End to End Integrity Protection for Web Application Using Blockchain

By

Muhammad Zaid

172169-MS(IS)-9-2016F

Supervisor

Dr. Shahzad Saleem

Department of Computing

A thesis submitted in partial fulfillment of the requirements for the degree of Masters in Information Security (MS IS)

In

School of Electrical Engineering and Computer Science,

National University of Sciences and Technology (NUST), Islamabad,

Pakistan.

Dedication

To my parent, my supervisor and my friends and who has always been supportive.

Approval

It is certified that the contents and form of the thesis entitled "End to End Integrity Protection for Web Application Using Blockchain" submitted by **Muhammad Zaid** have been found satisfactory for the requirement of the degree.

Advisor: **Dr. Shahzad Saleem**

Signature: _____

Date: _____

Committee Member 1: **Dr. Naveed Ahmed**

Signature: _____

Date: _____

Committee Member 2: **Dr. Abdul Ghafoor Abbasi**

Signature: _____

Date: _____

Committee Member 3: **Madam Haleemah Zia**

Signature: _____

Date: _____

Certificate of Originality

I hereby declare that this submission my own work and to the best of my knowledge. It contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgment, is made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistic is acknowledged.

Author Name: **Muhammad Zaid**

Signature: _____

Acknowledgment

Thanks to the Almighty ALLAH for letting me pursue and fulfill this research work. I could not have achieved anything without HIS utmost support and countless blessings.

I am thankful to my beloved parents for their support throughout my educational carrier. I dedicate this work to my lovely parents, honorable teacher. They have always supported and encouraged me to do the best in all matters of life.

I am truly grateful to my supervisor Dr. Shahzad Saleem for his guidance, supervision, and encouragement to complete this task. He has been a great source of inspiration for me during my research, thank you so very much. I am obliged to all my respectable teachers for providing me their valuable time and considerations. I believe that this work would not have been possible without their guidance and expert suggestions.

I am also thankful to my committee members Dr. Naveed Ahmed for sharing with me his valuable experiences regarding the web & network security, Dr. Abdul Ghafoor Abbasi for guiding me through his experience in blockchain, and Ms. Haleema Zia for their contribution and timely suggestions toward the successful completion of this thesis.

Despite all the assistance provided by the supervisor, committee members, and others, I take the responsibility for any errors and omissions which may unwittingly remain.

Abstract

Web application uses a remote web server to stores its wide range of data. The data could be financial records, news, stock prices, weather forecast or medical record of patients. So, a web application is totally dependent on a remote potentially hostile web server for the security of its data and query results. If an attacker gets control over the web server, one cannot guarantee the integrity of data and query results. If an attacker tampers the critical data like stock prices or diagnostic medical records of patient present on a web server that is to be used in decision making, it can cause some severe monetary and health damages.

Although we cannot prevent the data from getting tampered, however, we can detect if someone has illegally tampered it. Instead of being tamper resistant we have provided an efficient and secure tamper evident solution to this problem. Our solution provides strong evidence to data user (decision maker) that if data provided by the server is tampered or not, even if the server is complete control of an attacker. Using blockchain technology as a trust base, our practical solution guarantees the correctness and freshness of data with minimum overhead. To check if the solution is practically convenient, we have integrated our solution with remote medical web application and evaluated its results. In a remote medical web application, a patient remotely uploads its diagnostic data, and physician (decision maker) evaluates the patient uploaded data and prescribe the medications accordingly. Our solution provides an efficient, secure and strong proof to the decision maker or physician that if patient's data is been illegally tampered or not on compromised web server. The provided solution can be integrated with any web application by adding minimal changes in the application's existing structure.

Table of Contents

List of Figures

List of Tables

Introduction

This introductory chapter includes context and basic information to develop an understanding of this research work. The process and terminologies followed are briefly discussed to build an essential knowledge base for both novice and expert audience. It also includes the background, motivation behind this study, and problem statement to be addressed. Goals and objectives, intended audience, and scope of this work are also incorporated. At the end of this chapter, the organization of this research project is included. Following is the list of sections in this thesis.

- Section 1.1 Background
- Section 1.2 Motivation
- Section 1.3 Problem Statement
- Section 1.4 Goals and Objectives
- Section 1.5 Intended Audience
- Section 1.6 Scope of the Study
- Section 1.7 Organization of Dissertation

1.1 Background

Internet was first introduced in the late 1960s by U.S. Department of Defense and was then named APRANET. It connected computers and allowed them to communicate with each other. In 1989 Tim's Berners Lee[1] while working at CERN noticed to the problem of sharing information between scientists. He decided to solve this problem and by the end of 1990, he introduced the world first web application [2]. In 1993 it was

decided to share the underlying code for free and to make WWW open for everyone. Now there are more than 1.5 billion web applications available on world wide web [3].

The web application allows a user to perform different tasks using a web browser and the internet. These tasks may include e-Shopping, online banking, emailing, checking stock prices or weather updates, etc. Web applications consist of two parts [4] i) Client ii) Server. Client requests data from the server and displays it to the user part. The client-side interface of a web application is developed using HyperText Markup Language (HTML), JavaScript and it runs in a web browser. While Server stores and maintains the application's data. It serves the data to different clients of an application as per their demand. There are many different programming languages like Ruby, PHP, C#, etc. that can be used on the server side to process client requests or quires and to display requested web pages.

There are two basic types of the web application [5], static web application, and dynamic web application.

1.1.1 Static Web Application

Static applications [6] are less interactive (doesn't provide quires over data) and displays fixed content or web page to every user. As these applications display static web pages only so they are developed using client-side languages only like HTML, CSS, JavaScript, etc. It doesn't involve any server-side programming as these applications don't maintain any user database.

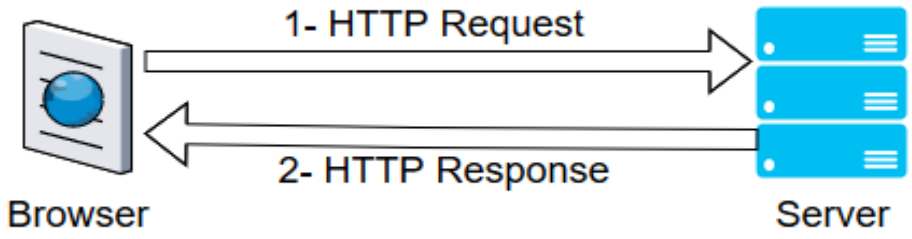

Figure 1-0-1 Data Flow of static web application

Figure-1.1 explains the basic workflow of a static web application. When browser or client makes an HTTP request for any web page from a server. The server simply locates the requested web page from its storage and sends it back to client/browser as an HTTP response.

1.1.2 Dynamic Web Application

Dynamic web application [7] maintains database like sports updates, stock prices, news, weather reports or any kind of user's data, etc. The client can make different quires on data and server generates a dynamic web page as per requested data. These applications require both client and server-side programming to develop.

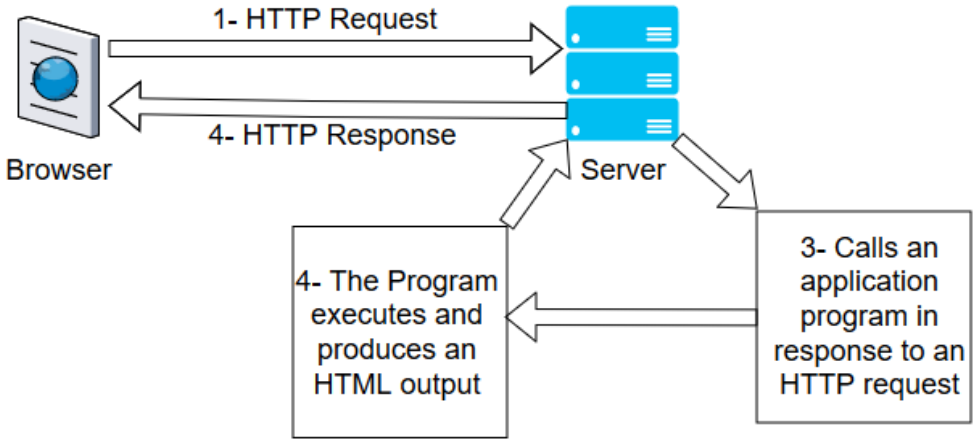

Figure 1-0-2 Data Flow diagram of dynamic web application

Figure-1.2 explains the basic flow of dynamic web application. When a user makes a request for specific data from an application server. The server calls applications program to deal with the request, the program executes and produces an HTML based output. The server sends the results back to the client. The browser runs the provided HTML code and displays the result to the user for its query.

1.2 Motivation

Web applications use data servers to store different type of data. This data can be about financial or forensic records, stock prices, weather forecast or medical record of

patients. Data owners or users are totally dependent on web servers for the integrity protection of their data. However, if the web server gets in control of an attacker, we cannot guarantee the integrity of data served by the web server.

By the increase in web applications, attacks on web application have also increased rationally. According to a Symantec report [8] attacks on web applications has increased by 58 percent in 2018 and it was approximately 0.9 Million attacks per day. In another report [9], approximately 67 percent of web applications in 2018 were highly vulnerable to information exposure attacks. Data hosting companies show reluctance to secure the data. According to a recent survey [10] nearly 47% percent of companies have not assigned an individual or team for the security of their data. These stats are motivational for an attacker to tamper the data available on a web server.

The integrity of the data present on a web server is very important, if data is to be used in decision making. Let's make it clear with some examples. A weather forecast agency (data owner) uploads weather-related data to its web application, if the integrity of weather-related data gets compromised it can affect the decisions to be taken by a farmer. Stock exchange (data owner) publishing latest stock prices to its web application, compromised integrity of stock prices on stock exchange web server will affect the decision making of broker and it can lead to monetary loss. Same is the case for a remote medical web application, where a patient (data owner) uploads its daily diagnostic data e.g. pulse rate, blood pressure, etc. Physician remotely accesses this data for the diagnostic purpose. Inaccurate or incomplete diagnosis will occur if medical data of patient present on web application server gets modified by an attacker. It can result in wrong medication which may lead to the death of the patient in a worst-case scenario. According to a study [11], misdiagnosis is the 3rd leading cause of death in the USA. According to another study, 40 thousand to 80 thousand people die annually due to diagnostic errors in the US only. Similarly, a Syrian hacker hacked the website of American news agency 'Associate Press' and uploaded the fake news of Obama's death [12], this cause the loss of $136 billion in stocks to AP.

1.3 Problem Statement

Web application stores its data on a remote web server. In this case, owner and the decision maker are totally dependent on the server for protection of the data. If the server gets compromised the authenticity and integrity of data stored on it cannot be guaranteed. A solution is required to provide end to end integrity for a web application with minimum overhead.

The solution should ensure integrity properties [13][14] including,(i) correctness and (ii) freshness. *Correctness* means that the data served by server is the one that's been queried by client. *Freshness,* on the other hand, means that the data served by server for client's query is up to date.

1.4 Goal and Objectives

The goal of our research is to provide end to end integrity to the web application. This integrity should be ensured even if an attacker gets control of the web server and tampers the data available on the server. In case of a compromised web server, if a user will request the data from the server for some decision making then he will get notified that some on has illegally changed their data.

Our objective is to provide the integrity protection framework. Using the provided framework, a web application developer will easily make its application secure against integrity attacks. He will need to make very few changes in its existing application structure. The provided solution will be fast and will add minimum overhead.

1.5 Thesis Organization

The thesis is divided into different chapters for better understanding and semantics. Following is the brief description of chapters.

- Chapter 1 provides the important basic concepts related to the problem and historical background of the problem

- Chapter 2 explains the previous work done by researchers in this domain.
- Chapter 3 describes the methodology.
- Chapter 4 explains the implantation details of our solution.
- Chapter 5 consists of the results obtained after solution implementation
- Chapter 6 concludes the thesis and discuss future work regarding the problem

2ND CHAPTER

2 Literature Review

Our security community has acknowledged the integrity problem and a lot of work has been done for both file system and database integrity protection. However, we have seen lesser amount of work when it comes to the integrity protection in the context of correctness and freshness while dealing with web servers.

So, in this chapter, first we will discuss the published literature available related to File and File System Integrity. Then we will discuss the published work related to the integrity protection of an outsourced data. At the end of this chapter, we will discuss the studies related to web application integrity protection.

2.1 File and File System Integrity

G. Ateniese, R. Burns [15] has provided the technique to efficiently authenticate the file data stored on a remote and untrusted server. In this approach client (data owner) generates metadata of the file and stores it locally before sending it to a remote server as shown in Figure-2.1. Later at any time, the client can send a challenge to the server if he wants to verify that if a server has a correct copy of the file. Instead of sending the whole file to the client for verification, server computes the results against the challenge using some function on the file and sends the results back to the client. A client can then efficiently verify the results by using its locally stored file metadata. If the file consists of 10 thousand blocks and compromised server deletes 1% of the blocks, then using the

proposed technique client can efficiently detect server behavior by randomly selecting 460 blocks i.e. 4.6% of the file.

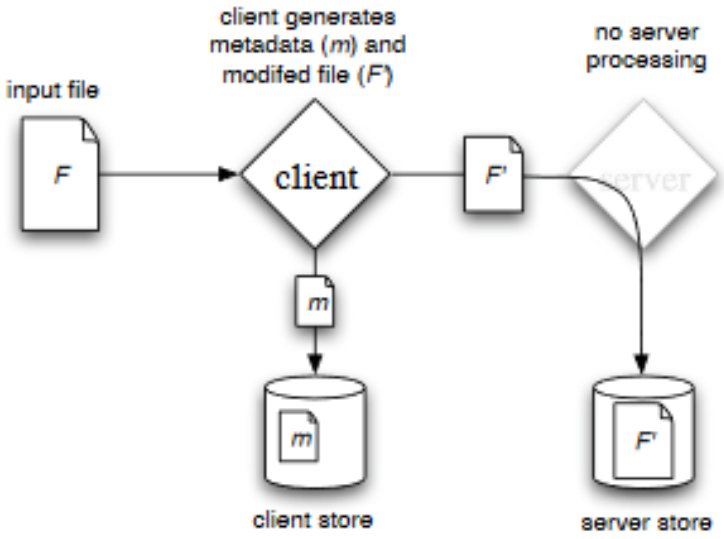

Figure 2-1 Pre-Process and File Storage

SUNDR [16] also addresses the problem and guarantees the detection of unauthorized modification of a file on the compromised server. Figure-2.2 explains the basic architecture. On retrieval of a file by an application, system calls are translated into 'modify' and fetch operations. SUNDR provides "fork Consistency" to clients i.e. clients can see each other updates to the file stored on a remote server. Clients maintain a version list against each stored file. When any client makes updates to the file, he also updates the version no. in list shared by other clients. In this way, each client knows a new version of the file. This allows the client to detect any unauthorized change in the file by using some out-of-bound communication.

