Preserving patient's privacy using proxy re-encryption

1st Devendra Meena
Computer Science and Engineering
Indian Institute of Technology, Kanpur

2nd Ras Dwivedi
Computer Science and Engineering
Indian Institute of Technology Kanpur

3rd Prof. Sandeep Shukla
Computer Science and Engineering
Indian Institute of Technology Kanpur

***Abstract*—In 2008, Satoshi Nakamoto introduced blockchain in his paper "bitcoin: peer to peer electronic file transfer system". Bitcoin, as of now, is a primitive blockchain supporting only cryptocurrencies, and some smart contract using its scripting language. Etherem and other subsequent blockchain also support the smart contrast. With numerous blockchain, there, there is a problem of interoperability, and most basic of it is the atomic swap. Almost every solution either uses hash-time locks or creates a network of trusted node over an overlay network. In this paper we present a solution that is independent of the time locks, and does not need a trusted third nodes, or an overlay network. We use simple Diffie-Hellman key exchange and the threshold BLS signature scheme to achieve our objective**

***Index Terms*—Patient Privacy, Proxy Re-encryption, Hyperledger Fabric, Permissioned Blockchain**

I. Introduction

The blockchain concept was introduced in 2009 with the release of the Bitcoin white paper. Bitcoin was the first cryptocurrency achieving a market capitalization of more than 117 billion US dollars in 2020[1]. After its release, several cryptocurrencies were created taking advantage of the blockchain technology and the success of Bitcoin. As of 2020, there exist more than 6000 different cryptocurrencies [1]. These cryptocurrencies either follow the Bitcoin blockchain protocol, or design their own blockchain protocol and implementation focusing on special features, such as smart contract support, privacy, digital identities, or data.

Currently, there are 86,000 blockchain projects proposed in GitHub and growing at a rate of 20,000 projects per year but the average lifespan of these projects is just 1.22 years. As the blockchain ecosystem is very heterogeneous satisfying different applications, it is clear that one blockchain is not going to be satisfactory for all. An interoperable ecosystem enables multiple users from separate blockchain networks to interact without needing to migrate from their preferred blockchain.

Hyperledger Fabric is one such project which is among the most widely used blockchain project for setting up private blockchain networks. This paper introduces an algorithm that allows atomic cross-chain transactions between two Hyperledger Fabric networks using Diffie Hellman Swaps and Boneh–Lynn–Shacham(BLS) Signature scheme.

The remainder of the paper is organized as follows. Section 1 ….. ¡Can complete as the paper is written¿

II. Backgound and Related Work

Electronic Health Records(EHR) should be a nation's priority to achieve health targets and are necessary for efficient treatment of the patient based on his medical history. However, the patient's willingness to share his health records depends upon his attitude toward its utility in research and privacy guarantees [3]. Currently, EHR is maintained by different organizations in silos. Medrec [4] tries to share EHRs between organizations using PoW blockchain, such that blockchain only has encrypted pointer to the storage and not the actual patient's data. MedRec is not able to exploit the distributed architecture of blockchain and have a single point of failure at the end of institutions. Also, because of Proof of Work blockchains, they are computation intensive.

We use proxy re-encryption to give access control to the patient over his data. We also separate diagnosis from prescription and bills, adding an extra layer of privacy for the patient. As a result, the pharmacy could only view prescribed medicine and not the diagnosis of the disease. Similarly, Insurance could only see the final amount to be reimbursed and not the diagnosis or medication. It prevents them from storing any patient-related data against his choice.

III. Blockchain Overview

Blockchain [5] is a shared, immutable ledger in a decentralized network, where the data is arranged in the form of the blocks, arranged sequentially and linked with a cryptographic hash function. Depending upon who can participate in the distributed network, blockchain could be classified as public, private, or permissioned blockchain. Bitcoin [6] and Ethereum [7] are examples of the public blockchain, while Hyperledger is an example of permissioned blockchain. Since membership of permissioned blockchain is restricted and verified, it is immune from the Sybil attack.

Every participant node in the blockchain has to reach a consensus on the final state of blockchain, i.e., the hash of the latest block. Public blockchain uses proof of work to achieve this consensus, because they have to resist the Sybil attack, while the permissioned blockchain, could use less computationally expensive mechanisms like Kafka, Raft [8] or Byzantine fault-tolerant consensus. Currently, Hyperledger Fabric [9] uses solo, Kafka and raft mechanism, and it has plans to introduce BFT consensus in future releases. A simplified version of the blockchain is shown in Fig. 1. We

have implemented our work in Hyperledger Fabric, which is one of the many blockchains developed under Hyperledger Project. We describe the organization structure of Hyperledger Fabric below.

