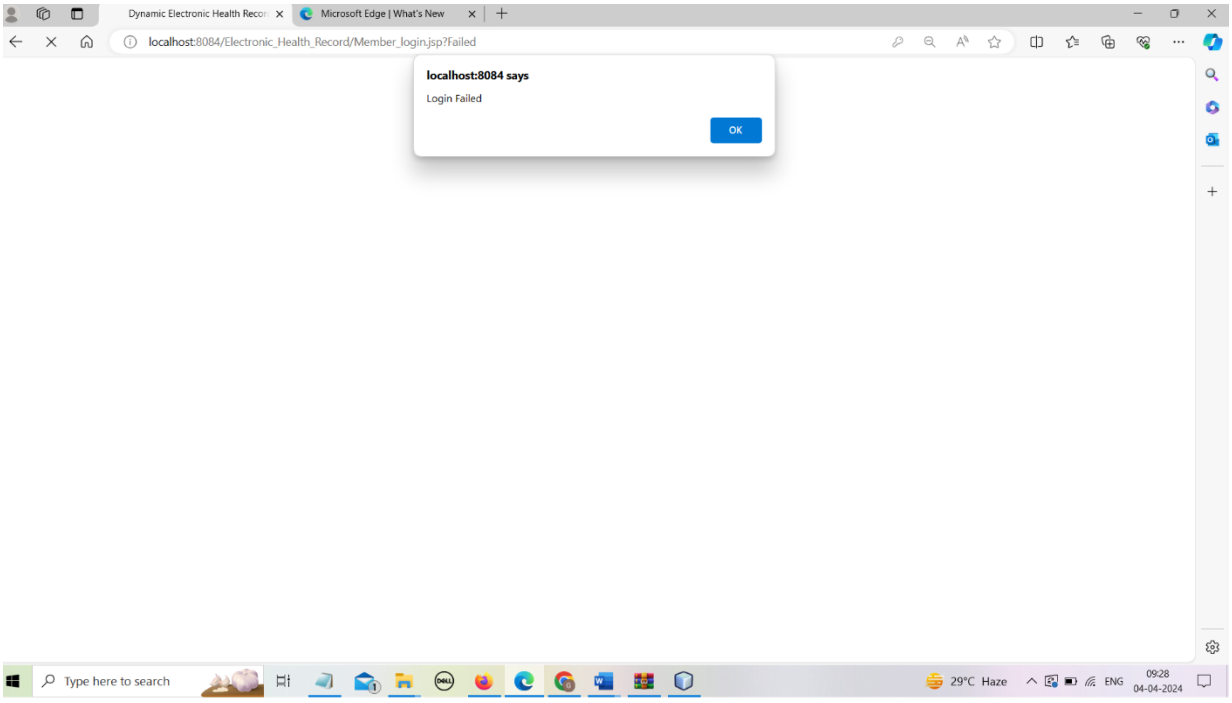

Figure 8.2

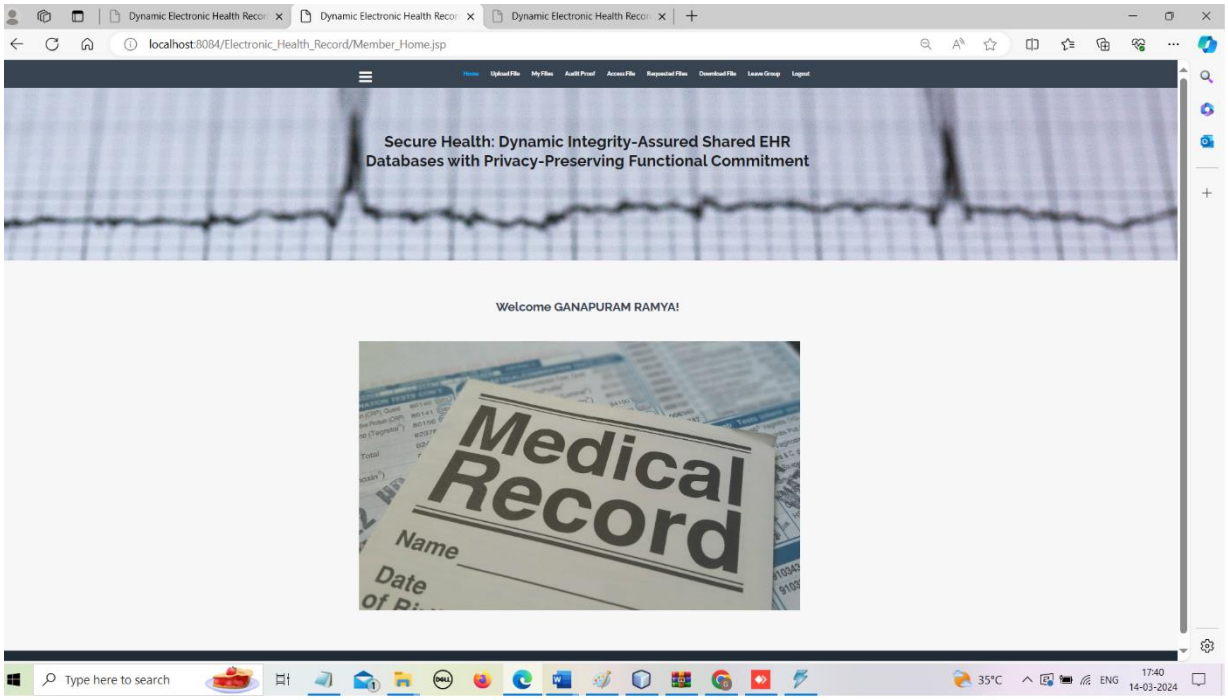

Figure 8.3

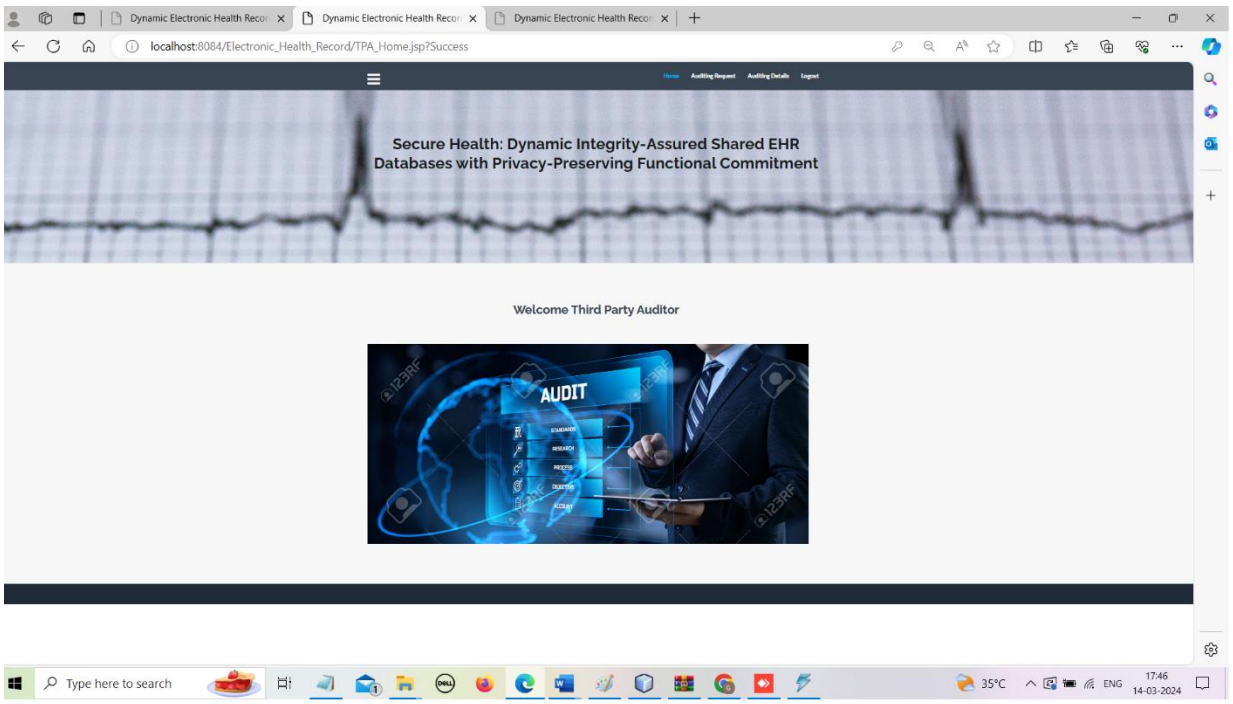

Figure 8.4

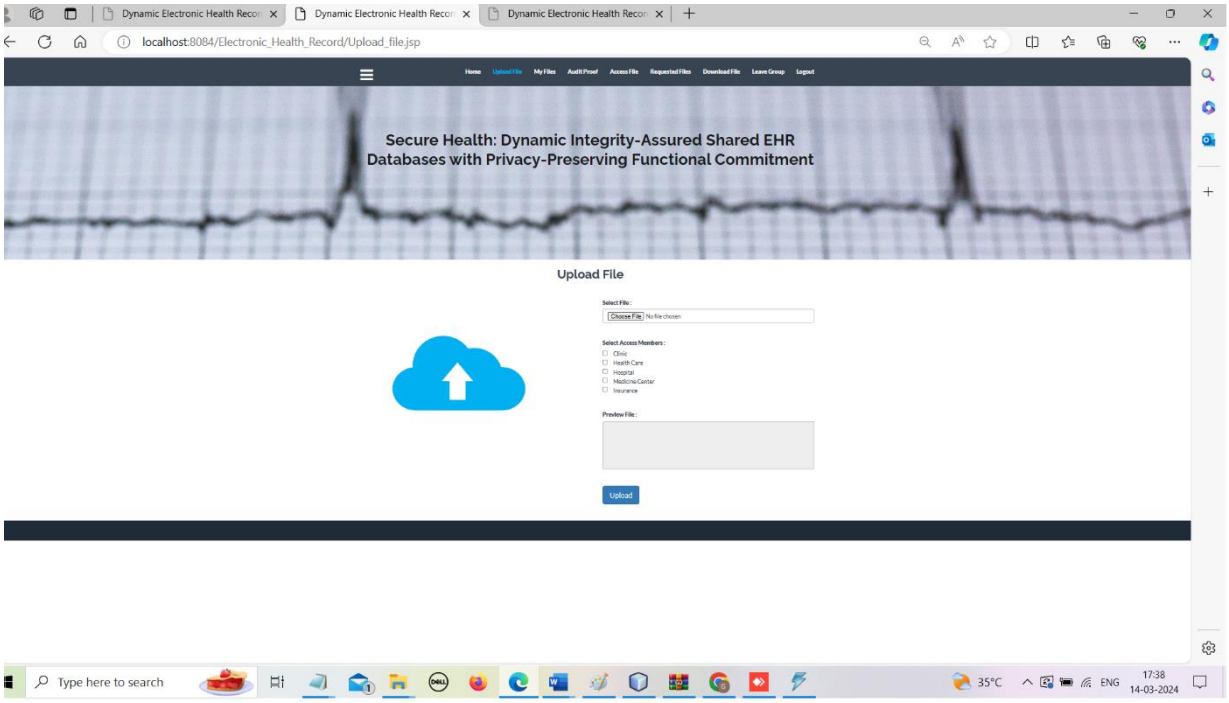

Figure 8.5

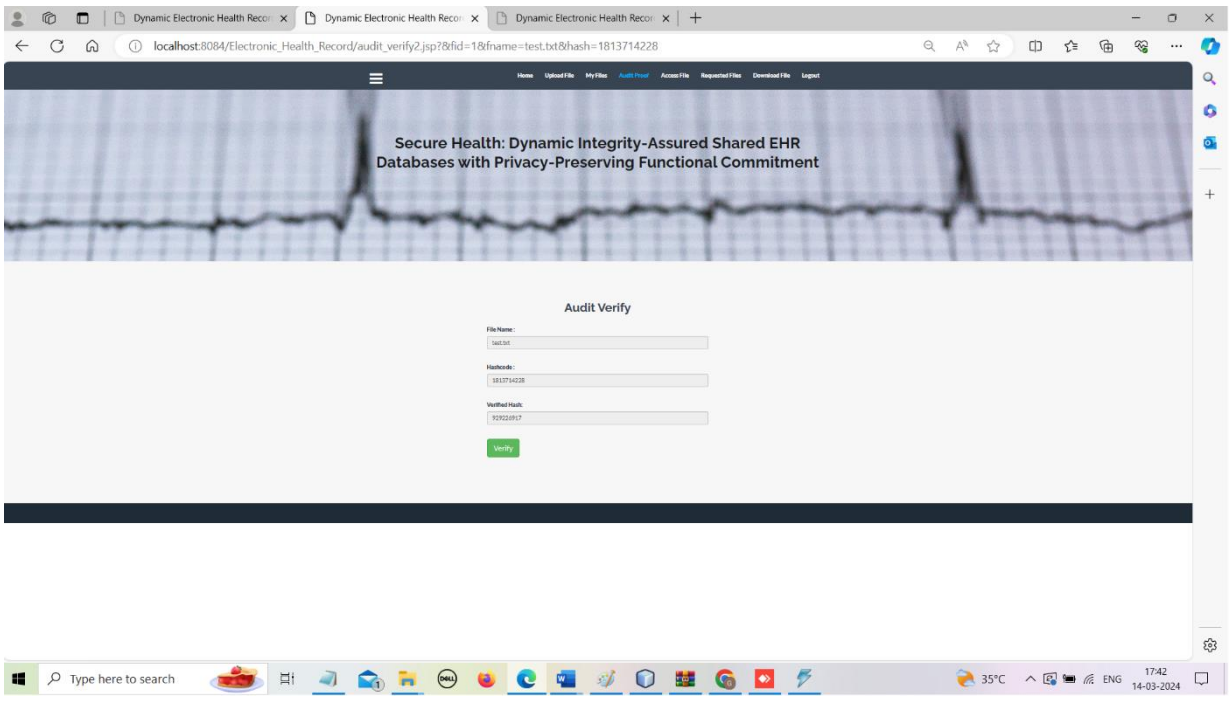

Figure 8.6

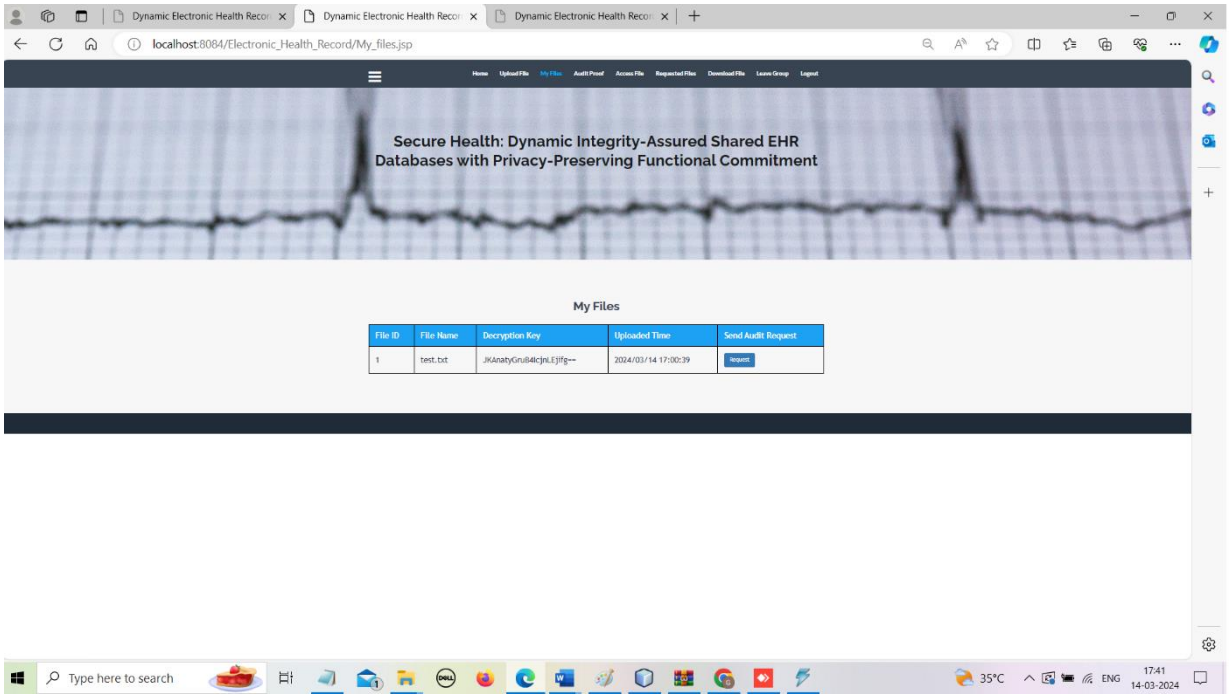

Figure 8.7

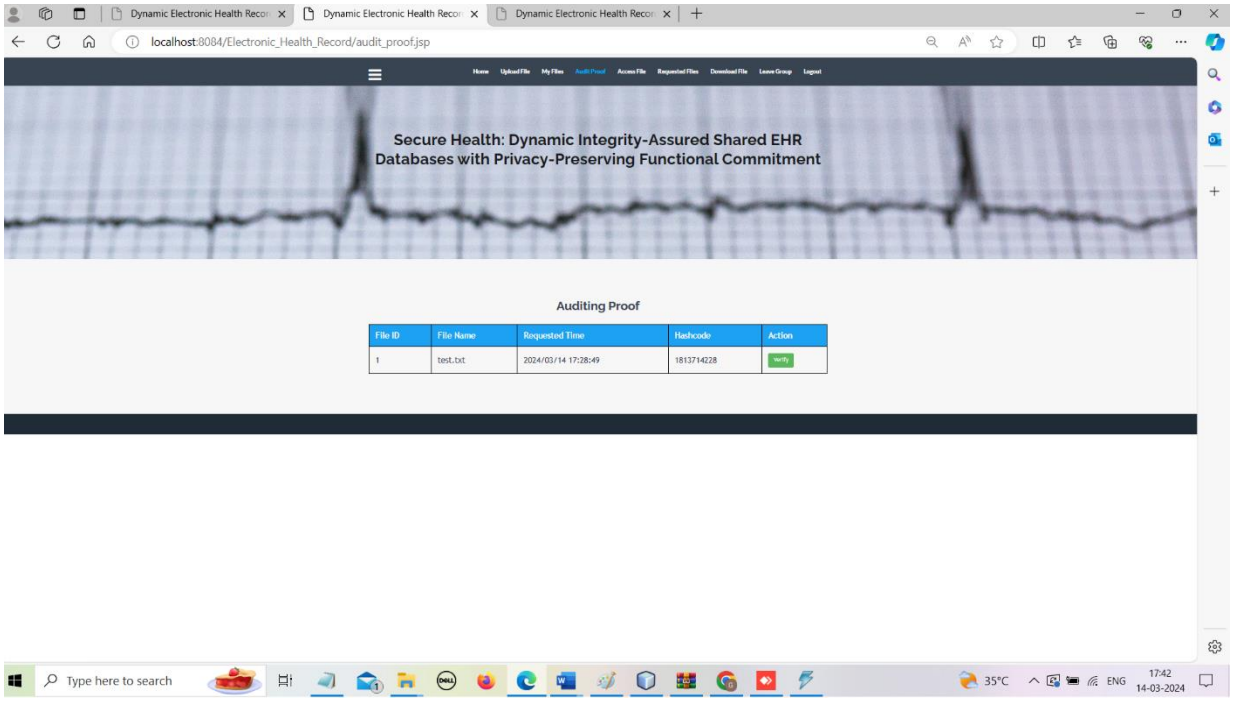

Figure 8.8

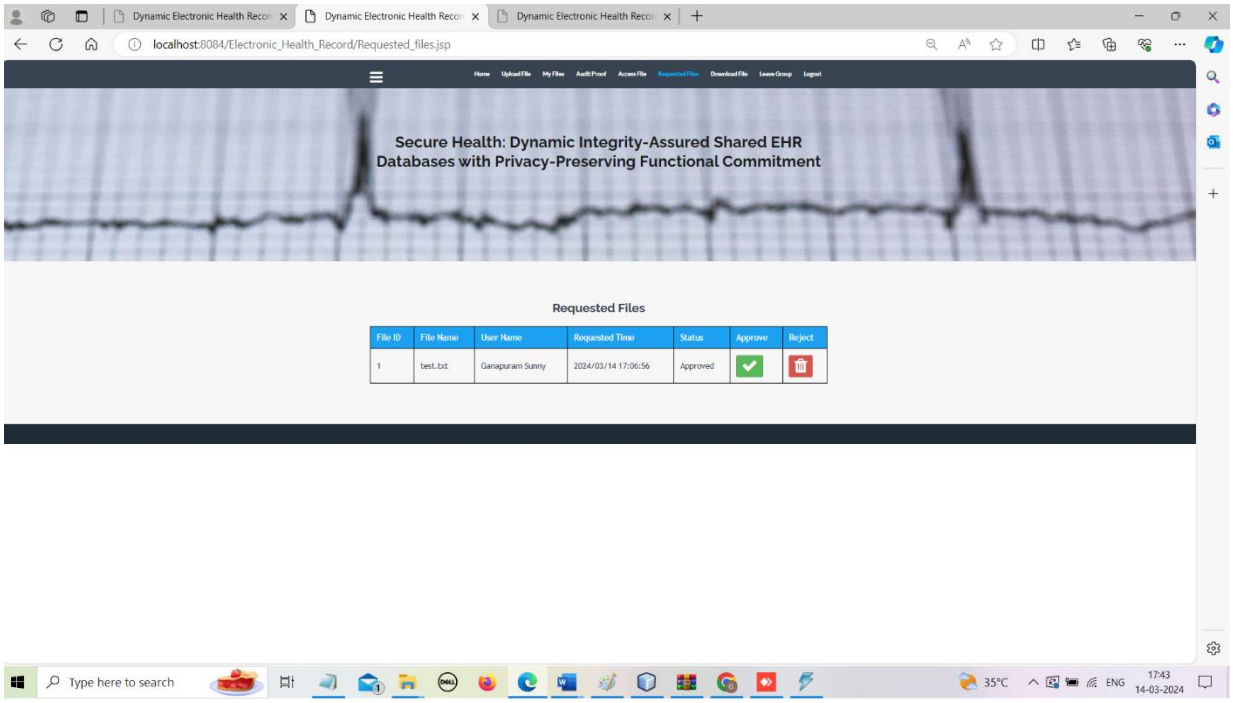

Figure 8.9

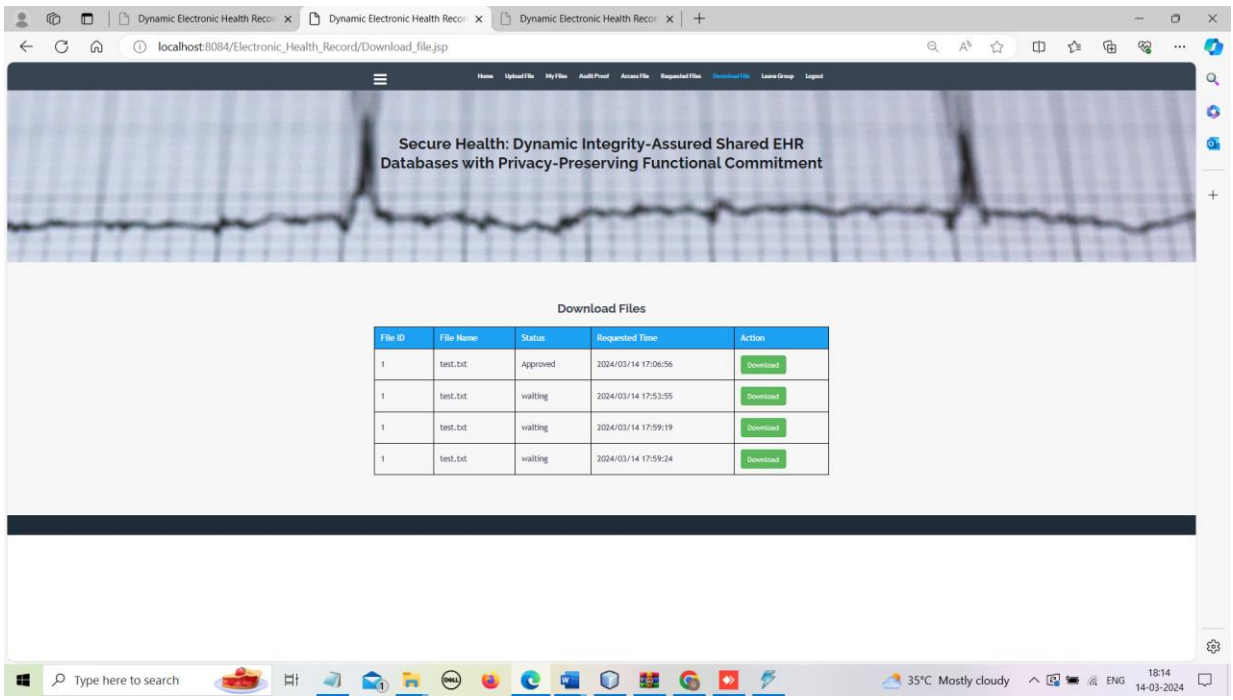

Figure 8.10

9.CONCLUSION

