JADAVPUR UNIVERSITY

MASTER DEGREE THESIS

Design of Secure Storage and Access for Cloud Based Data

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Technology in Distributed and Mobile Computing

in the

School of Mobile Computing and Communication

by
SONALI MANDAL
University Roll Number: 001730501008
Examination Roll Number: M4DMC19011
Registration Number: 141105 of 2017-2018

Under the Guidance of
Dr. SARMISTHA NEOGY
Department of Computer Science and Engineering
Faculty of Engineering and Technology
Jadavpur University
Kolkata-700032

May 24, 2019

Declaration of Authorship

I, Sonali Mandal, declare that this thesis titled, "Design of Secure Storage and Access for Cloud Based Data" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a masters degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

To whom it may concern

This is to certify that Sonali Mandal has satisfactorily completed the work in this thesis entitled "Design of Secure Storage and Access for Cloud Based Data", University Roll Number: 001730501008, Examination Roll Number: M4DMC19011, Registration Number: 141105 of 2017-2018. It is a bonafide piece of work carried out under my supervision at Jadavpur University, Kolkata-700032, for partial fulfillment of the requirements for the degree of Master of Technology in Distributed and Mobile Computing from the School of Mobile Computing and Communication, Jadavpur University for the academic session 2017-2019.

Dr. Sarmistha Neogy
Professor
Department of Computer Science & Engineering,
Jadavpur University
Kolkata-700032.

Director
School of Mobile Computing and Communication,
Jadavpur University
Kolkata-700032.

Prof. Pankaj Kumar Roy
Dean, Faculty of Interdisciplinary Studies, Law and Management
Jadavpur University
Kolkata-700032.

Certificate of Approval

(Only in case the thesis is approved)

This is to certify that the thesis entitled "Design of Secure Storage and Access for Cloud Based Data" is a bona-fide record of work carried out by Sonali mandal, University Roll Number: 001730501008, Examination Roll Number: M4DMC19011, Registration Number: 141105 of 2017-2018, in partial fulfilment of the requirements for the award of the degree of Master of Technology in Distributed and Mobile Computing from the School of Mobile Computing and Communication, Jadavpur University for the academic session 2017-2019. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein but approve the thesis only for the purpose for which it has been submitted.

_____

(Signature of the Examiner)
Date:

_____

(Signature of the Examiner)
Date:

Jadavpur University

Abstract

Faculty of Interdisciplinary Studies, Law and Management, Jadavpur University
School of Mobile Computing and Communication

Master of Technology in Distributed and Mobile Computing

Design of Secure Storage and Access for Cloud Based Data

by
SONALI MANDAL
University Roll Number: 001730501008
Examination Roll Number: M4DMC19011
Registration Number: 141105 of 2017-2018

With the increasing usage of wearable smart devices for health monitoring, providing health care facilities in remote areas are being developed. For an organization managing all these data from health-care devices and other patient information becomes difficult in a local database. The situation is similar for large hospitals with lots of departments as well. The cloud computing technologies can provide a low-cost and scalable solution to this huge volume of data. However, with the advantages of a cloud database, comes various security issues. Firstly, secure data transfer through a network is needed so that data cannot be stolen or tampered. Also data stored in the cloud database needs to be protected because if the cloud provider is untrusted or is attacked by an intruder, data confidentiality and integrity can be lost. Encrypting all the data stored in the cloud database provides the required level of security. However it produces several challenges in searching a encrypted database. This thesis describes the challenges and presents an approach to secure a Cassandra database used in an untrusted cloud environment. Specifically, the proposed model provides a secure interface to the data that is stored in a distributed database in cloud, by tackling threats of both internal and external attackers. It stores patient's information in encrypted form in the database and also modification-sensitive information is hashed and stored in blockchain. Thus, it provides seamless access to the encrypted data for permitted users while limiting the access for other users in the system, internal and external attackers; along with preventing unauthorized data modification. Storage of hashes allows the system to validate any information at any point of time, while access to the hash values is provided with a secure protocol similar to a blockchain.

Acknowledgements

On the submission of "Design of Secure Storage and Access for Cloud Based Data", I wish to express gratitude to the School of Mobile Computing & Communication for sanctioning a thesis work under Jadavpur University under which this work has been completed.

I would like to convey my sincere gratitude to Dr. Sarmishta Neogy, Professor, Department of Computer Science & Engineering, Jadavpur University for her valuable suggestions throughout the duration of the thesis work. I am really grateful to her for her constant support which helped me to fully involve myself in this work and develop new approaches in the field of Cloud Security.

I would like to express my sincere, heartfelt gratitude to Mrs. Sayantani Saha, Assistant Professor, Department of Computer Science & Engineering, Maulana Abul Kalam Azad Universtity Of Technology, Kolkata, for her helpful suggestions and guidance.

I would also wish to thank Dr. Punyasha Chatterjee, Director of the School of Mobile Computing & Communication, Jadavpur University and Prof. Pankaj Kumar Roy, Dean, Faculty of Interdisiplinary Studies, Law and Management, Jadavpur Universi for providing me all the facilities and for their support to the activities of this research.

Lastly I would like to thank all my teachers, classmates, guardians and well wishers for encouraging and co-operating me throughout the development of this thesis.

I would like to especially thank my parents whose blessings helped me to carry out my thesis in a dedicated way.

Regards,
SONALI MANDAL
University Roll Number: 001730501008
Examination Roll Number: M4DMC19011
Registration Number: 141105 of 2017-2018
School of Mobile Computing and Communication
Jadavpur University

Signed:

_____

Date:

_____

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Digitization of essential services paved the way to overcome lots of real-life problems across many sectors such as education, banking, transportation, healthcare etc. Interest in data-driven healthcare facilities has been growing rapidly. This resulted in generation huge amount of healthcare related data across many different health care services and organizations. Moreover different organization related to the healthcare services can contribute to a shared pool of data that is simultaneously generated and accessed by organizations such as hospitals, clinics and government bodies. This requires efficient and scalable storage and pervasive access to the data. Cloud computing is a natural candidate for storing and processing this huge amount of data. Typically such data is stored in a relational database (RDBMS) such as Oracle, SQL Server, DB2, SQLite, etc. Recent trends shows the use of NoSQL databases such as Cassandra, MongoDb, HBase, etc. to process such unstructured data, that allows more flexibility in data generation and processing.

Most of these data are sensitive and private, such as: medical reports, radio-graphic images, laboratory reaction history, insurance information, patient's personal information, doctor information and many more. Naturally, secure methods are required for storage, processing and transfer of such data. Not only protection of these data interests the patients and the medical organizations, but also laws are regulated by governments for ensuring the protection. Encrypting all the data after generation, can provide the required security and privacy. For example, sensitive information can be stored in the database in encrypted form. Also, while transferring the data between an application and a cloud server storing the database, Secure Socket Layer (SSL) can be used to maintain privacy in end-to-end data transfer. However, this thesis reports several challenges in storing and querying data in encrypted form, and provides directions to tackle them. Also, providing authorized access to different kinds of encrypted data based on the role of an user (e.g doctor, nurse, pathologist, admin personnel) is another challenge highlighted and tackled in this thesis.

1.1 Overview

This thesis presents an efficient and secure system model for managing e-Healthcare data stored in a public cloud environment. In short, this model provides a secure interface to the data that is stored in a distributed database, by tackling threats of both internal and external attackers, also provide efficient query management scheme. This model uses encryption to store data securely and thereby protecting it's confidentiality. A secure interface is provided to the authenticated users (such as the patients, doctors, nurses and other administrative personnel) for accessing the required data on-demand for both read and write requests. Specifically, the model allows encrypted data to be shared by several concerned users in a secure manner; i.e it provides seamless access to the encrypted data for permitted users, while limiting the access for other users in the system along with the internal or external attackers.

To preserve data integrity from external attackers and malicious internal users, we propose a system that uses a *blockchain* to check and prevent unauthorized data modification. A blockchain is a list of records, called *blocks*; such that each block contains a cryptographic hash of the previous block, a timestamp and some arbitrary data. We also propose an efficient strategy to search in the blockchain that is applicable to medical database systems.

Specifically, hash values of each sensitive data record are stored in a blockchain. These hash values of the data cannot be modified due to the immutable property of a blockchain. This means, once hash values of data records has been written to a blockchain, not even an authorized user can change it. Also, because of the immutable property of a blockchain, only the parts of the data that should not be modified, are stored in the blockchain.

