HEALTHCARE IDENTITY REVOCATION MODELS WITH REVOCATION LISTS

Prithivi Raaj K	(RollNo:21Z238)
Rakesh Kumar S	(RollNo:21Z241)
Sureya Narayanan K	(RollNo:21Z261)
Sushanth S	(RollNo:21Z262)
Aswin Sailesh V S	(RollNo:21Z265)

Dissertation submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

Branch: COMPUTER SCIENCE AND ENGINEERING

of Anna University

November 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PSG COLLEGE OF TECHNOLOGY

(Autonomous Institution)

COIMBATORE – 641 004

CONTENTS

SYNOPSIS

The Healthcare Identity Revocation System is designed to address the need for secure and efficient management of healthcare identities, a critical aspect of safeguarding sensitive patient data and preventing unauthorized access in healthcare settings. The system provides functionalities for user registration, identity verification, and identity revocation, ensuring that only authorized individuals can access or modify healthcare information. Built with security as a primary concern, the system leverages cryptographic techniques like OpenSSL for digital signatures and SHA-256 hashing for password protection, ensuring data security throughout the system.

The project utilizes React.js for the frontend, providing an intuitive user interface, while the backend is developed with Flask to handle server-side logic. MySQL serves as the database to securely store user credentials and revocation records. During user registration, passwords are hashed and stored securely, and identity verification ensures that only authorized users can access protected functionalities. This seamless integration between frontend, backend, and database ensures a smooth user experience while maintaining strict access control.

A critical feature of the system is the ability to revoke healthcare identities when necessary, ensuring that compromised or invalid identities are promptly removed. The system maintains a revocation list, recording the reasons for revocation, thus creating an audit trail for future reference and compliance with healthcare regulations. This enhances the system’s role in maintaining data privacy and security, aligning it with legal standards. The system underwent extensive testing, primarily using Postman for API validation.

In conclusion, the Healthcare Identity Revocation System offers a secure and effective framework for managing healthcare identities. With robust cryptographic techniques and well-structured data handling, it provides healthcare organizations with a scalable solution to protect sensitive information and ensure compliance with data security standards.

CHAPTER 1

INTRODUCTION

In today's digital healthcare environment, ensuring the security and proper management of patient identities is critical to maintaining trust and safeguarding sensitive information. The Healthcare Identity Revocation System is a comprehensive solution designed to meet this need by providing a secure and efficient framework for handling healthcare identities. The system offers core functionalities such as user registration, identity verification, and, most importantly, identity revocation. By integrating modern cryptographic techniques, this system guarantees that all sensitive user data is protected through strong encryption and audit trails. It is built with the healthcare industry's stringent security requirements in mind, ensuring compliance and maintaining the privacy and integrity of patient information. The system caters to both healthcare providers and users, offering them an intuitive, secure, and reliable platform for managing identities while mitigating the risks of identity theft or misuse.

1.1 PROBLEM STATEMENT

Managing healthcare identities presents a significant challenge, particularly when it comes to revoking access to compromised or outdated identities. Without a secure and efficient system in place, unauthorized users may continue to exploit revoked credentials, leading to potential data breaches and violations of privacy. Furthermore, traditional identity management systems often lack the necessary auditability and cryptographic strength needed to prevent identity misuse in a rapidly evolving digital landscape. The Healthcare Identity Revocation System addresses these issues by providing a robust mechanism for revoking healthcare identities, ensuring that only authorized individuals can access healthcare services. This solution not only allows healthcare providers to revoke identities seamlessly but also maintains a transparent record of all revocations, along with the reasons for action. Through a combination of secure data handling practices, including the use of digital signatures and hash-based encryption, this system provides a safe and trustworthy environment for managing healthcare identities, thus preventing unauthorized access and enhancing patient data security.

CHAPTER 2

SYSTEM ANALYSIS

This chapter focuses on the hardware and the software requirements essential to develop and implement the system and its module. It also discusses the feasibility of the system.

2.1 HARDWARE REQUIREMENTS

- **Server:** A moderately powerful server is required to host the backend services, including Flask and the MySQL database. The server should support multi-core processing and have sufficient memory (minimum of 8GB RAM) to handle concurrent requests and perform cryptographic operations efficiently.
- **Client Devices:** End-users, including healthcare providers and patients, can access the system via modern web browsers on laptops, desktops, or mobile devices with internet access. The system does not demand high-end client hardware, but it is recommended to have at least 4GB RAM and dual-core processors for seamless interaction.
- **Storage:** Adequate storage is needed for user records, revocation lists, and logs. A minimum of 100GB of disk space is recommended to store hashed user credentials, cryptographic signatures, and other metadata securely.

