ABAC Health Care Model

Bonafide record of work done by

AKIL K (21z204)
JAVAGAR M (21z221)
S BHARATH (21z246)
SANTHOSH A (21z250)
VENKATPRASADH MARI (21z268)

19Z701 – CRYPTOGRAPHY

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

BACHELOR OF ENGINEERING
BRANCH: COMPUTER SCIENCE AND ENGINEERING

OCTOBER 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
PSG COLLEGE OF TECHNOLOGY
(Autonomous Institution)
COIMBATORE – 641 004

PSG COLLEGE OF TECHNOLOGY
(Autonomous Institution)
COIMBATORE – 641 004

ABAC Health Care Model
Bona fide record of work done by

AKIL K (21z204)
JAVAGAR M (21z221)
S BHARATH (21z246)
SANTHOSH A (21z250)
VENKATPRASADH MARI (21z268)

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

BACHELOR OF ENGINEERING
Branch: COMPUTER SCIENCE AND ENGINEERING
of Anna University
October 2024

Certified that the candidate was examined in the viva-voce examination held

on …….

TABLE OF CONTENTS

Synopsis:

The objective of this project is to develop a secure and efficient web-based system that enables doctors and patients to manage healthcare records. This system provides separate login functionalities for doctors and patients, ensures secure data handling using encryption, and implements Attribute-Based Access Control (ABAC) for managing permissions. The project focuses on the confidentiality of patient information, where prescriptions and sensitive medical details are encrypted and can only be accessed by authorized users.

Introduction:

In today's rapidly evolving healthcare environment, the need for secure and efficient management of patient data has become more critical than ever. Medical professionals require systems that not only streamline the management of patient records but also ensure the confidentiality and integrity of sensitive health information. As healthcare organizations move toward digital transformation, ensuring data security and privacy, particularly for patient information, is a top priority.

The Doctor-Patient Management System aims to address these needs by providing a secure web-based platform where doctors and patients can manage healthcare information in a streamlined and user-friendly manner. This system leverages cutting-edge web technologies like Python Flask for the backend, MongoDB for data storage, and Fernet encryption to secure sensitive information, particularly patient prescriptions. It also implements Attribute-Based Access Control (ABAC) to ensure that only authorized users—either doctors or patients—can access and manage the data relevant to their role.

The system is designed with two key roles in mind: Doctors and Patients. Doctors can log in to the system to view and manage their patients' medical records, including writing prescriptions that are securely encrypted before being stored in the database. Patients, on the other hand, can log in to view their personal information and prescriptions, but cannot modify any data, thus ensuring data integrity.

One of the primary focuses of the system is security. The use of BCrypt for password hashing ensures that user credentials are stored securely. Additionally, patient prescriptions are encrypted using Fernet encryption, guaranteeing that sensitive medical data is protected, even if the database is compromised. ABAC further strengthens the system's security by enforcing role-based access control, ensuring that a doctor can only view the records of their own patients, while a patient can only access their own personal information and prescriptions.

By combining ease of use with robust security measures, this system provides a solution that enhances the way doctors and patients interact with healthcare information. Whether in a clinic, hospital, or private practice, this Doctor-Patient Management System offers a secure, scalable, and efficient approach to managing patient data, ensuring that healthcare professionals and patients alike can confidently access and manage medical records without compromising on security.

Problem Statement:

Title: Attribute-Based Access Control Models with Threshold Cryptography for Healthcare

The management of sensitive patient data in healthcare requires robust and fine-grained access control mechanisms to ensure that only authorized personnel can access or modify this information. Traditional Role-Based Access Control (RBAC) systems often lack the flexibility needed to handle the dynamic and complex access requirements of modern healthcare environments. Attribute-Based Access Control (ABAC) models offer a solution by defining access policies based on a combination of user, resource, and environmental attributes.

This project aims to design mathematical models for ABAC specifically tailored for healthcare applications, ensuring that access policies are both secure and adaptable to various contexts. To enhance the security of these access control mechanisms, the project will explore the use of cryptographic threshold schemes. These schemes distribute cryptographic operations among multiple parties, ensuring that no single party can unilaterally control access to sensitive data. By integrating ABAC with threshold cryptography and implementing these models within smart contracts, the project seeks to enforce fine-grained access policies in a decentralized and automated manner. This approach will not only improve the security and flexibility of access control in healthcare but also provide a practical framework for real-world implementation, making it an ideal subject for a college project.

Objectives:

1. Design Tailored ABAC Models: Create mathematical models for ABAC that are specifically suited to the dynamic and sensitive nature of healthcare data. These models will define access policies based on a comprehensive set of user, resource, and environmental attributes.

2. Enhance Security with Threshold Cryptography: Incorporate threshold cryptography schemes to distribute cryptographic operations among multiple parties, thereby enhancing the security of the access control mechanism. This ensures that no single entity can unilaterally access or control sensitive patient data.

