

CSCI 531
Applied Cryptography
Semester Project

Designing a Secure Decentralized Audit System for healthcare

Submitted by
Shobana Chandrasekaran
3724112894

Problem statement

The aim of the project is to design and analyze a secure decentralized audit system for the healthcare department. The implementation of the system should meet the following goals:

- *Privacy*: Patient records cannot be viewed by any unauthorized entity.
- *Identification and authorization*: Access to all data should follow the step of authentication. Only authorized and identified entities can access the patient's records.
- *Queries*: An authorized, identified auditor can query the audit logs
- *Immutability*: All modifications to the medical records should be detectable
- *Decentralization*: A trusted third-party can vouch for the integrity of the data in the medical records.

1. Introduction

Audits are a useful tool for an organization. They provide credibility for the data that thrive in the organization and give other organizations the confidence that the records are fair and true. It helps build trust between the organizations and between the organization and the public. Audits in the healthcare department aim to bring forth the discrepancies between the actual medical practice and the ones written in books. This enhances the quality of healthcare the patients receive. To make auditing easier and more trustworthy, the process of auditing has become decentralized.

This project aims at implementing such a decentralized auditing system for the healthcare department to meet the aforementioned goals. Section 2 talks about the proposed system architecture and Section 3 discusses the cryptographic components that are incorporated in the proposed architecture. The implementation details are given in Section 4. Section 5 provides a conclusion to the thesis, noting its assumptions and limitations and provides scope for future work.

2. Proposed system architecture

The proposed system consists of three participants: doctor, patient and auditor. The roles of each of the participants are:

- A patient can book an appointment, print the medical records, give access to a doctor to view the medical records, give access to an audit to view the medical records and delete an appointment.
- A doctor, who has been authorized by the patient, can view the medical records of the patient, copy the records, make changes to the records and delete the records
- An auditor, who has been authorized by the patient, can query the audit logs of the record he/she/they selects and also view the medical records if the audit logs don't tally.

The main component of the system is the audit logs. Every action performed on a medical record by the participants of the system generates an audit log. It consists of:

- Recorded message: It tells about the action that was carried out. Depending on the action performed, the message in the audit log varies also the status of the record varies.
- Date and time: A timestamp of when the action was performed
- Patient account address: Address of the patient whose record is viewed
- Doctor account address: Address of the doctor who looked after the patient
- Auditor account address: Address of the auditor who views the audit logs

The proposed system contains a front-end for easy user interaction with the system and uses a Flask development web server. The whole system is built over a Blockchain. It uses smart contracts to execute the actions performed on the medical records and also stores the transactions on the blockchain. The smart contract is also written to emit the actions into an event log. This log will be rendered in the front-end for the auditor's approval.

2.1 System Goals Satisfaction

Privacy:

Patient is the sole participant who can grant access to his/her medical records for a doctor or an auditor. This way no unauthorized entity will be able to view, modify or delete the patient's records. Any attempt to access the records without authorization, the system redirects to a page showing exception.

The patient also has the right to revoke access to a doctor or an auditor. On top of this, all the activities are monitored and put in the blockchain.

Identification and authorization:

All the participants of the system are required to signup, providing their credentials, in order to be able to use the system. Each participant is given a blockchain address and a patient is given a contract address for login purposes. The information is stored as a database in an encrypted form. The passwords are stored using salted-hash-encryption. More on this later. All registered users pass through a login page. This guarantees identification and authentication.

The system does not deal with authorization per se in any form like Access Control Lists(ACLs), policies etc. It is the patient who grants access to the records for other participants. In this system, the patient acts as an authority who authorizes.

Queries:

A patient can book an appointment specifying the feasible date and time. Booking an appointment creates a medical record with unique ID. A patient can give access to a doctor and auditor and also revoke access. The patient can also delete a medical record. Deleting a medical record by the patient is considered equivalent to cancelling a doctor's appointment.

A doctor, authorized by the patient, can view, modify, copy or delete the records. When a doctor's access is revoked, he/she will no more be able to do any of those actions on the record.

An auditor, given access by the patient, can view all the audit logs. On the worst case, the auditor can view the medical records if he/she finds any discrepancies with the logs. A record has a feature called record status. It is a numerical value that maps to the event performed on the medical record.

All actions performed by any of the participant is recorded as a Blockchain transaction. So anyone will be able to see the changes and verify the correctness of the data.

Immutability:

Blockchain is an immutable ledger. All the actions are recorded in the Blockchain as individual transactions. Each transaction block is internally chained using cryptographic hashes. A record cannot be changed without altering the cryptographic hash as well as all hashes in subsequent blocks. So anyone in the network can check the integrity of the data and no changes will go undetected.

Decentralization:

The proposed system takes advantage of two main technologies: Blockchain and Smart Contracts.

There are two major categories of implementing a Blockchain technology: permissioned and permissionless. Because a patient's privacy is at concern here, the system is implemented as a permissioned blockchain technology. Participants of the blockchain network are given permission by agreed upon administrators. For this system, the administrator is the designer of the system.

The system uses smart contracts which contain code on what should be done under specified circumstances. For example, the smart contract will tell what action should be taken when a doctor tries to view a patient's record. The contract will first check if the doctor has been authorized by the patient; then it checks if the record exists only then further steps are carried out. Smart contracts ease the process of monitoring and enforcement of actions by automating the whole process.

The system is completely decentralized and any trusted third party will be able to detect the changes in the system.

2.2 Cryptographic components of the proposed system

Ethereum Blockchain

Blockchain is the backbone of the system. It is used to preserve and exchange patient data in a secure manner. It satisfies the requirement of decentralization. A transaction is requested to write a new block of data to the blockchain. Every participant in the blockchain checks hashes to validate the transaction., thus making it a secure and immutable mode of storing data.

