A Project Report
On

HealthCare Data Management

BY

GROUP 13

Yashank Garg (2019A7PS0347H)

Manan Agrawal (2019A3PS0400H)

Udit Varshney (2019AAPS0295H)

Under the supervision of

Prof. Geetha Kumari

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF

BITS F463: CRYPTOGRAPHY PROJECT

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI

HYDERABAD CAMPUS

(April 2022)

Problem Statement

With the healthcare business becoming increasingly data-driven, the focus is increasingly on how to automate, simplify, and maximise data use. Large amounts of healthcare data may be managed, integrated, and harnessed to improve operational performance.

However, with the influx of so much healthcare data, healthcare providers are under even more pressure to handle it efficiently while maintaining integrity, interoperability, and security while adhering to laws and regulations. The following are some of the most typical obstacles and issues that health data professionals face:

1) **Fragmented Data**: Data can come from a variety of places and in a variety of formats. Data is collected by healthcare providers, payers, and patient network communities, but combining it poses certain issues.

2) **Lack of Integration between Clinical and Administration Systems**: Even inside, there might be a disconnect between patient care and administration. For administrative and analytical purposes, the data management system must be setup to guarantee that treatment codes match and that the care provided is appropriately tracked.

3) **Data Security**: Data may be compromised with each transfer from one repository to another. That is why data security has become such a major worry in recent years. Organizations must be aware of new and growing international privacy and security standards as the number of cyber assaults, data breaches, and invasions of private personal data grows.

With each transfer from one repository to another, data may be compromised. That is why, in recent years, data security has become such a huge concern. As the frequency of cyber attacks, data breaches, and invasions of private personal data grows, organisations must be aware of new and evolving international privacy and security regulations.

How does the Blockchain Work?

A blockchain is a decentralized, peer-to-peer database that stores an ever-increasing number of transactions. Each transaction, referred to as a "block," is encrypted, timestamped, and authenticated using consensus techniques by every authorized member of the database (i.e., a set of rules). A transaction is not added to the database if it has not been validated by all database members. Every transaction is linked to the one before it in a sequential order, forming a transaction chain (or blocks). Because a transaction can't be deleted or modified, it creates an audit trail that can't be changed. Only by adding another transaction to the chain can a transaction be altered.

How it can help solve our problem

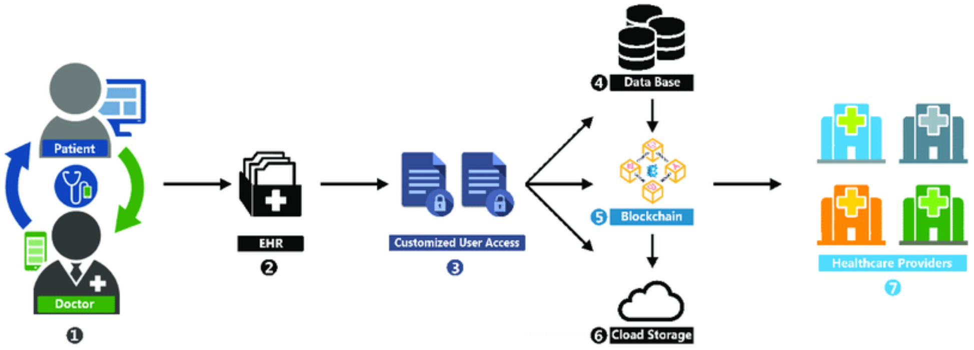
1) Blockchain, which is based on distributed ledger technology, provides a mechanism to decentralise data while maintaining its integrity and security.

2) By their very nature, blockchains are immune to data tampering. Blockchain ledgers are immutable, which means that once data is added or a transaction is made, it cannot be changed or removed, hence resolving the data integrity issue.

3) Using encryption, blockchain provides a way to safeguard data against fabrication or modification. Specifically, blockchain will assist with the following, so resolving the interoperability issue.
   - Disintermediation of databases to improve visibility of patient information
   - Removal of central authority to manage database reducing hacking attacks
   - Immutability of transactions
   - Validation without the involvement of any central verification agency

4) Blockchain platform ensures that your data is encrypted, which means that data modification is a difficult task.