2.2 SOFTWARE REQUIREMENTS

- **Operating System:** The system is cross-platform, running efficiently on Linux, Windows, and macOS servers. Linux-based servers are preferred for better performance, security, and compatibility with the cryptographic libraries.
- **Frontend:** The frontend is built using React.js, which runs on all modern browsers (Chrome, Firefox, Safari, Edge).
- **Backend:** The backend is powered by Flask, a lightweight Python framework, and requires Python 3.x to run.
- **Database:** MySQL is used for database management, offering a robust, relational structure for securely storing user data.

- **Cryptographic Libraries:** OpenSSL is required for implementing digital signatures, and SHA-256 hashing for password security is implemented via Python's built-in libraries.
- **Version Control:** Git for managing the codebase and ensuring seamless collaboration among developers.

2.3 FUNCTIONAL REQUIREMENTS

The Healthcare Identity Revocation System is designed to meet key functional objectives that ensure secure and effective identity management. The system's main functional requirements are:

- **User Registration:** The system must allow new users (healthcare providers, patients) to register by securely providing their details. Passwords should be hashed using SHA-256 and stored securely in the MySQL database.
- **Identity Verification:** Users must be able to log in using their credentials. The system should verify credentials against the stored hashed password and grant access to authorized features.
- **Identity Revocation:** Authorized users should be able to revoke their healthcare identities if needed. The system must log the reason for revocation and securely delete the user's data from active records while maintaining an entry in the revocation list for future auditing.
- **Manage Revocation List:** The system must maintain a revocation list that tracks all revoked identities, reasons for revocation, and timestamps. This list should be queryable for auditing and compliance purposes.
- **Audit Logs:** The system must keep detailed logs of identity management activities, including registrations, verifications, and revocations, for auditing purposes.

2.4 NON-FUNCTIONAL REQUIREMENTS

In addition to core functionalities, several non-functional requirements ensure that the system operates efficiently, securely, and reliably:

- **Security:** The system must ensure data confidentiality and integrity through robust cryptographic techniques like digital signatures and hashed passwords (SHA-256). All sensitive transactions should be encrypted (HTTPS) to prevent unauthorized access.

- **Scalability:** The system must be scalable to handle an increasing number of users and identity revocations as the healthcare organization grows. The database and application infrastructure should be capable of expanding to meet future demands.
- **Reliability:** The system must be highly reliable, with minimal downtime. Backup mechanisms must be in place to prevent data loss and ensure that the system is operational even in case of hardware or software failures.
- **Performance:** The system should respond quickly to user requests, especially for identity verification and revocation. The backend operations, including hashing and database queries, should be optimized to ensure that response times are minimized.
- **Usability:** The user interface must be intuitive and user-friendly, providing healthcare providers and patients with easy access to the system's core functions without requiring extensive technical knowledge.

2.5 FEASIBILITY

The Healthcare Identity Revocation System is both technically and operationally feasible. The technologies selected—React.js, Flask, MySQL, OpenSSL—are proven, widely adopted, and offer the necessary flexibility to implement a secure identity management system.

- **Technical Feasibility:** The system relies on well-established frameworks and libraries. Flask’s lightweight nature allows for quick development cycles and easy maintenance. The cryptographic techniques, including digital signatures and SHA-256 hashing, are industry-standard methods for secure identity management and are well-supported by existing libraries.
- **Economic Feasibility:** From a cost perspective, the system can be developed and deployed on relatively low-cost hardware, and it leverages open-source software (React.js, Flask, MySQL, OpenSSL), reducing licensing costs. This makes the system affordable for healthcare organizations, especially those with budget constraints.
- **Operational Feasibility:** The system can be seamlessly integrated into the daily operations of healthcare providers. Users with minimal technical expertise can easily manage identities and perform revocation tasks through the system’s intuitive interface, minimizing the need for extensive training. The modular design also allows for future enhancements without major architectural changes, ensuring long-term operational viability.

In conclusion, the Healthcare Identity Revocation System meets the required technical, economic, and operational criteria, offering a secure, scalable, and user-friendly solution for managing healthcare identities efficiently.

CHAPTER 3

SYSTEM DESIGN AND IMPLEMENTATION

The Healthcare Identity Revocation System is designed to provide a secure, scalable, and user-friendly platform for managing healthcare identities. This chapter outlines the system architecture, the design choices made to meet the security and functionality requirements, and the implementation details, including both frontend and backend components. The system employs modern web technologies, cryptographic techniques, and a relational database to ensure robust and efficient identity management.