Smart contracts

The smart contract in the system is Patient.sol written using Solidity. It takes care of the following actions:

- Contains the declaration of a medical record
- Creates a new medical record
- Deletes a medical record
- Prints a record
- Lists all the records of a patient
- Updates a record
- Changes the status number of the record depending on the action performed on the record like queried, copied, printed, deleted etc..
- Checks permission for the doctor and auditor
- Logs every event and writes it as a transaction on the blockchain.
- Lets the doctor or auditor to perform the queries
- Grants / revokes access for the doctor/ auditor

After every patient registers into the system, individual contracts are created for each of the patients. Patients are assigned a unique contract address along with the blockchain address. To distinguish the participants, doctors and auditors dont have a contract address for these participants. They are asked to input the value 0 during Login.

SSL/TLS certificates

The system uses Flask-based web server in the development environment. The development webserver runs on HTTPS using SSL certificate. Because its is a development web server, the certificate is self-signed. The SSL certificate acts as identification for the server, containing the server name and domain details.This helps in establishing a trusted and secure connection between the server and the client.

The code snippet that implements this component is :

```
if __name__ == "__main__":
    app.run(ssl_context='adhoc')
```

This adds encryption to the Flask application by creating an SSL certificate on-the-fly.

Hashing passwords

Passwords should never be stored or sent in clear. If an attacker finds a database of plaintext passwords, they can easily be matched with the usernames (mostly emails) to log in to a system. A common method used today is to use salted hashing when a password is provided to the system. The salt is stored along with the salted-hashed password in the database.

Hashing algorithms like SHA-256 are very easy to try and match with a particular hash using brute-force approach with the computation power we have now. Another tactic to matching hashes is using rainbow tables, which implements the process of randomly generating passwords in a grouped fashion.

Due to the above reasons, the system is implemented using HMAC with a combination of salting. HMAC is a slower hashing method. In technical terms, it takes in a password, random salt and number of iterations to produce a certain key length. The larger the number of iterations the longer it is to get the result.

All the salted-hashed passwords are stored in the database (.csv) file along with the salt that was used in the process. The verification step goes like this: Get the salt and the hashed password from the database. Carry out salted hashing using the salt from the database for the password that the user has supplied in the login page. Grant access if the hashed values are same. Else, the user cannot get into the system.

AES256(CTR) Encryption and Decryption

Not only passwords, all data about the users of the system are stored in encrypted form in the database. The system uses AES256 in CTR mode for encryption and decryption purposes. During SignUp, the data is encrypted and stored in the file. During Login and while querying the audit logs, the data is first decrypted for further processing.

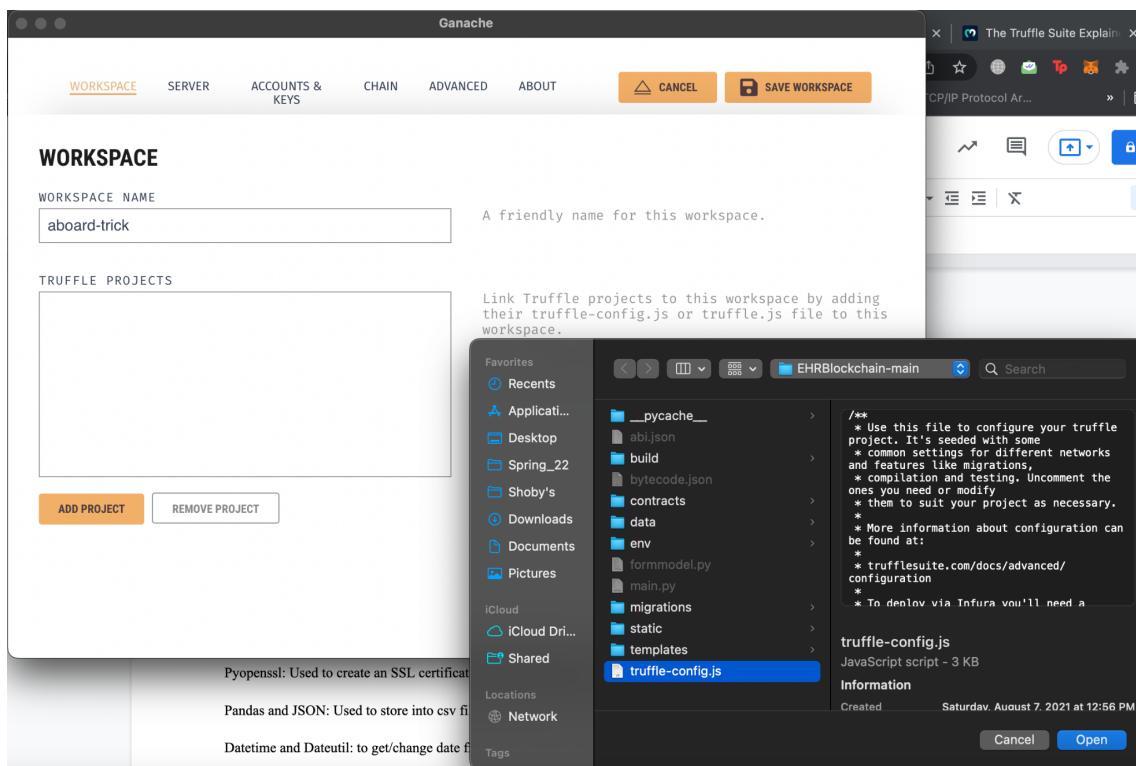
AES in CTR mode is chosen considering the factor of scalability (in terms of hardware) and parallelization. The parallel nature of CTR mode where the blocks are encrypted independently of each other can help speed things up in a multicore processor setting. Also, unlike CBC, CTR mode can perform certain calculations offline to prepare the key stream thereby reducing the latency of the system.