In this way, we can see how every challenge mentioned earlier can be solved using blockchain. Now, we will see how we can implement blockchain to manage healthcare data.



Zero-Knowledge Proof

Zero-knowledge proofs are a type of proof that allows you to persuade someone that you know something without having to divulge any details about what you know.

In cryptography, this can be used to prove difficult things. For example, given a hash, the prover could persuade the verifier that there is a number that the verifier knows, that meets specific criteria, and hashes to this value without exposing any of the number's properties. This is what is used to create anonymous transactions, in which the address and transaction balances are hidden on the blockchain behind hashes, or in which you must own the private key in order to spend a note, thus concealing your identity.

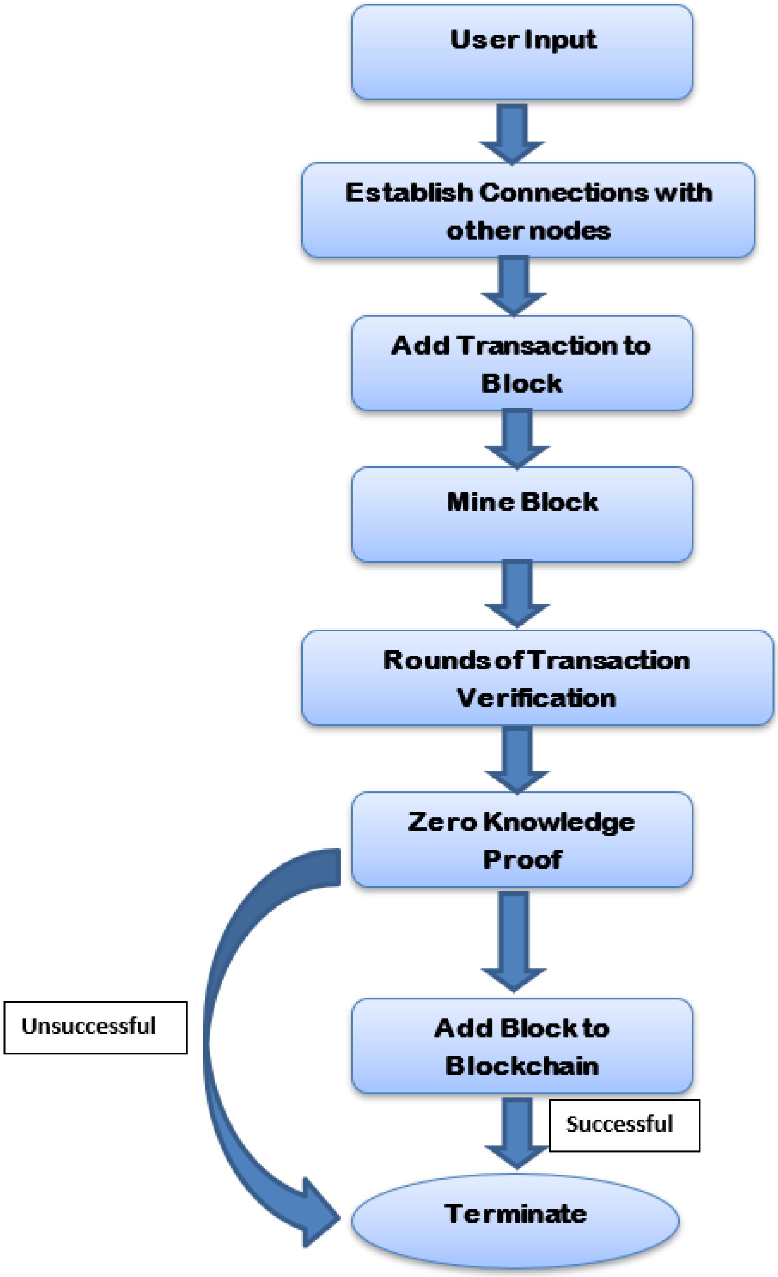
Implementation

We have also implemented something similar to the aforementioned zero-knowledge proof. Let Alice has sensitive data x for which she chooses two numbers p and g, and Bob is the verifier. We had chosen nonce value to be our sensitive data and followed the below-mentioned steps to perform zero-knowledge proof:

1) We assumed p = 11, g = 2, where g is the generator of p.
2) We calculated g^x mod(p) and assigned it to a new variable y.
3) Alice chooses a random number r in the range [0, p-1), calculates a new value $h = g^r \, mod(p)$ and sends it to Bob.
4) Bob receives h and sends back a random bit b (could be 0/1).
5) Alice sends a new calculated value $s = (r + bx) mod(p - 1)$ to Bob.
6) Bob computes l=g^s mod(p) which it send to Alice
7) Alice receives this value which should equal val=$hy^b \, mod(p)$.

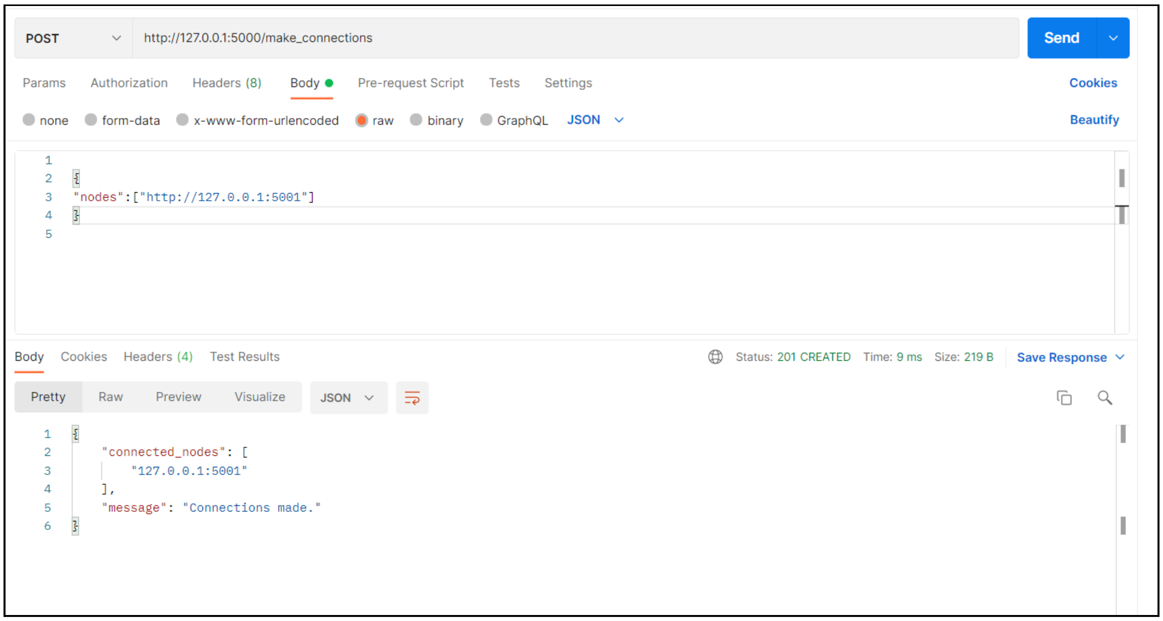
Here Bob acts as a verifier and checks if Alice knows the value of x without actually getting to know what x is.

Project Flowchart



Functions used

- **Making connections between nodes - [/make_connections , Request type= POST]**
  Here we establish connection between the nodes defined at port number 5000 and 5001.
  Below is the screenshot of the same