In the medical data context, there is much information that need not be changed after inserting in the database, e.g medication history, patient name, age, etc but some of the information need to append according to patient treatment going on such as medication list, diagnosis reports, etc. We describe a system that handles the security and integrity of such sensitive data, reducing the risk of information leakage.

1.2 Outline

The rest of the thesis is organized as follows:

- Chapter 2: Describes a brief review of literature on security issue cloud database systems.
- Chapter 3: Provides a brief introduction of the concepts and the technologies used in this thesis.
- Chapter 3: Describes the proposed system and highlights the specific contributions.

- Chapter 4: Describes the implementation details and reports the performance of the implemented system following the proposed approach.
- Chapter 5: Draws the summary of proposed system and shows the directions for future work.

Chapter 2

Literature Survey

There are many approaches to medical data security in public cloud. Kantarcioğlu and Clifton, 2005 proposed a theoretical overview of a secure database server that provides probabilistic security guarantees. The authors loosely elaborated in the paper about how research in this area should proceed. They presented an efficient encrypted database and query processing model. Some security norms are also proposed in the encrypted database, such as any two tables with the same schema and the same number of tuples must have indistinguishable encryption methods. Ganapathy et al., 2011 presented a distributed architecture that provides both privacy as well as fault tolerance to the client. In this paper a query partitioning approach is taken for the query at the client side to the servers, which has satisfies the privacy constraints even after local changes to a partition. However, these most of the proposed approaches in the literature focuses on general data and their applicability is limited for a medical data storage. The limitations are in both in terms of the unique challenges that the properties of the medical data posses, and also in terms of the scope of exploiting domain-specific information to make the system more efficient. This thesis focuses on explicitly tackling the problem of querying encrypted data. Also, most of the previous approaches to secure a cloud database overlooks the threat of losing data integrity by the internal attacks, i.e the possibility of an authorized, but malicious user modifying the stored data. In this chapter overview of the related works in these two directions are provided.

2.1 Performing query in encrypted data

A tutorial by Arasu et al., 2014 informs that querying encrypted data quite challenging, highlighting the obstacles in trusted hardware and performing encryption in the client-based encrypted. Li et al., 2014 proposed a fast range query processing scheme for mdeical database that works against chosen keyword attack (INDCKA). The key idea in this paper is to organize indexing elements in a complete binary tree called PBtree, which satisfies structure indistinguishability (i.e., two sets of data items have the same PBtree structure if and only if the two sets have the same number of data items) and node indistinguishability(i.e., the values of PBtree nodes are completely random and have no statistical meaning). For efficient query processing, they used PBtree traversal width minimization and PBtree traversal depth minimization algorithm. The worse

case complexity of the query processing algorithm using PBtree need O(R*log n) time, where n is the total number of data items and R is the set of data items in the query results.

Saha et al., 2016 proposed an approach to secure sensitive medical data stored in the public cloud with the benefits of symmetric and asymmetric encryption keys using AES encryption algorithm. The authors proposed a framework for data-centric WSN application. Another objective of the is to establish a secure channel for data communication that could tackle attacks like MITM, DoS. Different segments of the patient data are encrypted using different symmetric keys. These keys are distributed on-demand basis to authorized users only. Before data transmission, data encrypted using AES symmetric key encryption algorithm. The scheme is made scalable by distributing static symmetric keys only to the legitimate users, with fine grained access control mechanism. Data integrity during transfer is achieved using SHA-1 hashing. Saha, Saha, and Neogy, 2018 proposed a methodology for protecting the privacy and confidentiality on e-Healthcare data using sensitivity association, also providing efficient searching scheme using metadata based search. A secure and efficient data retrieval strategy was proposed by Kumari, Saha, and Neogy, 2018. Here entire patient relation fragmented into different segments according to sensitivity. The data is divided into clusters using correlation between patient and authorized users to whom they are assigned. In the query evaluation phase, a co-relation metadata was used to validate a patient and the authorized-user combination. Fu et al., 2017 proposed a content-aware semantic search scheme for encrypted data. They used a graphical model, called conceptual graphs as a knowledge representation tool. Specifically they vectorized the plaintext to transform them into real valued vectors and performs query in the vector space. However the vectorizaion process can lead to deterministic behavior, reducing the security standard.

The related works in the literature provides many directions for encrypted search but overlooks an unique challenge in the medical data domain. Often multiple users have to access the same attribute of a table, while having different read-write access control. Also different rows in the same table can have many such associations with different users. However, if the data is stored in encrypted form, the problem of sharing such encrypted data in the same table and performing query while maintaining data integrity is not well investigated. Typically, role based access management are used for ensuring only authenticated users get access to the data Mitra et al., 2018, but such roles are defined in the database management system, not the the application level, which is a requirement for sharing encrypted data.

2.2 Preserving data integrity from internal attacks

The thesis focuses on the blockchain technology for preserving data integrity, that recently has been popular as a verifiable storage for electronic health records. For example, Azaria et al., 2016; Kuo, Kim, and Ohno-Machado, 2017; Liu, 2016 presented various secure blockchain storage for medical data. The first key benefit of blockchain that

it is a peer-to-peer, decentralized database management system. Therefore, blockchain is suitable for applications where independently managed health care entities collaborate with one another. Secondly, DDBMSs support create, read, update, and delete functions, while blockchain only supports create and read functions, ie, it is very difficult to change the stored data. Thus, blockchain is suitable as an unchangeable ledger to record critical information (eg, insurance claim records). Although blockchain is based on distributed technology and thus do not suffer from single-point-of-failure, it would be costly for DDBMS to achieve that high level of data redundancy blockchain does (ie, each node has a whole copy of whole historical data records). Thus, blockchain should be used only for storing data that are important and are modification sensitive.

In Muzammal, Qu, and Nasrulin, 2019, how a blockchain can be used as a relational database was studied. Furthermore, blockchain as a storage for medical data for auditing purposes was proposed in Azaria et al., 2016. However, blockchain being a linear data structure, querying time in a blockchain also increases linearly. This becomes a huge problem as the number of nodes in a blockchain can be in thousands or millions in practical medical database for a large organization. However, fast querying in such large blockchain that stores healthcare information is not very well studied. The approaches most related to this work are done by Roehrs et al., 2019 and Xu et al., 2017. Roehrs et al., 2019 implemented a distributed architecture that distributes data among servers and reintegrates when queried, which increases the average response time and availability. Their architecture is formed using a P2P network, where health records are organized into data blocks comprising a linked list and a distributed ledger of health data. Xu et al., 2017 proposed a hierarchical data structure to enable efficient querying in a blockchain that stores educational certificates.

The proposed approaches to minimize query time in blockchain do not exploit the data characteristics present in a typical medical database. For example, in a medical information system, data records are often searched by time ranges to know about the history of the diagnosis, medication and pathological information. Also these types of information are actually the prime candidates to be altered by malicious users. In the proposed approach, a hybrid security model is used where only hash values of such modification-sensitive data are stored in a blockchain to reduce overhead and the actual data is stored in encrypted form in a Cassandra database. Also, the blockchain storage uses a hierarchical data structure using the timestamp information present in the medical data, that supports a much faster query.

Chapter 3

Background

3.1 Security in cloud database

The cloud computing model is transfers computing infrastructure and data to third-party service providers that manage the hardware and software resources which enables on-demand, anytime-anywhere access and cost reductions. Many medical organizations have started shifted electronic health information to the cloud storage. Not only it simplifies the exchange medical records between the hospitals, clinics and other participating organizations, but also makes the cloud a medical record storage center which is permanent and remotely accessible by many users. The medical data stored in cloud makes the treatment better by retrieving patient's medical history from the database before going for the treatment and get to know about the health issues of the patient.

However, maintaining the confidentiality of the data stored in cloud is a major issue. The data stored in such a cloud database can be stolen in a number of ways:

1. Data stolen while transferring using network
2. Direct abuse by untrusted cloud service provider
3. Third party attack in the network or on the cloud service provider
4. Legal issues: Law in the nation of the cloud service provider may enforce it to reveal all the data.

One way to implement security of the data in a cloud database is to encrypt the data before sending to the cloud this involves sharing a key with the cloud provider. In this method all the encrypted data sent by client are first decrypted with the key and decrypted data is stored in the DBMS. Now the client application can send an encrypted query which is decrypted and run on the server DBMS. To ensure confidentiality the result of the query is encrypted by the security module in the server. The client also having the key can decrypt this encrypted result and view the data. An overview of the scheme with a trusted cloud provider is shown in Fig. 3.1.