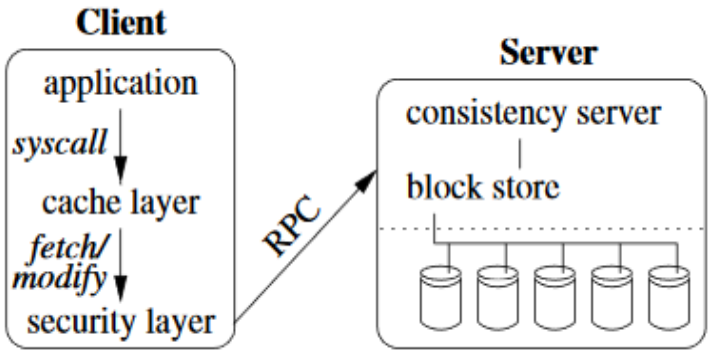

Figure 2-2 Basic SUNDR Architecture

Deswarte Y, Quisquater JJ [17] also has worked on integrity protections of files, stored on a remote server using RSA-based hash function. In their solution data owner generates the multiple challenges of the file, using its metadata, before sending it to a server. The owner also computes the response of challenges and saves it locally. So, when the owner needs to check the integrity of the file it sends multiple challenges to the server. Although this solution is efficient for data owner (verifier) but not for the server. As server needs to access the whole file to compute the challenge sent by the verifier.

I. Zikratov, A. Kuzmin [18] has used blockchain technology to ensure the integrity of files uploaded on the cloud. When any file in a system gets modified, OS sends a notification to the "FileSystemWatcher" class. The class then creates a transaction, contains File Hash, Previous Hash, Timestamp, and Signature of the modifier. The transaction then pushed on the blockchain. This solution will not be able to provide file integrity if the cloud gets compromised. If an attacker gets control of the cloud, he can easily change the "FileSystemWatcher", hence integrity cannot be promised to the files stored on a cloud.

2.2 Outsourced Database Integrity

Instead of hosting database locally most of the companies for their ease, outsource their databases to the cloud or to remote servers. By this, they lose control over data, so it

arises new challenges related to the authenticity and integrity of their data stored on untrusted servers. Much work has been done in recent years to detect any unauthorized tampering in the outsourced database. Three different approaches have been used to solve this problem.

- Authenticated Data Structures
- Signature Aggregation
- Probabilistic integrity verification

2.2.1 Authenticated Data Structures for Integrity Protection

The first approach is to use Authenticated Data Structure (ADS) e.g. Merkle Has Tree (MHT) [19] to protect the integrity of the outsourced database. MHT is a tree data structure in which leaf nodes of the tree consists of a hash of the data and non-leaf node or intermediate nodes are produced by hashing the concatenated hashes of child nodes. MHT Provides efficient verification of data. Some researchers [20][21][22][23] have used this approach to make database tamper resistant to unauthorized entities. The key idea is to produce an index based on Merkle Hash Tree, have DB data at its leaf nodes. So, if the data owner needs authenticate the data, he needs a verification object consists of the tree root, and value of intermediate nodes making a path to the root.

R. Jain and S. Prabhakar [20] have used an extended version of Merkle Tree called Merkle B Plus Tree (MB Tree) as a building block in their solution. In their solution, multiple clients can access and update the data upload by data owner on the untrusted server. Every time a client makes changes to the uploaded data, server needs to assure it by sending root hash of MB Tree, that the provided change has executed correctly and completely with client's signature who had made this change. R. Jain's model is not suitable for web setting. As in the proposed model client who changes the data, also sends the proof (MB Tree root) to the data owner. This is not possible in web applications because web settings don't allow clients to communicate with each other directly, bypassing the server.

MichaelT. Goodrich [24] has used skip list based authenticated data structure to provide an efficient authentication for a outsource databases. Although the provided solution can verify efficiently and can verify 0.7 million records in 0.7 milliseconds but, it only supports the basic insert, update, and delete quires and does not authenticate the aggregated queries.

IntegriDB [23] is a practical implementation of ADS based technique. It allows JOINS and multidimensional range quires on integrity secured database. As in [20] IntegriDB also requires some direct and out-of-bound communication between data owner and client to share the digest.

2.2.2 Signature Aggregation for Integrity Protection

The second approach to protect the integrity of data for an outsourced DB is signature aggregation. Signature aggregation generates one signature by the aggregation of multiple signatures generated by different signers on distinct messages. M. Narasimha and G. Tsudik [25][26] has used signature aggregation to protect the integrity of outsourced data. Although this technique provides less computational and communication overhead as compared to ADS but, it fails to guarantee the completeness (complete results) for queried data.

2.2.3 Probabilistic integrity verification

Using this approach [27] data owner adds some fake tuples in the database before uploading it to a remote server. The added fake tuples are also known to all other users or clients. When data is queried from the server, it's been checked that fake tuples returned by the server are the one that satisfies the query. The technique gets failed, if dishonest server colludes with a dishonest user or if any dishonest user gets control of server.

2.3 Web application integrity protection:

As discussed earlier that web servers store the data of a web application. To protect the integrity of data Verena [14] uses authenticated data structures as a building block. It

ensures the integrity of the data stored on the untrusted web server. As web applications are stateless, the user loses its state stored in a browser when he signs out of an application. Although an authenticated data structure can prove the correctness and completeness of data, it can't ensure the freshness of data on a web server in case of seamlessness. Verena uses a hash server to ensure the freshness of queried results as shown in Figure-2.3. Hash server stores hash of the data, its version, and identity of the last modifier. However, Verena doesn't guarantee the security of Hash Server. So, if the hash server gets compromised, Verena cannot guarantee the freshness of queried data.

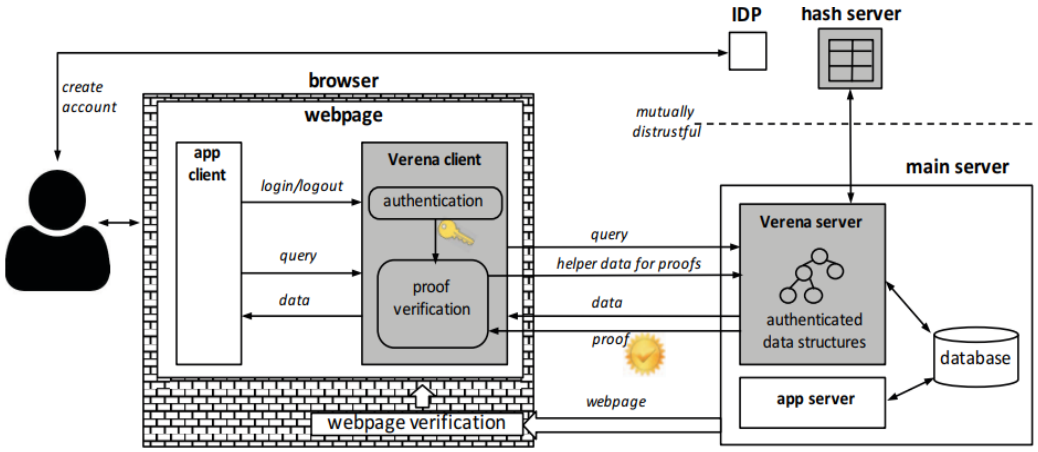

Figure 2-3 System Overview of Verena [14]

Hallgren and Mauritzson have presented GlassTube [28], it protects the integrity of data communication between client and server. It is an alternate and lightweight approach to HTTPS. However, GlassTube doesn't guarantee the integrity of data stored on a server.

Pedro Fortuna and Nuno [29] also has presented a framework for web integrity protection. But, it only protects the integrity of data available at the client side and doesn't provide server-side data integrity protection.

3 Design and Methodology

In this chapter, we will present the basic design of our solution to provide end to end integrity for a web application with minimum overhead. The solution also ensures integrity properties such as correctness (web server has executed the client query correctly) and freshness (web server has sent an up-to-date data to the client). The building blocks that we used produce our solution are following.

- One Way Hash Function
- Merkle Hash Tree
- Digital Signatures
- Blockchain

First, we will explain the basic concept of tools used in our solutions.

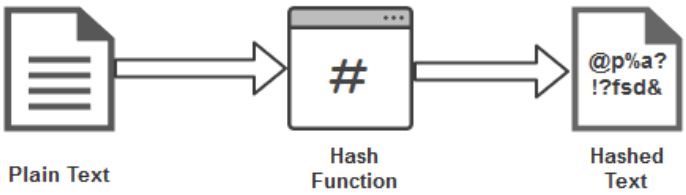

Figure 3-1 Working of Hash function

3.1 One Way Hash Function

A mathematical function h which takes a data x of variable length as an input and provides a binary sequence $y = h(x)$ of fixed length called hash as an output as shown in Figure 3.1 [30]. There are three important properties of a hash function:

i) It is nearly impossible to produce actual data x from a given hash y.

ii) The hash function also makes it impossible to find two different strings of actual data, x and z, that produces same hashed values i.e. $h(x) = h(z)$.

iii) In a hash function changing only one bit of input string, will at least change the half of bits of hash value (output).

3.2 Public Key Cryptography:

Public key cryptography is also known as asymmetric cryptography [31], was first purposed in 1976 by Deffi and Hallmen. The key idea is to use two different keys for encryption and decryption of data. Private key is generated locally and only known to owner, while the public key can be distributed publicly among other parties of the system. In Public Key cryptography, it is impossible to compute private key from provided public key. RSA [32] and ElGamal [33] are the famous algorithms of the system. Public key cryptography provides both confidentiality and authentication.

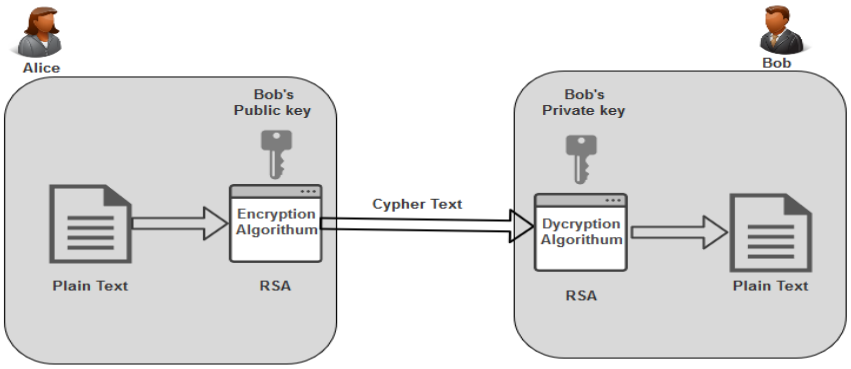

Figure 3-2 Confidentiality in PK Cryptography

First, we will discuss how PK cryptography provides confidentiality. In Figure 3-2, Alice needs to send a confidential message to Bob, that no other than Bob can read the message. She uses Bob's public key for data encryption using the RSA algorithm and to produce ciphertext. Now this generated cyphertext can only be possible to decrypt by Bob's using the private key only. No other than Bob can decrypt the cipher text. So, in this way, Public Key cryptography provides confidentiality.

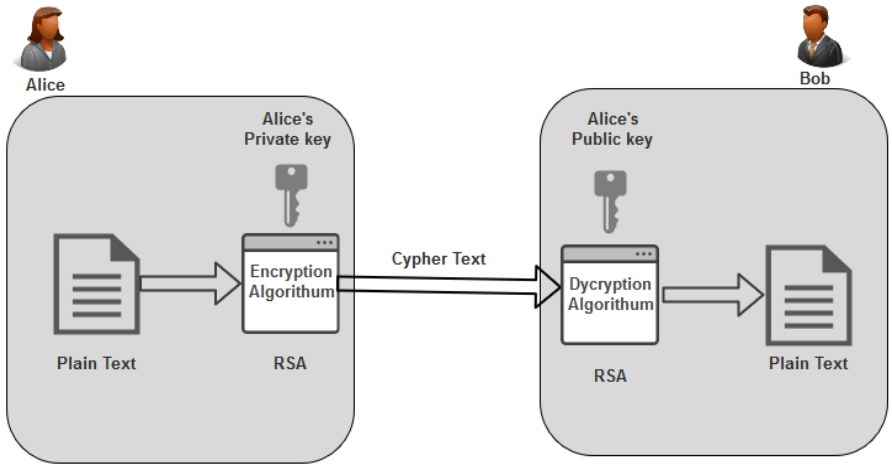

Figure 3-3 Authentication in PK Cryptography

For authentication, as shown in Figure 3-3, Alice encrypts the data with his private key using RSA and sends the ciphertext to Bob. Now Bob will authenticate that the cipher text is signed and sent by Alice. For the purpose, he will use Alice's public key. Authentication will be successful if cipher text decrypts successfully using Alice's public key. In our solution, we have used the authentication feature of Public Key cryptography.