The concept of verifiable database is a great tool for verifiable EHR storage. However, proof reuse and the technique of proof updating by the server to improve system efficiency fails to achieve data integrity checking. In this work, we propose a novel updatable VDB scheme based on the functional commitment that supports privacy-preserving integrity auditing and group member operations, including join and revocation. Two security requirements of HER are implemented: the server response correctness and the data storage integrity. Our VDB scheme achieves the desired security goals without incurring too much computational increase. And our VDB scheme provides the minimum communication cost for the terminal with limited performance. To design a functional commitment scheme that applies to our program, two algorithms are added to make the FC scheme updatable. A practical improved concrete VDB scheme under computational $l-$BDHE assumption is presented. In addition, batch auditing for our VDB scheme supports multi-cloud server, multi-user and multi-storage vector scenarios. It makes the auditing process more efficient. Furthermore, we prove that our functional commitment scheme with updates and our VDB scheme can achieve the desired security properties. The performance of our scheme is more efficient compared with other different algorithms.

10.FUTURE ENHANCEMENT

❖ The scheme preserves data privacy from the auditor by using a random masking technique and the sparse vector is used for sampling auditing.

❖ Our scheme supports dynamic group member operations which include join and revocation. In addition, our VDB supports batch auditing and it supports multi-cloud server, multiuser and multi-storage vector scenarios.

❖ Security analysis and experimental comparison with existing schemes are provided and it shows that our VDB is secure and efficient.

❖ Our VDB scheme can securely and efficiently query and update database stored in the cloud and publicly audit data storage integrity.

11.BIBLIOGRAPHY

[1] Wei L, Wu C, Zhou S. efficient verifier-local revocation group signature schemes with backward unlinkability. Chinese Journal of Electronics, 2021, e90-a(2):379-384.

[2] Dan B, Shacham H. Group signatures with verifier-local revocation. Acm Conference on Computer & Communications Security. 2022.

[3] Chaum, David, and T. P. Pedersen. Wallet Databases with Observers. International Cryptology Conference on Advances in Cryptology 2020.

[4] B. Dan, X. Boyen, E. J. Goh, “Hierarchical identity based encryption with constant size ciphertext”, International Conference on Theory and Applications of Cryptographic Techniques. Springer-Verlag, pp. 440- 456, 2021.

[5] A. Kate, G. M. Zaverucha, I. Goldberg, “Constant-Size Commitments to Polynomials and Their Applications”, Advances in Cryptology - ASIACRYPT 2010 -, International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings. DBLP, pp. 177-194, 2023.

[6] Official Website of The Office of the National Coordinator for Health Information Technology (ONC). (2021). Available: https://www.healthit.gov/

[7] Canada Health Infoway. (2023). Available: https://www.infoway-inforoute.ca/en/