Although this simple solution works well, it can be only implemented with a trusted and secure cloud provider. Also there is still a risk of leakage of sensitive data because

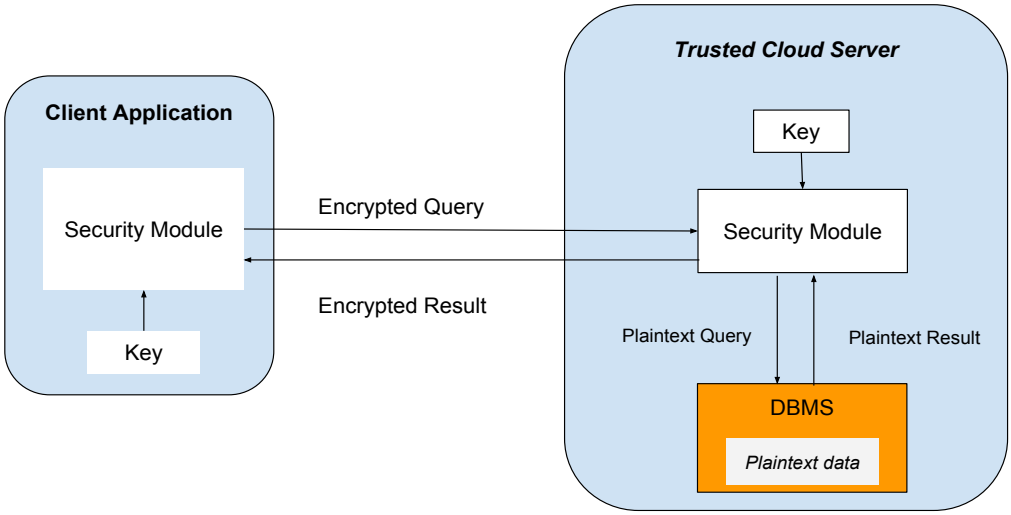

FIGURE 3.1: Data transfer in trusted cloud provider.

securing a server in the cloud is not easy and there is a chance of third party attack. Also legal issues can cause loss of confidentiality.

In an alternative approach called the *data-at-rest encryption*, no key is shared with the server and only encrypted data is stored in the cloud DBMS, shown in figure 3.2. Client application sends encrypted data to the cloud, and the cloud provider directly stores the encrypted data in the DBMS. In this case, the cloud provider can never see the actual data. So any attack on the cloud server, or an untrusted cloud provider only gets the encrypted data which is of no use without the decryption key. Also this data is safe from legal issues.

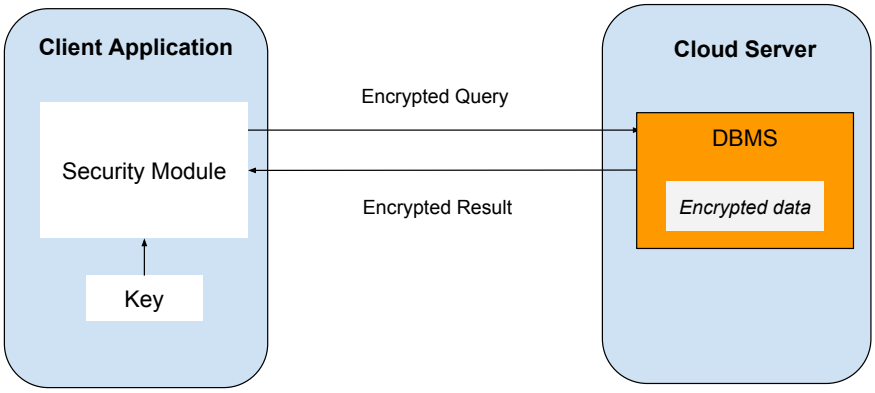

FIGURE 3.2: Data-at-rest encryption in DBMS.

Even though this mechanism conforms the required security standards, there are several issues with storing encrypted data in the database management system, as reported by Arasu et al., 2014:

1. Performing search query is difficult on encrypted data. This is because the user is expected to provide a query containing attributes in plaintext, which are to be matched with encrypted data.

2. Some queries require data to be ordered by some column value. Since a primary requirement of any encryption algorithm that there should be no relation between the characters of the plaintext and the ciphertext, i.e without the key the plaintext can not be predicted from the ciphertext. So obviously, the order of some data records(plaintext) determined alphabetically or numerically will not be the same if the records are encrypted. So performing such queries where the actual ordering of the data is required, is challenging in a encrypted database.

3. Another set of queries like sum, average etc. require mathematical operation to be perform on the data records. The encryption scheme that allows performing an mathematical operation on encrypted data and getting the same result as if perform on actual data is called Homomorphic encryption. This is challenging and there are only a few Homomorphic encryption schemes available, that to allows only a restricted set of operations.

3.2 Cassandra

In this section the motivation for using Cassandra as the database is provided, along with a brief description of its security aspects.

3.2.1 Introduction to Cassandra

Cassandra is a highly scalable, distributed, structured key-value store NoSQL database that is chosen to demonstrate the security model presented in the thesis. It is a mixture of the distributed system technology like the Dynamo database and the data model is column family based like Google's BigTable. Cassandra is eventually consistent, i.e it may not be consistent all the time but will be consistent eventually after a finite time. Cassandra supports four types of storage model: Wide Column Store/Column Families, Document store, Key Value/Tuple Store, Eventually Consistence Key Value store, Graph Databases. The design goal of Cassandra is to handle large amount of workload across multiple nodes without single point of failure by replicating data. All the nodes in a cluster play the same role. Each independent node interconnected to other nodes at the same time.

Cassandra deals with unstructured data and flexible schema. Internal data model of Cassandra and the equivalent relational database terminologies are shown in Table 3.1. Attributes are only blob-values in the internal model and the attributes of one row are always stored sorted by the name of the attribute, when written to the internal storage called the *SSTable*, shown in Figure 3.3).

TABLE 3.1: Terminologies of Relational DBMS and Cassandra.

Relational Model	**Cassandra Model**
Database	Keyspace
Table	Column Family(CF)
Primary Key	Row Key
Column Name	Column Name/Key
Column Value	Column Value

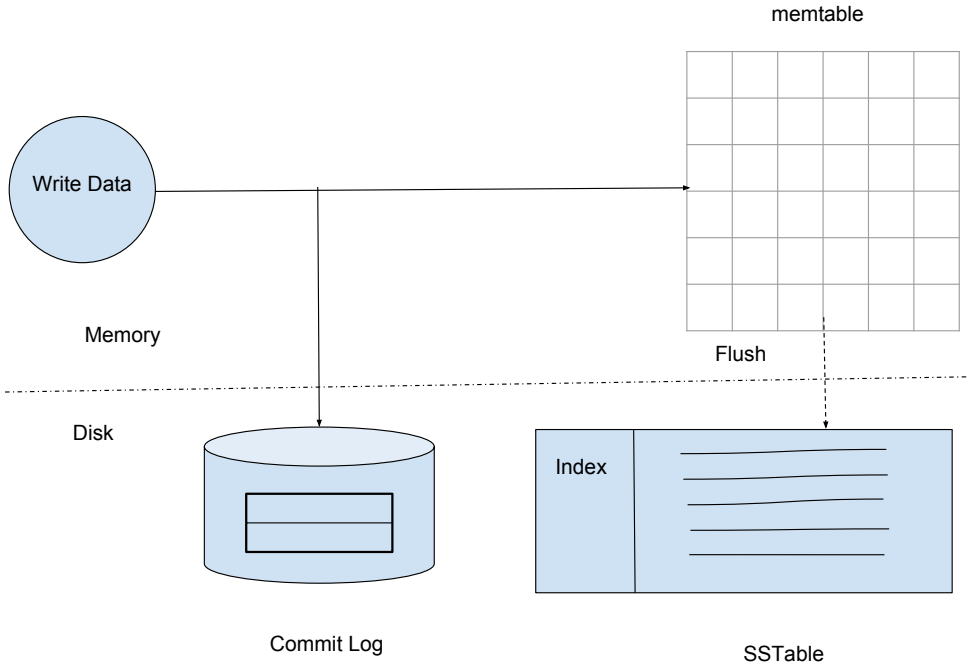

FIGURE 3.3: Data storage model of Cassandra.

Every tuple also contains one special attribute: the primary key, which is used for distributing and replicating the tuples across the nodes of the database cluster, in addition to its traditional use for indexing. Cassandra does not repeat the names of the columns in memory or in the SSTable. In the SSTable on disk, Cassandra stores data after flushing the *memtable*. Any data written to Cassandra, is first written to a commit log before being written to a memtable. This provides durability in the case of unexpected shutdown. On starting Cassandra any mutations in the commit log is be applied to memtables.