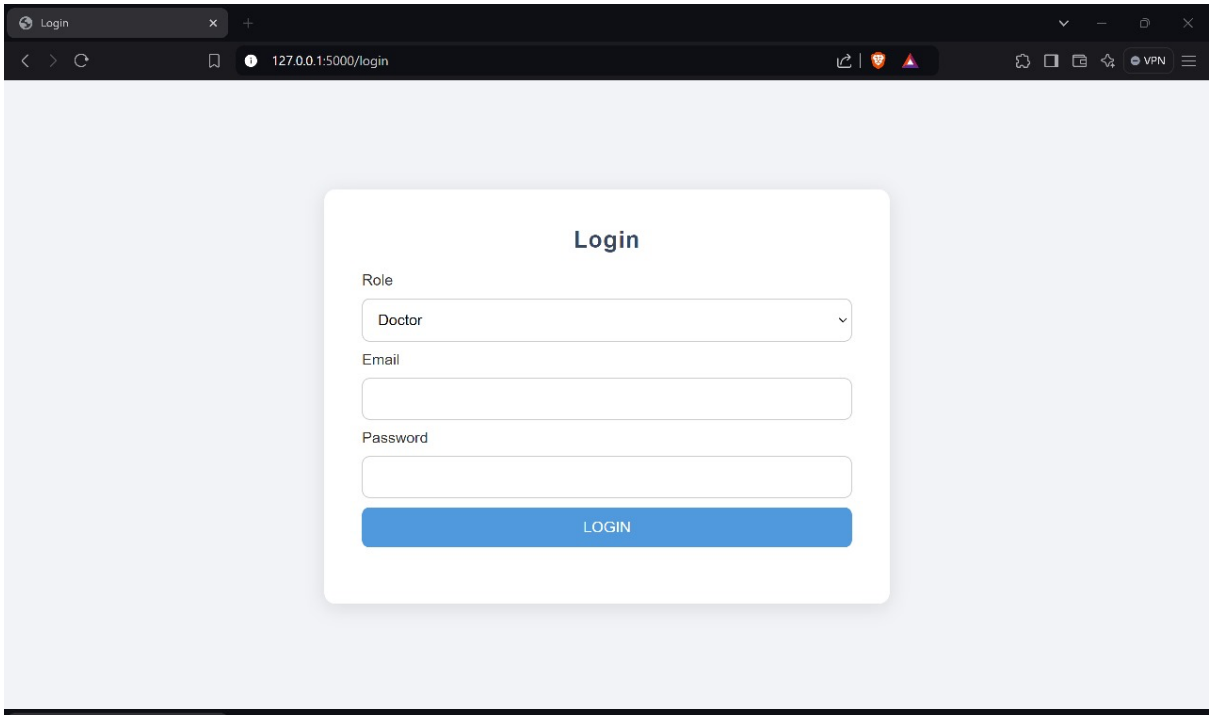
3. Implement Smart Contracts: Develop smart contracts to automate the enforcement of ABAC policies and the execution of threshold cryptographic operations on a blockchain platform. This will provide a decentralized, transparent, and tamper-resistant framework for managing access to healthcare data.

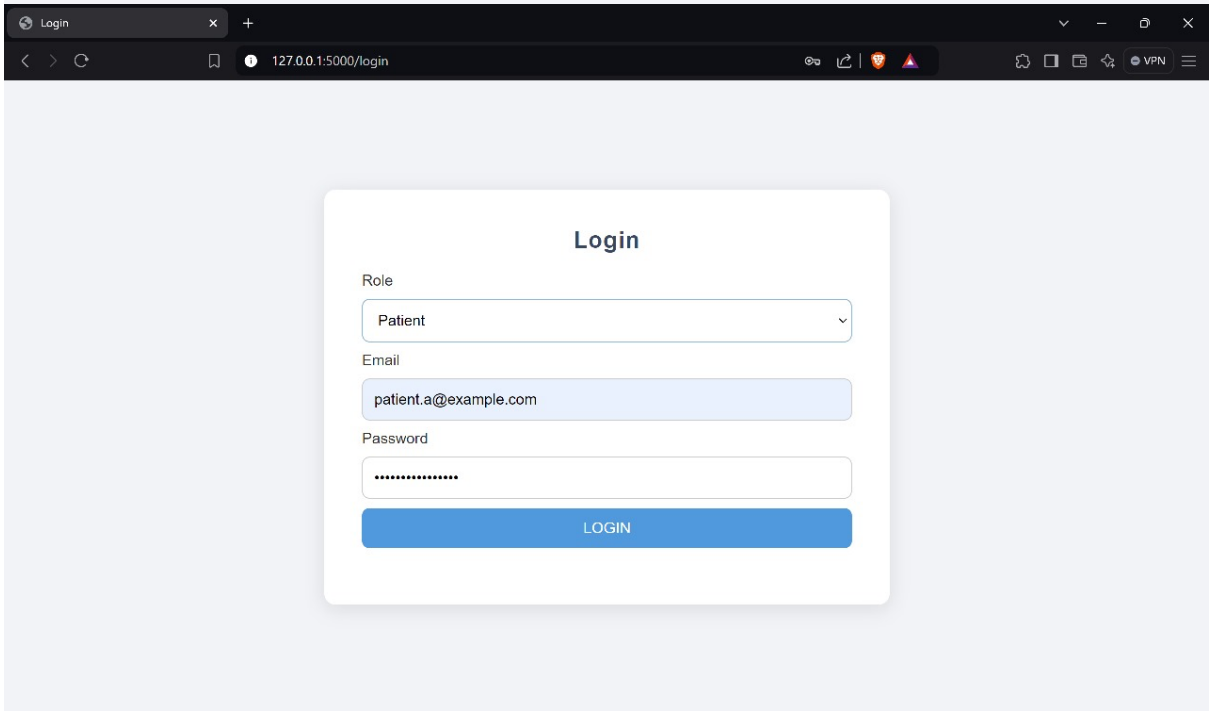
4. Evaluate and Test: Conduct thorough testing and evaluation of the proposed models to assess their security, performance, and scalability in simulated healthcare scenarios. The goal is to demonstrate the practical feasibility and effectiveness of the integrated ABAC and threshold cryptography approach in real-world healthcare environments.