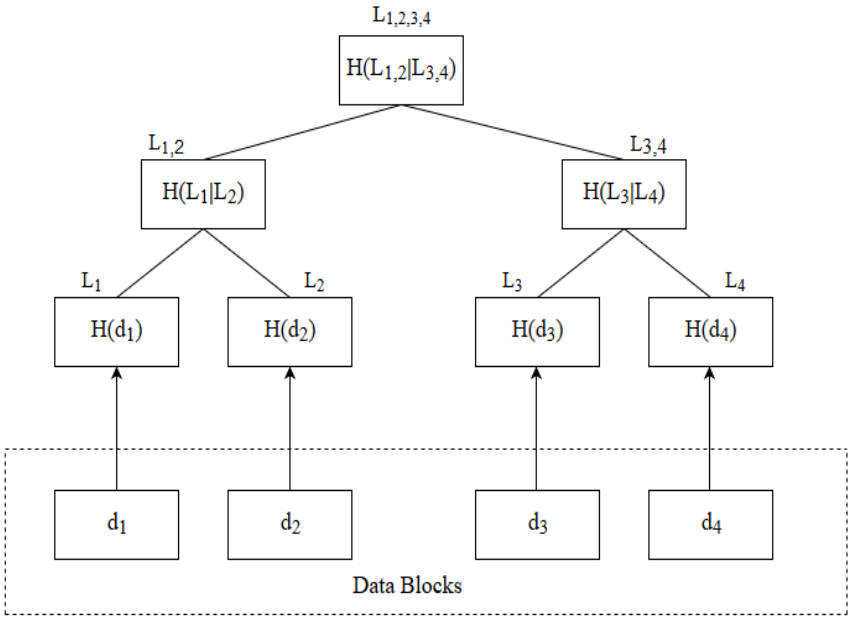

Figure 3-4 Example of Merkle Hash Tree

3.3 Merkle Hash Tree

Merkle Hash Tree (MHT) [19][34] is a hash-based tree data structure in which every leaf node contains a hash of the data and every non-leaf node is a hash of concatenated hash of its two children. As in Figure-3-4 non-leaf node $L_{1,2}$ is produced by hashing the concatenated hashes of its children i.e. L_1 and L_2.

MHT is useful in efficient authentication of data with the help of verification object (*VO*). For Example, to authenticate d1, the *VO* consists of L_2, $L_{34,}$ and the root node $L_{1,2,3,4.}$ With help of provided *VO*, an entity can easily verify the authenticity by computing *H(d1)* and then checking if *H(H(H(d1)|L2)|L3,4)* matches with the root node

$L_{1,2,3,4}$. If both matches successfully, d1 is accepted: else, $d1$, L_2, $L_{3,4}$ or the root node $L_{1,2,3,4}$ has been tampered.

3.4 Digital Signature

A digital signature is a cryptography-based electronic analog of regular physical signature [35]. It serves the same purpose as of regular signatures:

i) Digital signatures offer authentication i.e. it ensures recipient that the digitally signed message is coming from the right person.

ii) It provides non-repudiation, which means that the sender can't deny the authenticity of his signatures on a message.

iii) Digital signature ensures the integrity of signed data. As if message/data got changed, it will not produce the same signatures

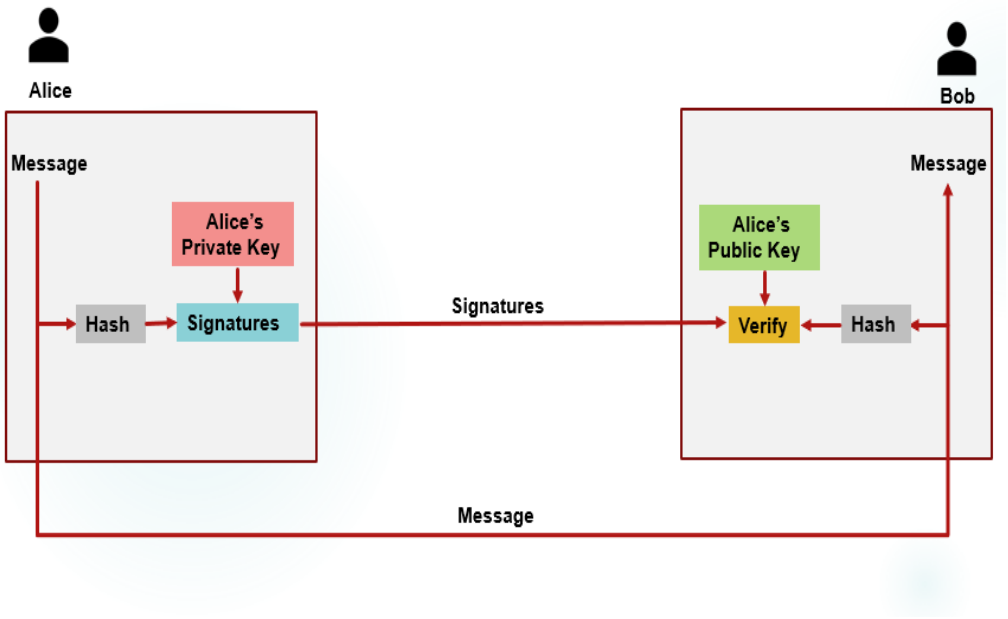

Figure 3-5 Working of Digital Signatures

Figure 3-5 shows the workflow of digital signatures. If Alice wants to send the signed message to Bob, she will first take the hash of the message. After taking a hash, she will then sign it by encrypting the message using her private key. The message will then be sent to Bob along with the signature. Bob receives the message and signatures. Now to

verify the signatures Bob takes a hash of the received message. He also needs to decrypt the signature sent by Alice using Alice's public key. Signature verification will be successful if the decrypted hash matches Bob's hash of Message.

3.5 Blockchain

Blockchain [36] can be regarded as a public distributed ledger. It consists of a collection of records called blocks, linked with each other, these blocks are strongly resistant to alteration and are protected using cryptography. A block in blockchain contains records, a hash of the block and hash of the preceding block. Different types of Blockchains store different types of records. For a healthcare blockchain, the data in a block would be medical records of patients and for bitcoin blockchain, the record would be transaction details. As block also contains the hash of the preceding block, so it effectively produces the chain of blocks called ledger. This chain of blocks also protects the data in a block against tampering. Modification of data in any block in the chain will also change its hash, it makes all following blocks invalid because they no longer store a valid hash of the preceding block. In a blockchain system, each node is connected to a peer-to-peer network and each node has a full copy of the blockchain. When someone in the blockchain network creates a new block before it is added to the chain it needs to be verified by everyone in the network. Consensus is made by the nodes in a network. If more than 50 % of the nodes in the network verify it, then the block is added to chain by everyone in the network. In this way, distributive nature of blockchain also makes the data in the blockchain, tamper resistant. That's the property of blockchain that had made it famous. It becomes very difficult to change the data after it gets recorded inside a blockchain. For an attacker to change the data, he needs to change all blocks of chain and should have control over 50% of the network.

The blockchain that is open to anyone in the network is called public blockchain. While the other which is closed to the selected group of authorized users is called permissioned blockchain [37].

3.5.1 Public Blockchain:

Public blockchains [38] are open to anyone, which means anyone can join the network without any permission. Anyone can make transactions, can read transactions, and can take part in the census. To become a part of the network one need to download the open source code, run it on its local machine. In public blockchains, participant will have an anonymous identity. With an anonymous identity participant can make a transaction, can also take part in a consensus mechanism i.e. the process of validating blocks or group of the transaction and adding it to the chain. So, in public blockchains, anybody can make transitions on the network and this transaction will be added to chain if it is valid. Public blockchain uses Proof of Work and Proof of Stake as consensus algorithms. Bitcoin and Ethereum are examples of public blockchains. In bitcoin, it takes 10 min for a transaction to be a part of the chain.

3.5.2 Permissioned Blockchain:

Permissioned blockchain [37] maintains the identity of participants and regulate the role of participants, which provides more trackability and efficiency. So, a permissioned blockchain deployed in an organization, organization controls the read or write access to the blockchain. As the participants do not have any anonymous identity, so it helps in better auditing and trust; who does what on blockchain at what time. Permissioned blockchains are faster than public blockchains as it do not need Proof of Work or Proof of stake like census algorithms. In our solution, we have used the Hyperledger fabric [39] blockchain.

3.5.3 Hyperledger Fabric:

Hyperledger fabric [39] is an opensource permissioned blockchain managed by Linux foundation. The modular architecture for fabric accommodates the diversity of enterprise use cases through the plug and play components such as consensus, privacy and membership services. One of the many compelling fabric features is the enablement of a network of networks. Members of a network work together but because business needs some of their data to remain private, they often maintain separate relationships within their networks. Rather than an open, permissionless/public system, the fabric

offers a modular, scalable, and secure platform that supports private transactions and confidential contracts. The fabric helps members manage confidential obligations to each other without first passing it through a central authority. That way, personal data isn't available to the entire network. If a member is not agreed upon a party, the transaction shouldn't appear on their ledger. This architecture allows for solutions developed with fabric to be adapted for any industry, thus ushering in a new era of trust, transparency, and accountability for businesses.

Fabric peers usually have three kinds of roles. They can be an endorser, committer and ordered. **Endorsers** are the ones which simulate the transaction to check if it gives the consistent and deterministic outcome. **Committers** verify the integrity of the transaction before adding it to the chain. **Orderer's** job is to maintain a consistent state of ledger over the network.

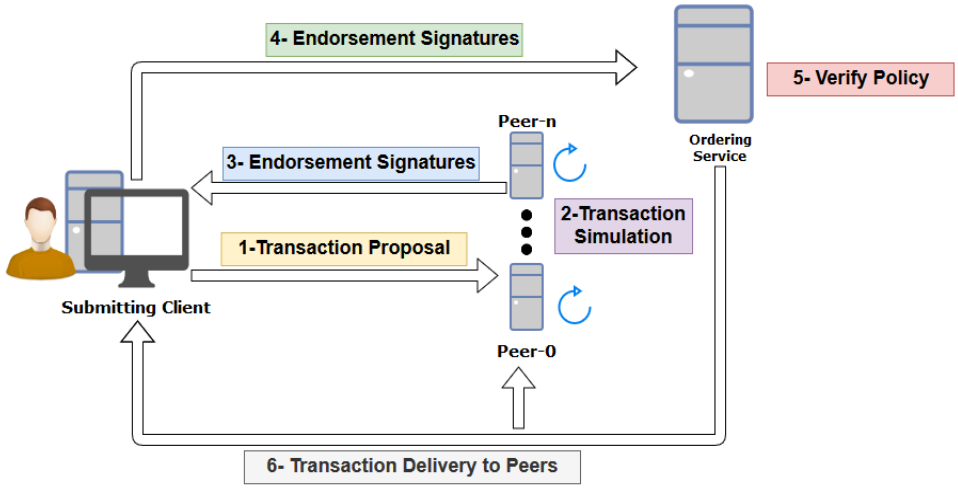

Figure 3-6 Transaction Flow in Hyperledger Fabric

3.5.4 Transaction Flow in Hyperledger Fabric:

Transactional flow [40] of Hyperledger Fabric consists of six steps as shown in Figure-3-6. Description of each step is as following:

Step 1: In the first step the client sends the transactional proposal to endorsing peers

Step 2: Endorsing peers simulates the transaction proposal over the copy of ledger they have. Peers then create read/write set i.e. what is read from the ledger and what will be written on the ledger while simulating the transaction proposal on the current state of the ledger.

Step 3: After cryptographically signing, peers then send the results i.e. read/write set back to the client.

Step 4: Client then sends the signed and endorsed transaction along with read/write set to the ordering service.

Step 5: Ordering service consists of a cluster of orderers peers. It accepts the endorsed transactions to verify its signatures along with policy or chaincode. Ordering service then uses the algorithms like Kafka, Solo to produce an order in which these transactions will be added to the ledger. Ordering service then sends the data to committers. Committing peers verifies that if the read/write set from different endorsing peers matches the current state of the ledger.

Step 6: If the match is successful committers write the transaction to their ledger and inform the client about it.

3.6 Basic Setup of Solution

We will explain the basics of our solution with the help of a use case. Remote medical web applications are very common nowadays. Using this application patient (data owner) uploads his diagnostic data e.g. heart rate, blood pressure, sugar level, etc. The application saves this data in the database on its server. The application also provides an interface to a physician (Decision maker). The physician can remotely view the patient's data and prescribes him medications accordingly. Now if, medical data of patient present on web application server gets modified by an attacker, it can result in wrong medication which may lead to the death of the patient. Our solution will alert the physician when he accesses the patient's data tampered by an attacker.

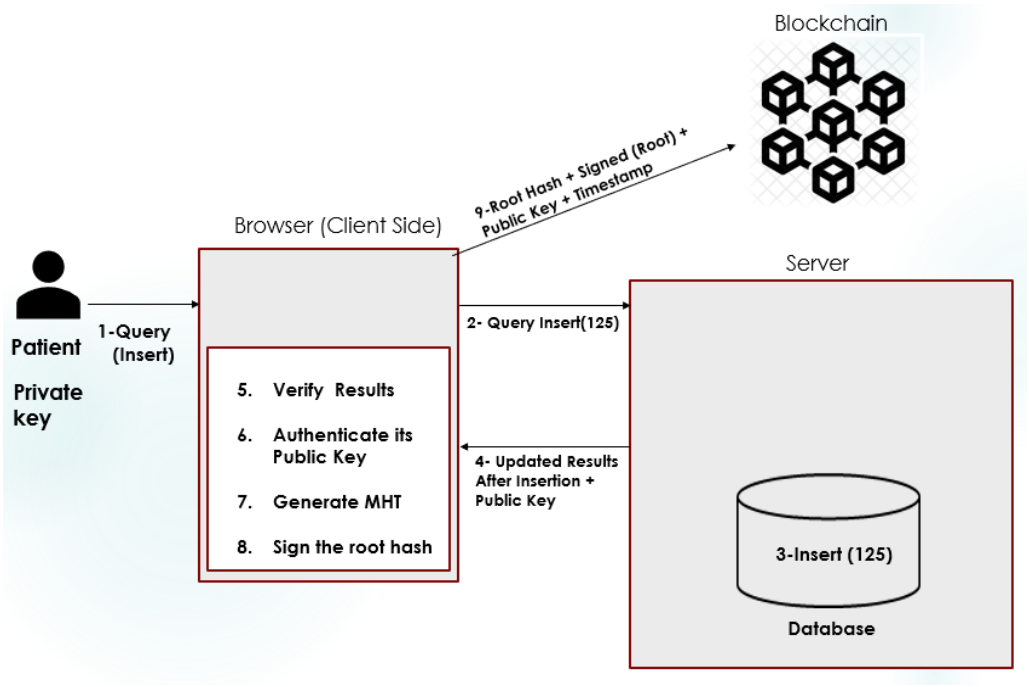

Figure 3-7 Data Flow diagram for patient

3.7 Data Flow for Data owner/Patient:

We will explain our solution with the help of a data flow diagram Figure 3-7. When a new patient registers/signup to medical application his public-private key pair gets generated. The public key is stored in a database along with his other credentials, while private key is stored at his local machine. The patient will later use this key pair for digital signature.