3.2.2 Querying in Cassandra

Cassandra provide its own query language called Cassandra Query Language(CQL), available in the online documentation *Cassandra Query Language*. CQL is somewhat similar to Structured Query Language(SQL), that offers an easy interface to interact with the database. This is because the Thrift API, that was being used prior to CQL, was found difficult to comprehend. CQL supports various set of data type i.e native types, collection types, user-defined types, tuple types and custom types. CQL uses database roles to represent users or group of users.

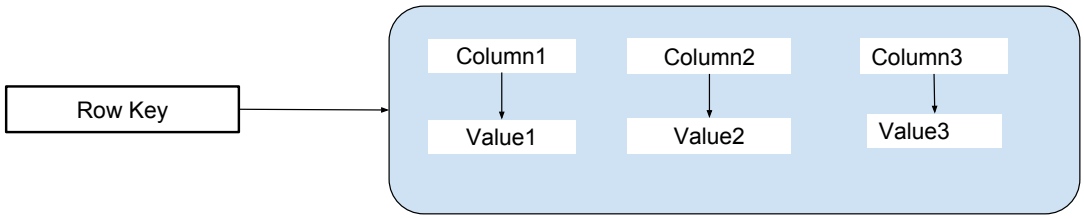

FIGURE 3.4: Cassandra look-up map using row and column keys.

CQL uses a map for efficient key look-up, and the its sorted nature gives efficient scans. In Cassandra, both the row keys and column keys to do efficient look-ups and range scans, as shown in Figure 3.4. The number of column keys is unbounded, In other words, i.e very wide rows are supported.

The difference between the internal data model used by Cassandra and the CQL data model is that the columns have fixed types in the CQL data model, i.e. they are no longer just blob values, and the primary key can be a compound key, i.e. consisting of multiple columns. Additionally, the primary key is further divided into two different parts: the partitioning primary key and the clustering primary key, both of which can be made of an arbitrary amount of columns. The partitioning primary key is the one that is used as the primary key in the internal data model, and thus it is used to share and replicate the tuples. The clustering primary key is used when transporting CQL tuples into internal tuples to define their ordering, i.e. multiple CQL tuples with a common partitioning primary key will be sorted in one internal tuple based on their clustering primary key, when written to the internal storage.

3.2.3 Security aspects of Cassandra

There are three major components to the security features provided in Cassandra by default: Internal authentication, Internal authorization and SSL/TLS encryption for client and inter-node communication. A detailed documentation on the security provided by Cassandra is available online *Cassandra Security Documentation*.

Internal Authentication

Internal authentication is based on Cassandra-controlled roles and passwords. Role-based authentication encompasses both users and roles, i.e different users can have different roles in the database and the specifc roles are granted using password based authentication. The roles can represent either actual individual users or roles that those users have in administering and accessing the Cassandra cluster.

Internal Authorization

Cassandra supports object permissions, that are assigned using Cassandra's internal authorization mechanism for the following objects: keyspace, table, function, aggregation, roles and MBeans (available only in Cassandra 3.6 and later). Authenticated roles with passwords stored in Cassandra are authorized for selective access. The permissions are stored in Cassandra tables.

Permission is configurable for CQL commands that are used to create or manipulate data. The allowable commands are: *CREATE, ALTER, DROP, SELECT, MODIFY, and DESCRIBE*, which are used for the generic interaction with the database. The *EXECUTE* command may be used to grant permission to a role for the *SELECT, INSERT, and UPDATE* commands. Additionally, the *AUTHORIZE* command can be used to grant permission for a role to *GRANT, REVOKE or AUTHORIZE* other role's permissions.

JMX Authentication and Authorization

Cassandra runs under a Java virtual machine, and it provides various statistics and management operations via the Java Management Extensions (JMX). JMX is a Java technology that supplies tools for managing and monitoring Java applications and services. JMX authentication and authorization allows only a set of users to access JMX tools and JMX metrics. In Cassandra version 3.5 and earlier, JMX is configured with password and access files. In Cassandra 3.6 and later, JMX connections may use the same internal authentication and authorization mechanisms as CQL clients.

Encrypted communication

Cassandra provides secure communication between a client machine and a database cluster and between nodes within a cluster. Enabling encryption using SSL ensures that data being transferred is not stolen or tampered. The options for client-to-node and node-to-node encryption are managed separately and may be configured independently. In both cases, some default values of JVM for supported protocols and cipher suites are used when encryption is enabled. These can be modified using the settings in *cassandra.yaml* file. However, this is not recommended unless there is a need to disable vulnerable ciphers or protocols in cases where the JVM itself cannot be updated.

The settings for managing inter-node encryption can be set using the *Cassandra.yaml* file. To enable inter-node encryption, the inter-node setting should be changed from its default, *none* to any of these: *rack, dc or all*. If neither is set to true, client connections are entirely un-encrypted. If *enabled* option is set to true and the *optional* option is set to false, all client connections is encrypted. If both options are set to true, both encrypted and un-encrypted connections are supported using the same port. Client connections using encryption with this configuration is automatically detected and handled by the server.

To summarize, Casandra provides some basic security protocols such as user authentication and SSL encryption for data transfer between the clients and the distributed

nodes. However, if Casandra itself runs in a cloud computing environment, then the stored data can be compromised. To store data in an untrusted cloud provider, the thesis presents a secure application layer on top of Cassandra.

3.3 Cryptographic algorithms

3.3.1 AES Algorithm

To perform encryption, a non-deterministic encryption algorithm called Advanced Encryption Standard(AES) is chosen. AES is a symmetric key block cipher algorithm. This cryptographic algorithm perfroms very secure encryption, overcoming the drawbacks of the DES encryption algorithm. In DES algorithm 64-bit blocks and the 56-bit key is used for data encryption and decryption, but in AES 128 bit blocks and 128-bit key/256-bit is used. Plaintext is converted to a target ciphertext using substitution and transposition techniques after a minimum of 10 and a maximum of 14 rounds of execution.

3.3.2 SHA Algorithm

A hashing algorithm is used to map a variable length data to a fixed length data, in such a way that the process becomes irreversible, i.e it is impossible to get the original data from the hashed data. In this work, Secure Hashing Algorithm(SHA) is used, that is a modified version of the previous MD5 hashing algorithm. This algorithm takes less than 2^{64} bits length input and converts into 128-bits length digest. This algorithm follows several steps for execution. The first step is used for padding zero or one in input text as multiple of 512-bit blocks then extra 64 bits are appended to end of the message, that is carrying information about the length of the original message. The message is divided into 512 bit-blocks, then the chaining variables are initialized and the actual mathematical process (OR, AND, NOT, XOR operations) is performed. In this way, the produced hash digest length is large that's why 2^{160} operations need to break this algorithm, which is infeasible to do.

Chapter 4

Proposed Model

4.1 Overview

In this section, an efficient and secure system model is introduced for managing e-Healthcare data stored in a public cloud environment. In short, this model provides a secure interface to the data that is stored in a distributed database, by tackling threats of both internal and external attackers. This model uses encryption to store data securely and thereby protecting it's confidentiality. A secure interface is provided to the authenticated users (such as the patients, doctors, nurses and other administrative personnel) for accessing the required data on-demand for both read and write requests. Specifically, the model allows encrypted data to be shared by several concerned users in a secure manner; i.e it provides seamless access to the encrypted data for permitted users, while limiting the access for other users in the system along with the internal or external attackers.

Another part of this work to prevent unauthorized modification of data by internal attackers blockchain has been used. Internal attackers may be able to modify the data in the database. To prevent this, hash values of each data record are stored in a blockchain. These hash values of the data cannot be modified due to the immutable property of blockchain. This means, once hash values of data records has been written to a blockchain, not even an authorized user can change it. However, this can be only applied to be data that can not be modified. In the medical data context, there are many such information that need not be changed after inserting in the database, e.g medication history, diagnosis reports etc.

4.2 Approach

The proposed approach of the secure system model consists of different modules as described below. An overview of the system architecture is shown in Figure 4.1. In the following sub-sections different modules of the system are described.