3.1 SYSTEM ARCHITECTURE

The system follows a client-server architecture, where the frontend, developed using React.js, interacts with the backend, implemented in Flask. The backend handles all business logic and communicates with the MySQL database for storing and retrieving user data. Cryptographic operations, such as password hashing and digital signatures, are performed using OpenSSL and SHA-256, ensuring the security of sensitive information throughout the process.

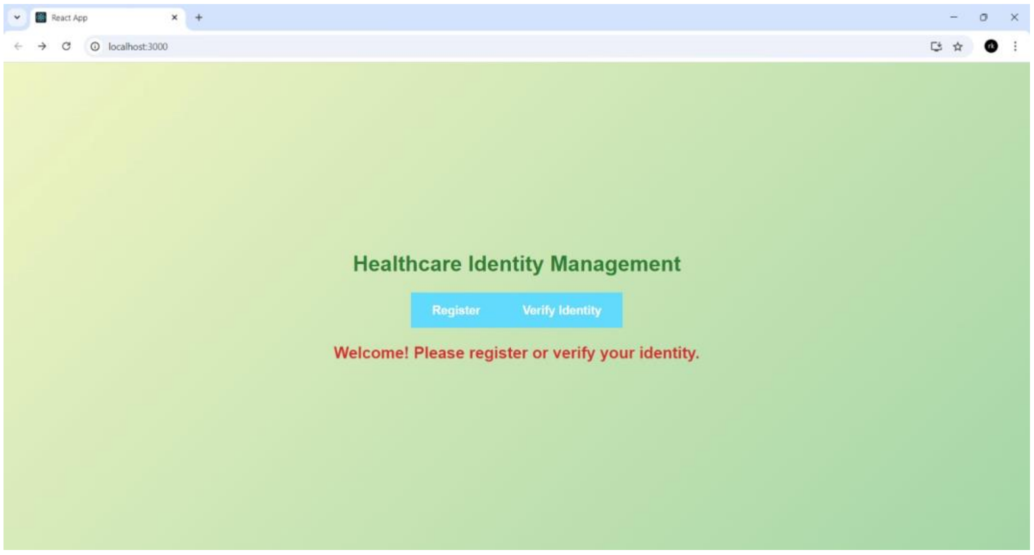
The architecture is modular, with a clear separation of concerns between the frontend, backend, and database layers. This not only ensures scalability and ease of maintenance but also allows for secure communication between components via RESTful APIs.

- **Frontend**: The client-side user interface built with React.js, providing an interactive experience for user registration, identity verification, and revocation of healthcare IDs.
- **Backend**: The server-side logic handled by Flask, responsible for processing requests, performing cryptographic operations, and managing the database.
- **Database**: MySQL serves as the relational database for securely storing user credentials, revocation records, and audit logs.

3.2 MODULES IMPLEMENTED

3.2.1 FRONTEND IMPLEMENTATION

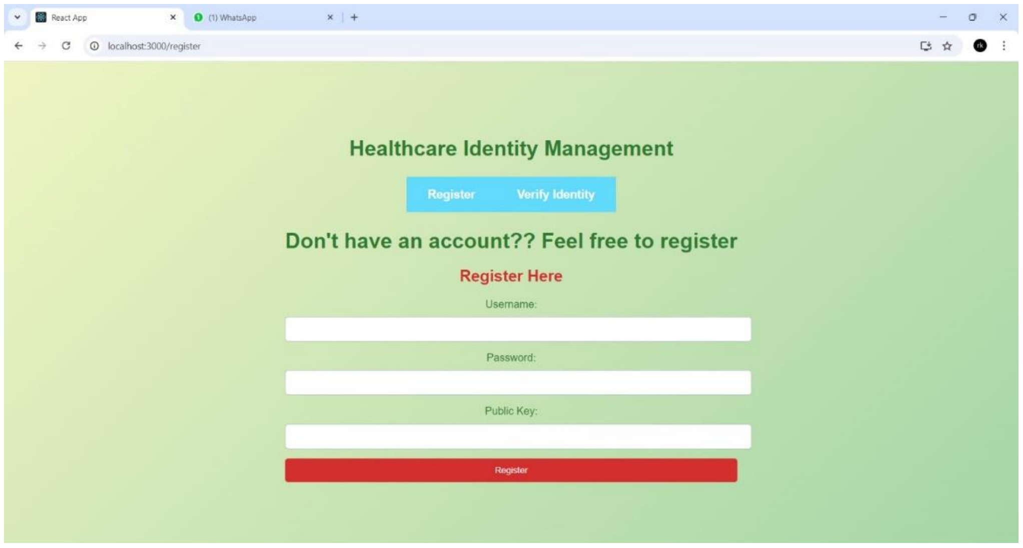
The frontend of the system is developed using **React.js**, a popular JavaScript library for building dynamic and interactive web applications. React allows for the efficient rendering of components and provides a smooth user experience.