3. Prototype Implementation

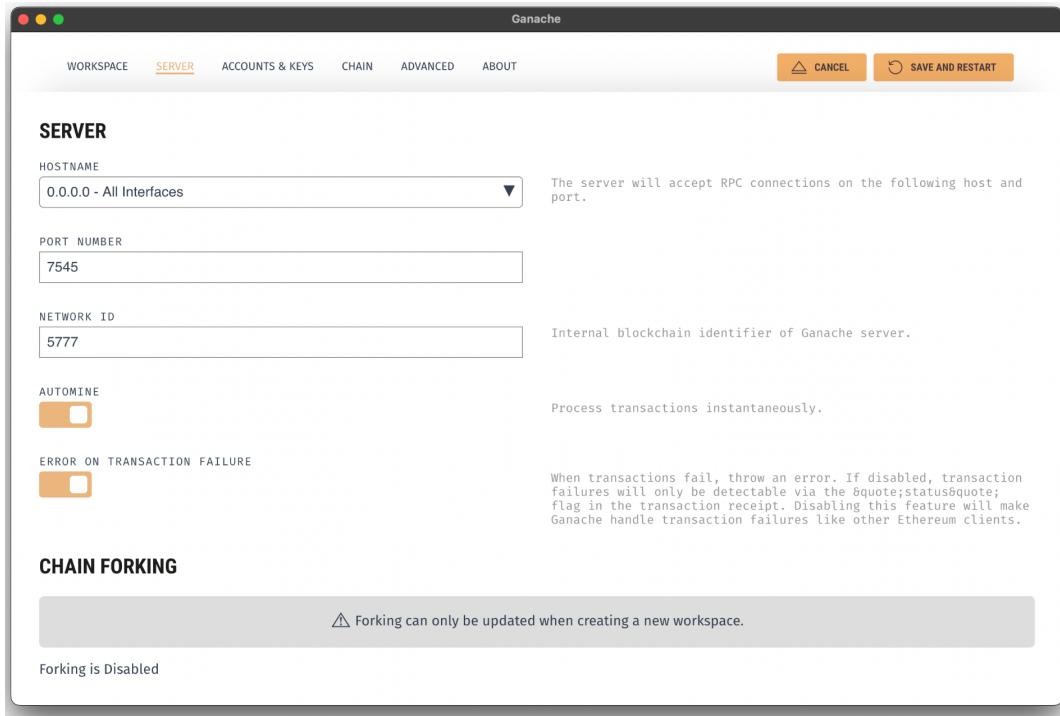
To implement the Ethereum Blockchain, Truffle and Ganache are used.

- Truffle: Truffle is the development environment, testing framework, and the dApp pipeline for an EVM blockchain. (<https://github.com/trufflesuite/truffle>)
- Ganache GUI: Ganache is a personal Ethereum Blockchain used to deploy smart contracts, develop applications, run tests and perform other tasks free of cost. (<https://trufflesuite.com/ganache/>)

In the Ganache GUI, create a new workspace and add the truffle-config.js file from the project folder as shown below



Make sure the server settings are as shown below:



To run the system, create a python virtual environment.

Using the commands: (Unix/MacOS)

```
python3 -m venv env  
source env/bin/activate
```

After activating the environment, download the following packages using ‘pip install’ :

- ❖ Web3.py (<https://web3py.readthedocs.io/en/stable/>): It is a Python library for interacting with Ethereum. It's commonly used in decentralized apps (dapps) for sending transactions to the Blockchain, interacting with smart contracts, and many more.
- ❖ Flask (<https://flask.palletsprojects.com/en/2.1.x/quickstart/>): Flask is a small and lightweight Python web framework for creating web applications in Python. It is easy to create a web application with Flask because you only need a single Python file.
- ❖ Eth_utils(<https://github.com/ethereum/eth-utils>): Used for access the event logs generated by the smart contract from the Blockchain.

Hashlib: Used to hash passwords and file names in /data/

Cryptography: Used for fernet encryption/decryption and generation of key.

- ❖ JSON(<https://docs.python.org/3/library/json.html>): Contract ABI is represented in JSON format. The Contract Application Binary Interface (ABI) is the standard way to interact with contracts in the Ethereum ecosystem, both from outside the blockchain and for contract-to-contract interaction.
- ❖ Pyopenssl(<https://pypi.org/project/pyOpenSSL/>): Used to create SSL certificate for the webApp.
- ❖ Pandas(<https://pandas.pydata.org/>): Used to store and retrieve the encrypted data in .csv file.
- ❖ Datetime(<https://docs.python.org/3/library/datetime.html>) and Dateutil(<https://dateutil.readthedocs.io/en/stable/>): to convert human readable time to unix timestamp and vice-versa
- ❖ Clipboard(<https://pypi.org/project/clipboard/>): Used in the copy query. It copies a unique patient record into clipboard.

Once all the packages are imported, you are all set to run the application. Do the following:

```
truffle compile && migrate
```

in your terminal to migrate all the contracts into the blockchain. You output will look like this:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1_initial_migration.js
=====
Replacing 'Migrations'
> transaction hash: 0xeec3e0hd246403abe919fa98de3e161ea20a78d66ab9caa5ccf61fc158597ef7
> Blocks: 0 Seconds: 0
> contract address: 0x2CF3aaCA3AaE76F5c512919096b6568c768fdfCe
> block number: 1
> block timestamp: 1651266346
> account: 0x69240F001c53dFC48E1EEe461632365e8ca95E6c
> balance: 99,99903914
> gas used: 198053 (0x305cd)
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.00396186 ETH

> Saving migration to chain.
> Saving artifacts
> Total cost: 0.00396186 ETH

2_deploy_contract.js
=====
Replacing 'Patient'
> transaction hash: 0x277cb04931f631b6efffc938029bc8f553fa409221115ef29ce37e99117fc299
> Blocks: 0 Seconds: 0
> contract address: 0x31d0F90590D7b2C5d1e84E4d2f7fe71f6f1c736
> block number: 3
> block timestamp: 1651266348
> account: 0x69240F001c53dFC48E1EEe461632365e8ca95E6c
> balance: 99,91364786
> gas used: 4077171 (0x3e3673)
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.08154342 ETH