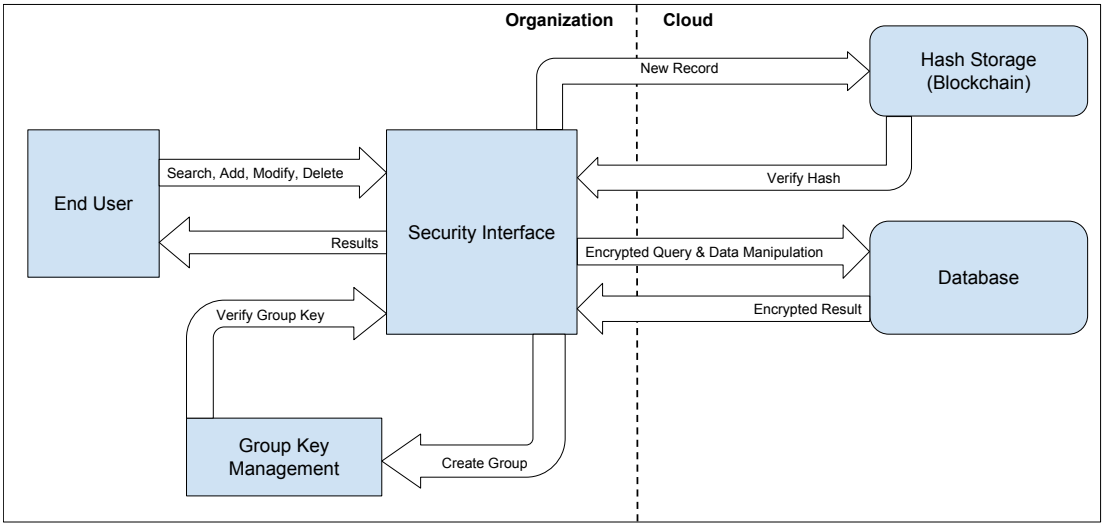

FIGURE 4.1: An overview of the system model.

4.2.1 Cassandra database

The Security Module as shown in Figure 4.1 is responsible for getting various patient's and doctor's information and storing them in the Cassandra database after encryption. Information includes general information of the patient that are unchanged: patients unique identification number(*pid*), name(*pname*),phone number(*phno*), date of birth (*dob*), blood group (*blood_group*) and patients diagnosis details that can be updated including blood pressure(*blood_pressure*), pulse rate (*pulse_rate*), diagnosis date (*date*), etc. and the current treatment particulars(*medication*). All these information are received from the users as plaintext and stored in the database as ciphertext after encryption. The tables in the database used in our model, along with their schema are shown in Table 4.1.

TABLE 4.1: Database tables and schema details.

Table Name	**Schema**
patient_general	pid, pname, dob, blood_group, gender, phone_number, date
diagnosis_data	pid, blood_pressure, pulse_rate, hemoglobin, date, medication, x-ray report
doctor_patient_metadata	pid, dp_metadata
nurse_patient_metadata	pid, np_metadata
doctor_nurse_metadata	pid, dn_metadata

The *patient_general* table is used to store personal information about the patients. A row from this table is shown in Table 4.2. Information stored in this table is accessed by all the users in the system and some users have read-only access. Data in this table is generally added first by nurse and administrative personnel.

The *diagnosis_data* table contains details of ongoing treatment and medication. A

TABLE 4.2: Patient_general table for storing personal information.

pid	**pname**	**dob**	**blood_group**	**gender**	**phno**
40122	O+	19/3/1968	Sisir Sarkar	Male	8112637289

sample row from this table is shown in Table 4.6. This table is accessed by doctors,

TABLE 4.3: Diagnosis_data table for storing medical information.

pid	blood-pressure	pulse-rate	hemoglobin	date	medication	x-ray report
40122	143/12	66	12.3	27/03/05	Tab. Flex...	Image

nurses, and pathologists. Some users may have read-only access to some parts (columns) to a data record. For example, only doctors have read-write access to medication information, while other users have read-only access.

The *doctor_patient_metadata*, *nurse_patient_metadata* and *doctor_nurse_metadata* tables contains pid and diffrent types of metadata based on the relationships. The metadata is used to enable querying on encrypted data. The metadata is created for unique user-to-patient and user-to-user relationships.

For example, the doctor_patient metadata is a user-to-patient relationship which stores metadata for patients and the doctor who is treating the patient. The doctor-nurse metadata is a user-to-user relationship which stores metadata for nurse-doctor pairs, where the nurse is attending to patients treated by the doctor. The metadata is created by first appending the identification numbers of the concerned users. Then a hashing algorithm is used on the appended numbers, and the hashed value is stored in the table. For the patient-to-user relationship, the patient identification number (pid) is used in the same way.

The metadata is not encrypted so that it can be used in a query. However, metadata contains the identification number of the users and the patient. To avoid leakage of these identification numbers, a hashing scheme is used. This is because from the hashed value, original identification numbers cannot be extracted. A deterministic hashing algorithm is used, so that for the same identification number pair, the same hashed value is always generated. The pid is used as a reference key among all the tables.

An example of populated rows from the *doctor_patient_metadata* table is shown in Table 4.4.

TABLE 4.4: Example rows from the Doctor_patient_metadata table.

pid	meta
40122	4012231
40123	4012331
40124	4012450

The database used in our model can handle heterogeneous data types such as text, numeric data and image. This is because before performing encryption, the data is converted to binary form as array of bytes, irrespective of the original data type. After

encryption, the data is stored in the database, as the *blob* data type that supports storing raw binary data. Specifically this makes it possible to store encrypted images (such as x-ray reports), then retrieve and decrypt the same with ease.

4.2.2 User validation for shared data

A model for a user validating other users if they share common data is presented here. This is made available to the *Security Interface* by the *Group Key Management* module. Users may have different privileges on the data, based on their roles in the organization. For example, a doctor might have access to more sensitive data than a nurse. Also, two users groups may share a set of data, while some other sets of data may have mutually exclusive access rights. For example, a nurse might have access to medication history, which is inaccessible to a pathologist; whereas pathologist might have reports protected from nurses.

In the proposed system, sharing of encrypted data by specific user groups is enabled in the application level, instead of the database level. This makes it possible to encrypt the shared data, before sending it to the cloud database. Encrypted data sharing is performed by using a shared key (that is shared by the user groups) to encrypt and decrypt all the shared data in a user group. An example workflow for inserting a data row by a doctor, where some of the data is also shared by a nurse is shown in Fig. 4.2.

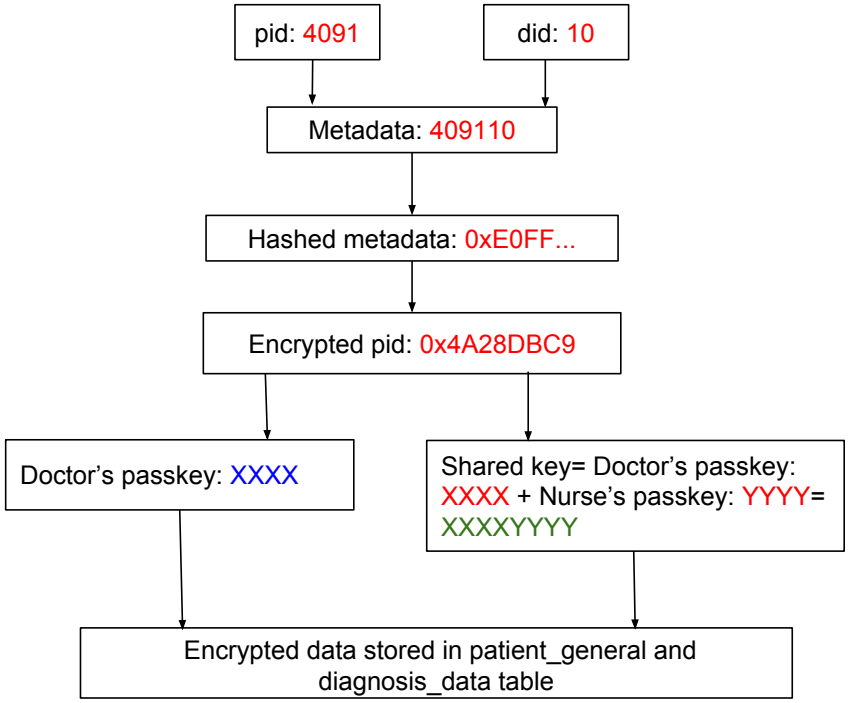

FIGURE 4.2: Workflow for inserting data shared by a doctor and nurse.

The shared key is dynamically generated by collecting pass-codes from all the users in the group. For inserting data row that contains shared data, a shared key is generated from individual keys of the members of the group. Then the shared parts of the data

row is encrypted with the shared key and non-shared parts are encrypted with the key of the user, who is performing the insertion. It can also happen that, different parts of a data row is shared by different groups.