React App
localhost:3000
Healthcare Identity Management
Register
Verify Identity
Welcome! Please register or verify your identity.

Key components of the frontend include:

- **Registration Component (Registration.js):**

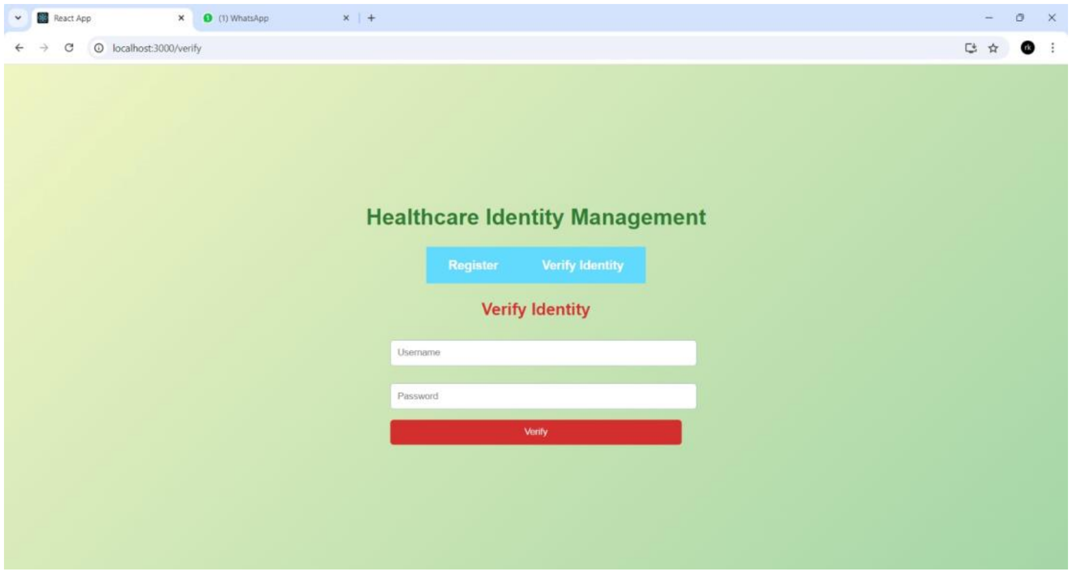
  This component allows users to register by entering their username and password. The frontend ensures that the password is never transmitted in plain text. Upon form submission, the credentials are sent to the backend API for further processing, including password hashing and storage.



- **Verify Identity Component (VerifyIdentity.js):**

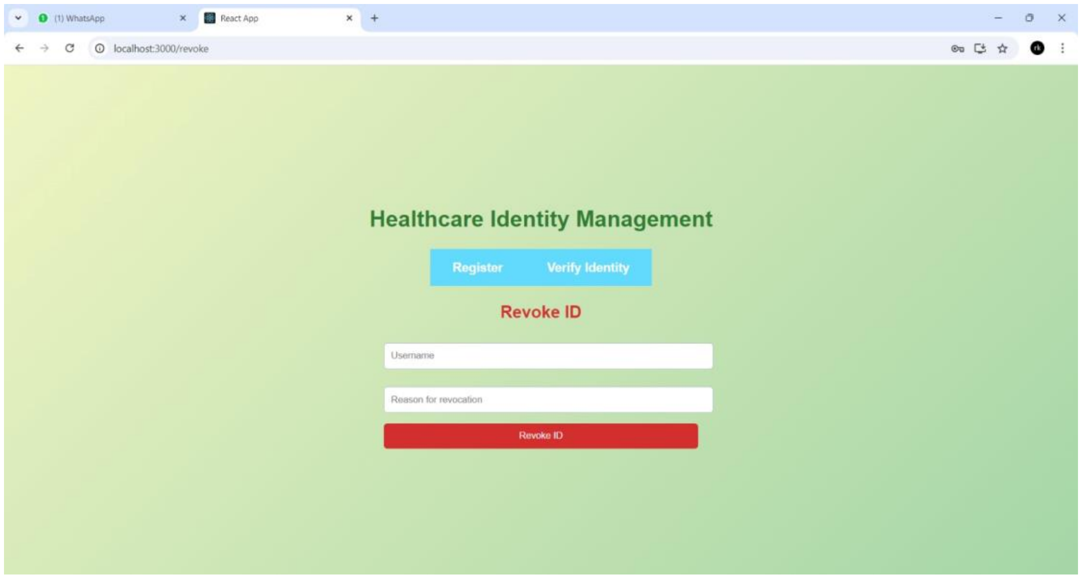
  This component is responsible for accepting the user's login credentials and sending them to the backend for verification. The component captures the username and password inputs, sends them in a POST request to the backend