Pseudo code: Patient operations on server-side

```
1:  Run the Insert() query to save the patients data in database
2:  patient_records = Run find() query to fetch the patient's all records
3:  pub_key= Run find() query on 'public_key' table to fetch the patient's
    Public key
4:  return  patient_records and  pub_key
```

Figure 3-8 Pseudo code of patient operations on server-side

After registration to an application, when a patient inserts his diagnostic data like heart rate, it is forwarded to server by client and is inserted into database by server. Pseudo code in Figure 3-8 Pseudo code of patient operations on server-side provide better understanding of the process. The server then sends all patient record back to the client along with patient's public key. Patient on client side verifies that the server has inserted his readings correctly. Patient also authenticates his public key sent by the server. After verification of public key, application client then generates the Merkle Hash Tree (MHT) on readings sent by the server. The patient signs the root of MHT. The signatures and root then pushed on the blockchain along with a timestamp, and a public key of the patient as shown in Figure 3-12. It then gets distributed to every node on blockchain. This flow repeats every time a patient will insert, update or delete his reading on medical application.

Pseudo code: Patient operations on client-side

```
1:  Display/Print 'patient_records' and 'pub_key' to patient on web page
2:  Print prompt "If displayed data is correct click 'Yes' otherwise click 'No'"
3:      If patient clicks 'Yes' then
4:          MHT_root = Generate MHT on results
5:          Sign_data = Sign the MHT_root+public_key+timestamp
6:          Make an API call to push 'Sign_data', MHT_root, public_key and
            timestamp on Blockchain
7:      else stop the process
```

Figure 3-9 Pseudo code of patient operations on client-side

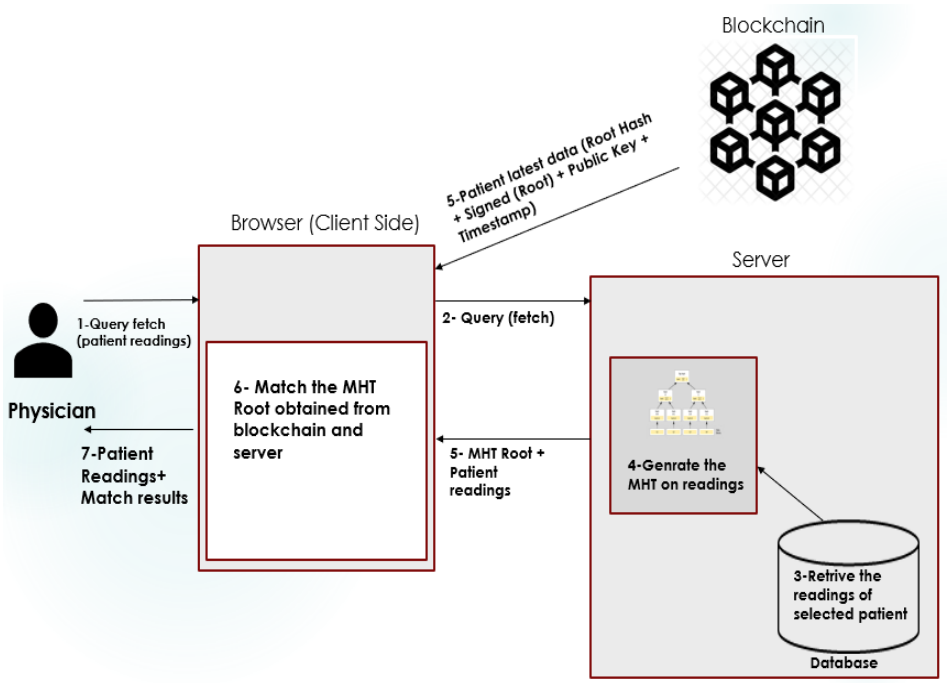

Figure 3-10 Data flow diagram for physician

3.8 Data Flow for Decision Maker / Physician:

Now in Figure 3-10 when physician login to the application, he fetches the diagnostic readings of a specific patient. The browser forwards his request to server. Server retrieves the readings from database and produce an MHT on it Figure 3-11.

Pseudo code: Physician operations on server-side

```
1:  patient_records = Run find() query to fetch complete record
    the selected patient.
2:  MHT_root_Server= Compute MHT on 'patient_records'
3:  return  patient_records and MHT_root_Server
```

Figure 3-11 Pseudo code of physician operations on server-side

Now server sends the requested readings of patient along with MHT root to the application client. Application client meanwhile also fetches the latest record (MHT root, signature, public Key and time stamp) of patient from blockchain. App Client then

validates the signatures of MHT root by using public key of patient fetched from blockchain. After that app client compares the both MHT Roots i.e. the one obtained from server and the other that it fetched from blockchain. If both roots do not match it means that the data provided by server has been illegally tampered, as shown in pseudo code Figure 3-12.

Pseudo code: Physician operations on client-side

```
1:   API call to Get the selected patient's data ('Sign_data' i.e. Signature, MHT_root,
     public_key and  timestamp) from Blockchain
2:   Verify Signature i.e. 'Sign_data' using MHT_root, timestamp and public_key
3:   If Signature verification is 'true' then
4:      Display prompt "Signature Verified Successfully'"
5:     If MHT_root (from Blockchain) is equal MHT_root_Server
6:        Display prompt "Data Verification is Successful'"
7:     else  Display prompt "Data Verification Failed'"
8:   else Display prompt "Signature Verification Failed'"
```

Figure 3-12 Pseudo code of physician operations on client-side

Now if the physician needs the data of patient for specific date/time as in Figure 3-13, he queries it to the server. Server retrievs the asked data from its database. Server then makes MHT over the patient's complete data. It then sends the asked record of patient for specific data/time along with MHT root and verification object (VO). As mentioned earlier in section 3.2, VO consists of MHT nodes that can assure the client that sent data belongs to corresponding MHT. App client asynchronously also sends the API GET request to blockchain to fetch the patients MHT root along with signatures, public key and time stamp. Now client has MHT Roots of patient's data both from server and from blockchain. App client now compare the both MHT roots and if match failed, this mean that data on server is tampered with. Now to make sure that the query result sent by server belongs to the sent MHT root, client verifies the proof sent by server. If proof validates successfully it means the records sent by server for specific date/time are

correct and genuine. Else our solution alert the user/decision maker that the data on server has been tampered.

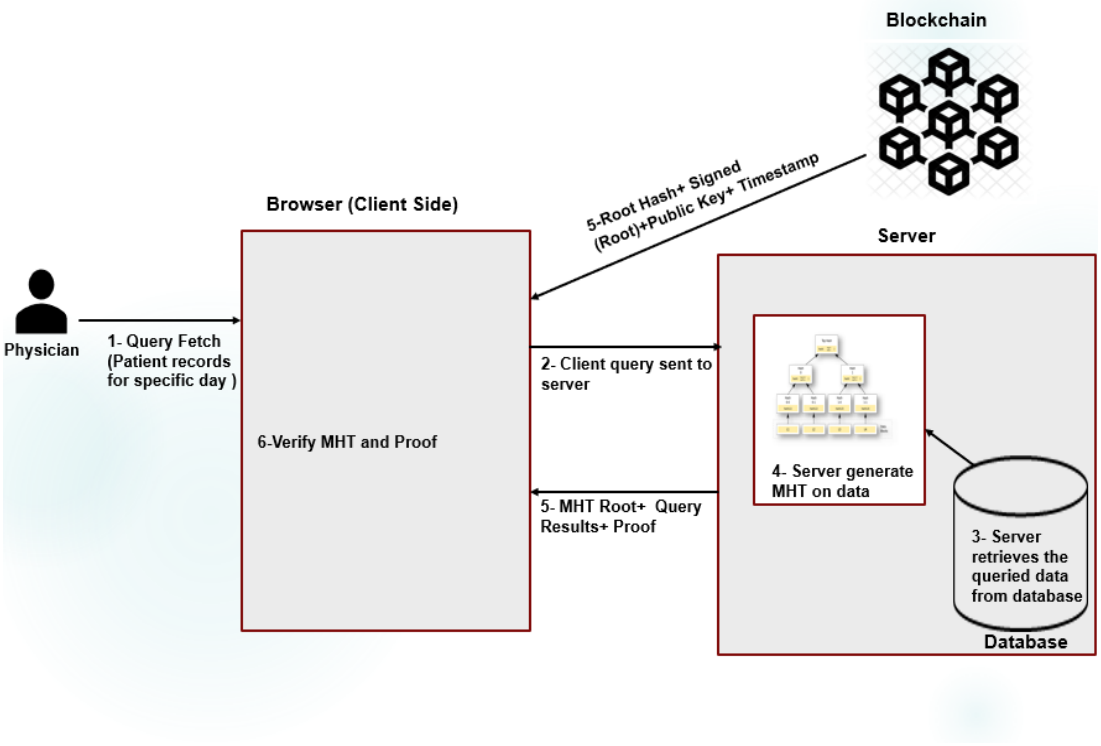

Figure 3-13 Data flow diagram for physician when query single record

3.9 Integrity Guarantee

Now the question is "How this solution protects the integrity of data against a compromised web server?".

As each time patient or data owner saves his reading, he also computes an MHT over his saved data and pushes its root over the blockchain. So, if an attacker having control over web server tampers the patient data, he would also need to change the root over blockchain illegally, which is impossible due to tamper resistant property of blockchain. If attacker only changes the data on web server, physician will get notified as MHT root stored on blockchain will not match with MHT root from server. In this way patient (decision maker) will get correct record from server. We have used timestamp to ensure the freshness of data. Time gets pushed on blockchain each time when data owner or

patient save new record on server. So, timestamp keeps record that when the owner changed his data. In this way blockchain provides the protection against fork attacks [41][42] and allows the client to check that if server has not sent any old root hash and

Solution also supports multiple data owners, as multiple patients can upload their diagnostic data on web application. Patient will compute MHT over its own data and will send its root to blockchain. Now when physician will access the data for a specific patient. Client of the application will fetch the data respective to the chosen patient from blockchain. Server sever will compute a separate MHT on the data of chosen patient.

4 Implementation

In this chapter we will discuss the implementation details and technologies that we used in our solution. First, we will discuss the implementation details of web application and then we will describe the implementation of blockchain for our solution.

4.1 Web Application

As we are dealing with integrity protection for a web application, so it is required to implement the solution over a functioning web application. The application should have following functionalities

- It must allow a data owner to insert, update and delete his diagnostic data.
- The application must be providing an interface to the decision maker, to view the data uploaded by the owner.
- The application should allow multiple data owners to store their data.

By keeping all these functionalities in mind, we developed a remote medical web application according to use case Figure 4-1. Using this application, a patient uploads his diagnostic data, and physician remotely accesses the patient's data to make a decision regarding patient's medication. In web application page gets assembled at the client side of an application with data coming from server and static code like HTML, CSS, JS, etc. Earlier work has been done to ensure the integrity of static code. Mylar [43] check's integrity of static code in client's browser against signature from the developer.

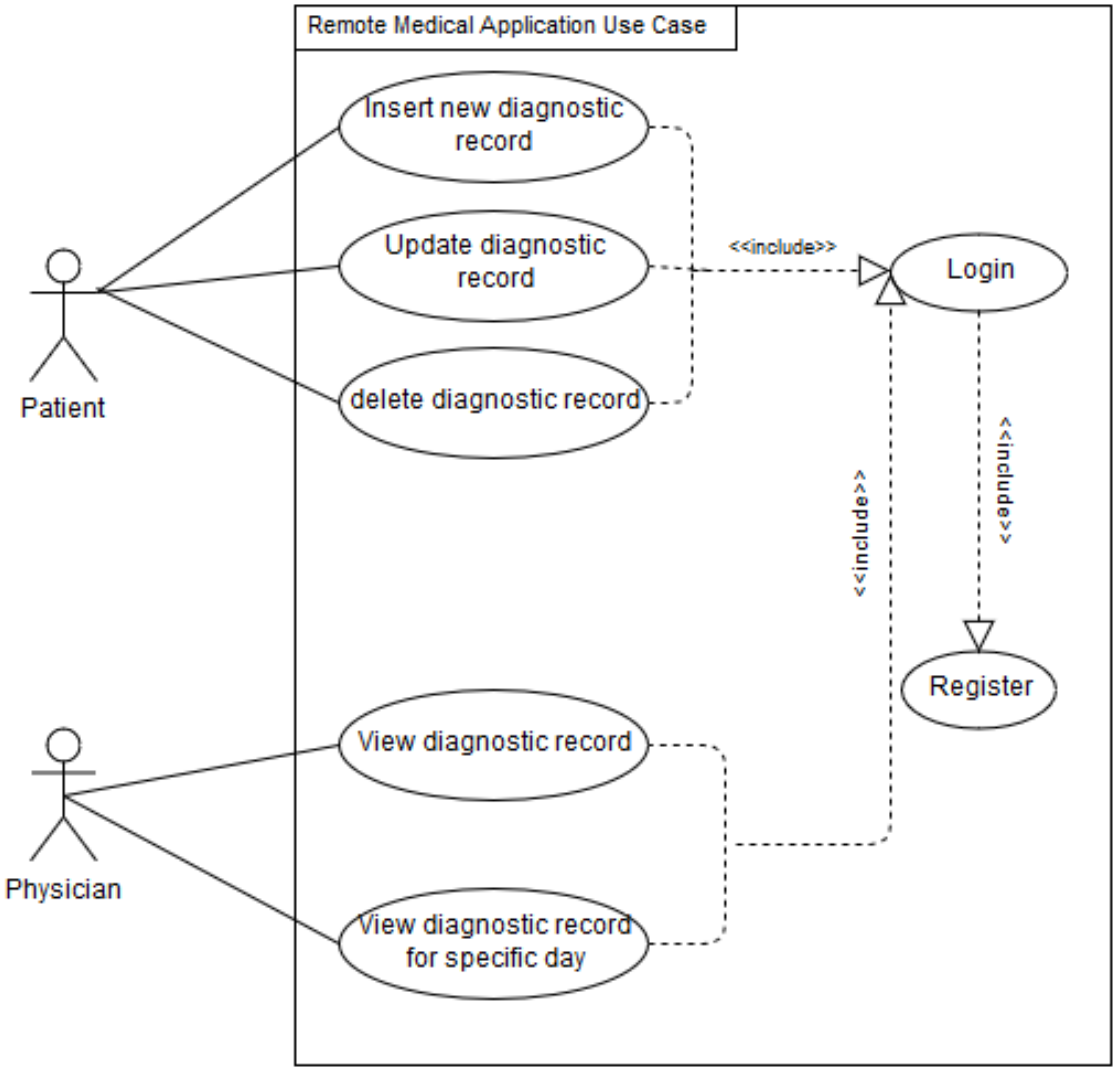

Figure 4-1 Use Case Diagram of remote medical web application

4.2 Basic Work Flow of Application for Patient (Data Owner)

For the very first time, a patient needs to register to medical application. While registering he needs to provide his email, password and the disease group (Heart patient, BP patient, etc.). After registration when patient login to the application, a page appears. This page has all previous reading of patient (if any) and it also allows the patient to enter his new diagnostic readings like BP Value or Heart rate etc. respective to his disease. As soon as the patient enters his readings, application client sends it to a server

which inserts it to a database and sends back an updated record to application's client. The client side of application then displays patient's newly added readings along with all his previous readings in the form of a table.