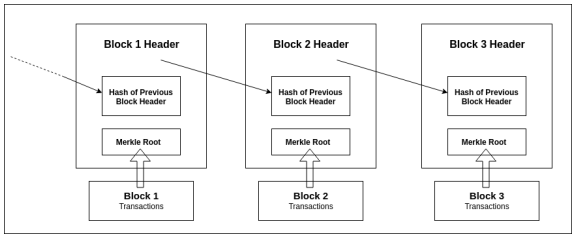

Fig. 1: Blockchain Simplified

A. Hyperledger Organization Structure

1) **MSP(Membership Service Provider)** : This node acts a certificate authority. Hyperledger Fabric architecture uses x.509 certificates through public key infrastructure, and each organization within a business network runs its own certificate authority.
2) **Client Node**: This node is used for the end-user where the middleware code resides and a rest API server to interact with the middleware code.
3) **Peer Nodes**: These are the backbone of the Hyperledger Fabric network. There are two types of peers:
   a) Committing peer: This node updates the transactions in the ledger.
   b) Endorsing peer: This node actually runs the chaincode to validate the transactions.
4) **Ordering Node** : These nodes runs the transactions, reorders them if required and decides in which block the transactions will be included. These nodes do not have chaincodes installed on them and also does not hold the ledger. This node does not need to be present in each organization as opposed to the MSP nodes, but at least one of the organization has to run this.

IV. Proxy Re-Encryption Algorithm

Public key asymmetric key encryption system works very well when there is a need to transmit confidential messages over an insecure channel. However, such a scheme does not allow delegation of messages. E.g., While Alice could easily send message to Bob, but if Bob wants to forward that same message to Charles, he has to do decryption by his private key followed by encryption by Charles's public key. This leads to unwanted computational overload. One of the efficient ways through which Bob could delegate reading authority for some of his messages to Charles, without leaking information about his private key is through Proxy re-encryption scheme. In our work, we use the re-encryption algorithm proposed by Ateniese et al. [2] which works on the assumption that Decisional Bilinear Diffie-Hellman [10] is hard to decide.

A. Decisional Diffie-Hellman Assumption

For a cyclic group G of prime order q and generator g and group operator '.,' Decisional Diffie-Hellman assumption is that given three elements of the group G as $x = g^a$, $y = g^b$, $z = g^c$ where $a, b, c \in \mathcal{Z}_q$ it is hard to decide whether $x.y = z$? Decisional Diffie-Hellman is based on the hardness of the discrete logarithm problem. For Bilinear group (which we define later) [2] Decisional Diffie-Hellman assumption is that given $x = g^a,\ y = g^b,\ z = g^c,\ Q \in G$, where $a, b, c \in \mathcal{Z}_q$ it is hard to decide whether $Q = e(g, g)^{abc}$. Along with this we also assume hardness of Discrete logarithm problem

B. Bilinear Groups

Bilinear Map: We say a map $e : G_1 \times \hat{G}_1 \rightarrow G_2$ is bilinear if

- $G_1, \hat{G}_1$ are groups of same prime order q
- for all $a, b \in Z_q,\ g \in G_1$ and $h \in \hat{G}_1$ then $e(g^a, h^b) = e(g, h)^{ab}$
- map is non-degenerate i.e if g generates G_1 and h generates $\hat{G}_1$ then $e(g, h)$ generates G_2
- there exists a computable isomorphism form $\hat{G}_1$ to G_1

C. Algorithm

While choosing a Re-encryption algorithm, few necessities are to be kept in mind.

- Not every message needs to be forwarded to another, and hence when a patient desire that only his doctor should be able to read a message, and should not be able to do that, without either doing decryption-encryption cycle or leaking his private key, he should be able to do that
- Patient should be able to choose, which messages to be forwarded, and which are not
- Even though a patient may allow the doctor to forward his message (patient's data) to another doctor, that should not give doctor right to allow another doctor to be able to forward the same message (patient's data)
- Delegation should be computationally inexpensive, and minimum changed must be done in the ciphertext

Based on our needs, we use the re-encryption algorithm suggested by Ateniese et. al. [2]. We provide a brief description of the algorithm below. Here **Type 1** encryption is such that the ciphertext could be decrypted only by the intended recipient and could not be forwarded and delegated. **Type 2** encryption could be decrypted by the recipient and could also be delegated and forwarded. During delegation, a new ciphertext is generated by a small modification of the original ciphertext.

- **Key generation:**
  - **Alice:** $pk_a = (Z^{a_1}, g^{a_2})$ and Secret key is $sk_a = (a_1, a_2)$. $Z^a = e(g, g)^a$
  - **Charles:** $pk_b = (Z^{b_1}, g^{b_2})$ and Secret key is $sk_b = (b_1, b_2)$. $Z^b = e(g, g)^b$
- Re-encryption key is: $rk_{A \rightarrow B} = g^{a_1 b_2}$
- **Type 1 Encryption**:

- **Encryption:** $c_{a,1} = (Z^{a_1 k},\ m \oplus Z^k)$ where k is any random element of the group.
- **Decryption:** Receiver knows a_1 as his first private key and can calculate a_1^{-1}, and hence can calculate Z^k from $Z^{a_1 k} = Z^{a_1 k / a_1} = e(g,g)^{a_1 k / a_1}$ message is reconstructed as $m \oplus Z^k \oplus Z^k$