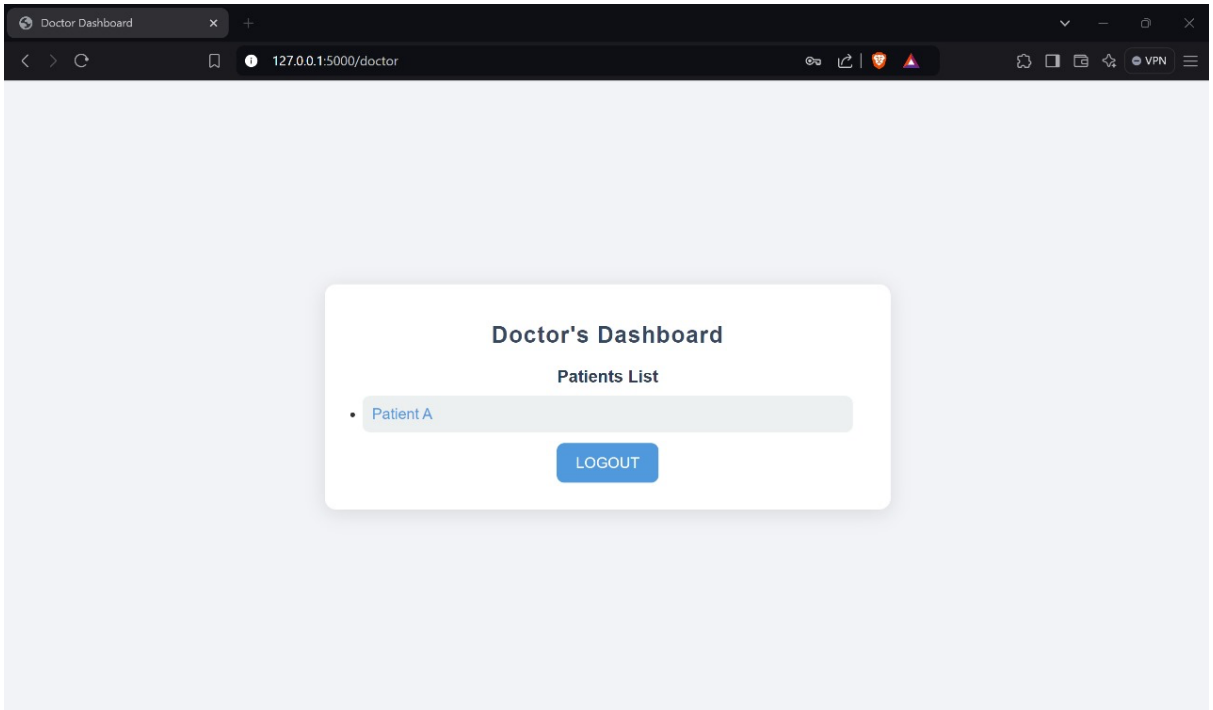
Tools to be Used:

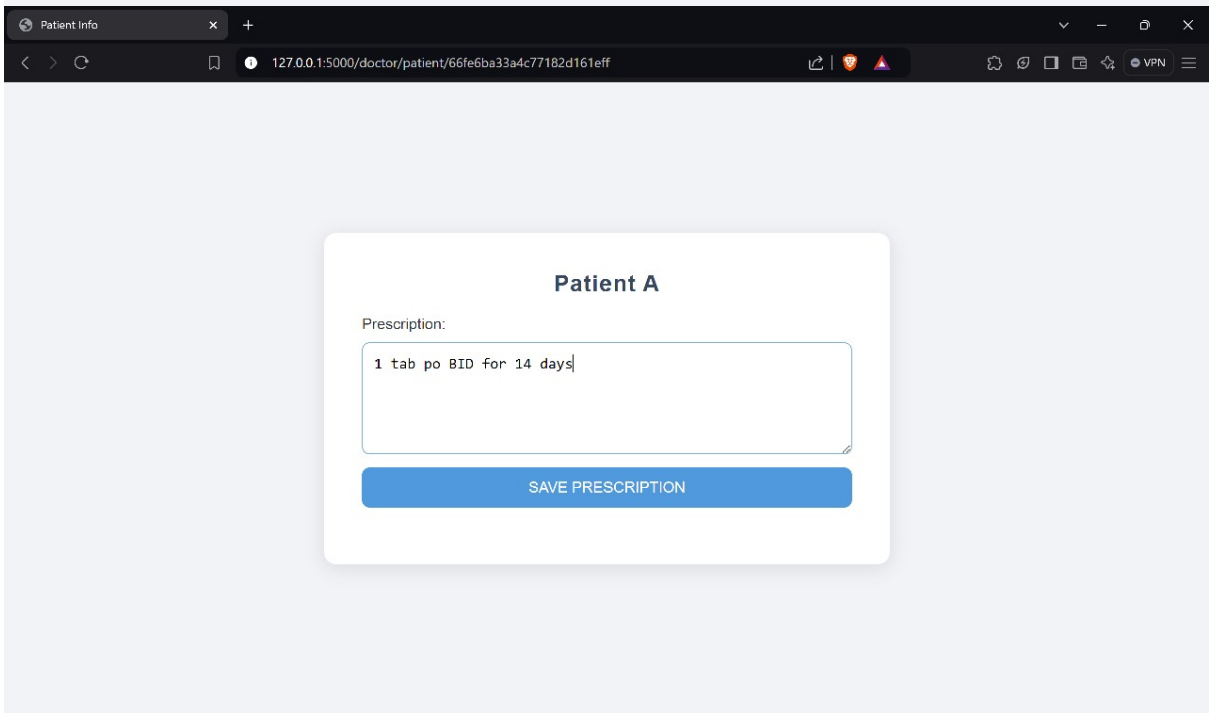
- Programming Languages: Python, Solidity (for smart contracts)
- Cryptographic Libraries: OpenSSL, PyCryptodome
- Blockchain Platforms: Ethereum, Hyperledger Fabric

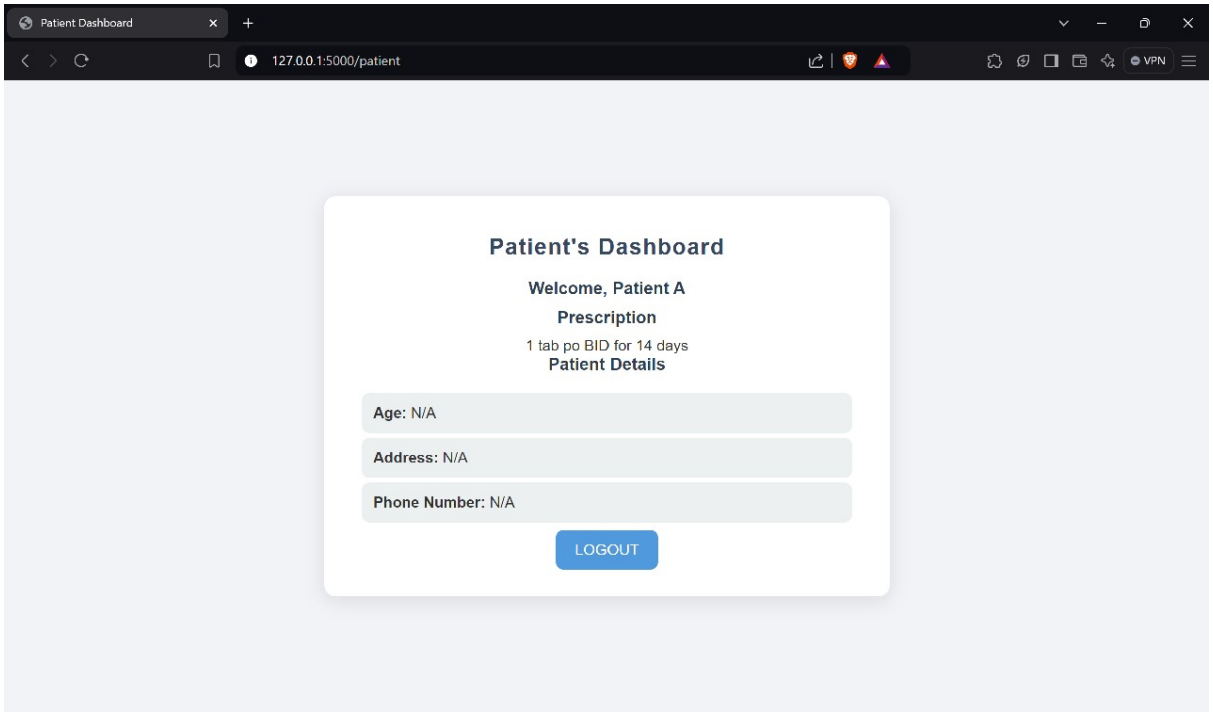
Implementation:






Doctor Dashboard
127.0.0.1:5000/doctor
VPN
Doctor's Dashboard
Patients List
Patient A
LOGOUT


Patient Info
127.0.0.1:5000/doctor/patient/66fe6ba33a4c77182d161eff
VPN
Patient A
Prescription:
1 tab po BID for 14 days
SAVE PRESCRIPTION



Scope:

For Doctors:

Login and view only the patients they are treating.
Add and manage patient prescriptions.
Ensure that prescriptions are stored in an encrypted format for security.

For Patients:

Login and view their own personal and medical information.
View the prescriptions written by their doctor, with the assurance that these prescriptions are stored securely.
Patients cannot edit any information to maintain data integrity.

Tools and Technologies:

Backend: Python Flask
Database: MongoDB
Frontend: HTML, CSS

Authorization & Security:

ABAC (Attribute-Based Access Control) for restricting access based on roles.
Fernet Encryption from the cryptography package for securing sensitive patient data like prescriptions.

Authentication: BCrypt for password hashing.

System Overview:

User Authentication:

The system allows two types of users—doctors and patients—to log in using a common login page.
Upon login, users are directed to different dashboards based on their role.

Doctor's Dashboard:

Doctors can view a list of patients assigned to them.
By clicking on a patient's name, doctors can access the patient's details and write or update prescriptions.
Prescriptions are encrypted using Fernet encryption before being stored in the MongoDB database.

Patient's Dashboard:

Patients can log in and view their personal information and the prescription provided by their doctor.
The prescription is stored securely in the database and decrypted when presented to the patient.

Security Features:

All passwords are hashed using BCrypt before storage to ensure secure authentication.
Prescriptions are encrypted using Fernet encryption, making sure that even if the database is compromised, sensitive information remains secure.
ABAC (Attribute-Based Access Control) ensures that:
A doctor can access only their patients' data.
A patient can access only their own medical information.
MongoDB is used as the database, where doctors and patients' records are stored securely.

Functional Requirements:

Doctor:

Login with credentials.
View a list of patients they are treating.
Click on a patient to view their details.
Write or update a prescription for the patient.
Prescription data is encrypted and stored securely.

Patient:

Login with credentials.
View their personal details and prescriptions.
The prescription is securely decrypted when displayed.

Non-Functional Requirements:

Data Security: The system ensures that all patient data and prescriptions are encrypted and stored securely.

Scalability: The system can be extended to accommodate more users, both doctors and patients, with minimal changes.

Usability: The interface is simple, allowing both doctors and patients to easily navigate and manage information.

Implementation Details:

Frontend:

A simple login page that redirects users based on their roles.
Two dashboards: one for doctors and one for patients, with respective functionality.

Backend:

The backend is developed using Python Flask, handling user requests, authentication, and encryption.
Data is stored in MongoDB, with separate collections for users (doctors and patients) and prescriptions.
User passwords are securely hashed using BCrypt, and sensitive data such as prescriptions are encrypted with Fernet encryption.

Database Schema:

Users Collection:

Stores doctors and patients with fields:
name, email, password (hashed), role (doctor or patient).

Patients Collection:

Stores patient data, including:

name, email, doctor_id (reference to the doctor), and prescription (encrypted).

Conclusion:

This project is an implementation of a secure, user-friendly, and scalable doctor-patient management system. By using MongoDB for data storage, Flask for the backend, and Fernet encryption for securing patient data, the system ensures both usability and security. The Attribute-Based Access Control (ABAC) mechanism further ensures that sensitive information is accessed only by authorized users.

This system can be a valuable solution for clinics or hospitals to manage patient records while ensuring that data privacy and security are maintained.

Future Enhancements:

Integrate appointment scheduling and notifications.
Provide API access for mobile applications.
Add support for more complex roles, such as administrators.