> Saving migration to chain.
> Saving artifacts
> Total cost: 0.08154342 ETH

Summary
=====
> Total deployments: 2
> Final cost: 0.08550528 ETH

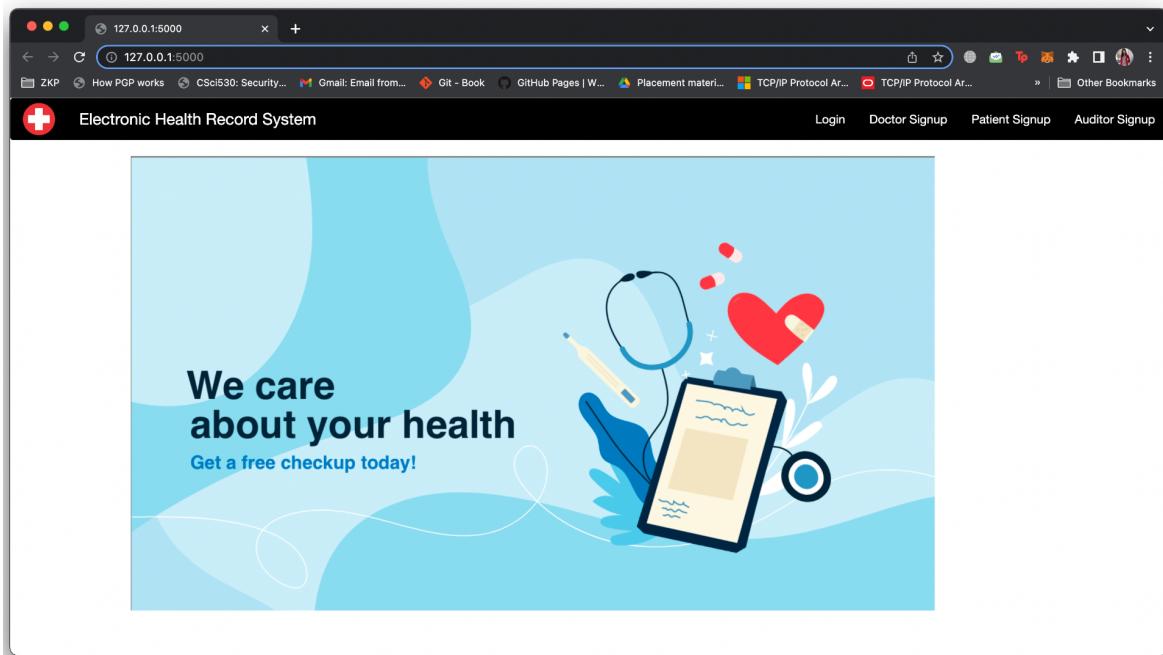
```

Then: `FLASK_APP=main.py FLASK_ENV=development flask run`

This will start the web application at <https://127.0.0.1:5000>. As you can see the application uses HTTPS.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
^C%
(env) shoby@Shobanas-MacBook-Air EHRBlockchain-main % FLASK_APP=main.py FLASK_ENV=development flask run
* Serving Flask app 'main.py' (lazy loading)
* Environment: development
* Debug mode: on
Web3 is connected = True
* Running on http://127.0.0.1:5000 (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 377-143-775
Web3 is connected = True
```

The homepage of the website is as follows:



3.1 File description

❖ Python files:

- *main.py*: This is like the main() function in C++. When you run this program, a series of activities gets triggered. It deploys the smart contracts, reads the symmetric key for encryption and decryption purposes. It links all the HTML files by telling what code to be executed when a particular action is performed (like patient signup). It also defines all the actions that can be performed by each participant and talks to the smartcontract at the backend to perform the query. The file is deployed using a Flask development web server.
- *Formmodel.py*: This file contains Flask forms i.e. it defines the components of a form in a specific html page. For example, in the patient signup form, the flask contians the components : first name, last name, address, zip code, email, password etc.

- ❖ Json files:
 - abi.json: As mentioned above, ABI is the way to interact with the smart contract and the ABI comes in JSON format. It contains the functions, events, variable definitions that are in the smart contract.
 - bytecode.json: The contracts written using Solidity, after running “truffle compile” gets converted into Ethereum Virtual Machine (EVM) bytecode. It is this bytecode that gets deployed on the blockchain and not the .sol files.
- ❖ Smart Contracts: (Found in the contracts directory)
 - Migrations.sol: This file is not that useful during the development phase but it comes into play when the app is for production. It keeps track of the changes in the network which is helpful when there is a situation to modify only the current deployment.
 - Patient.sol: This is the main contract in the system. It creates and assigns a unique contract address for each patient in the system. It also defines the structure of the medical record and the events that occur on the medical record. There are functions which takes care of rendering the record details to the front-end and also the function of granting and revoking access to the participants of the system.

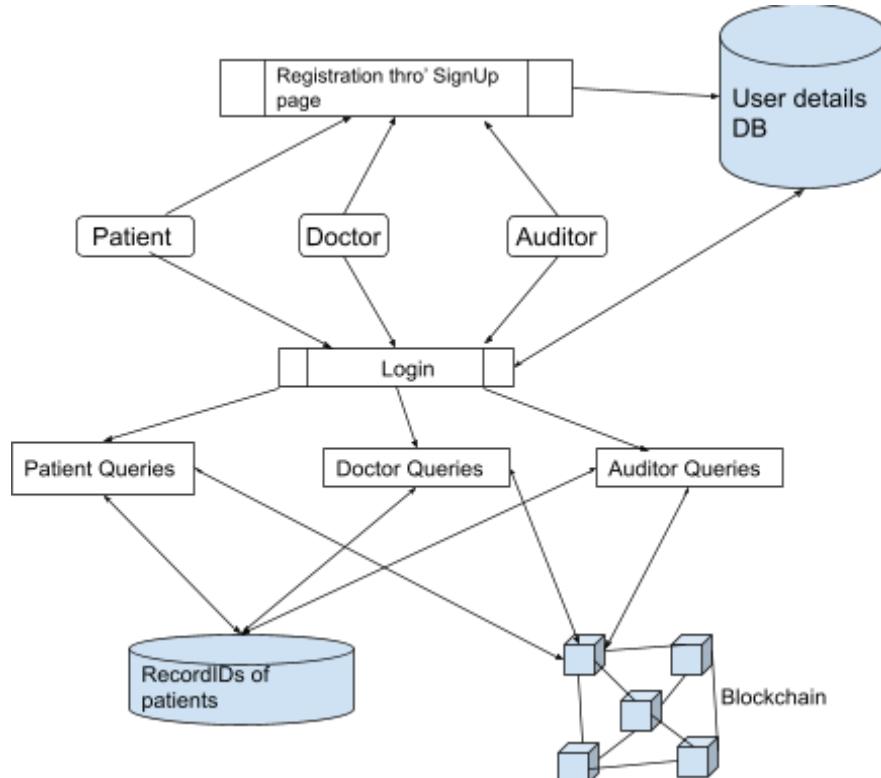
Once you compile the contracts, they get converted into a JSON file and gets stored in the build directory.