- **Type 2 encryption :**
  - **Encryption:** $c_{a,r} = (g^k, m \oplus Z^{a_1 k})$
  - **Re-Encryption:** Compute $e(g^k, g^{b_2 a_1}) = Z^{b_2 a_1 k}$. Now the re-encrypted cipher text $c_{b,r} = (Z^{b_2 a_1 k}, m \oplus Z^{a_1 k})$
  - **Decryption:**
    By 1st receiver: receiver can get $Z^{a_1 k}$ as $e(g^k, g^{a_1})$ and hence can get plain text as $m \oplus Z^{a_1 k} \oplus Z^{a_1 k}$
    By 2nd receiver (Charles): since Charles knows b_2 so Charles can get Z^{1/b_2} from which he can get $Z^{b_2 a_1 k} / Z^{b_2} = Z^{a_1 k}$ and hence can get m as $m \oplus Z^{a_1 k} \oplus Z^{a_1 k}$.

V. Design

The architecture is designed to achieve the following objectives:

1) Appointment requests, lab checkup reports, and dosage prescription must be committed to the ledger
2) Patient should be able to fetch his medical records
3) Sensitive data like dosage details, lab reports in prescription and amounts in bills must be encrypted with the patient's public key. So, the only patient should have access to his medical records
4) Patient should be able to allow anyone to see his medical records
5) A mechanism to capture all the bills and payment flow for lock/unlock of prescription dosage and lab reports
6) To involve insurance company in the architecture for automatic payments of bills
7) Patient's approval on bills for verification by the insurance company
8) Patient's should be able to register for an insurance plan with the insurance company.
9) Pharmacies should be able to use the prescription from the ledger to give medicines to user
10) Strict access policies ensuring assets are fetched by the rightful owner

A. Participants and Organizations

S.No.	Participants Involved	Organization
1.	Patients	PatientORG
2.	Doctors	HospitalORG
3.	Nurses	HospitalORG
4.	Labs	HospitalORG
5.	Insurance Companies	InsuranceORG
6.	Pharmacies	PharmacyORG

TABLE I: Participants and their organizations

B. Main Assets

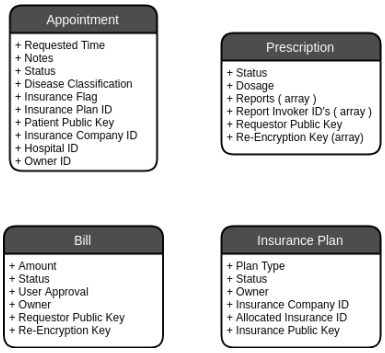

Fig. 2: Assets

C. Architecture

There can be two possibilities either the patients have registered for health insurance, and the insurance company pays a bill for them, or the patient himself pays the bill. An important assumption in the architecture is that all the money transfer are being done outside the Hyperledger fabric network. It's only the bill and other metadata details that are being updated on the ledger. This assumption was taken because all the bills amount that is generated against the patient have their amount encrypted with patient's public key. So, keeping a wallet in the network was not a good idea.

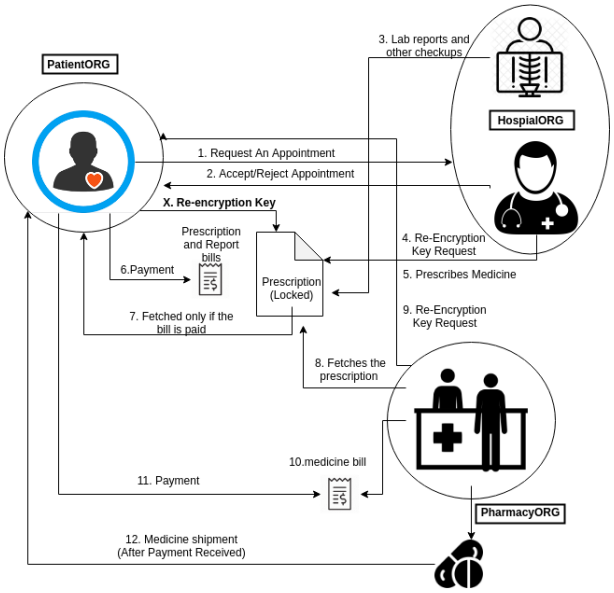

Fig. 3: CASE I: Without Health Insurance

1) Case I: Patient with no health Insurance:

1) Patient requests an appointment with a doctor for a particular time using a randomly generated appointment id, which acts as a token for patient till he gets the medicine.
2) Response by the doctor to accept or reject the appointment depending on his schedule

When the patient reaches the hospital, the doctor generates an empty prescription asset corresponding to the appointment token and adds it to the ledger. Now, the doctor can ask the patient to get some checkups done before getting a prescription.