Appendix:

Abac.py

```
def check_access(attributes):
    # Define required attributes and threshold
    required_attributes = ["doctor", "cardiology"]
    threshold = 2

    # Check how many required attributes the user has
    matched_attributes = [attr for attr in attributes if attr in required_attributes]

    # Allow access if the threshold is met
    return len(matched_attributes) >= threshold
```

app.py

```
from flask import Flask, render_template, request, redirect, url_for, session, flash
from flask_pymongo import PyMongo
from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user,
current_user
from cryptography.fernet import Fernet
import bcrypt
from bson import ObjectId

app = Flask(__name__)
app.secret_key = 'your_secret_key'
app.config['MONGO_URI'] = 'mongodb://localhost:27017/medicalapp'

# MongoDB Setup
mongo = PyMongo(app)
login_manager = LoginManager()
login_manager.init_app(app)

# Generate encryption key for prescriptions
fernet_key = Fernet.generate_key()
cipher_suite = Fernet(fernet_key)

# Load user for login session
@login_manager.user_loader
def load_user(user_id):
    user_data = mongo.db.users.find_one({"_id": ObjectId(user_id)})
    if user_data:
        return User(user_data['_id'], user_data['role'])
    return None

class User(UserMixin):
    def __init__(self, user_id, role):
```

```
        self.id = str(user_id)
        self.role = role

@app.route('/')
def index():
    return render_template('login.html')

# Login Route
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        role = request.form['role']
        if role=="doctor":
            user = mongo.db.users.find_one({"email": email})

            if user and bcrypt.checkpw(password.encode('utf-8'), user['password']):
                user_obj = User(user['_id'], user['role'])
                login_user(user_obj)
                if user['role'] == 'doctor':
                    return redirect(url_for('doctor_dashboard'))
        else:
            user = mongo.db.patients.find_one({"email": email})

        if user and bcrypt.checkpw(password.encode('utf-8'), user['password']):
            user_obj = User(user['_id'], user['role'])
            print("fs",user_obj)
            if user['role'] == 'doctor':
                return redirect(url_for('doctor_dashboard'))
            else:
                session['username']=email
                return redirect(url_for('patient_dashboard'))

        else:
            flash('Invalid email or password.')
    return render_template('login.html')

@app.route('/login1', methods=['GET', 'POST'])
def login1():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        user = mongo.db.patients.find_one({"email": email})

        if user and bcrypt.checkpw(password.encode('utf-8'), user['password']):
            user_obj = User(user['_id'], user['role'])
            print("fs",user_obj)
            if user['role'] == 'doctor':
                return redirect(url_for('doctor_dashboard'))
```

```
            else:
                session['username']=email
                return redirect(url_for('patient_dashboard'))
        else:
            flash('Invalid email or password.')
    return render_template('login.html')

# Doctor's Dashboard
@app.route('/doctor', methods=['GET'])
@login_required
def doctor_dashboard():
    if current_user.role == 'doctor':
        patients = mongo.db.patients.find({"doctor_id": ObjectId(current_user.id)})
        return render_template('doctor.html', patients=patients)
    return redirect(url_for('login'))

# View Patient Info and Write Prescription
@app.route('/doctor/patient/<patient_id>', methods=['GET', 'POST'])
@login_required
def view_patient(patient_id):
    if current_user.role == 'doctor':
        patient = mongo.db.patients.find_one({"_id": ObjectId(patient_id), "doctor_id":
ObjectId(current_user.id)})
        if request.method == 'POST':
            prescription = request.form['prescription']
            encrypted_prescription = cipher_suite.encrypt(prescription.encode())
            mongo.db.patients.update_one(
                {"_id": ObjectId(patient_id)},
                {"$set": {"prescription": encrypted_prescription}}
            )
            return redirect(url_for('doctor_dashboard'))
        return render_template('view_patient.html', patient=patient)
    return redirect(url_for('login'))

# Patient's Dashboard
# Patient's Dashboard
@app.route('/patient', methods=['GET'])
@login_required
def patient_dashboard():
    # Use find_one to get a single document
    patient = mongo.db.patients.find_one({"email": session["username"]})
    if patient:  # Check if the patient exists
        if patient.get('prescription'):
            decrypted_prescription = cipher_suite.decrypt(patient['prescription']).decode()
        else:
            decrypted_prescription = None
        # Fetch patient details
        name = patient.get('name', 'N/A')
        age = patient.get('age', 'N/A')
        address = patient.get('address', 'N/A')
```

```
        phone_number = patient.get('phone_number', 'N/A')

        return render_template('patient.html', patient=patient,
prescription=decrypted_prescription, name=name,age=age, address=address,
phone_number=phone_number)

    return redirect(url_for('login'))



# Logout Route
@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('login'))

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

i.py

```
from pymongo import MongoClient
import bcrypt
from bson.objectid import ObjectId

# Connect to MongoDB
client = MongoClient('mongodb://localhost:27017/')
db = client['medicalapp']  # Database

# Hash password using bcrypt
def hash_password(password):
    return bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())

# Insert Doctors
doctors = [
    {
        "_id": ObjectId(),  # MongoDB generates a unique _id if not specified
        "name": "Dr. John Doe",
        "email": "john.doe@example.com",
        "password": hash_password("password123"),  # Hash password before inserting
        "role": "doctor"
    },
    {
        "_id": ObjectId(),
        "name": "Dr. Jane Smith",
        "email": "jane.smith@example.com",
        "password": hash_password("password456"),
        "role": "doctor"
```

```
    }
]