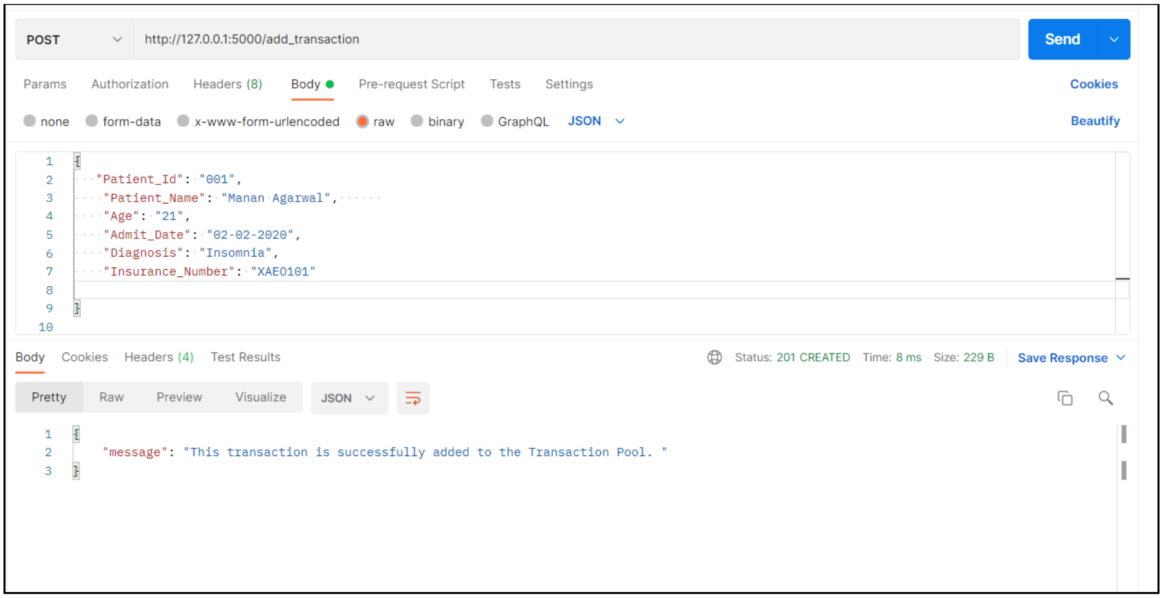
http://127.0.0.1:5000/make_connections

```
{
"nodes":["http://127.0.0.1:5001"]
}
```

Status: 201 CREATED Time: 9 ms Size: 219 B

```
{
    "connected_nodes": [
        "127.0.0.1:5001"
    ],
    "message": "Connections made."
}
```

- **Adding transaction - [/add_transaction , Request type= POST]**
  The transaction details are provided in a JSON file which includes patient name, patient, id, age, date of admission, insurance number, and diagnosis. The timestamp of the order is auto-fetched and need not be provided explicitly



http://127.0.0.1:5000/add_transaction

```
{
  "Patient_Id": "001",
   "Patient_Name": "Manan Agarwal",
   "Age": "21",
   "Admit_Date": "02-02-2020",
   "Diagnosis": "Insomnia",
   "Insurance_Number": "XAE0101"

}

```

Status: 201 CREATED Time: 8 ms Size: 229 B

```
{
    "message": "This transaction is successfully added to the Transaction Pool. "
}
```

- ## createBlock()

  This creates a new block which contains all transaction present in the transaction pool ,along with the necessary block header information

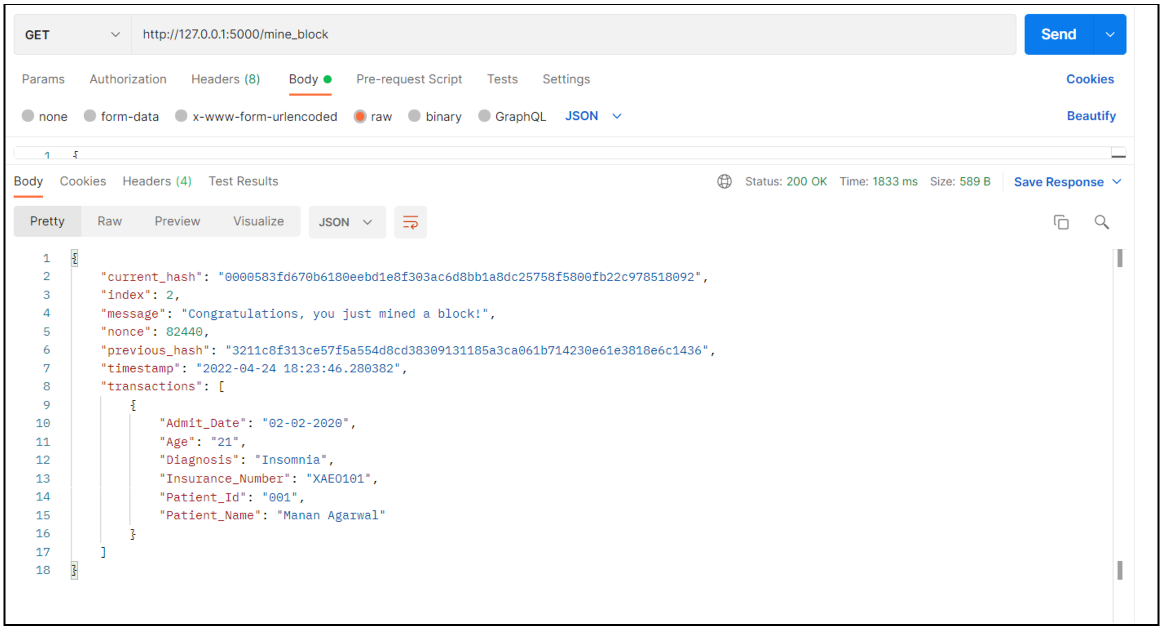
- ## mineBlock()