/verify API, and handles the response to determine if access is granted or denied.



- **Revoke ID Component (RevokeID.js):**

  This component enables users to revoke their healthcare ID for a specified reason. The reason for revocation is recorded and sent to the backend, where the revocation is processed, and the user's data is removed from the active user records.



3.2.2 BACKEND IMPLEMENTATION

The backend of the system is implemented using **Flask**, a lightweight Python web framework. Flask provides the necessary routing and request-handling capabilities to manage the core functionalities of the system, including user registration, identity verification, and identity revocation. Key backend components include:

- **User Registration:**

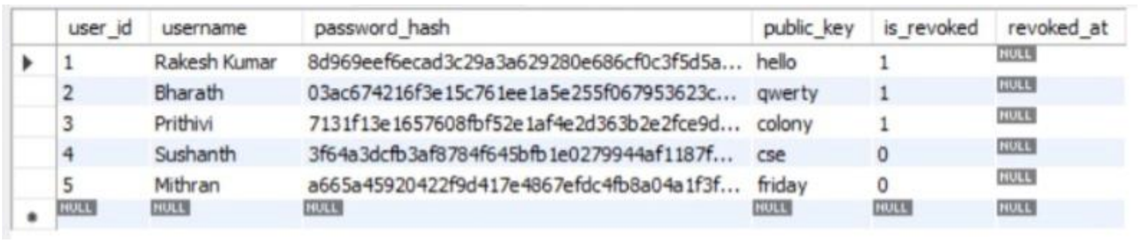
  When a user registers, their password is hashed using SHA-256 before being stored in the MySQL database. This ensures that even if the database is compromised, attackers cannot easily retrieve plain-text passwords.
- **Identity Revocation:**

  Upon receiving a revocation request, the system removes the user's record from the active user list and adds an entry to the revocation list, storing the reason for revocation and the timestamp.

3.2.3 DATABASE IMPLEMENTATION

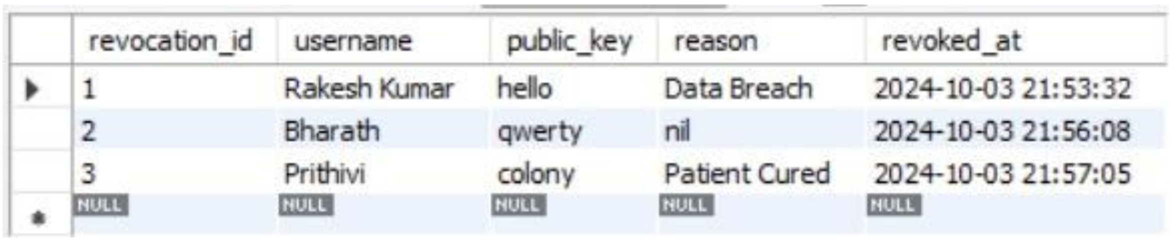
The database is implemented using MySQL, with two primary tables: **users** and **revocation_list**. The users table stores usernames and hashed passwords, while the revocation_list table logs revoked identities, reasons for revocation, and timestamps.