4.3 Basic Work Flow of Application for Physician (Decision maker)

The physician also needs to register himself when he accesses the application for the first time. On registration, he needs to enter his email, password and the disease group which he is specialized in. After registration when physician login to an application a page appears which shows him, different disease groups. Physician can choose any group and can see the diagnostic data of patients of those group. But, he can only give remarks to the patients of his specialized group. On choosing his specialized disease group let's say "Heart Disease Group", he will see the diagnostic records of patients of that heart patients. Now when the physician will enter the ID of the specific patient, Client of application will forward the request to the server. Server will fetch the data (diagnostic records) of that patient and will serve the asked data it back to application-client.

4.4 Web Application Implementation

We implemented the above mentioned remote medical web application in MeteorJS [44]. Meteor is a JavaScript [45] based opensource web application development framework. It is written in node.js. We used BlazeJs [46] to produce HTML templates. Following are some reasons that we choose Meteor for our implementation.

- Meteor provides fast development with JavaScript on client and server side.
- It also provides a clear separation between client and server-side code.
- Meteor render the page on client-side, it uses the HTML templates which get filled with data retrieved from a server. This provides a clear separation between client-side and server-side data.

To manage the backend database of our application we used MongoDB [47]. It is opensource, NoSQL type database. MongoDB provides high flexibility to manage the

database. It stores the data in BSON (a binary record format like JSON) and this allows variation of fields from document to document (each record). Following Table-4.1 is the list of collections (table) and their fields for our remote medical web application.

Table 4-1 Collections in Database

Collections	**Fields in documents**
Users	userID, email, password, groupID, account_type, creation date
Disease_Groups	groupID, group_Name
Records	recordID, userID, readings, timestamp, remarks

'Users' collection saves the userID, along with email, password, groupID (Disease Group ID), account_type (patient/physician), and account creation date. Similarly, 'Disease_Groups' save different disease groups like Heart Patient, Sugar Patients, Blood pressure etc. 'Records' collection saves the diagnostic records of different patients.

4.5 Solution Implementation

After the development of remote medical web application, we implemented our solution for end-to-end integrity protection over it. For solution implementation, we need to add solution components i.e. Merkle Hash Tree, Blockchain, and Digital Signature into our existing application.

4.5.1 Blockchain Implementation

As we discussed earlier in the previous chapter, we used Hyperledger Fabric a private blockchain for our solution. For the fast and simple implementation of fabric, we used Hyperledger Composer [48]. Hyperledger Composer provides a set of tools which helps to produce the proof of concept implementation easier and faster. It provides a rest

server to handle the “GET” and “POST” API calls from application, to post and retrieve the data from blockchain. This Website [49] provides a detailed installation guide for Hyperledger Composer development environment.

4.5.2 Digital Signatures Implementation

Public and Private key pairs of patient or data owner gets generated by OpenSSL [50] when he registers himself on an application. Following are the commands we used to generate public/private key pairs using OpenSSL

- openssl genrsa -out rsa_1024_priv.pem 1024
- openssl rsa -in rsa_1024_priv.pem -out rsa_1024_pub.pem -outform PEM -pubout

As discussed earlier; in our solution, patient signs his diagnostic data on client side using his private key. So, we used “jsrsasign” [51] a JavaScript based library, as JavaScript can run on client-side/browser. We used SHA-256 for hashing and RSA 1024 for encryption and decryption of data. Following is the code, that explains use of “jsrasign” library for signature generation.

1. *var rs = require('jsrsasign'); // loads ‘jsrsasign’ library to your code*
2. *var sign = new rs.Signature ({alg: 'SHA2withRSA'});// Object initialization*
3. *sign.init(private key); //initialization of private key*
4. *sign.updateString('data’); //updates the data to be signed*
5. *var signatures = sig.sign(); //returns hexadecimal string of signatures*

Following is the codes which enable the verification of signatures using actual data, signatures, and public key of signer:

1. *sig.init(public key); // initialization of public key of signer*
2. *sig.updateString('data’); // actual data that was signed by signer*
3. *var isValid = sig.verify(signatures) //return true or false after validating signatures*

So if ‘isValid’ is true means signatures validates successfully, else validation failed.

4.5.3 Merkle Hash Tree Implementation

To generate MHT over the data we have used a JavaScript based library "merkle-tools" [52]. It helps in generation of MHT, generates a proof, and helps to verify the proof. We have used the following functions of merkle-tools library to generate MHT.

- **addLeaf (value, doHash):** As we discussed earlier in Section 3.3 MHT contains the hash of data at its leaf nodes. addLeaf () function of merkle-tools library adds that hash of data to tree leaf e.g. merkleTools.addLeaves("hash-of-data"). If data is not hashed already then addLeaf () will take the hash of it by setting it's "doHash" argument 'true' and the "value" will the actual data e.g. merkleTools.addLeaves("a", true).
- **makeTree (doubleHash):** This function creates the MH tree for added leaf nodes. 'doubleHash' can set 'true', if it is required to take twice the hash for extra security, otherwise it can set to 'false'
- **getMerkleRoot ():** Returns the root hash of Merkle Hash Tree.
- **getProof ():** Provides the proof for selected leaf node of MH tree. For example, if required is the proof of 3rd leaf node in the tree, merkleTools.getProof (2) will provide the proof for 3rd leaf node as index starts from '0'.
- **validateProof (proof, targetHash, merkleRoot):** Used by the client-side of our web application to validates the provided proof using target hash i.e. the hash of leaf node whose proof is sent by the server, and the root node of MHT.

We also tried another NodeJS library "merkletree" for the generation of MHT. Using "mekletree" it is required to calculate the hash of data before adding it to the tree. To calculate the hash of data one can, use "Crypto" class of NodeJs. However, "merkletree" is slower than "merkle-tool". For 1000 values "merkletree" takes 288 milliseconds to compute the root of the tree, while "merkle-tool" does the same operation in 196 milliseconds.

4.5.4 Demo of Solution

Now we will show the demo of our solution on a remote medical web application.

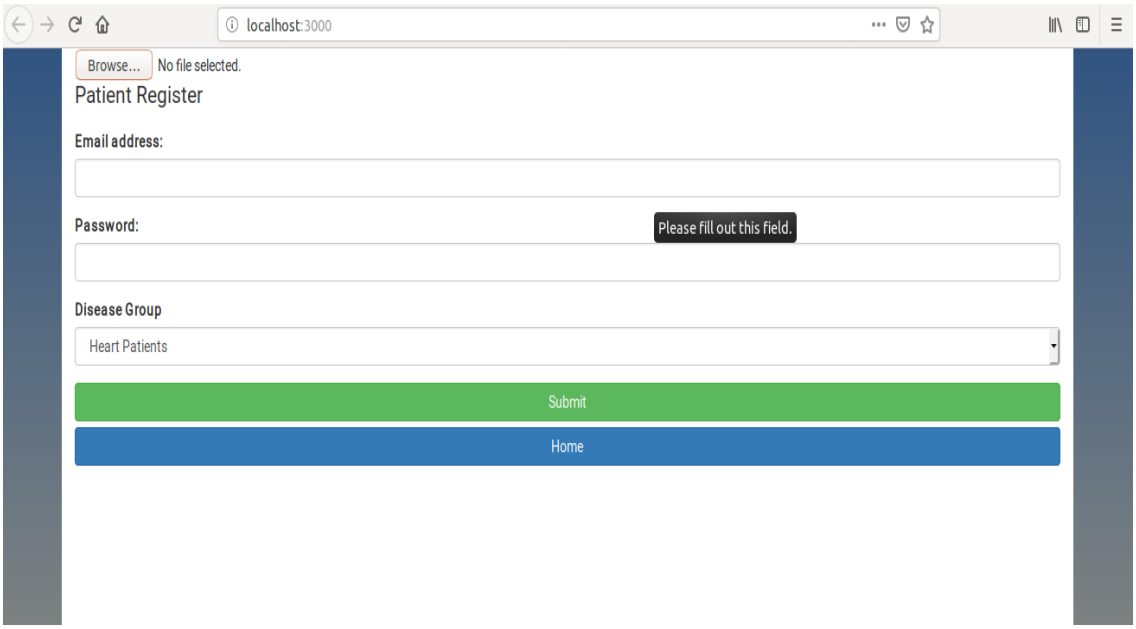

Figure 4-2 Registration Panel for Patient

Figure 4.2 displays the patient registration screen. The patient will add his public key by browsing it from its local storage. Then he will enter his email, password and will select his disease group when he will click "Submit" he will get registered to an application.

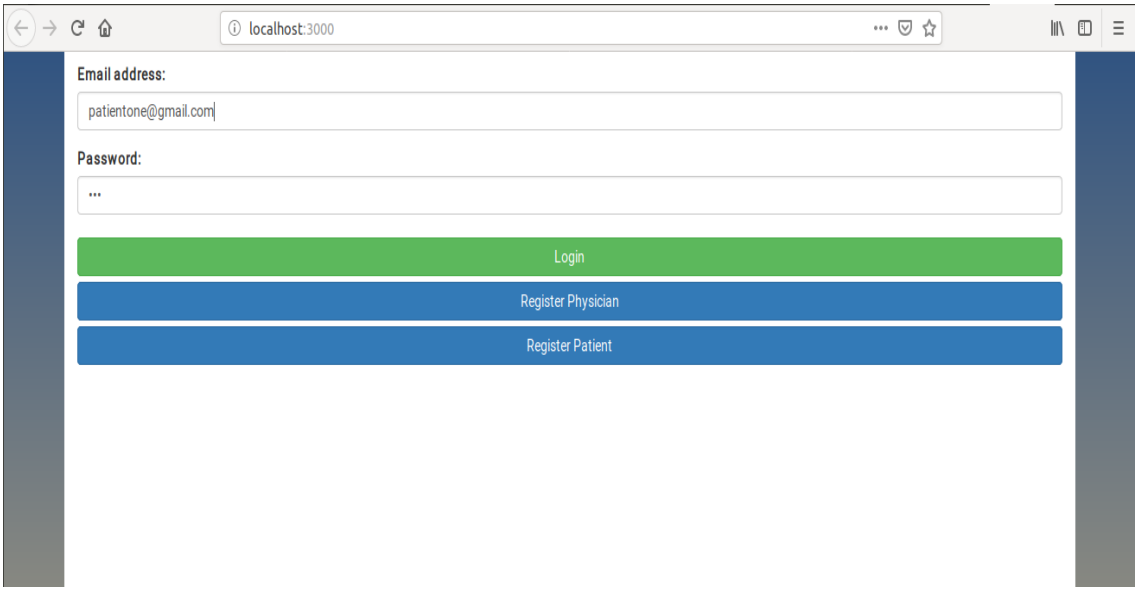

Figure 4-3 Application Login Panel

After registration, Figure-4.3 he will be able to login the application by providing his credentials i.e. Email address and password.

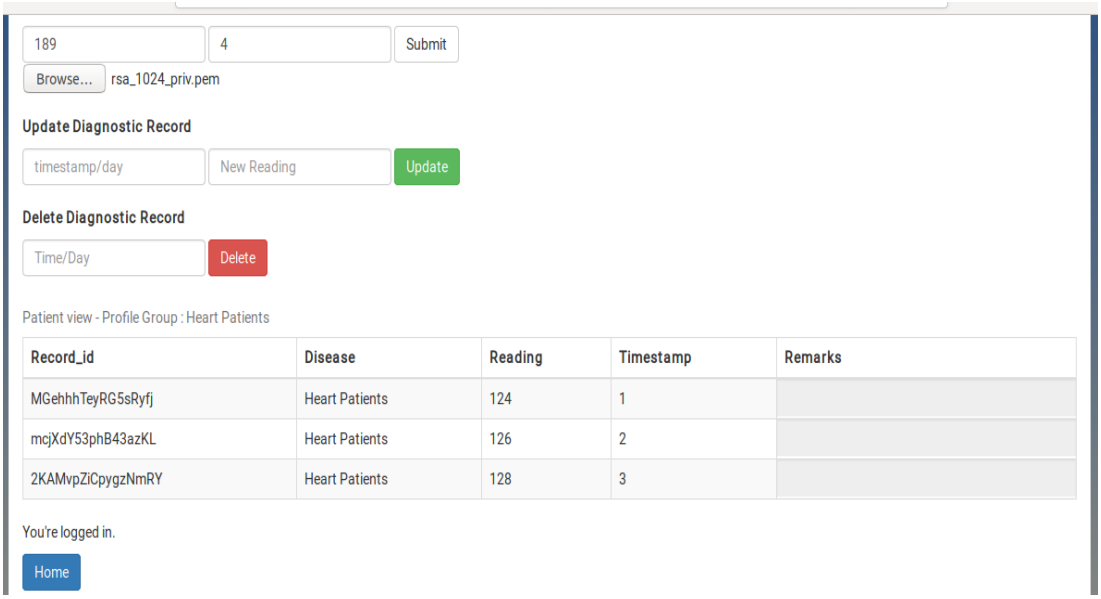

Figure 4-4 Patient's record insert panel

Figure-4.4 displays the application after the patient's login. He will enter his diagnostic data i.e. heart rate etc. like 124 in the above figure. After adding timestamp and his private key when he clicks "submit". Data is sent to the server and added to the database. Server then sends back newly added readings along with his previous readings (if any) to application-client. The client displays the readings to patient and signs the data with