3) Patient gives his token to Labs, and encrypted lab reports with patient public key are added to the same prescription asset
   Whenever a lab report is added to the prescription, a corresponding bill is also generated.
4) To see the lab reports doctor asks the patient to update re-encryption key (**step X**) in prescription
5) Doctors updates prescription asset with dosage detail, and also a bill is generated corresponding to that prescription
   when updating the prescription with dosage details the prescription is marked as 'COMPLETE'
6) Patient pays for the lab reports and prescription bill
   Whenever a patient pays for a bill, he marks that bill as 'PAYMENT SENT'. When the corresponding generator of that bill successfully verifies his account with the money transfer, then he marks the bill as 'PAYMENT RECEIVED'. When the patient pays for all the generated bills, then the prescription is marked as 'PAID', which unlocks the prescription.
7) After the payment for all bills, the patient can now fetch the prescription
8) Patient goes to the pharmacy and gives them their appointment token, with this appointment token pharmacy, fetches the prescription dosage
9) Pharmacy asks the patient to update the re-encryption key (**step X**)
10) Bill is generated by the pharmacy
11) Payment is made by the patient for the bill generated by the pharmacy
12) Once the payment is confirmed then the pharmacy will ship the medicine

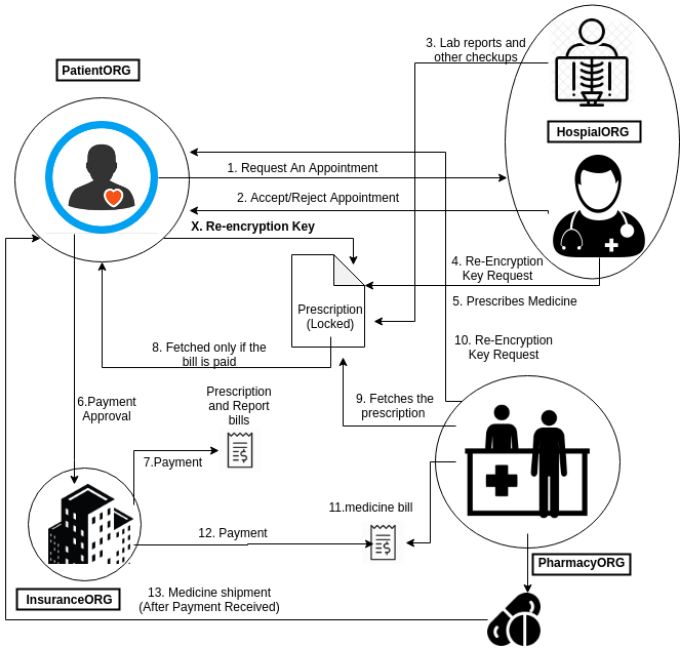

Fig. 4: CASE II: With Health Insurance

2) Case II: Patient with health Insurance: In case II Fig 4 when the patient has opted for the insurance plan, then he can provide his insurance id and insurance company id with at the time of appointment. At any further stages, payment of bills will be made automatically by the insurance company and the main differences from the case I are as follows :

Step 6 After generation of bills by Labs, doctors or pharmacies, the patient will provide user approval on the bills by marking the bill as *APPROVED*.
While providing the approval, the patient will also provide a re-encryption key for the insurance company, which will be updated in the corresponding bill. When the patient provides the approval to the bill, payment requests for that bill are automatically added via smart-contract to the corresponding insurance company's payment requests database.

Step 7 Insurance company will verify the user approval status on the bill, and it also checks the validity of the insurance plan of that user. If the insurance plan is not expired, then the company sends the payment on behalf of the patient.
When the insurance company pays for some bill then it also marks it as 'PAYMENT SENT' similar to the previous CASE I Fig. 3 There are few more smart contracts implemented for CASE II to interact with the insurance company which provides other services like requesting for registration of an insurance plan, checking of the request status and also a smart contract for insurance company which can be used to accept or reject the registration request. Whenever the company accepts a new registration, it also places its public key inside the insurance plan asset Fig. 2 which the user uses to provide re-encryption key. Rest of the steps are

the same as CASE I.

The application provides a rest API interface using express JS [11] server where the users can invoke the smart contracts. 25 smart contracts were developed in this architecture to simulate transaction flow in Fig 3 and Fig 4

An admin dashboard is also provided to analyze the network, and it's transactions which are powered by Hyperledger explorer and is shown in Fig. 5