- **Users Table:**
  - username (Primary Key): The unique identifier for each user.
  - password_hash: The SHA-256 hash of the user’s password.



	user_id	username	password_hash	public_key	is_revoked	revoked_at
▶	1	Rakesh Kumar	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a...	hello	1	NULL
	2	Bharath	03ac674216f3e15c761ee1a5e255f067953623c...	qwerty	1	NULL
	3	Prithivi	7131f13e1657608fbf52e1af4e2d363b2e2fce9d...	colony	1	NULL
	4	Sushanth	3f64a3dcfb3af8784f645bfb1e0279944af1187f...	cse	0	NULL
	5	Mithran	a665a45920422f9d417e4867efdc4fb8a04a1f3f...	friday	0	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL

- **Revocation List Table:**
  - username: The username of the revoked user.
  - reason: The reason for revocation.
  - timestamp: The date and time the revocation occurred.



	revocation_id	username	public_key	reason	revoked_at
▶	1	Rakesh Kumar	hello	Data Breach	2024-10-03 21:53:32
	2	Bharath	qwerty	nil	2024-10-03 21:56:08
	3	Prithivi	colony	Patient Cured	2024-10-03 21:57:05
*	NULL	NULL	NULL	NULL	NULL

3.2.4 API IMPLEMENTATION

The backend exposes several RESTful API endpoints that allow the frontend to interact with the system. The key API endpoints include:

- **POST /register:** Registers a new user and hashes their password before storing it in the database.
- **POST /verify:** Verifies a user's identity by comparing the provided password's hash with the stored hash.
- **POST /revoke:** Revokes a healthcare ID, removes the user from the active list, and records the revocation.

CHAPTER 4

TESTING AND VALIDATION

Testing and validation are critical aspects of any software development project, ensuring that the system meets both functional and non-functional requirements. For the Healthcare Identity Revocation System, extensive testing was conducted to validate the correct operation of the system's components, focusing on user interactions, data security, and system robustness.

4.1 API TESTING USING POSTMAN

One of the primary tools used during the testing phase was Postman, a platform that facilitates API development and testing. Since the Healthcare Identity Revocation System relies heavily on RESTful APIs to handle user registration, identity verification, and revocation, Postman was essential for verifying the functionality of these endpoints.

To ensure comprehensive coverage, the following test scenarios were conducted using Postman:

- **User Registration:**
  - **Objective:** Validate the ability of the system to register new users.
  - **Test Case:** A user submits valid credentials (username and password). Postman sends a POST request to the **/register** endpoint, and the system stores the user's information securely in the database. The password is hashed using **SHA-256** before storage to prevent plaintext password exposure.
  - **Expected Result:** The system should return a success message indicating that the user was registered. The user's details, including the hashed password, should be stored correctly in the MySQL database.
- **Identity Verification:**
  - **Objective:** Ensure that the system correctly verifies user credentials.
  - **Test Case:** A user provides their username and password to log into the system. Postman sends a POST request to the **/verify** endpoint, which hashes the provided password and checks it against the stored hash.
  - **Expected Result:** If the credentials are correct, the system should return a success response, allowing the user access. If incorrect, the system should return an error response indicating a login failure.

    - **Special Case:** Incorrect password input was tested to ensure the system does not allow access to unauthorized users. For this, invalid credentials were intentionally provided to test the robustness of the verification process.

- **ID Revocation:**
    - **Objective:** Test the functionality of revoking a user's healthcare identity.
    - **Test Case:** A logged-in user attempts to revoke their healthcare ID for a given reason. Postman sends a POST request to the **/revoke** endpoint with the user's credentials and revocation reason. The backend is expected to delete the user's record from the active users list and add an entry to the **revocation_list** table.
    - **Expected Result:** The system should return a success message confirming that the identity has been revoked. The user should no longer exist in the users table, and a new entry in the revocation_list table should reflect the revocation reason and timestamp.

CHAPTER 5

CONCLUSION

The Healthcare Identity Revocation System successfully addresses the critical need for secure management of healthcare identities, ensuring both privacy and security in an increasingly digital healthcare environment. With the rising concerns surrounding data breaches, identity theft, and unauthorized access to sensitive medical information, this system serves as a robust solution by employing state-of-the-art cryptographic techniques and secure database management practices.

One of the major strengths of the system is its use of OpenSSL for digital signatures and SHA-256 hashing for password protection. These cryptographic measures guarantee that sensitive information, such as passwords and revocation data, remains secure from external threats. By adhering to security best practices, the system ensures that data integrity and confidentiality are maintained at all times.

The system's core functionalities—user registration, identity verification, and identity revocation—are designed to provide a seamless user experience while ensuring strict access control. Users can easily register and verify their credentials, while healthcare providers can securely revoke identities when necessary, ensuring that outdated or compromised credentials are effectively removed from the system. This dual focus on security and usability makes the system highly applicable to real-world healthcare operations.

Additionally, the Healthcare Identity Revocation System has been designed to be scalable and adaptable to meet the needs of healthcare institutions of varying sizes. Whether dealing with a small clinic or a large hospital network, the system can handle growing user bases and complex identity management requirements, while maintaining its core principles of data protection and access control.