To validate a user in a group, for every request the user makes to access shared data, the system asks all the users in the group for pass-codes. If correct pass-codes are received from all the users in the group, then the correct shared key is generated and encryption/decryption is performed. By this mechanism, role-based data accessed is implemented in an implicit manner i.e only if a user provides a valid pass-key then the required shared key will be generated and decryption will be successful. If a user is not authorized to access some data, he/she will not be able to provide the correct pass-key and thus decryption will fail.

An example workflow for a doctor searching for information about a patient is shown in Fig. 4.3.

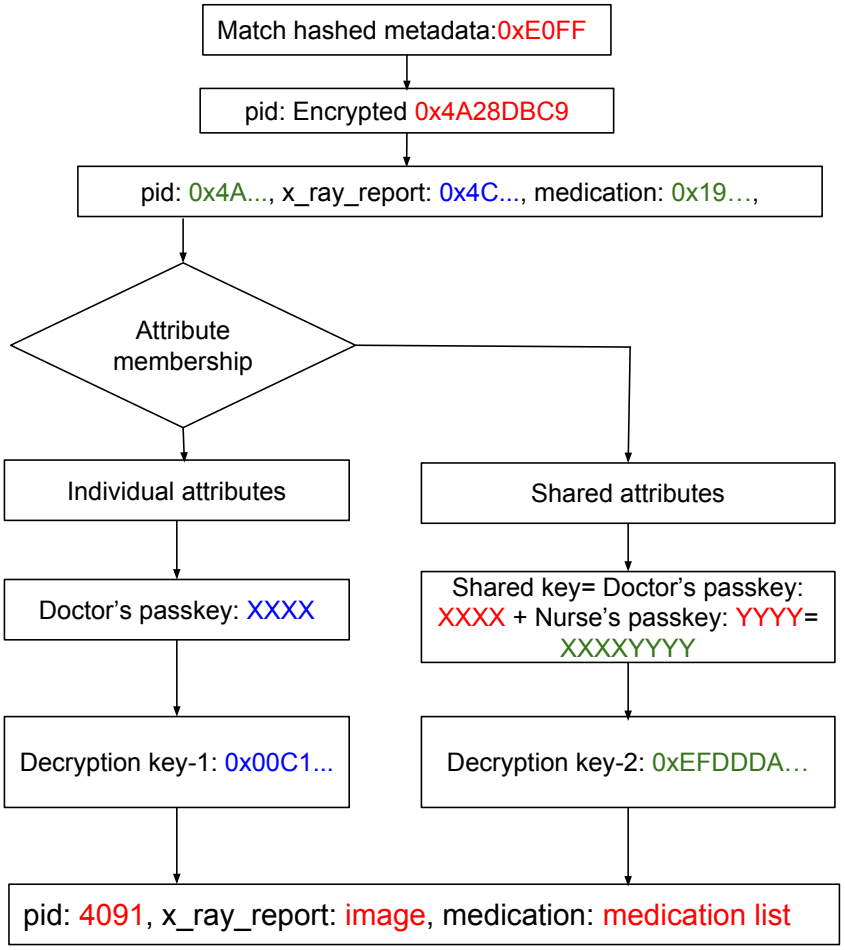

FIGURE 4.3: Workflow for a doctor searching for information about a patient.

The full procedure of extracting decrypted information from the database for the same example is given below.

1. The doctor provides patient identification number and his/her own identification number to the system.

2. Metadata is generated by appending the doctor identification number and the patient identification number an then hashing the appended value by a deterministic hashing scheme .

3. The doctor-patient relation table is queried to find the row containing the doctor-patient metadata and the corresponding encrypted patient identification number is extracted.

4. Depending upon the query, the *patient_general* or the *diagnosis_data* or both is queried using the encrypted patient identification number. After this step, information about the patient is found in encrypted form.

5. If some part of the queried information is shared by multiple users then the system asked for passkeys from all the users including the doctor. The passkeys are appended then hashed to generate the shared key.This is done for all possible doctor-to-other-users relationship. If some information is only accessed by the doctor then only the doctor passkey is used as the decryption key.

6. The above step generates a set of decryption keys for different parts of the patient information that is possibly shared by disjoint user groups. The decryption key is used to decrypt specific parts of the patient information. Finally, the decrypted information is shown to the doctor.

4.2.3 Blockchain storage

Although storing data in encrypted form prevents attacks from external threats, internal attackers who are maliciously logged in to the system as authorized users, can still modify sensitive data. An approach to maintain data integrity is proposed in this section, that uses a *blockchain* to check and prevent unauthorized data modification.

A blockchain is a list of records, called *blocks*; such that each block contains a cryptographic hash of the previous block, a timestamp and some arbitrary data. We also propose an efficient strategy to search in the blockchain that is applicable to medical database systems. Specifically, hash values of each sensitive data record, that are in the encrypted form, are stored in a blockchain. These hash values of the encrypted data cannot be modified due to the immutable property of a blockchain. This means, once hash values of encrypted data records has been written to a blockchain, not even an authorized user can change it. Also, because of the immutable property of a blockchain, only the parts of the data that should not be modified, are stored in the blockchain.

In the medical data context, there is much such information that need not be changed after inserting in the database, e.g medication history, patient name, age, etc but some of the information need to append according to patient treatment going on such as

medication list, diagnosis reports, etc. The blockchain storage only stores the hash values of the ciphertext of such modification-sensitive sensitive data, preventing the risk of information modification my malicious users.

The proposed model uses both the encrypted Cassandra database and the blockchain storage to provide data integrity. When data is queried from the encrypted database, the system first extracts authorized user data from the cloud storage using proper authentication system and also extracts the hash values of the data stored in the blockchain storage. After this, the calculated hash value of the encrypted data is matched with the hash value queried from the blockchain. If the two hashes match, then the data is unmodified. The workflow for checking data integrity is shown in Figure 4.4.

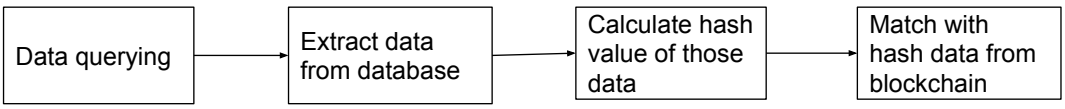

FIGURE 4.4: Workflow to detect unauthorized data modification.

Data in a block can't be modified even by authorized user, because this will invalidate the internal hashes of the blockchain that are dependent on previous blocks, i.e the hashes are calculated in a cascading manner. So, even if an authorized user changes some parts of the data in the encrypted database, the authorized user can't store the changes in blockchain. Data is always inserted in the last block, i.e the blockchain only supports insert and select queries, no update queries are allowed. The data integrity check is performed during a select query. So if in the meantime, if a malicious user modifies a data record in the encrypted database with a valid key, still the data alteration can be checked using the blockchain. The main motivation for using the blockchain to store the hash values is that, it is very difficult to change the data stored in the blockchain, so the hash values are protected from the internal attackers. Also, no database user has access to the blockchain. The data in the blockchain is inserted and queried internally by the application. This prevents unauthorized users from forcefully tampering the blockchain itself.

4.2.4 Efficient query in blockchain

The blockchain storage is used for storing hash values of the modification-sensitive parts of the patient's data. However, as blockchain is essentially a linked list of blocks, searching in the blockchain for hash validation must be performed sequentially, i.e random access to a block in the blockchain is not permitted. In practice, a medical organization can have hundreds of patients getting admitted per day. This can result in sequential search in a blockchain with millions of blocks, which is very inefficient. Asymptotically, the time complexity is $O(n)$ in the worst case, where n is the number of blocks in the blockchain. The data in the block is always timestamped. We exploit this property to implement the blockchain in a hierarchical manner to support efficient

search. An overview of this hierarchy is shown in Fig. 4.5.