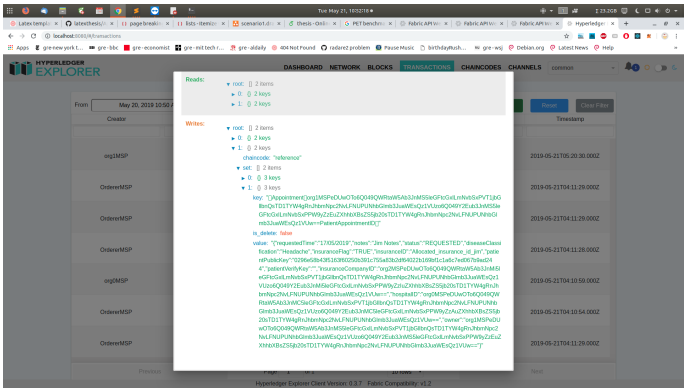

Fig. 5: Admin Dashboard

To build the healthcare application services, 25 smart contracts were developed with each providing different functionality to the end-user of the application. Express JS web application framework was used to expose REST API to the end-user for submitting transaction proposal. These transactions are intercepted at the middleware node where cryptographic material of the invoker identity is added. This cryptographic material includes unique id assigned to the user who is invoking the transaction and also the unique organization id to which the user belongs. Also, this is the stage where the x.509 certificate is attached to the transaction proposal. If this user was not enrolled earlier, then the middleware will interact with MSP to register it with the fabric network, and a secret key is returned to the user. Once the transaction proposal is formed with information of the organization name, channel name,chaincode id, smart contract name, and arguments, then this transaction is then submitted to the committing peer of that organization. The peer then validates the transaction by executing it, and at the same time, it also sends the transaction proposal to other endorsers according to the endorsement policy. Other endorsers also validate the transaction by executing it and prepares a Read set and write set. Read set contains the information that what did the endorsers read from database and write set includes the information that what did the endorsers write into the database. Once the read set and write set are prepared and attached to the proposal response, then the endorser attaches a signature in proposal response saying that it is a valid transaction. After getting a valid response from all the endorsers, the transaction is committed to the ledger by the peer but not updated in the world state, and then it is submitted to the ordering service. The architecture is using raft as the consensus protocol, so if the orderer to which the transaction is submitted is not the leader, it submits it to the leader first then it is the leader's responsibility to assign one of the orderers and sends the transaction to it. The orderer then with many other transactions that came within that second orders it into a block whose maximum number of transaction capacity is 100. The size of the block, i.e., 100 transactions per block is already decided before the bootstrapping of the network. This block is then sent to each peer in the network. Transactions in the block are executed again for verification, and the signatures are verified. Valid transactions are added to the world state. The application will then be notified of the state of the transaction, i.e., whether it was a valid transaction or not.

The complete code of the architecture can be broken into the following modules :

1) Network setup module: This consists of docker and docker-compose files having information related to the network like the number of organizations, number of peers in each organization, the volume that will be mounted in the docker containers, a path to certificates and keys.
2) Network deployment module: This mainly comprises of ansible playback modules [12] which are used to deploy the network. This module starts the runs the docker containers on each server. The modules specify that after the generation of cryptographic material at a single node which files will be sent to which host. Executes commands on each server to create a channel, joins peer on the channel according to the configuration, then installs the chaincode and instantiate it as well.
3) Business logic module: This comprises the actual logic of architecture, i.e., the chaincode where the 25 smart contracts written in go language.
   Each user in the fabric network is assigned an id that is unique within an organization. This id looks something like Fig. 6

```
eDUwOTo6Q049QWRtaW5Ab3JnMC5leGFtcGxlLmNvbSxPVT1jbGllbnQsTD1TYW4gRnJhbmNpc2NvLFNUPUNhbGlmb3JuaWEsQz1VUzo
6Q049Y2Eub3JnMC5leGFtcGxlLmNvbSxPPW9yZzAuZXhhbXBsZS5jb20sTD1TYW4gRnJhbmNpc2NvLFNUPUNhbGlmb3JuaWEsQz1VUw
==
```

Fig. 6: User id

This unique id does not need to be unique across organizations in the network. Also, each organization is assigned an id which is unique across the network. So, in this architecture, org0 is assigned the unique id 'org0MSP' which is actually the MSP id. When the organization id is attached to the id in Fig. 6 then it becomes a unique id across the complete network. These unique ids can be used to implement access policies within smart contracts.

4) Middleware module: Comprises of nodejs files, responsible for preparing transaction proposal for creating a channel, chaincode installation, chaincode instantiation, chaincode invoke, chaincode query, and user enrollment.
5) Application module: Contains express js server files containing code for (Java-based token) JWT authentication

for the user, preparation of request to execute files in middleware. Contains code to intercept the user's request and do the required encryption for some particular functions. Encryption occurs by spawning a synchronous thread on the Express JS server executing a python file which uses the umbral library [13] for the encryption. This server is running on a docker container whose docker image hash value is uploaded on the ledger.

6) Testing module: Since the architecture is pretty big, so python scripts were written having curl requests with various test cases to test the complete chaincode in one go.

The architecture is deployed over four servers running on the private network of IITK, each having the following specifications :

- Operating System: ubuntu 18.04 LTS
- RAM: 16 GB
- Processor: Intel® Core™ i7-4710HQ CPU 3.3GHz
- Cores: 4

Server0 in Fig. **??** is denoted as root orderer because the cryptographic node is generated at this node and then the keys and certificates are transferred to the correct server. Ansible playbook has been used to deploy the network since the policies inside ansible-playbook can be defined with full control during deployment.