- build/contracts/Migrations.json: JSON file of Migrations.sol
- build/contracts/Patient.json: JSON file of Patient.sol
- ❖ Database: (Database directory)
 - 3f91fb273e0c... .csv: Participants’ (doctor, auditor, patient) signup
 - data is stored in the encrypted format. The reason that the filename is also hashed is to not reveal any information about the data stored in the file.
 - enc_key.key: This file contains the symmetric key for all the encryption and decryption purposes. It should be kept safe.
 - f415ea3131a... .csv: Medical record ID and patient’s contract address is stored here after encrypting with the symmetric key.
- ❖ JavaScript:(Present in the migrations folder)
 - 1_initial_migration.js: Part of the truffle implementation. It keeps track of the record on the Ethereum network.
 - 2_deploy_contract.js: Creates a sample Patient contract for testing purposes and deploys to the Ethereum Blockchain.
- ❖ CSS: (present in the static directory)
 - Contains all the images and .css files for the DApp website.

- ❖ HTML: (Can be found in the templates directory)
 - index.html: This is the landing page of the website. Has navigation to Login, patient signup, audit signup and doctor signup.
 - auditsignup.html: Auditors enter their details to signup. After signup, the user is redirected to result.html page
 - doctorssignup.html: To set up the account, doctors provide the required data in the Flaskform displayed on the webpage. After signup, the user is redirected to result.html page
 - patientssignup.html: Patients enter their details to get registered into the system. After signup, the user is redirected to result.html page
 - result.html: After signup, users get a blockchain address which they have to keep note of. On top of it, patients also get a contract address.
 - login.html: Requires blockchain address and password to login. Additionally patients are required to provide their contract address.
 - auditor.html: Audit actions are shown in the webpage.
 - auditquery.html: Displays the audit logs of the particular record of the particular patient.
 - patient.html: Queries of the patients are taken care in this page.
 - Doctor.html: Doctors can perform their actions on the record.

- ❖ Truffle Configuration file:
 - truffle-config.js: Configuration file to connect Ganache GUI to truffle. Contains details of interface address and port address

3.2 System Workflow



The Registration step:

Patient registration

127.0.0.1:5000/patientsignup

Electronic Health Record System

Patient Account SignUp

Account count: 1

First Name: Shobana

Last Name: Chandrasekaran

Email Address: shobanac@usc.edu

Phone Number: 4084441738

City: Los Angeles

Zipcode: 90007

Insurance #: 34546

New Password:

Repeat Password:

Submit



127.0.0.1:5000/patientsignup

Electronic Health Record System

Result

Transaction Hash:
0x3e9d307f13eec697752690dc4b514146c06294ea7527f4222d363664898587d7

Please note your Blockchain address and contract address for future Login!
WE DO NOT STORE primary keys.

You will get a mail with the key pair, contract address and blockchain address!

User Name: Shobana Chandrasekaran

Blockchain Address: 0x9b5d3220A6DB7cEf43069C5906f232a54F84cCAc

Primary key: Will be sent to your email

Contract Address: 0xF435113Ea60D5019cCd2763F6E667C334F46bDaF

Ganache

ACCOUNTS BLOCKS TRANSACTIONS CONTRACTS EVENTS LOGS SEARCH FOR BLOCK NUMBERS OR TX HASHES

CURRENT BLOCK 5 GAS PRICE 20000000000 GAS LIMIT 6721975 HARDFORK MUIRGLACIER NETWORK ID 5777 RPC SERVER HTTP://127.0.0.1:7545 MINING STATUS AUTOMINING WORKSPACE GODLY-PROFIT SWITCH

TX HASH **0x3e9d307f13eec697752690dc4b514146c06294ea7527f4222d363664898587d7** CONTRACT CREATION

FROM ADDRESS 0x9B5d3220A6DB7cEf43069C5906f232a54F84cCAC CREATED CONTRACT ADDRESS 0xF435113Ea60D5019cCd2763F6E667C334F46bDaF GAS USED 4143690 VALUE 0

TX HASH **0x88d684f6070163c1409213039925884a9c7e5037454e6cd73ea60967efb89135** CONTRACT CALL

FROM ADDRESS 0xE7E10ebDA1301614281a94D6Ec0F9d5B2a8E85d1 TO CONTRACT ADDRESS Migrations GAS USED 27343 VALUE 0

TX HASH **0x6da23e639cc25f5b64d49feefdc264ec6965e3ada8e186214aa2db8918827a2e** CONTRACT CREATION

FROM ADDRESS 0xE7E10ebDA1301614281a94D6Ec0F9d5B2a8E85d1 CREATED CONTRACT ADDRESS 0x2c44E59072b8f823Dbc2Ab2aA9f22F2D739bcf5B GAS USED 4077171 VALUE 0

TX HASH **0x81e136d44d6dd43bf687861ae1969433dd36df324466a88957335f52fd2d2604** CONTRACT CALL

FROM ADDRESS 0xE7E10ebDA1301614281a94D6Ec0F9d5B2a8E85d1 TO CONTRACT ADDRESS Migrations GAS USED 42343 VALUE 0

TX HASH **0x51c69b731e6f660054fa50c73835deadf16ad07a61cc8a260d796b8a6d5dba5a** CONTRACT CREATION

Doctor registration

Doctor Account SignUp

Account count

First Name

Last Name

Email Address

Unique Employee ID

New Password

Repeat Password



Result

Please note your Blockchain address and contract address for future Login!
WE DO NOT STORE primary keys.

You will get a mail with the key pair, contract address and blockchain address!

User Name: Joe M

Blockchain Address:
0x9B5d3220A6DB7cEf43069C5906f232a54F84cCAC

Primary key: Will be sent to your email

Auditor Registration

Screenshot of the Auditor Registration page:

The page title is "Audit Account SignUp". It features a sidebar with account count (2), first name (Xavier), last name (Z), email address (Zavier@gmail.com), unique employee ID (2334), and password fields for new and repeat password. A "Submit" button is at the bottom.