In conclusion, the Healthcare Identity Revocation System provides a reliable and secure framework for managing healthcare identities, contributing to the safeguarding of sensitive medical information. Through its use of modern web technologies, secure cryptographic methods, and rigorous testing, the system offers a comprehensive solution for managing identities in the healthcare sector. The system not only prevents unauthorized access but also builds trust between healthcare providers and patients, ensuring a secure and efficient identity management process moving forward.

APPENDIX

Backend Code:

App.py

```
from flask import Flask, request, jsonify
from utils import hash_password, verify_password
import mysql.connector
from datetime import datetime
from flask_cors import CORS

app = Flask(__name__)
CORS(app, resources={r"/register": {"origins": "http://localhost:3000"},
                     r"/verify": {"origins": "http://localhost:3000"},
                     r"/revoke": {"origins": "http://localhost:3000"}})

# Database connection
db = mysql.connector.connect(
    host="localhost",
    user="root",
    password="root",
    database="healthcare"
)
cursor = db.cursor()

# Register User Route
@app.route('/register', methods=['POST'])
def register():
    data = request.json
    username = data['username']
    password = hash_password(data['password'])
    public_key = data['public_key']

    query = "INSERT INTO users (username, password_hash, public_key, is_revoked) VALUES
(%s, %s, %s, %s)"
    cursor.execute(query, (username, password, public_key, False))
    db.commit()
    return jsonify({"message": "User registered successfully"}), 201

# Verify User Identity
@app.route('/verify', methods=['POST'])
def verify_identity():
    data = request.json
    username = data['username']
    password = data['password']
    password_hashed = hash_password(password)
    # Fetch the password hash for the given username
    query = "SELECT password_hash FROM users WHERE username = %s"
    cursor.execute(query, (username,))

    # Check if a user was found and verify the password
    result = cursor.fetchone()
    print(result[0],password_hashed)
    if result and result[0]==password_hashed:
        print("Success")
        return jsonify({"message": "Identity verified successfully"}), 200
    else:
        print("Failed")
        return jsonify({"error": "Invalid username or password"}), 401

# Revoke User ID
@app.route('/revoke', methods=['POST'])
def revoke_id():
    data = request.json
    username = data['username']
    reason = data['reason']

    # Check if user is verified before revoking
    verification_query = "SELECT public_key, is_revoked FROM users WHERE username = %s"
    cursor.execute(verification_query, (username,))
    result = cursor.fetchone()

    if result and not result[1]:  # If user is found and not already revoked
        public_key = result[0]

        # Update the user's revocation status
        query = "UPDATE users SET is_revoked = TRUE WHERE username = %s"
        cursor.execute(query, (username,))

        # Insert into the revocation list
        revocation_query = "INSERT INTO revocation_list (username, public_key, reason, 
revoked_at) VALUES (%s, %s, %s, %s)"
        cursor.execute(revocation_query, (username, public_key, reason, datetime.now()))

        db.commit()  # Commit changes to the database
        return jsonify({"message": "ID revoked successfully", "reason": reason}), 200
    else:
        return jsonify({"error": "User not verified or already revoked"}), 403


# Handle 404 errors
@app.errorhandler(404)
def not_found(e):
    return jsonify(error=str(e)), 404

if __name__ == '__main__':
    app.run(debug=True)
```

Utils.py

```
from hashlib import sha256
from Crypto.PublicKey import RSA
from Crypto.Signature import pkcs1_15
from Crypto.Hash import SHA256

def hash_password(password):
    """Hashes a password using SHA-256."""
    return sha256(password.encode()).hexdigest()

def verify_password(stored_password, provided_password):
    """Verifies a provided password against a stored hashed password."""
    return hash_password(provided_password) == stored_password

def generate_signature(message, private_key_str):
    """Generates a digital signature for a given message using a private key."""
    private_key = RSA.import_key(private_key_str)
    h = SHA256.new(message.encode())
    signature = pkcs1_15.new(private_key).sign(h)
    return signature.hex()
```

Frontend Code:

App.js

```
import React from 'react';
import { BrowserRouter as Router, Route, Routes, Link } from 'react-router-dom';
import './App.css';
import Registration from './components/Registration';
import VerifyIdentity from './components/IdentityVerification';
import RevokeID from './components/RevokeId';

function App() {
  return (
    <Router>
      <div className="App">
        <h1>Healthcare Identity Management</h1>
        <nav>
        <Link to="/register" className="nav-button">Register</Link>
        <Link to="/verify" className="nav-button">Verify Identity</Link>
        </nav>
        <Routes>
          <Route path="/" element={<h2>Welcome! Please register or verify your identity.</h2>} />
          <Route path="/register" element={<Registration />} />
          <Route path="/verify" element={<VerifyIdentity />} />
          <Route path="/revoke" element={<RevokeID />} />

        </Routes>
      </div>
    </Router>
  );
}

export default App;
```