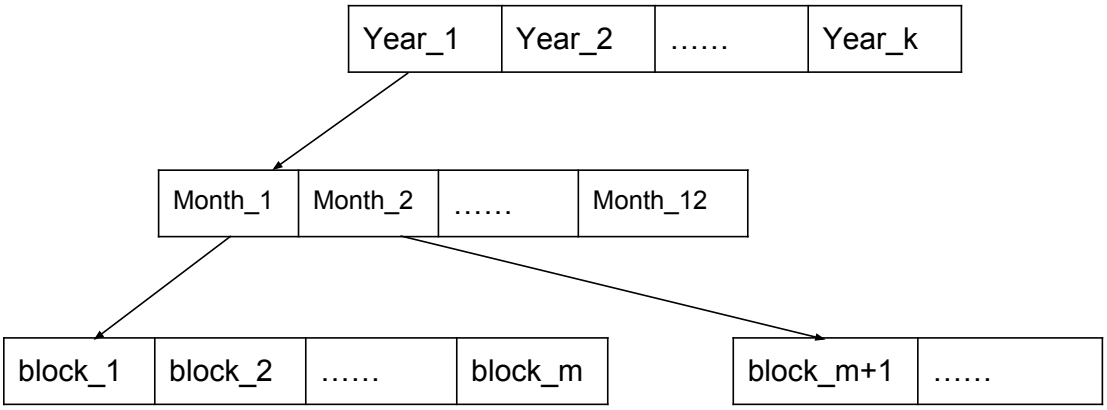

FIGURE 4.5: Multilevel lists for efficient searching in blockchain.

The topmost level of this hierarchical data structure consists of a list ordered by year, which is extracted from the timestamp. The middle level is a list of lists, where each list consists of months that belong to the year, determined by the position in the outer list. Similarly, the lowest level consists of the actual blockchain, stored as a nested list, where individual lists contain the blocks for a month. The procedure for validating the data integrity of a patient is described below:

1. First, the encrypted data record is fetched from the database, which includes the same timestamp stored in the blockchain.
2. Hash value of the encrypted data record is calculated using SHA-256, that yields H_1
3. The timestamp is partitioned into year and months.
4. Then the top level list is searched using the year as search-key.
5. Using the returned position and the month, the mid level nested list is searched and so on.
6. At the lowest level, data is searched in a linear fashion, and the corresponding data block is extracted.
7. From the extracted data block, the has value of the encrypted data record is calculated using SHA-256, that yields H_2.
8. If $H_1 = H_2$ the data is returned, else an error is raised to show that data integrity is lost.

In the worst case, total number of comparisons are $k + l + m$, where k is the length of the top level list, l is for the mid level and m is for the bottom level list. As the number of months in a year is a constant number, the total asymptotic complexity is given by $O(k + m)$, where $k + m$ is much lesser than n.

To describe the working principle with an example, consider the data stored in the *Patient-general* table shown in Table 4.5, that stores personal information about the patient that are collected when the patient is admitted for the first time.

TABLE 4.5: Patient-general table for storing personal information. The data marked in blue color are modification-sensitive.

Patient-id	Patient_name	Gender	age	Phn_no	Date
40122	Purusottam Sarkar	Male	45	8112637289	15/02/17
40125	Swarup Ghosh	Male	16	9091906754	16/02/17
40126	Balaram Bagdi	Male	54	9891901757	16/02/17
40127	Jamaluddin Ahamed	Male	80	9748190043	17/02/17

The blockchain for the patient is created by inserting the hash values of the modification-sensitive personal information in the first block of the blockchain, also called the genesis block. Also, consider the data stored in *diagnosis_data* table that contains details of ongoing treatment and medication, shown in Table 4.6. The first row is storing the diagnosis information for the first visit, and the second row stores some updated information after the patient came for a re-checkup. Similarly, subsequent addition of information for

TABLE 4.6: Diagnosis-data table for storing ongoing diagnosis information. Data marked with blue color are modification sensitive.

Patient -id	Blood-pressure	Pulse-rate	Hemoglobin	Date	Medication	Test report
40122	143/12	66	12.3	15/02/17	Tab. Flex ...	-
40122	140/11	65	-	05/03/17	Repeat	Chest X-Ray
40125	99/44	57	-	16/02/17	Tab Inderal 10mg	ECG
40126	145/85	110	-	16/02/17	Tab. Glu-formin G2	-
40127	138/78	94	-	16/02/17	Tab.ESLO 2.5mg	ECG,HB, CR,FBS, HDL

the same patients are stored in the successive blocks in the blockhain. When adding a new block to the blockchain of a particular patient, the timestamp information in the data record generation is used to find the allowed slot in the hierarchy.

For the example shown in Figure 4.6, the genesis block contains the patient id and personal information, the next update of diagnosis results and medication are added to next block and stored in the same month slot. Then for a visit in the the next month, X-ray and ECG reports are stored in the Block2. As the two data records are generated in different months, they are stored in the blockchain by linking with hierarchy accordingly.

The proposed hierarchical searching approach is generic and it can be extended to an arbitrary number of levels. This can be useful to further partition the blockchain, so that at the lowest level, the number of clustered blocks becomes manageable, thereby

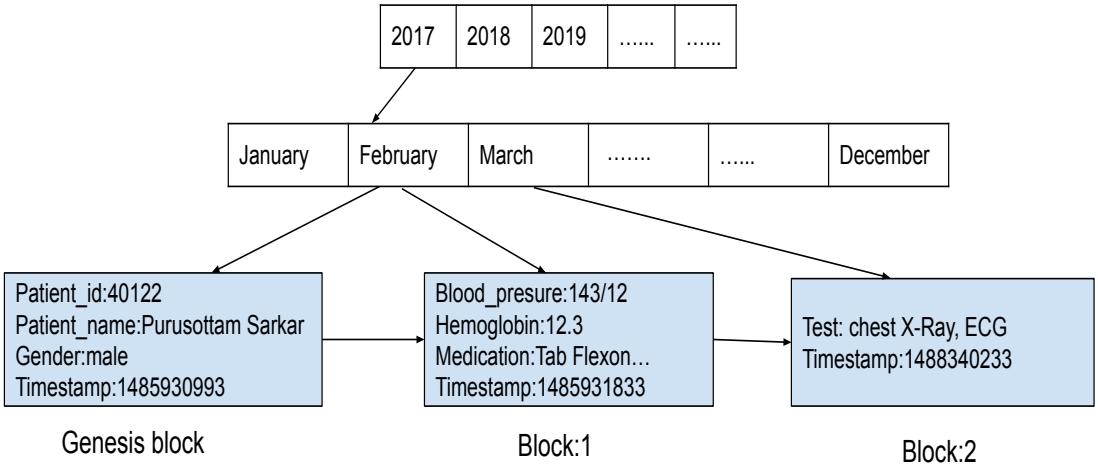

FIGURE 4.6: Data integrity check using hierarchical block chain data storage.

limiting the linear searching time to a tolerable limit. Another important aspect is to be noted that, there is a small amount of overhead incurred due to managing the hierarchy. So, insertion time in the blockchain is increased by a small factor. However, as in the healthcare data domain, the number of searches is usually much greater than the number of insertions, the overhead in insertion can be traded off for the benifit of faster search.

Chapter 5

Implementation and Evaluation

In this chapter the details of the implementation of a system that follows the security model proposed in Chapter 4 are described.

5.1 Implemented system

The proposed work is implemented using the python programming language, using Cassandra as the database. The CQL commands for using the Cassandra database are presented below:

- **Creating the keyspace:**
  CREATE KEYSPACE *userdatabase* WITH
  REPLICATION=*{'class':'SimpleStrategy', 'replication_factor':'3'}*
  AND DURABLE_WRITE= *true*;

- **Creating the doctor-patient metadata table:**
  CREATE TABLE *userdatabase.dpmeta*
  (dpmeta text PRIMARY KEY,
  pid blob);

- **Creating the patient_general table:**
  CREATE TABLE *userdatabase.patient_general*
  (pid blob PRIMARY KEY,
  dod blob,
  gender blob,
  gurdianname blob,
  phone blob,
  pname blob);

- **Creating diagnosis_data table:**
  CREATE TABLE *userdatabase.diagonosis_data*
  (pid blob PRIMARY KEY,
  blood_group blob,
  blood_pressure blob,
  date blob,

hemoglobin blob,
pulse_rate blob,
test_report blob);

The software components of the system and their versions are shown in Table 5.1.

TABLE 5.1: Versions of the software components of the implemented system.

Component type	**Name**	**Version**
Programming environment	Python	3.6.5
Database	Cassandra	3.11.3
Query engine	Cqlsh	5.0.1
Operating system	Ubuntu	14.04

For performing encryption, hashing and for connecting to the Cassandra database several python libraries were used. Table 5.2 shows the details of the libraries.

TABLE 5.2: Details of the python libraries used.