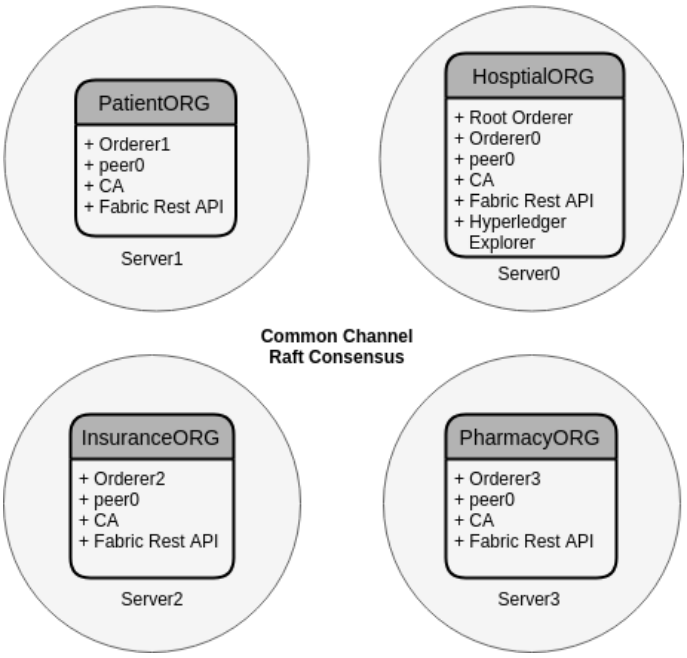

Fig. 7: Physical View

VI. Network Benchmark

To analyze the performance of the network, throughput, and latency's at various stages must be obtained while simulating the real usage scenario. This was done by using the Benchmark tool, Performance Traffic Engine (PTE) [14] provided by the Hyperledger community. Following are some definitions :

1) **Throughput**: The rate at which the transactions are committed to the ledger. Its measurement unit is Transactions per Second (TPS)
2) **Latency**: It's the intermediate time between the transaction proposal generation at the application end and committing of that transaction in the ledger. There can be many latencies in the network, and a few of the important ones under the analysis metrics are the following:
   a) **Peer Latency** : Latency in intercommunication of peers while endorsement
   b) **Orderer Latency** : Time taken at orderers while making batches of transactions
   c) **Event Latency** : This is the overall latency from the transaction proposal generation by the user until the user received the response against that transaction.

The following benchmark has been performed with a new smart contract which was designed computationally heavy involving encryption and decryption, which consumes most of the time at the smart contract level. We performed a benchmark with different configurations to analyze how different parameters affect the network performance :

A. Configuration 1: Solo orderer with multiple hosts

The setup was the same as the physical view in Fig. **??** with the changes being that there is only one orderer in the network instead of 4, which is residing on server0. Following were the configuration used in this benchmark test :

S.No.	Parameter	Value
1	Number of Organizations	4
2	Number of channels	3
3	Number of Peers per organization	1
4	Number of Orderers	1
5	Ordering consensus	Solo
6	Number of Processes	4
7	Number of Processes per organization	1
8	Total transaction per process	1000
9	Transaction frequency	Any Random value between 1ms - 5ms
10	Block Size	100

TABLE II: Configuration I

Analysis: Total throughput of 118.12 TPS was observed in this case and total execution time for sending 4000 transactions was $\sim$ 34 sec where avg latency between peers while endorsing the transaction was 17.55 ms (peer latency) and the average time taken by orderer for preparing a block of 100 transactions was 12.83 ms.