patients provided private key and sends it to the blockchain

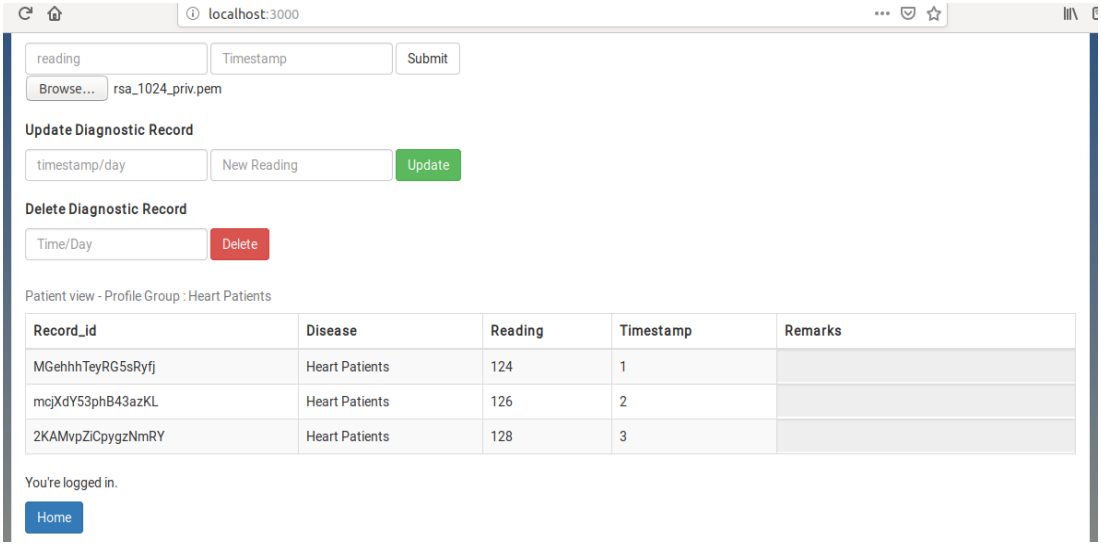

Figure 4-5 Patient's record insert panel

along with timestamp and public key of patient as shown in Figure-4.5 . Same is the process for Update and Delete.

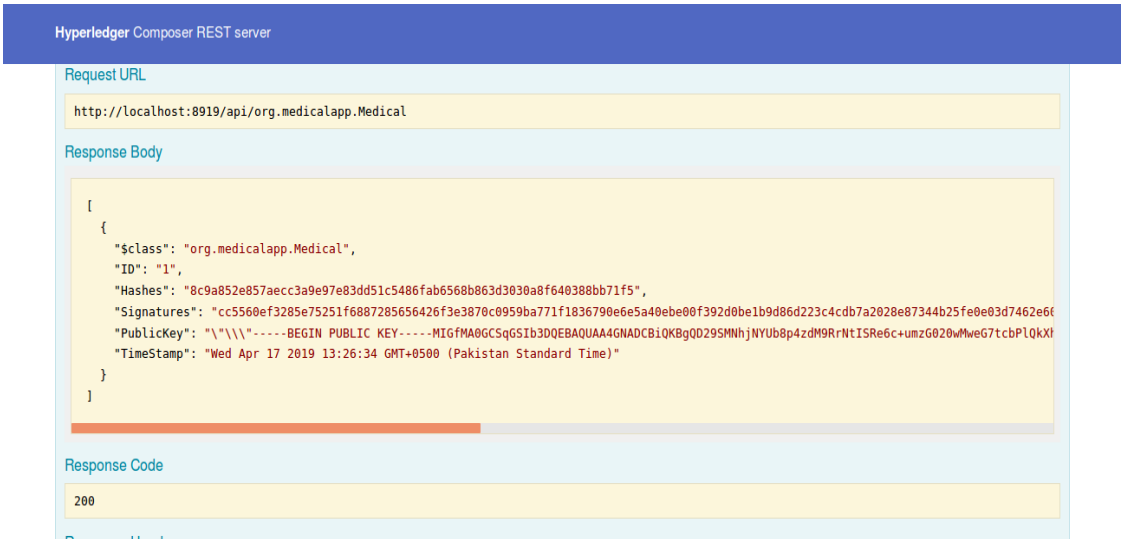

Figure 4-6 Hyperledger Composer REST Server

Figure-4.6 displays the data stored blockchain, sent by application-client. Blockchain distributes this data i.e. hash, signatures, PublicKey, and timestamp to all its peers.

Figure 4-7 Physician Registration Screen

Physician registers himself for the first time on application by filling the registration form as in Figure-4.7.

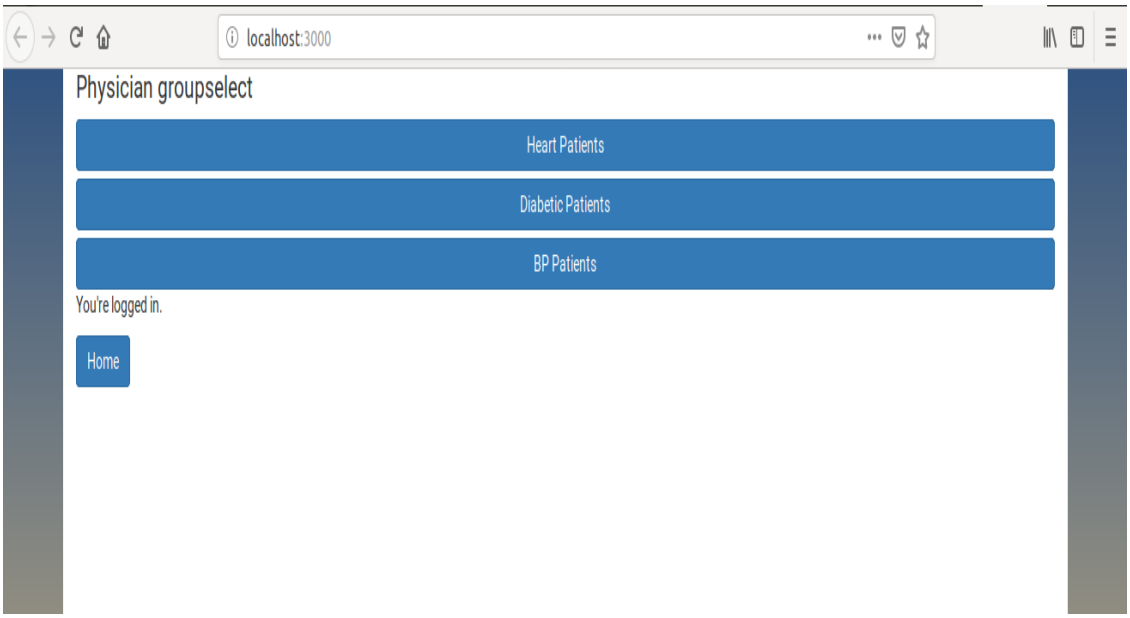

Figure 4-8 Physician Group Select Screen

After registration when physician logins, as in Figure-4.8 he needs to select the disease group which he is specialized in.

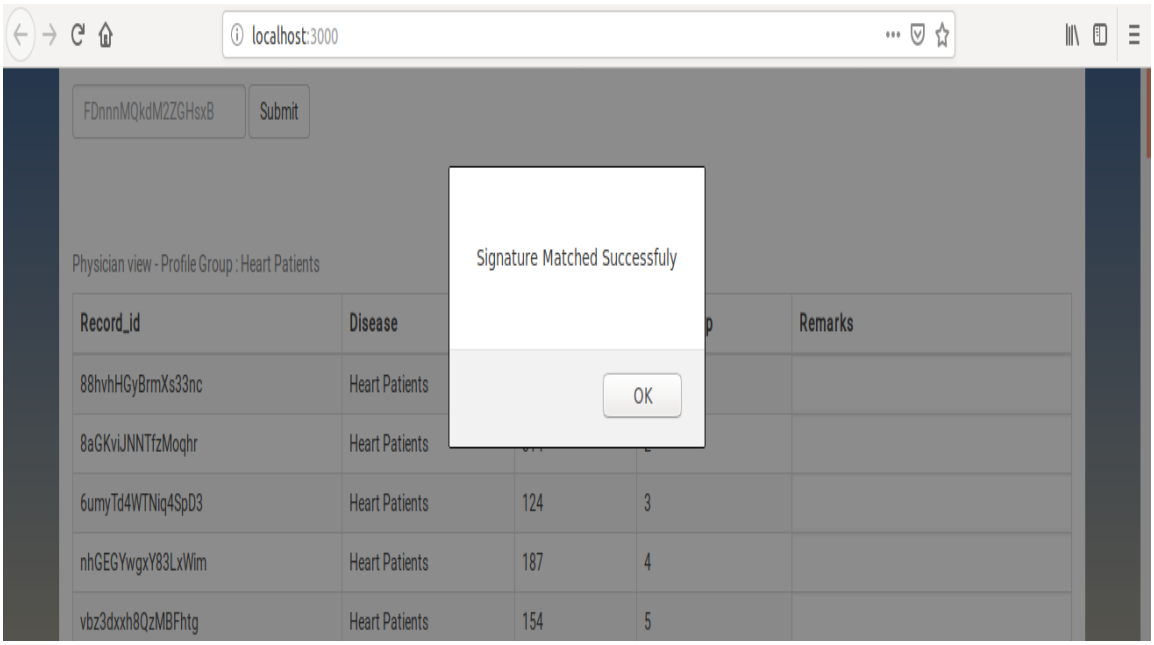

Figure 4-9 Physician data view panel

After selecting the disease group, a page appears Figure-4.9, where he needs to enter the specific patient's id. On clicking the "Submit" button, application-client requests the diagnostic records of the specific patient from application server. Meanwhile, client also fetches the patients latest uploaded data i.e. hash, signatures, Public Key, and timestamp from blockchain. Server on receiving the client's request, fetches the data from the stored database, builds an MHT over it and sends the root of MHT along with data and public key of patient back to client. Client verifies the signature of data received from blockchain using public key and displays verification result to the physician.

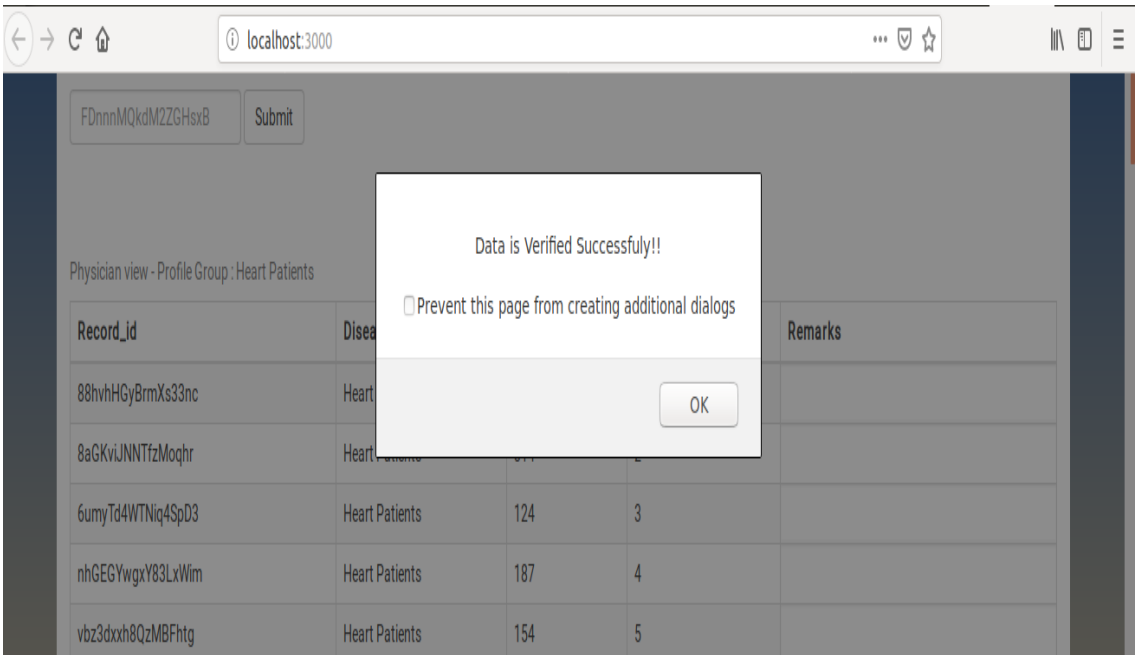

Figure 4-10 Data Verification Panel

After signature verification, application-client then matches the root obtained from fabric blockchain with the root obtained from application server, to ensure the integrity of data. And displays the results back to the physician as in Figure-4.10. If both roots match it shows “Data is verified successfully” else it shows “Data Verification Failed”. Means someone has tampered the data illegally.