  Mine A Block - - [/mine_block , Request type= GET]

  To mine a block, a Node has to call mine_block().

  This will first check if the current node has the latest blockchain or not, then accordingly add all the transactions currently in the transaction pool into a new block by calling the create_block function.

  It then calls the verify_transaction function, which asks other nodes to verify the proof of work using zero-knowledge proof. If successful, it appends that block in the blockchain. The transaction pool is now empty. After mining, details of the newly created block are displayed. We can see that the hash values of the blocks begin from 0000, which acts as proof of work.



GET http://127.0.0.1:5000/mine_block

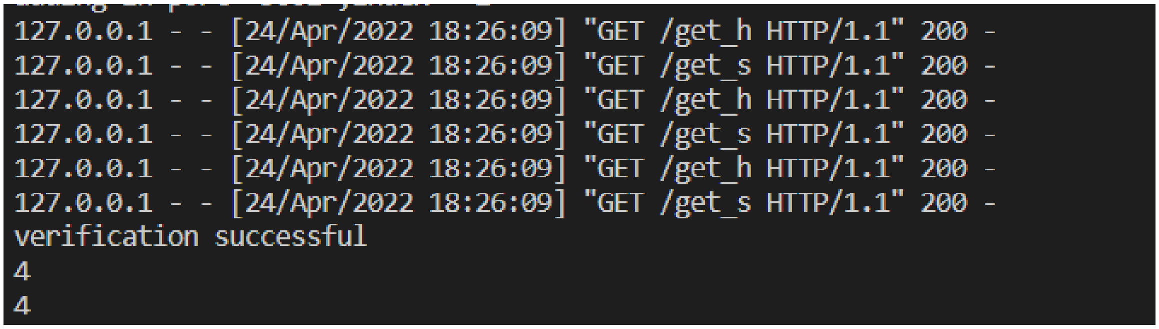
Body ● raw JSON

Body — Pretty — JSON — Status: 200 OK Time: 1833 ms Size: 589 B

```
{
    "current_hash": "0000583fd670b6180eebd1e8f303ac6d8bb1a8dc25758f5800fb22c978518092",
    "index": 2,
    "message": "Congratulations, you just mined a block!",
    "nonce": 82440,
    "previous_hash": "3211c8f313ce57f5a554d8cd38309131185a3ca061b714230e61e3818e6c1436",
    "timestamp": "2022-04-24 18:23:46.280382",
    "transactions": [
        {
            "Admit_Date": "02-02-2020",
            "Age": "21",
            "Diagnosis": "Insomnia",
            "Insurance_Number": "XAE0101",
            "Patient_Id": "001",
            "Patient_Name": "Manan Agarwal"
        }
    ]
}
```