Library	**Purpose**	Version
Cassandra-driver	The driver connects to the Cassandra cluster from python. The driver also supports performing CQL queries.	3.13
Pycryptodome	Performs encryption and decryption using AES algorithm.	3.5.1
Bcrypt	Performs hashing using SHA-256 algorithm.	3.1.4
Pillow	Performs image manipulation	5.0.0
Pickle	Performs python object serialization, used to store blockchain objects as files.	11.0.1

5.2 Results

The system was experimented with using two different machines, both running Ubuntu 14.04 operating system. Table 5.3 shows the hardware specification for these two machines. One of the machine was used as a cloud server and another was used to run the client application. The two machines were connected by a personal WiFi network with a link speed of 150 megabytes per second.

TABLE 5.3: Hardware specifications of experimented machines.

Machine	**CPU speed**	**Memory (Speed)**	**Disk (Speed)**
Server machine	2.7 GHz, 4cores	4 GB (2400 MHz)	1 TB (5400 rpm)
Client machine	2.2 GHz, 2cores	4 GB (1600 MHz)	512 GB (5400 rpm)

In the following sections the experiments with the proposed system, performed in these machines are described and the results are reported.

5.2.1 Evaluation of encrypted search

Search query is performed on the database by specifying a search term, which consists of a column name and an expected value to be matched. Consider a search query: *"SELECT * FROM patient_general WHERE pid=3457"*. In this query, information about a patient whose pid is 3457 is to be found. If non-deterministic encryption scheme is used, then different ciphertexts are generated for every encryption. This means, if data is stored by encrypting non-deterministically, then the ciphertext generated from the search term will be different than that is stored in the database, and no matching could be done. To perform query in such a non-deterministically encrypted database, we have analyzed two methods for querying data:

1. The columns containing the search term are stored as plaintext. This is a trivial approach but it does not enable fully encrypted database. Also, all the columns for possible search terms has to be kept in plaintext. For example, users may want to search by patient's name, date of admission etc. These information can be considered semi-sensitive, but they have to be stored in plaintext in this method.

2. The ciphertext of the search term is stored in a the metadata table, along with the metadata generated for the user. When the data is to be queried, the metadata is used as an secondary index, to first find the ciphertext of the search term. Then the ciphertext of the search term is used to find the actual data row.

Table 5.4 shows the comparison of the above mentioned system variants. The results were calculated by averaging the measured time of 100 query in a database with 5000 inserted rows.

TABLE 5.4: Average query times for searching in a non-deterministically encrypted database.

Query type	Metadata used	Search term	Time
Select	Yes	Encrypted	270.1 ms
Select	No	Plaintext	13.6 ms

Note that the metadata-less method performs query faster because it does very minimal processing. But the security of this method can be compromised as the search keys can not be stored in encrypted form. Whereas, the query using metadata takes longer, as it has to perform one extra query to get the metadata-key pair. But, as all the information in the actual table is encrypted, this method is much more secure. Even though the time taken for the metadata based search takes longer time, it does not affect the uer experience, as the time is within seconds, i.e it performs the query in real time.

5.2.2 Evaluation of blockchain search

Detailed experiments were performed to compare the naive linear data structure and the proposed hierarchical data structure approach in a blockchain, by varying the number of blocks and by measuring both the insertion and the query times. Table 5.5 shows

the results of the comparison.

TABLE 5.5: Query time comparison between linear and the proposed hierarchical data structures.

Types	**No. of blocks**	**Insertion time(ms)**	**Query time (ms)**
Linear	2052	0.001515	0.202277
Hierarchical	2052	0.010424	0.005797
Linear	11172	0.005885	5.996496
Hierarchical	11172	0.046613	0.027285
Linear	45372	0.018346	127.751428
Hierarchical	45372	0.114271	0.091386
Linear	113772	0.044489	707.559454
Hierarchical	113772	0.294859	0.198096

From analysis of the results, it can be seen that insertion time for the linear data structure is always less than the proposed approach, but the difference between the measured times of the two approaches is not large. This is because in the proposed approach many calculations and insertions are performed for the different levels of the data structure. Also, it is important to consider that the number of insertion operations in a health-care organization are much less than the search operations. So, the small increase in the insertion time in our approach can be traded off for much faster searching performance.

Whereas, the proposed approach always takes much lesser time that the linear one. Also, as the size of the blockchain increases, the proposed approach becomes more and more faster than the linear approach.

Chapter 6

Concluding Remarks and Future Directions

6.1 Conclusion

NoSQL and distributed database such as Cassandra is a good choice for storing medical data of large organizations. Cassandra clearly have advantages like unstructured data model and high scalability. However, Cassandra only provides security for client to database node or inter-node communication. But there are several security vulnerabilities that can lead to stolen or tampered data. This thesis presented an approach to protect the confidentiality of sensitive data of a patient stored in a cloud database. The approach provides the directions to store data in a non-deterministically encrypted form and to search in such a encrypted database. The proposed approach tackles the problem of sharing encrypted data by different groups of user having different roles and permissions. Also, an approach to provide data integrity in a secure blockchain storage, and a method to efficiently search in such a blockchain, exploiting the properties of medical data is presented. The proposed method uses a hierarchical data structure that provides an asymptotic lower bound than a linear search. The proposed system provides real time query performance, while conforming to a high security standard.

6.2 Future work

There are many scopes of future research from the work presented in this thesis. In its current form, the proposed system is limited to simple select queries only. How to extend the proposed approach to support more complex queries can be investigated in future. Also, if a health-care organization has to continuously monitor a patient and securely store such real time streaming data, sharing of such encrypted, real-time data by different user groups can be of interest. The proposed system provides a method to validate data integrity in a secure manner, but in case of un-authorized modification, restoring the orignal data can also be a challnge.

Bibliography

Arasu, Arvind et al. (2014). "Querying encrypted data". In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, pp. 1259–1261.

Azaria, Asaph et al. (2016). "Medrec: Using blockchain for medical data access and permission management". In: *2016 2nd International Conference on Open and Big Data (OBD)*. IEEE, pp. 25–30.

Cassandra Query Language. `http://cassandra.apache.org/doc/latest/cql/index.html`. [Online], Accessed: 2019-05-20.

Cassandra Security Documentation. `http://cassandra.apache.org/doc/latest/operating/security.html`. [Online], Accessed: 2019-05-20.

Fu, Zhangjie et al. (2017). "Privacy-preserving smart semantic search based on conceptual graphs over encrypted outsourced data". In: *IEEE Transactions on Information Forensics and Security* 12.8, pp. 1874–1884.

Ganapathy, Vignesh et al. (2011). "Distributing data for secure database services". In: *Proceedings of the 4th International Workshop on Privacy and Anonymity in the Information Society*. ACM, p. 8.

Kantarcioğlu, Murat and Chris Clifton (2005). "Security issues in querying encrypted data". In: *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, pp. 325–337.

Kumari, Kritika, Sayantani Saha, and Sarmistha Neogy (2018). "Cost Based Model for Secure Health Care Data Retrieval". In: *International Symposium on Security in Computing and Communication*. Springer, pp. 67–75.

Kuo, Tsung-Ting, Hyeon-Eui Kim, and Lucila Ohno-Machado (2017). "Blockchain distributed ledger technologies for biomedical and health care applications". In: *Journal of the American Medical Informatics Association* 24.6, pp. 1211–1220.

Li, Rui et al. (2014). "Fast range query processing with strong privacy protection for cloud computing". In: *Proceedings of the VLDB Endowment* 7.14, pp. 1953–1964.

Liu, Paul Tak Shing (2016). "Medical record system using blockchain, big data and tokenization". In: *International conference on information and communications security*. Springer, pp. 254–261.

Mitra, Gaurav et al. (2018). "Accessing Data in Healthcare Application". In: *International Symposium on Security in Computing and Communication*. Springer, pp. 301–312.

Muzammal, Muhammad, Qiang Qu, and Bulat Nasrulin (2019). "Renovating blockchain with distributed databases: An open source system". In: *Future Generation Computer Systems* 90, pp. 105–117.

Roehrs, Alex et al. (2019). "Analyzing the performance of a blockchain-based personal health record implementation". In: *Journal of biomedical informatics*, p. 103140.

Saha, Sayantani, Priyanka Saha, and Sarmistha Neogy (2018). "Hierarchical metadata-based secure data retrieval technique for healthcare application". In: *Advanced Computing and Communication Technologies*. Springer, pp. 175–182.

Saha, Sayantani et al. (2016). "A cloud security framework for a data centric wsn application". In: *Proceedings of the 17th International Conference on Distributed Computing and Networking*. ACM, p. 39.

Xu, Yuqin et al. (2017). "ECBC: A high performance educational certificate blockchain with efficient query". In: *International Colloquium on Theoretical Aspects of Computing*. Springer, pp. 288–304.