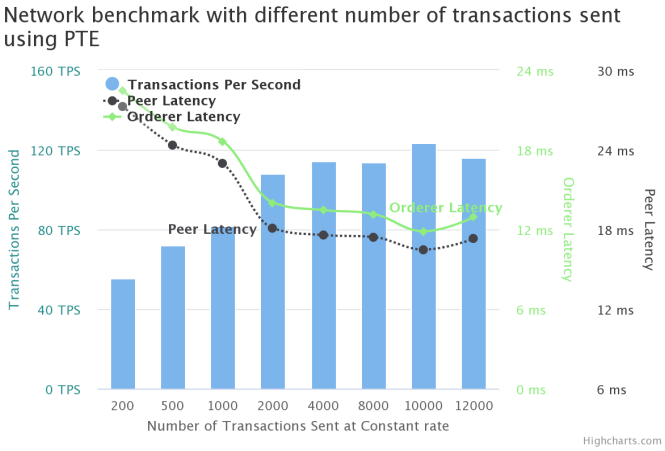

Fig. 8: Transactions in solo orderer network

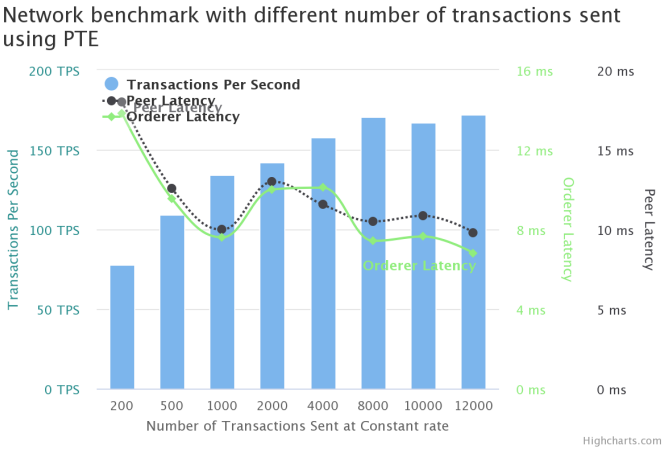

Fig. 9: Transactions in Raft orderer network

Analysis: In the PTE configuration file, the total number of transactions sent in the network was varied to observe the throughput and corresponding latencies in each case. Each transaction was being sent after a time interval of any random value between 1-5 ms. We can observe in Fig. 8 that with an increased number of transactions arrival in the network, the throughput saturated at $\sim$ 115 TPS and the Orderer latency saturated at $\sim$ 13 ms. Reason for the saturation of throughput value can be explained with latency in the network. We can also observe in Fig. 8 that as the latency in the network decreased, the throughput value increased along with it, but with the saturation of latency in the network, the throughput value also reached a saturation level.

B. Configuration II: Raft orderer with multiple hosts

The setup was the same as the physical view in Fig. 7 with raft consensus. Following were the configuration used in this benchmark test with only a few changes in TABLE II:

S.No.	Parameter	Value
1	Number of Orderers	4
2	Ordering consensus	Raft

TABLE III: Configuration II

Analysis: In this case, the throughput saturated at $\sim$ 170 TPS while the orderer latency saturated at $\sim$ 7ms. The orderer latency is less in comparison to solo consensus Fig. **??** because there is only one orderer in solo consensus and with an increase in the number of transaction arrival there is more stress over a single orderer where raft consensus network which has four orderers has less stress on the network. Reason for the saturation of throughput value is the same as the reason given in the analysis of configuration I.

C. Configuration III: Raft orderer with multiple common channels

The healthcare architecture can be deployed with multiple common channels, and we tried to observe the performance in this case. The setup was the same as the physical view in Fig. 7 . The configuration settings were the same as Table III with the only change is that now the network is deployed with three common channels. Raft ordering consensus was used with four orderers residing on each server. The same chaincode was deployed on all the three common channels. PTE was set up to invoke 4000 transactions on each channel at the same time. Rest of the benchmark configuration settings were kept to be same as mentioned in TABLE II

Output:

S.No.	List of Parameters	Observed values
1.	**Channel I**	
	TPS	87.43 TPS
	Peer Latency	22.56 ms
	Orderer Latency	15.92 ms
2.	**Channel II**	
	TPS	86.69 TPS
	Peer Latency	22.29 ms
	Orderer Latency	16.15 ms
3.	**Channel III**	
	TPS	86.15 TPS
	Peer Latency	22.52 ms
	Orderer Latency	15.99 ms

TABLE IV: Participants and their organizations

Analysis: Total throughput : 260.27 TPS = 87.43 (channel1) + 86.69(channel2) + 86.15(channel3) and effective average orderer latency in the network : 5.34 ms = ((15.92 (channel1) + 16.15(channel2) + 15.99(channel3))/3)/3 . One important observation here is that even if the orderer latency is high per channel in comparison to Fig. 9, we still managed to get higher throughput in this case because overall effective average orderer latency in the network is less.

D. Throughput Trend in different configurations

Analysis: In all of the three configurations above, we observed that raft is a better option in comparison to solo ordering service for higher throughput. But with an increase in the number of common channels from 1 to 3, we observed that

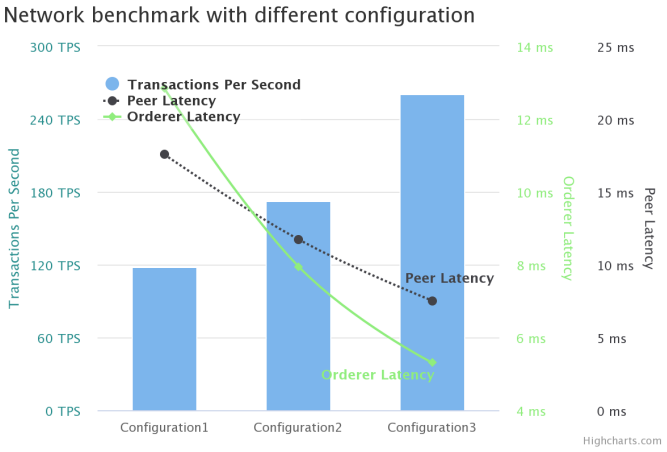

Fig. 10: Network Benchmark with different configurations

the load on the network is distributed among these channels and therefore we received the highest throughput in this configuration. Average orderer latency decreased significantly from 15ms in configuration 1 to $\sim$ 8 ms in configuration 2. In configuration 3, even though per channel orderer latency was $\sim$ 15 ms, it was 5.34 ms for the network.