- **verifyTransaction()**

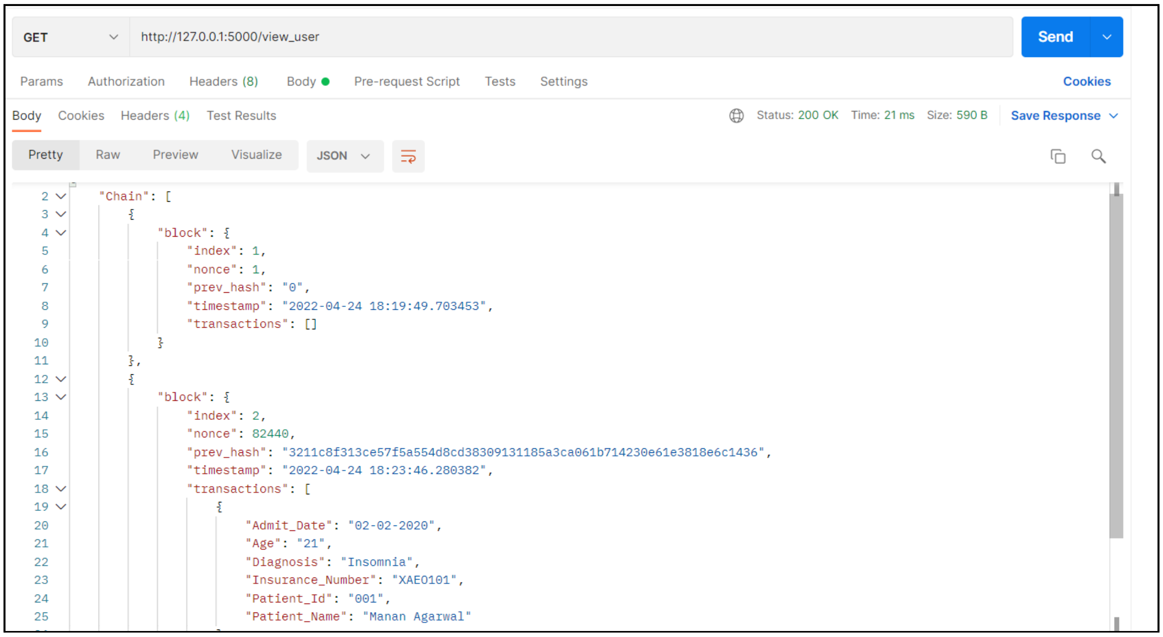
  Verifying the sensitive data using Zero-Knowledge Proof



```
127.0.0.1 - - [24/Apr/2022 18:26:09] "GET /get_h HTTP/1.1" 200 -
127.0.0.1 - - [24/Apr/2022 18:26:09] "GET /get_s HTTP/1.1" 200 -
127.0.0.1 - - [24/Apr/2022 18:26:09] "GET /get_h HTTP/1.1" 200 -
127.0.0.1 - - [24/Apr/2022 18:26:09] "GET /get_s HTTP/1.1" 200 -
127.0.0.1 - - [24/Apr/2022 18:26:09] "GET /get_h HTTP/1.1" 200 -
127.0.0.1 - - [24/Apr/2022 18:26:09] "GET /get_s HTTP/1.1" 200 -
verification successful
4
4
```

- **viewUser()**

  Display only those transaction/patient details that were added by this specific node, preventing unauthorized access to data



http://127.0.0.1:5000/view_user

Status: 200 OK   Time: 21 ms   Size: 590 B

```
    "Chain": [
        {
            "block": {
                "index": 1,
                "nonce": 1,
                "prev_hash": "0",
                "timestamp": "2022-04-24 18:19:49.703453",
                "transactions": []
            }
        },
        {
            "block": {
                "index": 2,
                "nonce": 82440,
                "prev_hash": "3211c8f313ce57f5a554d8cd38309131185a3ca061b714230e61e3818e6c1436",
                "timestamp": "2022-04-24 18:23:46.280382",
                "transactions": [
                    {
                        "Admit_Date": "02-02-2020",
                        "Age": "21",
                        "Diagnosis": "Insomnia",
                        "Insurance_Number": "XAE0101",
                        "Patient_Id": "001",
                        "Patient_Name": "Manan Agarwal"
```