Registration.js

```
import React, { useState } from 'react';
import './Registration.css';
const Registration = () => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [publicKey, setPublicKey] = useState('');
  const [error, setError] = useState('');
  const [successMessage, setSuccessMessage] = useState('');

  const handleRegister = async (event) => {
    event.preventDefault(); // Prevent the default form submission

    const userData = {
      username,
      password,
      public_key: publicKey,
    };

    try {
      const response = await fetch('http://localhost:5000/register', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(userData),
      });

      if (response.ok) {
        const data = await response.json();
        setSuccessMessage('Registration successful: ' + data.message);
        setError(''); // Clear any previous errors
      } else {
        const errorData = await response.json();
        setError('Registration failed: ' + (errorData.message || 'Unknown error'));
        setSuccessMessage(''); // Clear any previous success messages
      }
    } catch (error) {
      setError('Error during registration: ' + error.message);
      setSuccessMessage(''); // Clear any previous success messages
    }
  };
```

```
    return (
        <div>
            <h1>Don't have an account?? Feel free to register</h1>
            <h2>Register Here</h2>
            <form onSubmit={handleRegister}>
                <div>
                    <label>Username:</label>
                    <input
                        type="text"
                        value={username}
                        onChange={(e) => setUsername(e.target.value)}
                        required
                    />
                </div>
                <div>
                    <label>Password:</label>
                    <input
                        type="password"
                        value={password}
                        onChange={(e) => setPassword(e.target.value)}
                        required
                    />
                </div>
                <div>
                    <label>Public Key:</label>
                    <input
                        type="text"
                        value={publicKey}
                        onChange={(e) => setPublicKey(e.target.value)}
                        required
                    />
                </div>
                <button type="submit">Register</button>
            </form>
            {error && <p style={{ color: 'red' }}>{error}</p>}
            {successMessage && <p style={{ color: 'green' }}>{successMessage}</p>}
        </div>
    );
};
```

```
export default Registration;
```

IdentityVerification.js

```
import React, { useState } from 'react';
import './IdentityVerification.css';
const VerifyIdentity = () => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');

  const handleVerify = async (e) => {
    e.preventDefault();
    try {
      const response = await fetch('http://localhost:5000/verify', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ username, password }),
      });

      if (!response.ok) {
        throw new Error('Verification failed');
      }

      // If verification is successful, redirect to the revoke page
      window.location.href = '/revoke'; // Redirect to the revoke ID page
    } catch (err) {
      setError(err.message);
    }
  };

  return (
    <div>
      <h2>Verify Identity</h2>
      <form onSubmit={handleVerify}>
        <input
          type="text"
          placeholder="Username"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
```

```
                required
            />
            <input
                type="password"
                placeholder="Password"
                value={password}
                onChange={(e) => setPassword(e.target.value)}
                required
            />
            <button type="submit">Verify</button>
            {error && <p style={{ color: 'red' }}>{error}</p>}
        </form>
    </div>
  );
};

export default VerifyIdentity;
```

RevokeID.js

```
import React, { useState } from 'react';
import './RevokeId.css';
const RevokeID = () => {
  const [username, setUsername] = useState('');  // Store username here
  const [reason, setReason] = useState('');

  const handleRevoke = async () => {
    try {
      const response = await fetch('http://localhost:5000/revoke', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ username, reason }),
      });

      if (response.ok) {
        alert('ID revoked successfully');
      } else {
        const errorData = await response.json();
        alert(`Failed to revoke ID: ${errorData.error}`);
      }
```

```
        } catch (err) {
            console.error(err);
            alert('An error occurred');
        }
    };

    return (
        <div>
            <h2>Revoke ID</h2>
            <input
                type="text"
                placeholder="Username"
                value={username}  // Change to use username state
                onChange={(e) => setUsername(e.target.value)}
                required
            />
            <input
                type="text"
                placeholder="Reason for revocation"
                value={reason}
                onChange={(e) => setReason(e.target.value)}
                required
            />
            <button onClick={handleRevoke}>Revoke ID</button>
        </div>
    );
};

export default RevokeID;
```