On the right side, there is a large decorative graphic featuring a pie chart and a checkmark.

Screenshot of the Result page:

The page title is "Result". It displays a message: "Please note your Blockchain address and contract address for future Login! WE DO NOT STORE primary keys." Below this, it says "You will get a mail with the key pair, contract address and blockchain address!"

It shows the User Name: Xavier Z, Blockchain Address: 0x64278129B714B40A33E10EC8C993b7EFcDC186EC, and Primary key: Will be sent to your email.

The Login step:

Patient: Once a patient logins to their account, their interface looks as follows:

The screenshot shows a web browser window with two tabs open. The active tab is titled "127.0.0.1:5000/login". The page itself has a header "Electronic Health Record System" with a red cross icon. Below the header is a "Login" form. The form contains three input fields: "Account Address" (containing "0x9B5d3220A6DB7cEf43069C5906f232a54F84cCac"), "Contract Address (If Audit or Doctor, enter 0)" (containing "0xF435113Ea60D5019cCd2763F6E667C334F46bDaF"), and "Password" (with a masked value). A "Start" button is located below the password field. At the bottom of the form are three links: "Patient SignUp", "Auditor SignUp", and "Doctor SignUp". To the right of the form is a decorative illustration of a doctor and a patient in a medical office setting.

The screenshot shows a web browser window with multiple tabs open. The active tab is titled "127.0.0.1:5000/patient?address=0x9B5d3220A6DB7cEf43069C5906f232a54F84cCac&contract=0xF435113Ea60D5019cCd2763F6E667C334F46bDaF". The page title is "Electronic Health Record System". The main content area is titled "Patient actions". It includes several sections: "Get an Appointment (Creates a new Medical Record)" with a date input field and "Start Visit" button; "Add doctor audits" with a "Doctor Id" input field and "Add Doctor" button; "Remove doctor audits" with a "Doctor Id" input field and "Remove Doctor" button; "Add other audits" with an "Audit Id" input field and "Add Audit" button; "Remove other audits" with an "Audit Id" input field and "Remove Audit" button; "Print Medical Records" with a "Record Id" input field and "Print Medical Record" button; and "Delete Record" with a "Record Id" input field and "Delete Medical Record" button.

Auditor: The auditor's actions are as follows once he logs in to the system.

127.0.0.1:5000/login

Electronic Health Record System

Login

Account Address
0x64278129B714B40A33E10EC8C993b7Ef0DC186EC

Contract Address (If Audit, enter 1 Else enter 0)
1

Password
.....

Start

Patient SignUp
Auditor SignUp
Doctor SignUp



127.0.0.1:5000/auditor?address=0x9B5d3220A6DB7cEf43069C5906f232a54F84cCAC&contract=1

Audit actions

Account Address: 0x9B5d3220A6DB7cEf43069C5906f232a54F84cCAC

Contract Address to Audit:

Patient Contract Address
Patient Contract Address to View Medical Records

Submit

Print Medical Records
Record ID

Print

Query Medical Records
Record ID

Query List Unique ID's

Doctor: The doctor's actions that are possible once he logins to the system and also provided the access by the patient.

Screenshot of the Electronic Health Record System Login page (127.0.0.1:5000/login). The page features a logo of a red cross inside a white circle. The title bar says "Electronic Health Record System". The main form has fields for "Account Address" (containing 0x9B5d3220A6DB7cEf43069C590f232a54F84cCAC), "Contract Address (If Audit or Doctor, enter 0)" (containing 0), "Password" (containing), and a "Start" button. Below the form are links for "Patient SignUp", "Auditor SignUp", and "Doctor SignUp". To the right of the form is a decorative illustration of a doctor and a patient with medical icons like a heart rate monitor and a brain.

Screenshot of the Electronic Health Record System Doctor actions page (127.0.0.1:5000/doctor?address=0x9B5d3220A6DB7cEf43069C590f232a54F84cCAC&contract=0). The page title is "Doctor actions". It contains several sections: "Patient Contract Address" (with a "Submit" button), "Print Medical Records" (with a "Print" button), "Copy Medical Records" (with a "Copy" button), "Update Medical Records" (with a "Record ID" field and an empty "New Record" area), and "Delete Medical Records" (with a "Record ID" field and a "Delete" button).

Patient Queries

1. Booking appointments (Creates a record)

The screenshot shows a web browser with multiple tabs open, all displaying the URL `127.0.0.1:5000`. The active tab is titled "Patient actions". The page contains fields for "Account Address" and "Contract Address", both showing long hex strings. Below these are sections for "Get an Appointment (Creates a new Medical Record)" and "Add doctor audits". The "Get an Appointment" section includes a date input field set to "07/19/2022 09:00" and a "Start Visit" button. The "Add doctor audits" section has a "Doctor Id" input field and a "Add Doctor" button. To the right, a panel displays "Action: Patient initiated visit." and "Details:" followed by "Appointment Date: 07/19/2022 09:00", "Tx Hash: 0x48784c4affd0fd4e156f3e5175d62968fb6312...", and "Unique Record ID: 0x9951cF278c4E9E2752c513f6bEAbdd11fD3C8". It also features a QR code and a "Record Message: New Medical Record is created". At the bottom, it shows "Record Status: 0 (0-Created, 1-Deleted, 2-Changed, 3-Queried, 4-Printed, 5-Copied)".

2. Grant access to a doctor

This screenshot shows the same web interface after a doctor has been added. The "Add doctor audits" section now contains the doctor's ID and the "Add Doctor" button. The "Details" panel on the right now shows "Action: Patient added a doctor to access their medical records." and "Event Log: A doctor is added." It also lists the "Tx Hash", "Address", and "Epoch time".

3. Grant access to an audit

Get an Appointment (Creates a new Medical Record)

MM/DD/YYYY hh:mm

Start Visit

Add doctor audits

Doctor Id

Add Doctor

Remove doctor audits

Doctor Id

Remove Doctor

Add other audits

0x64278129B714B40A33E10EC8C993b7EFcDC186EC

Add Audit

Action: Patient added an audit to view/change their medical records.