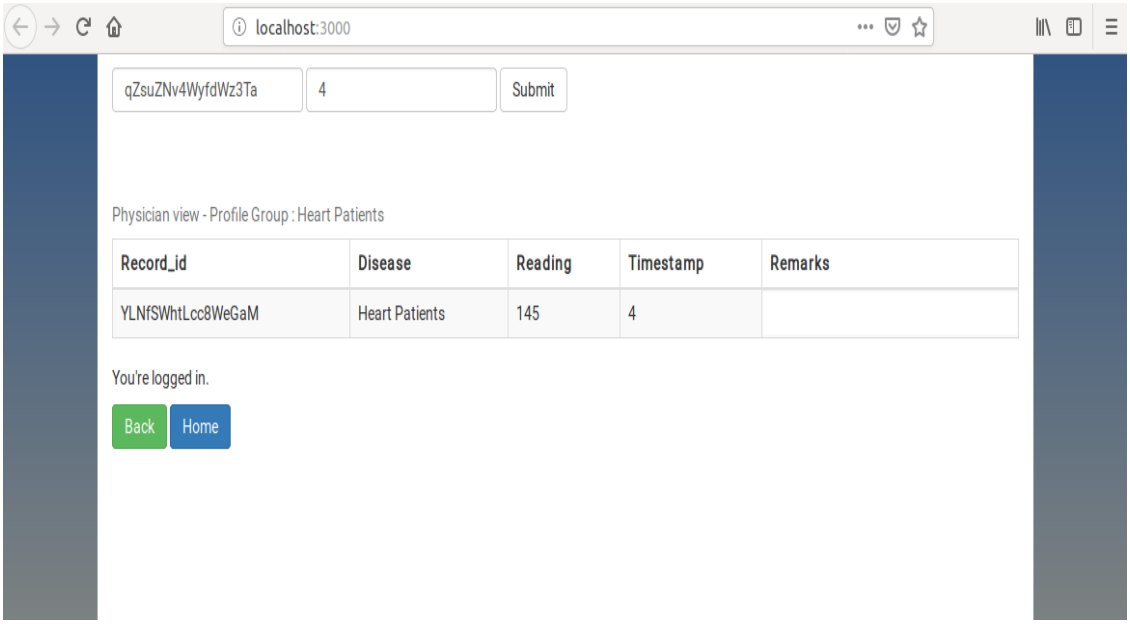

Figure 4-11 Physician Interface for query

When the physician needs the diagnostic record of a patient for a specific time/date. He simply enters the patient ID and timestamp and clicks the submit button as in Figure-4.11. When 'Submit' button gets pressed client of web application sends an asynchronous API Get request to Hyperledger Composer (blockchain) rest server. Meanwhile client also forwards the query request to web server. Web server retrieves the asked data from its database. The medical application server will then make MHT over the patient's complete data. It will then sends the asked record of a patient for specific date/time along with MHT root and verification object. Meanwhile, app-client will also receive root hash, signatures, Public Key, and timestamp from fabric blockchain. Now client will verify signer signatures using public key. Then client will match the MHT root hash from block chain with MHT root hash from server to validate the data. After successful validation of data client will use the verification object, MHT root hash to verify the record sent by server. This will be done using 'validateProof ()' function of 'merkle-tools' library. Results will then be shown to user as in figure.

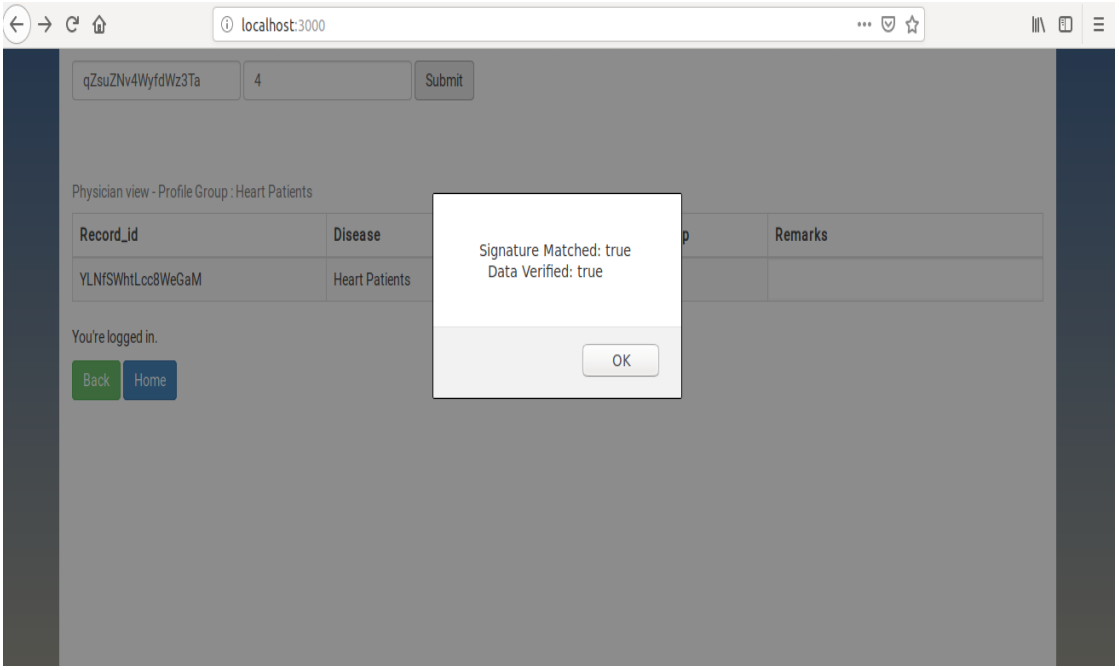

Figure 4-12 Integrity Check Alert

5TH CHAPTER

5 Performance Analysis

In this chapter, we will discuss the results of our solution with respect to our objectives discussed in Section 1.4 that we have made. We will also discuss how our solution caters the deficiencies of previously published solutions as discussed in Section 2.3.

Following were some objectives regarding to our solution for integrity protection:

1- Our solution will ensure the integrity of web application against the compromised web server.
2- The solution will ensure the integrity of data on a web application server even if it belongs to multiple data owners.
3- Proposed solution will support practical applications with minimum changes in existing structure of application.
4- Solution will add moderate overhead when deployed over an existing web application.

As per first objective our solution should guarantee the integrity of data even if web server is completely in control of an attacker. An attacker can change the data on server but cannot change the root hash of patient's data (diagnostic readings and time/day) on blockchain. As data on blockchain gets distributed to every peer on network, so it's almost impossible for an attacker to change the data on blockchain.

Our solution also fulfills objective 2. It solves the integrity problem of multiple data owners for data on web server. It builds separate MHT for each data owner and stores its root on blockchain with his owners id.

The solution also supports the practical web application. As discussed above we deployed our solution on a remote medical web application with minimum changes in the existing structure of an application as discussed in Section 4.5 . We have evaluated our solution and analyzed its cost on bases of three parameters i.e. memory, monetarily, and performance.

5.1 Memory Cost Analysis:

The proposed solution requires very few any changes in existing schema of a web application database. As in purposed solution each data owner has public/private key pair for digital signatures. So, we need a collection/table to store the public keys of respective data owners (patients) in the database.

Table 5.1 Public Key Collection in Database

Collections	**Fields in documents**
public_key	userID, publickey

“public_key” collection (‘Table’ in SQL) Table-5.1 need to be added in database to store the Public key of patient or data owner. “public_key” collection has two fields “userID” and “publickey”. “userID” stores the unique identification string and “publickey” stores the Public key of patient.

Using “db.collection.stats()”[53]we can get the size of a document in a mongo collection. For each document (‘row’ in SQL) in “public_key” collection, it takes 0.346 KB or 0.000346 MB extra space in memory. So, for 1 Million users it will take 346MB of extra space in existing web application. Figure-5.1 shows the relations of no. of documents in ‘public_key’ collection with increase memory.

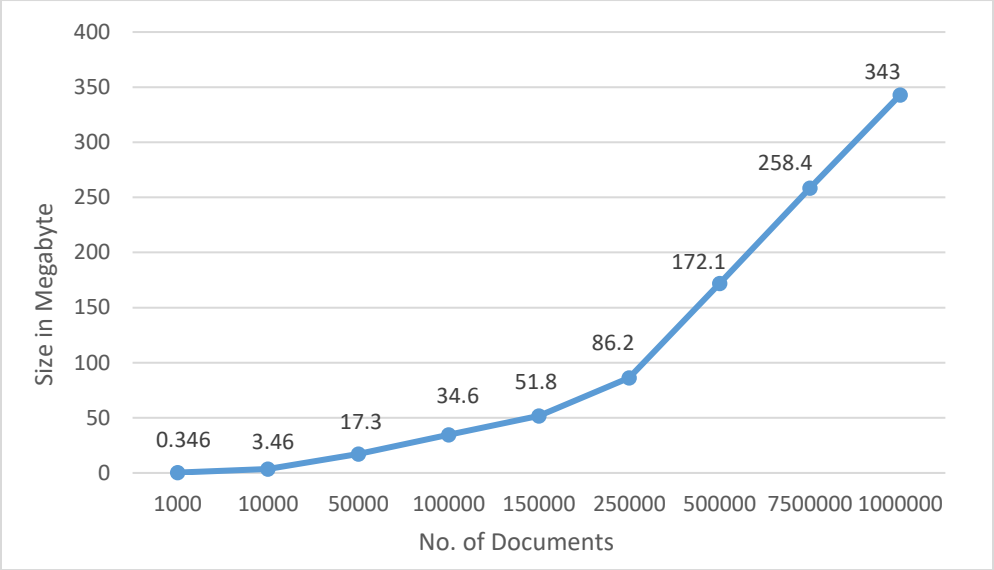

Figure 5-1 Documents to Size Relation for ‘public_key’ Table

5.2 Monetary Cost Analysis:

The purposed solution does not add any financial cost to an existing system. All the building blocks like ‘merkle-tools’ (MHT Library), ‘jsrsasign’ (Digital Signature Library), OpenSSL (public/private key pair generator), Hyperledger Fabric (Blockchain) that have been used to construct this solution are totally opensource and easily available.

5.3 Performance Analysis:

To evaluate the performance of our solution, we hosted the web application locally on a Core i3 system with 4 GB of RAM. Then analyzed the performance of the web application with integrity protection solution and without integrity protection solution using “Performance.now ( )” [54] function of JavaScript. It returns the code execution

time in milliseconds. Figure-5.2 shows the comparison results for Insert, Update, Delete, and View Operations.

When a patient or data owner inserts a single diagnostic data on remote medical web application integrated with our solution it takes 93 milliseconds and application without integrity solution takes 18 milliseconds to complete this operation. Similarly, for update operation integrity protected web application takes 59 millisecond and application without integrity solution takes 23 milliseconds. For delete operations application with integrity protection takes 57 milliseconds and without integrity protection it takes 14 milliseconds to complete the operation. This shows that our integrity protection solution adds 54 % overhead for insert operation , 36 % overhead for update operation and 38 % overhead for delete operation.

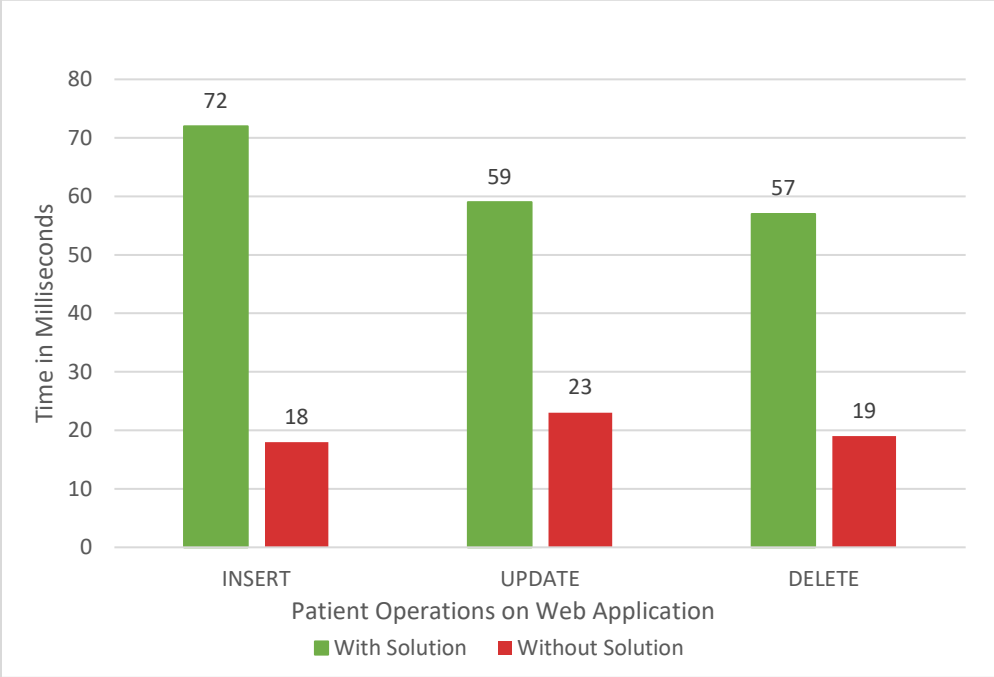

Figure 5-2 Performance Comparison of application with and without Integrity Solution

As explained earlier in Section 3.6 that, in our solution when patient insert/upload a diagnostic data on server, server after adding the data in database sends all the records of the respective patient back to app-client. So, patient on the client side can check if his data has been inserted in a database by the server, and all his previous records are

correct. After verification, patient's client side computes the MHT over diagnostic data and sends it blockchain.

As MHT is computed at app-client, so we evaluated the computation cost of an MHT with respect to the number of patient's diagnostic records on a Core i3 system with 4 GB of RAM. For 10 thousand records of one patient, "merkle-tools" takes 0.742 seconds to compute an MHT root and the graph grows linearly as shown in Figure-5.3.

