# TITLE PAGE

**A BLOCKCHAIN IMPLEMENTATION**

**FOR SECURED VACCINE CERTIFICATES**

A Capstone Project Presented to the Graduate Program

College of Engineering and Technology

Pamantasan ng Lungsod ng Maynila

In Partial Fulfillment of the Requirements for the Degree

Master in Information Technology

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

By

Jennifer L. Fadriquela

Dr. Khatalyn E. Mata

Adviser

January 2022

# **TABLE OF CONTENTS**

[TITLE PAGE 1](#_Toc92700641)

[TABLE OF CONTENTS 2](#_Toc92700642)

[LIST OF FIGURES 3](#_Toc92700643)

[LIST OF TABLES 5](#_Toc92700644)

[INTRODUCTION 6](#_Toc92700645)

[1.1 Background of the Study 6](#_Toc92700646)

[1.2 Statement Problem 8](#_Toc92700647)

[1.3 Objectives of the Study 8](#_Toc92700648)

[1.4 Scope and Limitations 9](#_Toc92700649)

[1.5 Significance of the Study 9](#_Toc92700650)

[1.6 Definition of Terms 11](#_Toc92700651)

[REVIEW OF RELATED LITERATURE 12](#_Toc92700652)

[THEORETICAL FRAMEWORK 24](#_Toc92700653)

[METHODOLOGY 37](#_Toc92700654)

[4.1 Requirements Modeling 38](#_Toc92700655)

[4.2 Quick Design 41](#_Toc92700656)

[4.2.1 Context Diagram 41](#_Toc92700657)

[4.2.2 Data Flow Diagram 42](#_Toc92700658)

[4.2.3 Use Case Diagram 43](#_Toc92700659)

[4.2.4 Transactional Operation Diagram 44](#_Toc92700660)

[4.2.5 System Flowchart of the Proposed Application 45](#_Toc92700661)

[RESULTS AND DISCUSSION 51](#_Toc92700662)

[5.1 Functionalities of the System 51](#_Toc92700663)

[5.2 Implementation of Proof of Authority and Keccak Hash 59](#_Toc92700664)

[5.3 Implementation of Merkle DAG 75](#_Toc92700665)

[5.4 Security Scans 77](#_Toc92700666)

[CONCLUSIONS AND RECOMMENDATIONS 90](#_Toc92700667)

[LIST OF REFERENCES 92](#_Toc92700668)

# 

# **LIST OF FIGURES**

[Figure 1: Diagram of Proposed Solution 23](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700398)

[Figure 2: Hashing Process 24](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700399)

[Figure 3: Generic Blockchain Transactions 27](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700400)

[Figure 4: Clique PoA Block Creation Process 28](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700401)

[Figure 5: Merkle Tree Implementation using Hashes 30](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700402)

[Figure 6: DAG Illustration 30](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700403)

[Figure 7: Merkle DAG Implemented on a File System 31](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700404)

[Figure 8: Conceptual Framework 34](#_Toc92700405)

[Figure 9: Prototype Model Phases and Process 36](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700406)

[Figure 10: Vaccine Certificate File 38](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700407)

[Figure 11: Context Diagram 40](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700408)

[Figure 12: Data Flow Diagram 41](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700409)

[Figure 13: Use Case Diagram 42](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700410)

[Figure 14: Transactional Operation Diagram 43](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700411)

[Figure 15: Patient Registration Flowchart 44](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700412)

[Figure 16: Vaccine Record Creation Flowchart 45](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700413)

[Figure 17: File storage via IPFS and blockchain Flowchart 46](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700414)

[Figure 18: File Validation Flowchart 47](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700415)

[Figure 19: File Retrieval Flowchart 48](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700416)

[Figure 20: Patient Registration Screen 50](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700417)

[Figure 21: Patient Login Screen 51](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700418)

[Figure 22: Patient Home Screen (Upper Section) 52](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700419)

[Figure 23: Patient Home Screen (Lower Section) 53](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700420)

[Figure 24: Auto-Generated Vaccine Certificate 54](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700421)

[Figure 25: Vaccine Record Creation Screen 55](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700422)

[Figure 26: File Validation (Successful) 56](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700423)

[Figure 27: File Validation (Failure) 56](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700424)

[Figure 28: Scan Summary QR Code Screen 57](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700425)