VII. Conclusion and Future Scope

Privacy is everyone's fundamental right. Laws like Hippa [15], 1996 in the US ensures complete privacy of patients in health-sector, but it is a sad truth that in India there are no strict laws to enforce the privacy of patients, but it shall not remain true forever in the future. It is up to a person to decide which personal information he wants to share. This healthcare architecture can help the patient in preserving their privacy and at the same time allowing many other healthcare services. This can help the doctors to treat the patient quickly and efficiently because they can use the medical records of that patient whose integrity is ensured by the immutability property of the blockchain. With a few improvements, this architecture can be a big help in scenarios like disaster management where user don't have to worry about the payments if he has an insurance plan registered. The corresponding insurance companies will handle all the payments in such tragedy. From this project, one thing that we can conclude for sure is that blockchain can provide support to our progress towards a more digitized world for business networks like health sector and therefore making a significant improvement in living standards of humanity.

There are multiple scopes in terms of improvement in the architecture. Improvement can be made in both performance and business logic. More than 1000 comprehensive performance experiments on a very similar network comprising of 4 organizations are carried out with different configurations in [16]. With common channel but two peers on each organization, they received a throughput of 140 Transactions Per Second (TPS). Then the article experimented with different configurations including an increasing number of channels, peers and orderers in the network, horizontal and vertical scaling of processing power and RAM on the nodes, changing block sizes, switching between the state databases, i.e., goleveldb and CouchDB and so on. After conducting all those experiments, they were able to increase the throughput to 2700 TPS.

The same experiments could be performed on the chaincode of our architecture to get the most optimized configuration. Further improvements can be made by incorporating banks in the architecture, which could make the payments more easier to the patients. Also, delivery agencies can also be added to the architecture to ship the medicines, lab reports, and other assets which could make the life of patients more comfortable. Furthermore, research needs to be done on the amount of data generated in the health sector and how can it be managed in the most efficient way.

References

[1] Luc Desrosiers, Venkatraman Ramakrishna, Petr Novotny, Salman A. Baset, Anthony O'Dowd, "Hands-On Blockchain with Hyperledger: Building decentralized applications with Hyperledger Fabric and Composer," June 2018, "http://doi.acm.org/10.1145/1127345.1127346"

[2] Ateniese, Giuseppe and Fu, Kevin and Green, Matthew and Hohenberger, Susan, "Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage ", ACM Transactions on Information and System Security vol. 9.1, 1094-9224, pp. 1–30, "http://doi.acm.org/10.1145/1127345.1127346"

[3] Katherine K Kim, Pamela Shankar, Machelle D Wilson, and Sarah C Haynes. 2017.Factors affecting willingness to share electronic health data among California consumers.BMC Medical Ethics 18, 1 (2017), 25. "https://doi.org/10.1186/s12910-017-0185-x

[4] A. Azaria, A. Ekblaw, T. Vieira and A. Lippman, "MedRec: Using Blockchain for Medical Data Access and Permission Management," 2016 2nd International Conference on Open and Big Data (OBD), Vienna, 2016, pp. 25-30." http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7573685&isnumber=7573673"

[5] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends ", June 2017, "https://www.researchgate.net/publication/318131748_An_Overview_of_Blockchain_Technology_Architecture_Consensus_and_Future_Trends"

[6] Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System", "https://bitcoin.org/bitcoin.pdf"

[7] Dr. Gavin Wood, "Ethereum: A Secure Decentralised Generalised Transaction Ledger", "https://gavwood.com/paper.pdf"

[8] Diego Ongaro, John Ousterhout, "In Search of an Understandable Consensus Algorithm ", "http://raft.github.io/raft.pdf"

[9] Hyperledger documentation, "https://hyperledger-fabric.readthedocs.io/en/release-1.4/"

[10] Steven Galbraith, "Mathematics of Public Key Cryptography", ch-21, 2.0, "https://www.math.auckland.ac.nz/~sgal018/crypto-book/ch21.pdf"

[11] Azat Mardan, "Express.js Guide: The Comprehensive Book on Express.js", 2014-05-28, "https://leanpub.com/express"

[12] Gleb Iodo, "https://github.com/Altoros/Ansible-Fabric-Starter"

[13] Gleb Iodo, " python3 library for proxy re-encryption" , "https://pypi.org/project/umbral/"

[14] Performance Traffic Engine, "https://github.com/hyperledger/fabric-test/tree/master/tools/PTE"

[15] Medical Learning Network, "Hipaa Basics For Providers: Privacy, Security,And Breach Notification Rules", "https://www.cms.gov/Outreach-and-Education/Medicare-Learning-Network-MLN/MLNProducts/Downloads/HIPAAPrivacyandSecurity.pdf"

[16] Parth Thakkar, " Performance Benchmarking and OptimizingHyperledgerFabricBlockchain Platform", "https://arxiv.org/pdf/1805.11390.pdf"