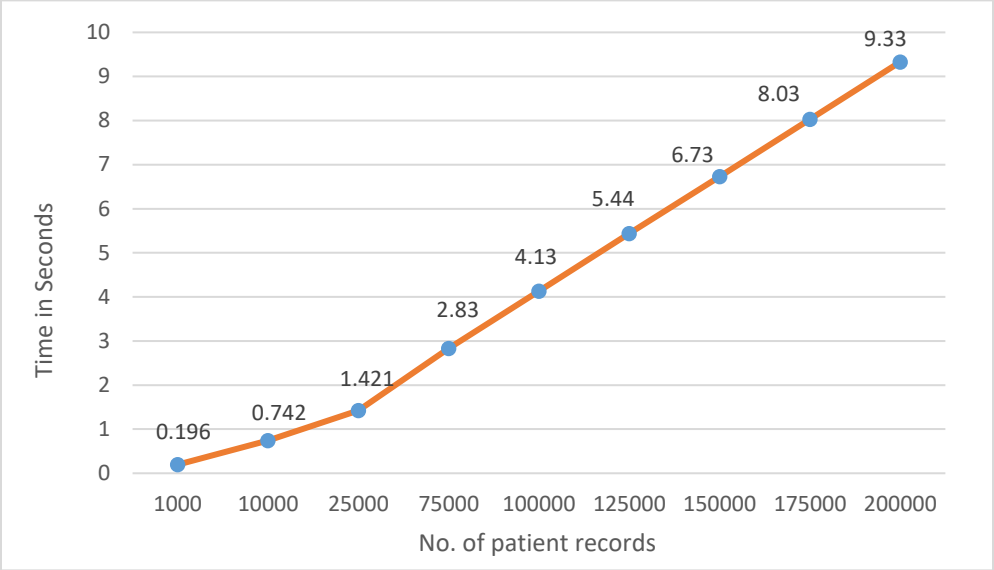

Figure 5-3 MHT time for no. of records with respect to time

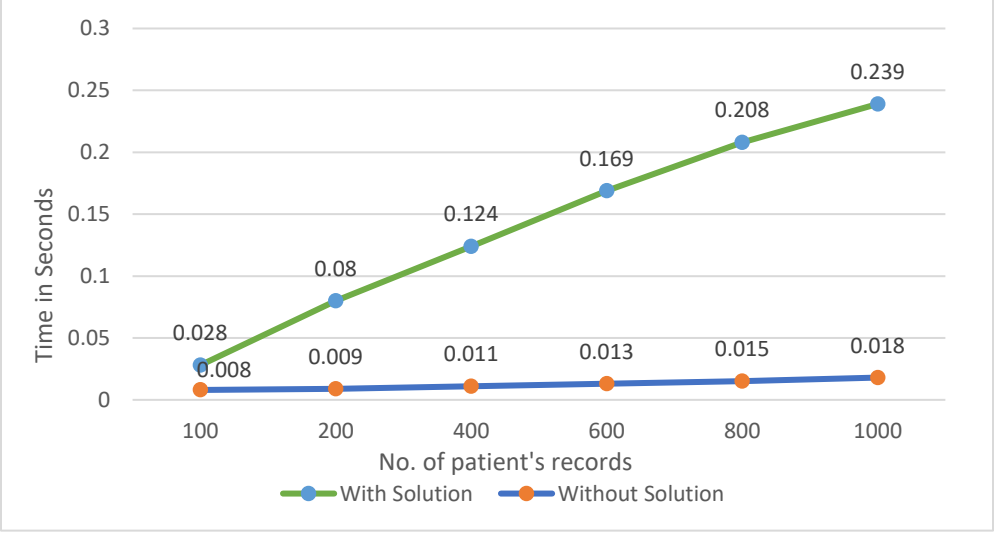

Figure 5-4 Physician view performance with and without integrity solution

When physician logins to the remote medical web application and access's the diagnostic record of a specific patient on remote medical web application. Server takes 0.008 seconds to retrieve 100 records of the asked patient when hosted locally on Corei3 machine with 4GB RAM. With our integrity protection solution when the same operation is carried out on 100 records it takes 0.028 seconds, as server needs to compute an MHT over data as shown in Figure-5.4

Verena [14] has used a vulnerable hash server for integrity protection while we have used blockchain. Although blockchain is slower than Verena's hash server but it is tamper resistant. Hash Server throughput for PUT requests is 2100 (±92) requests/second and for GET requests its throughput is 5890 (±548) requests/second. Hyperledger Fabric that runs at the base of Hyperledger Composer takes 0.09 seconds to execute a transaction. In our solution, an API call on blockchain on average takes 1.05 seconds [55] to respond for GET and POST requests. However, the response time is also dependent on internet connection speed. Our solution makes these GET and POST requests asynchronously, so it doesn't affect much on the overall performance of an application.

6 Conclusion and Future Work

Our solution completely protects the integrity of data of web application stored on web server. Blockchain technology builds the trust base in the proposed solution. It uses the blockchain to save the latest hash of data with the timestamp and public key of the data owner. When decision maker fetches the data from the server to make his decisions, our solution compares the server hash with blockchain hash to prove data integrity. The solution ensures strong integrity properties such as correctness, and freshness and supports practical applications with moderate overhead as explained in section 5.3.

6.1 Future Work:

Though the solution solves the core problem i.e. integrity protection of data on a remote web server. It enables the server to provide authentication proof against inert, update, delete and find quires. However, it doesn't cater the authentication proof against aggregated queries like SUM, Average, MAX, MIN, etc. For example, if the physician quires an average heart rate of a patient for a week, or physician quires the maximum heart rate over the month then our solution cannot provide the proof to the physician that it computed for the average or to find the maximum correctly. To upgrade the solution with the ability to deal with aggregated quires, we can use MHT [56][13] with the ability to support aggregated and projection quires.

7 References

[1] S. M. T. Berners-Lee, Dimitri Dimitroyannis, A. John Mallinckrodt, "World Wide Web," *Comput. Phys. 8*, p. 298, 1994.

[2] CERN, "A short history of the World Wide Web." [Online]. Available: https://home.cern/science/computing/birth-web/short-history-web. [Accessed: 26-Mar-2019].

[3] Internet Live Stats, "Latest Count of Web Application on Internet," 2019. [Online]. Available: https://www.internetlivestats.com/total-number-of-websites/. [Accessed: 25-Mar-2019].

[4] M. Burstein *et al.*, "A semantic Web services architecture," *IEEE Internet Comput.*, vol. 9, no. 5, pp. 72–81, 2005.

[5] Exposit, "Types of Web Applications." [Online]. Available: https://blog.exposit.com/what-are-types-web-applications/. [Accessed: 26-Mar-2019].

[6] Techopedia, "Static web application." [Online]. Available: https://www.techopedia.com/definition/5399/static-web-page. [Accessed: 03-Apr-2019].

[7] Computer Hope, "Dynamic Web Application," 2018. [Online]. Available: https://www.computerhope.com/jargon/d/dynasite.htm. [Accessed: 05-Apr-2019].

[8] Symantec, "Internet Security Threat Report 2019," 2019. [Online]. Available: https://www.symantec.com/security-center/threat-report. [Accessed: 06-Apr-2019].

[9] PositiveTechnologies, "Web Application Vulnerabilities Statistics for 2017,"

2017. [Online]. Available: https://www.ptsecurity.com/upload/corporate/ww-en/analytics/Web-application-vulnerabilities-2018-eng.pdf. [Accessed: 06-Apr-2019].

[10] Marketwired, “Osterman Research Study Finds Most Organizations Lack Necessary Visibility Into Data and Database Assets,” 2016. [Online]. Available: https://finance.yahoo.com/news/osterman-research-study-finds-most-121700890.html. [Accessed: 06-Apr-2019].

[11] M. Frellick, “Medical Error Is Third Leading Cause of Death in US,” 2016. [Online]. Available: https://www.medscape.com/viewarticle/862832. [Accessed: 06-Apr-2019].

[12] M. Fisher, “Associate Press Website got hacked,” 2013. [Online]. Available: https://www.washingtonpost.com/news/worldviews/wp/2013/04/23/syrian-hackers-claim-ap-hack-that-tipped-stock-market-by-136-billion-is-it-terrorism/?noredirect=on&utm_term=.696cce843ae4. [Accessed: 06-Apr-2019].

[13] H. Pang and K.-L. Tan, “Query Answer Authentication,” *Synth. Lect. Data Manag.*, vol. 4, no. 2, pp. 1–103, 2012.

[14] N. Karapanos, A. Filios, R. A. Popa, and S. Capkun, “Verena: End-to-End Integrity Protection for Web Applications,” *Proc. - 2016 IEEE Symp. Secur. Privacy, SP 2016*, pp. 895–913, 2016.

[15] G. Ateniese *et al.*, “Provable data possession at untrusted stores,” p. 598, 2007.

[16] J. Li, M. Krohn, D. Mazi, and D. Shasha, “Secure Untrusted Data Repository (SUNDR),” *Osdi*, vol. pp, pp. 121–136, 2004.

[17] Y. Deswarte and J. Quisquater, “Remote Integrity Checking,” *Integr. Intern. Control Inf. Syst. VI. IICIS 2003. IFIP Int. Fed. Inf. Process. vol 140. Springer, Boston, MA*, 2002.

[18] I. Zikratov, A. Kuzmin, V. Akimenko, V. Niculichev, and L. Yalansky, “Ensuring data integrity using blockchain technology,” *Conf. Open Innov. Assoc. Fruct*, vol. 2017-April, pp. 534–539, 2017.

[19] G. Becker, “Merkle Signature Schemes,” p. 28, 2008.

[20] R. Jain and S. Prabhakar, “Trustworthy data from untrusted databases,” *Proc. - Int. Conf. Data Eng.*, pp. 529–540, 2013.

[21] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine, “Authentic third-party data publication,” *Data Appl. Secur.*, pp. 101–112, 2002.

[22] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, “Authenticated index structures for outsourced databases,” *Handb. Database Secur. Appl. Trends*, pp. 115–136, 2008.

[23] Y. Zhang and J. Katz, “IntegriDB : Verifiable SQL for Outsourced Databases.”

[24] M. T. Goodrich, R. Tamassia, and A. Schwerin, “Implementation of an authenticated dictionary with skip lists and commutative hashing,” *Proc. - DARPA Inf. Surviv. Conf. Expo. II, DISCEX 2001*, vol. 2, pp. 68–82, 2001.

[25] M. Narasimha and G. Tsudik, “Authentication of outsourced databases using signature aggregation and chaining,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 3882 LNCS, pp. 420–436, 2006.

[26] M. Narasimha and G. Tsudik, “DSAC: Integrity for Outsourced Databases with Signature Aggregation and Chaining,” in *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, 2005, pp. 235–236.

[27] M. Xie, H. Wang, and J. Yin, “Integrity auditing of outsourced data,” *Very large data bases*, pp. 782–793, 2007.

[28] P. a Hallgren, D. T. Mauritzson, and A. Sabelfeld, “GlassTube: A Lightweight Approach to Web Application Integrity,” *Proc. Eighth ACM SIGPLAN Work. Program. Lang. Anal. Secur.*, pp. 71–82, 2013.

[29] P. Fortuna, N. Pereira, and I. Butun, “A Framework for Web Application Integrity,” in *Proceedings of the 4th International Conference on Information*

Systems Security and Privacy, 2018, no. April, pp. 487–493.

[30] P. Rogaway and T. Shrimpton, "Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance," pp. 371–388, 2011.

[31] N. Pinckney, D. M. Harris, N. Jiang, K. Kelley, S. Antao, and L. Sousa, "Public key cryptography," *Circuits Syst. Secur. Priv.*, no. June, pp. 109–182, 2017.

[32] K. Balasubramanian, "Variants of RSA and their cryptanalysis," in *2014 International Conference on Communication and Network Technologies*, 2014, pp. 145–149.

[33] Y. Desmedt, "ElGamal Public Key Encryption BT - Encyclopedia of Cryptography and Security," H. C. A. van Tilborg and S. Jajodia, Eds. Boston, MA: Springer US, 2011, p. 396.

[34] S. Electronics Laboratories and R. C. Merkle, "I Nformat I on Systems Laboratory Secrecy, Authentication, and Public Key Systems," 1979.

[35] R. Kaur and A. Kaur, "Digital Signature," in *2012 International Conference on Computing Sciences*, 2012, pp. 295–301.

[36] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, "Blockchain," *Bus. Inf. Syst. Eng.*, vol. 59, no. 3, pp. 183–187, Jun. 2017.

[37] X. Xu *et al.*, "A Taxonomy of Blockchain-Based Systems for Architecture Design," *Proc. - 2017 IEEE Int. Conf. Softw. Archit. ICSA 2017*, pp. 243–252, 2017.

[38] I.-C. Lin and T.-C. Liao, "A survey of blockchain security issues and challenges," *Int. J. Netw. Secur.*, vol. 19, pp. 653–659, 2017.

[39] E. Androulaki *et al.*, "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains," 2018.

[40] IBM, "Transaction Flow of Hyperledger Fabric." [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/txflow.html. [Accessed:

06-Apr-2019].

[41] J. Li, M. Krohn, D. Mazières, D. Shasha, and D. Shasha, “Secure Untrusted Data Repository (SUNDR),” in *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6*, 2004, p. 9.

[42] S. Chong, K. Vikram, and A. Myers, “SIF: Enforcing confidentiality and integrity in web applications,” *USENIX Secur.*, pp. 1–16, 2007.

[43] R. A. Popa *et al.*, “Building Web Applications on Top of Encrypted Data Using Mylar,” *Usenix Nsdi*, pp. 157–172, 2014.

[44] Meteor, “Why Meteor?” [Online]. Available: https://www.meteor.com/. [Accessed: 06-Apr-2019].

[45] NodeJs, “About Node.js.” [Online]. Available: https://nodejs.org/en/about/. [Accessed: 26-Apr-2019].

[46] Blaze, “BlazeJS an overview.” [Online]. Available: http://blazejs.org/. [Accessed: 06-Apr-2019].

[47] MongoDB, “What is MongoDB?” [Online]. Available: https://www.mongodb.com/what-is-mongodb. [Accessed: 06-Apr-2019].

[48] IBM, “Welcome to Hyperledger Composer.” [Online]. Available: https://hyperledger.github.io/composer/latest/. [Accessed: 06-Apr-2019].

[49] Hyperledger Composer, “Installing Hyperledger Composer.” [Online]. Available: https://hyperledger.github.io/composer/latest/installing/installing-index.html. [Accessed: 06-Mar-2019].

[50] OpenSSL, “Welcome to OpenSSL.” [Online]. Available: https://www.openssl.org/. [Accessed: 06-Apr-2019].

[51] npm, “‘jsrsasign’ digital signature library,” 2018. [Online]. Available: https://www.npmjs.com/package/jsrsasign. [Accessed: 06-Apr-2019].

[52] npm, “‘merkle-tools’ Merkle Tree library of JS,” 2017. [Online]. Available: https://www.npmjs.com/package/merkle-tools. [Accessed: 09-May-2019].

[53] MongoDB, “Document size in mongodb.” [Online]. Available: https://docs.mongodb.com/manual/reference/method/db.collection.stats/. [Accessed: 08-May-2019].

[54] Mozilla, “performance.now() function of javascript,” 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Performance/now. [Accessed: 08-May-2019].

[55] Q. Nasir, I. A. Qasse, M. Abu Talib, and A. B. Nassif, “Performance Analysis of Hyperledger Fabric Platforms,” *Secur. Commun. Networks*, vol. 2018, pp. 1–14, 2018.

[56] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, “Authenticated Index Structures for Aggregation Queries,” *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 4, pp. 1–35, 2010.