Details:

Tx Hash: 0xb1344938860fe5e22eb7675679aa8b606b4dd...

Event Log: An audit is added.

Address:

Epoch time: 1651366992

4. Revoke access of a doctor

Get an Appointment (Creates a new Medical Record)

MM/DD/YYYY hh:mm

Start Visit

Add doctor audits

Doctor Id

Add Doctor

Remove doctor audits

0x9B5d3220A6DB7cEf43069C5906f232a54F84cCAC

Remove Doctor

Add other audits

Audit Id

Add Audit

Remove other audits

Audit Id

Remove Audit

Action: Patient revoked access to a doctor their medical records.

Details:

Tx Hash: 0x22f3fd1f3ad661d25141efbd9efbd290d745...

Event Log: A doctor is removed.

Address: 0x9B5d3220A6DB7cEf43069C5906f232a54F84cCAC

Epoch time: 1651369042

5. Revoke access of an audit

The screenshot shows a web-based application interface for managing medical audits. It includes sections for adding and removing doctor and other audits, printing medical records, and deleting records. On the right side, there is a detailed log entry for an audit removal.

Action: Patient revoked access for an audit to view/change their medical records.

Details:

- Tx Hash: 0x03cbbe223e65d9ad23d68c40f07849d5a4454...
- Event Log: An audit is removed.
- Address:
- Epoch time: 1651374419

6. Print record details

The screenshot shows a web-based application interface for managing medical records. It includes sections for removing doctor audits, adding other audits, removing other audits, printing medical records, and deleting records. On the right side, there is a detailed log entry for a record printout.

Action: Patient printed their medical records.

Details:

- Tx Hash: 0xfc0c88d6d585b25fe55537a51b2a121c9a91e4...
- Record Message: Record is printed by patient.
- Record Status: 4
(0-Created, 1-Deleted, 2-Changed, 3-Queried, 4-Printed, 5-Copied)

Medical Record Details:

- Visit initiate

7. Delete an appointment (deletes the record)

The screenshot shows a web interface for managing medical records. In the top right corner, there is a message: "Action: Patient deleted their medical record." Below this, under "Details:", the Tx Hash is listed as "0x8837566878f2b4549b310a357dad0b35473d3f...". The Record Message is "Record is deleted by patient." and the Record Status is "1 (0-Created, 1-Deleted, 2-Changed, 3-Queried, 4-Printed, 5-Copied)".

Key sections visible include:

- Remove doctor audits:** A form with a "Doctor Id" input field and a "Remove Doctor" button.
- Add other audits:** A form with an "Audit Id" input field and an "Add Audit" button.
- Remove other audits:** A form with an "Audit Id" input field and a "Remove Audit" button.
- Print Medical Records:** A form with a "Record Id" input field and a "Print Medical Record" button.
- Delete Record:** A form with a "Record ID" input field containing the value "0xb4bAC2b5F4a996c8B195EE9aA82b1FE46426b28" and a "Delete Medical Record" button.

Doctor Actions:

1. Submit the contract address of the patient to view the records

The screenshot shows the "Doctor actions" section of the Electronic Health Record System. It includes fields for "Account Address" and "Contract Address", both of which are populated with the value "0x9B5d3220A6DB7cEf43069C5906f232a54F84cCAC". There is a "Submit" button below these fields.

Below this, there are three more sections:

- Patient Contract Address:** A field containing the value "0xF435113Ea60D5019cCd2763F6E887C334F#6bDaF".
- Print Medical Records:** A form with a "Record ID" input field and a "Print" button.
- Copy Medical Records:** A form with a "Record ID" input field and a "Copy" button.

2. Display the record details

The screenshot shows a web browser window titled "Electronic Health Record System". The main content area is labeled "Doctor actions". It contains several sections:

- Patient Contract Address:** A text input field containing "Patient Contract Address to View Medical Records". Below it is a "Submit" button.
- Print Medical Records:** A text input field containing "0x89651cF12788c4E9E2752c513f6bEABdd11fD3C8". Below it is a "Print" button.
- Copy Medical Records:** A text input field containing "Record ID". Below it is a "Copy" button.
- Update Medical Records:** A text input field containing "0x89651cF12788c4E9E2752c513f6bEABdd11fD3C8". Below it is a "Copy" button.
- Action:** "Action: Doctor printed patient medical records."
Details: "Tx Hash: 0xe3045127aac113183cc04ff5511138a698671e..."
Record Message: Record is printed by doctor/audit.
Record Status: 4 (0-Created, 1-Deleted, 2-Changed, 3-Queried, 4-Printed, 5-Copied)
- Medical Record Details:** "Visit initiate"

At the top right of the page is a "Log Out" link.

3. Copy the record

The screenshot shows a web browser window titled "Electronic Health Record System". The main content area is labeled "Doctor actions". It contains several sections:

- Patient Contract Address:** A text input field containing "Patient Contract Address to View Medical Records". Below it is a "Submit" button.
- Print Medical Records:** A text input field containing "Record ID". Below it is a "Print" button.
- Copy Medical Records:** A text input field containing "0x89651cF12788c4E9E2752c513f6bEABdd11fD3C8". Below it is a "Copy" button.
- Action:** "Action: Doctor copied patient medical records."
Details: "Tx Hash: 0x98a09abd63783c141d729e6b549e7b32e69d7a..."
Record Message: Record is copied by doctor/audit.
Record Status: 5 (0-Created, 1-Deleted, 2-Changed, 3-Queried, 4-Printed, 5-Copied)
- Medical Record Details:** "Visit initiate"

At the bottom left of the page is a "New Record" link.

4. Update the record

Print Medical Records

Record ID:

Print

Copy Medical Records

Record ID:

Copy

Action: Doctor updated patient medical records.