# Insert Patients
patients = [
    {
        "_id": ObjectId(),
        "name": "Patient A",
        "email": "patient.a@example.com",
        "password": hash_password("patientpassword1"),
        "role": "patient",
        "doctor_id": doctors[0]["_id"],  # Associate with Dr. John Doe
        "prescription": None
    },
    {
        "_id": ObjectId(),
        "name": "Patient B",
        "email": "patient.b@example.com",
        "password": hash_password("patientpassword2"),
        "role": "patient",
        "doctor_id": doctors[1]["_id"],  # Associate with Dr. Jane Smith
        "prescription": None
    }
]

# Insert doctors into 'users' collection
db.users.insert_many(doctors)
print("Doctors inserted successfully")

# Insert patients into 'patients' collection
db.patients.insert_many(patients)
print("Patients inserted successfully")
```

model.py

```
from flask_login import UserMixin

from flask_login import UserMixin

class User(UserMixin):
    def __init__(self, user_id, role):
        self.id = str(user_id)  # The user ID needs to be a string
        self.role = role

    def is_authenticated(self):
        return True  # You can adjust logic based on your needs

    def is_active(self):
        return True  # Can also be customized to check if user is active
```

```
    def is_anonymous(self):
        return False  # The user is not anonymous

    def get_id(self):
        return self.id  # Flask-Login needs this method

class Patient:
    def __init__(self, patient_id, doctor_id, name, prescription):
        self.id = patient_id
        self.doctor_id = doctor_id
        self.name = name
        self.prescription = prescription
```

threshold_cryptography.py

```
from cryptography.fernet import Fernet

# Generate a key for encryption
key = Fernet.generate_key()
cipher_suite = Fernet(key)

def encrypt_data(data):
    encrypted_data = cipher_suite.encrypt(data.encode())
    return encrypted_data

def decrypt_data(encrypted_data):
    decrypted_data = cipher_suite.decrypt(encrypted_data).decode()
    return decrypted_data
```

style.css

```
/* General Reset and Body Styling */
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: 'Poppins', Arial, sans-serif;
    background-color: #f0f3f7;
    color: #333;
    display: flex;
    align-items: center;
    justify-content: center;
    height: 100vh;
```

```
    margin: 0;
    text-align: center;
}

/* Container Styling */
.container {
    max-width: 600px;
    width: 100%;
    padding: 40px;
    background-color: #ffffff;
    box-shadow: 0 4px 20px rgba(0, 0, 0, 0.1);
    border-radius: 12px;
    animation: fadeIn 0.5s ease;
    position: relative;
}

/* Title Styling */
h1.title, h2.title {
    font-size: 24px;
    color: #34495e;
    margin-bottom: 20px;
    font-weight: 600;
    letter-spacing: 1px;
}

/* Form Styling */
.form {
    margin: 20px 0;
    text-align: left;
}

/* Input Fields */
.input, .dropdown, .textarea {
    width: 100%;
    padding: 12px;
    margin: 10px 0;
    border-radius: 8px;
    border: 1px solid #ccc;
    font-size: 16px;
    transition: border-color 0.3s;
}

.input:focus, .dropdown:focus, .textarea:focus {
    border-color: #3498db;
    outline: none;
}

/* Button Styling */
.btn, .btn-logout {
    background-color: #3498db;
```

```
    color: white;
    padding: 12px 20px;
    border: none;
    border-radius: 8px;
    font-size: 16px;
    cursor: pointer;
    text-transform: uppercase;
    transition: background-color 0.3s;
    width: 100%;
}

.btn:hover, .btn-logout:hover {
    background-color: #2980b9;
}

/* Patient Details and Messages */
ul.patient-details, ul.messages, ul.prescriptions {
    list-style: none;
    padding: 0;
    margin: 20px 0;
    text-align: left;
}

ul.patient-details li, ul.messages li, ul.prescriptions li {
    padding: 12px;
    background-color: #ecf0f1;
    margin: 5px 0;
    border-radius: 8px;
}

/* Hyperlinks Styling */
a {
    color: #3498db;
    text-decoration: none;
    transition: color 0.3s;
}

a:hover {
    color: #2980b9;
}

/* Titles and Highlight Text */
h3 {
    color: #2c3e50;
    font-size: 18px;
    margin-bottom: 10px;
}

/* Patient List */
.patient-list {
```

```
    text-align: left;
}

.patient-list li {
    background-color: #ecf0f1;
    padding: 10px;
    margin: 5px 0;
    border-radius: 8px;
}

.patient-list li:hover {
    background-color: #bdc3c7;
}

/* Fade In Animation */
@keyframes fadeIn {
    from {
        opacity: 0;
        transform: translateY(-20px);
    }
    to {
        opacity: 1;
        transform: translateY(0);
    }
}

/* Mobile Responsiveness */
@media (max-width: 768px) {
    .container {
        max-width: 100%;
        padding: 20px;
    }