Details:

Update Medical Records

0x89651cF12788c4E9E2752c513f6bEAbdd11fD3C8

New Record

Patient discharged

Medical Record Details:

Patient discharged

Update

Delete Medical Records

Auditor Queries

1. Submit the contract address of the patient to view the logs

Electronic Health Record System

Log Out

Audit actions

Account Address: 0x7Ba9D036CB734BB09D55BD526830A4beaBfA0534

Contract Address to Audit:

Patient Contract Address

0xd020E817097120a024051b1fd81F023125cE4098

Submit

Print Medical Records

Record ID:

Print

Query Medical Records

Record ID:

Query List Unique ID's

2. List all the medical records of that patient

The screenshot shows the 'Audit actions' section of the EHR system. It includes fields for 'Account Address' and 'Contract Address to Audit'. On the right, it displays 'Unique Medical Record IDs' and their corresponding hex values.

Unique Medical Record ID	Value
0xd020E817...	0x8ec78483D6526Fe78FB9D6024BAAb5c71A111035

3. Print the records

The screenshot shows the 'Audit actions' section after a print operation. It includes fields for 'Account Address' and 'Contract Address to Audit'. On the right, it displays audit details such as Tx Hash, Record Message, Record Status, and Medical Record Details.

Action	Value
Action	Audit printed patient medical records.
Details:	
Tx Hash:	0x01fdad5eb402f866831f4e2924ed348264100f...
Record Message:	Record is printed by doctor/audit.
Record Status:	4 (0-Created, 1-Deleted, 2-Changed, 3-Queried, 4-Printed, 5-Copied)
Medical Record Details:	Visit initiate

4. Query the audit log of that record

Audit data of the queried record					View Audit Logs for other records
	record_msg	record_status	record_time	audit_address	doctor_address
0	New Medical Record is created.	0	2022-07-19 20:30:00	-1	-1
1	An audit is added.	-1	2022-05-01 02:09:37	0x7Ba9D036CB734BB909D558BD526830A4beaBfA0534	-1
2	A doctor is added.	-1	2022-05-01 02:10:13	0x7Ba9D036CB734BB909D558BD526830A4beaBfA0534	
3	Record is printed by doctor/audit.	4	2022-07-19 20:30:00	-1	-1
4	Record is printed by doctor/audit.	4	2022-07-19 20:30:00	-1	-1
5	Record is queried by doctor/audit.	3	2022-07-19 20:30:00	-1	-1
6	A doctor is removed.	-1	2022-05-01 02:31:48	0x7Ba9D036CB734BB909D558BD526830A4beaBfA0534	
7	An audit is added.	-1	2022-05-01 02:31:52	0x7Ba9D036CB734BB909D558BD526830A4beaBfA0534	-1
8	An audit is added.	-1	2022-05-01 02:35:08	0x7Ba9D036CB734BB909D558BD526830A4beaBfA0534	-1
9	An audit is added.	-1	2022-05-01 02:37:47	0x7Ba9D036CB734BB909D558BD526830A4beaBfA0534	-1
10	A doctor is added.	-1	2022-05-01 02:38:27	0x7Ba9D036CB734BB909D558BD526830A4beaBfA0534	
11	Record is printed by doctor/audit.	4	2022-07-19 20:30:00	-1	-1

3.3 Assumptions

The following assumptions were made during the design and development of the system:

- Patient as the authorizer of the system who grants and revokes access to the doctor or auditor for access the medical records.
- An auditor or doctor is first required to submit the contract address of the patient to view the audit logs or record details respectively.
- Record status number :
 - 1 → Record doesn't exist
 - 0 → Record created
 - 1 → Record deleted
 - 2 → Record modified
 - 3 → Record logs queried
 - 4 → Record printed
 - 5 → Record copied
- The audit logs displayed is for a particular record of a particular patient. If the auditor wants to view a different record, he has to go back and submit the ID of that record. Or if the auditor wants to view a different patient's record, he has to submit the contract address of the patient he wants to view the records of.
- To distinguish the participants, patients are given a contract address to login and the auditors and doctors are required to enter 0 in place of contract address to login.
- It is assumed that the user will be sent the contract address and the details about the registration (key details) to their mail.

4. Conclusion

The proposed system satisfies all the requirements mentioned in the problem statement. However, the system suffers certain limitations. It doesnot address the criteria of authorization in its whole. It is assumed that the patient takes the role of authorizer. Also, the system stores all the encrypted data in a file. A sophisticated database system could be used for this purpose. All the address required during Login action are not user friendly. As these addresses are a bunch of numbers, it will be difficult for the users to remember their username. This can be further improved by converting the address into some human readable format. The UI of the system can be improved. When the record is not present in the DB, the UI takes the user to an exception page. This can be converted to display proper error messages.

In conclusion, the system can be further improved by taking care of the above mentioned limitations.

References

The project template was inspired from <https://github.com/shamil-t/ehr-blockchain> and <https://github.com/cumbul/EHRBlockchain>. I have used the same tech stack and tools as this project. The format and some functionalities of the patient contract and 2_deploy_contract.js was taken from

<https://github.com/NFhbar/Ethereum-Medical-Records>

Some other references:

https://github.com/samia17/Medical_Records_Miniproject

<https://github.com/yDeepak1889/Electronic-Medical-Record-management-using-Blockchain>

I have implemented the front-end of the project, the auditor queries, cryptographic components (Symmetric encryption, hashing of passwords) and the roles of a Doctor.

Web references

1. <https://blog.miguelgrinberg.com/post/running-your-flask-application-over-https>
2. <https://www.techtarget.com/searchitoperations/tip/Blockchain-An-immutable-ledger-to-replace-the-database#:~:text=Every%20block%20links%20to%20the,create%20secure%20and%20immutable%20records.>
3. <https://us.aicpa.org/content/dam/aicpa/interestareas/frc/assuranceadvisoryservices/downloadabledocuments/blockchain-technology-and-its-potential-impact-on-the-audit-and-assurance-profession.pdf>
4. <https://www.sciencedirect.com/science/article/pii/S266660302100021X>
5. <https://www.gsd.inesc-id.pt/~mpc/pubs/ECIS2020.pdf>
6. <https://nitratine.net/blog/post/how-to-hash-passwords-in-python/>
7. <https://academy.moralis.io/blog/the-truffle-suite-explained-what-is-truffle>