    .btn, .btn-logout {
        padding: 10px;
        font-size: 14px;
    }

    h1.title, h2.title {
        font-size: 20px;
    }
}
```

Access.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Access Page</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <div class="container">
        <h1 class="title">Access Granted</h1>
        <p class="decrypted-data">Decrypted Data: {{ data }}</p>
        <a href="{{ url_for('logout') }}" class="btn-logout">Logout</a>
    </div>
</body>
</html>
```

Doctor.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Doctor Dashboard</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <div class="container">
        <h2 class="title">Doctor's Dashboard</h2>
        <h3>Patients List</h3>
        <ul class="patient-list">
            {% for patient in patients %}
                <li><a href="{{ url_for('view_patient', patient_id=patient['_id']) }}">{{
patient['name'] }}</a></li>
            {% endfor %}
        </ul>
        <br>
        <a href="{{ url_for('logout') }}" class="btn-logout">Logout</a>
    </div>
</body>
</html>
```

Get_appointment.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Book Appointment</title>
```

```
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <div class="container">
        <h2 class="title">Book an Appointment</h2>
        <form action="/get_appointment" method="POST" class="form">
            <label for="doctor">Choose a Doctor:</label>
            <select name="doctor_id" class="dropdown" required>
                {% for doctor in doctors %}
                    <option value="{{ doctor['_id'] }}">{{ doctor['name'] }}</option>
                {% endfor %}
            </select><br>
            <button type="submit" class="btn">Book Appointment</button>
        </form>
        {% with messages = get_flashed_messages() %}
            {% if messages %}
                <ul class="messages">
                    {% for message in messages %}
                        <li>{{ message }}</li>
                    {% endfor %}
                </ul>
            {% endif %}
        {% endwith %}
    </div>
</body>
</html>
```

Index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Healthcare Access</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <div class="container">
        <h1 class="title">Welcome to the Healthcare Data System</h1>
        <a href="{{ url_for('login') }}" class="btn">Login</a>
    </div>
</body>
</html>
```

Login.html:

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Login</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
  <div class="container">
    <h2 class="title">Login</h2>
    <form action="/login" method="POST" class="form">
      <label for="role">Role</label>
      <select name="role" class="dropdown" required>
        <option value="doctor">Doctor</option>
        <option value="patient">Patient</option>
      </select><br>

      <label for="email">Email</label>
      <input type="email" name="email" class="input" required><br>

      <label for="password">Password</label>
      <input type="password" name="password" class="input" required><br>

      <button type="submit" class="btn">Login</button>
    </form>
    {% with messages = get_flashed_messages() %}
      {% if messages %}
        <ul class="messages">
          {% for message in messages %}
            <li>{{ message }}</li>
          {% endfor %}
        </ul>
      {% endif %}
    {% endwith %}
  </div>
</body>
</html>
```

Patient.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Patient Dashboard</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
```

```
    <div class="container">
        <h2 class="title">Patient's Dashboard</h2>
        <h3>Welcome, {{ name }}</h3>

        <h3>Prescription</h3>
        <p>{{ prescription }}</p>

        <h3>Patient Details</h3>
        <ul class="patient-details">
            <li><strong>Age:</strong> {{ age }}</li>
            <li><strong>Address:</strong> {{ address }}</li>
            <li><strong>Phone Number:</strong> {{ phone_number }}</li>
        </ul>
        <a href="{{ url_for('logout') }}" class="btn-logout">Logout</a>
    </div>
</body>
</html>
```

Register_patient.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Register Patient</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <div class="container">
        <h2 class="title">Register as a Patient</h2>
        <form action="/register" method="POST" class="form">
            <label for="name">Name</label>
            <input type="text" name="name" class="input" required><br>

            <label for="email">Email</label>
            <input type="email" name="email" class="input" required><br>

            <label for="password">Password</label>
            <input type="password" name="password" class="input" required><br>

            <label for="age">Age</label>
            <input type="number" name="age" class="input" required><br>

            <label for="address">Address</label>
            <input type="text" name="address" class="input" required><br>

            <label for="phone_number">Phone Number</label>
            <input type="text" name="phone_number" class="input" required><br>
```

```
        <button type="submit" class="btn">Register</button>
      </form>
      {% with messages = get_flashed_messages() %}
        {% if messages %}
          <ul class="messages">
            {% for message in messages %}
              <li>{{ message }}</li>
            {% endfor %}
          </ul>
        {% endif %}
      {% endwith %}
    </div>
</body>
</html>
```

View_patient.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Patient Info</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
</head>
<body>
    <div class="container">
        <h1 class="title">{{ patient.name }}</h1>
        <form method="POST" class="form">
            <label for="prescription">Prescription:</label><br>
            <textarea name="prescription" rows="5" cols="40" class="textarea"></textarea><br>
            <button type="submit" class="btn">Save Prescription</button>
        </form>
    </div>
</body>
</html>
```