[Figure 29: Node Blockchain Logs for File Upload 67](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700426)

[Figure 30: Detected Metamask Address of Log Sender 67](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700427)

[Figure 31: Metamask account details of vaccine file uploader 69](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700428)

[Figure 32: Metamask Notification of File Upload Transaction 70](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700429)

[Figure 33: Merkle DAG representing sample records 76](file:///C:\Repos\capstone\jeng\Capstone%202%20Docs\Chapters\FADRIQUELA,%20JENNIFER%20-%20Manuscript.docx#_Toc92700430)

# 

# **LIST OF TABLES**

[Table 1: Keccak Parameters Per Length 33](#_Toc92700441)

[Table 2: Generated Hash Value for Sample Record #1 76](#_Toc92700442)

[Table 3: Generated Hash Value for Sample Record #2 77](#_Toc92700443)

[Table 4: Securify Supported Vulnerabilities 79](#_Toc92700444)

[Table 5: Securify Results Summary 85](#_Toc92700445)

[Table 6: Slither Supported Vulnerabilities 86](#_Toc92700446)

[Table 7: Slither Results Summary 90](#_Toc92700447)

**Chapter One**

# **INTRODUCTION**

## **1.1 Background of the Study**

With the advancement of computer technology, electronic documentation and the use of electronic medical records have become more feasible. Medical records on a shared computer network that are read and written electronically on a relational database using a graphic user interface are referred to as electronic medical records. In the study entitled “A comparison of electronic records to paper records in mental health centers” (Tsai and Bond, 2007), they looked at three mental health facilities that had recently switched from paper to electronic medical records. Electronic records' documentation was shown to be more thorough and retrievable than paper records. As per the study, this finding can be a factor to take in when making treatment decisions.

In the study entitled “Perceived Benefits of Implementing and Using Hospital Information Systems and Electronic Medical Records” (Khalifa, 2018), they pointed out six ways EMRs could enable data accessibility and care organization: improving access to data during patient encounters, improving processes workflow, managing information overflow to clinicians, enhancing medical decision-making process care plans, supporting operational processes and improving financial data accessibility. They also emphasized that when a computer was used to retrieve patient information, physicians earned higher overall patient satisfaction rates, and when a computer was used to enter patient information, physicians received identical satisfaction rates.

The current technological advancements in the Philippines has yet to be manifested in its healthcare system. Though there were efforts from the government to adopt various modern tools, we are still miles behind other countries. On a study entitled “Barriers to the Adoption of Electronic Medical Records in Select Philippine Hospitals: A Case Study Approach” (Ebardo and Celis, 2019), identified barriers such as weak infrastructure, technology complexity and poor interface design of applications have made it difficult for various health organization to progress. Another study entitled “Barriers to Electronic Health Record System Implementation and Information Systems Resources: A Structured Review” (Gesulga et al., 2017), they determined another set of barriers to the adoption of EMRs in the Philippines namely: User resistance, lack of education and training, and concerns arising from data security. In a paper entitled “Identifying Healthcare Information Systems Enablers in a Developing Economy” (Ebardo and Tuazon, 2019), they discussed how the integration of existing information systems to be “paper-less” can produce potential savings. This is crucial given that the Philippines is still a developing country and has budget constraints to health systems.

The current pandemic situation poses another scenario for the state of EMRs in our country. Government has boosted efforts in immunizing majority of the population. Local Government Units (LGUs) had implemented varying strategies to keep proof and records of vaccinations. Areas inside the National Capital Region (NCR) have setup online web application to accommodate the vaccination process. Specifically, the city of Manila had employed a digitized way of keeping vaccination certificates and making them downloadable to its citizens. Other cities like Quezon City and Makati have a hybrid of online and manual processes. Although NCR cities have initiated the computerized way of the vaccination process, it is worth noting that majority of the Philippines (especially on province and remote areas) still utilize the pen and paper route.

## **1.2 Statement Problem**

At present, there is no unified system being implemented in the Philippines on Vaccination Certificates. Local Government Units (LGUs) have different strategies on their issuance of vaccination certificates. Most of them issue paper-based cards while some LGUs have web applications for their constituents to access the records. Security of these records is also in question as there are reports of people having tampered certificates to be used on various purposes. A news article from Philstar dated July 23, 2021 reports local LGUs warning the public against fake COVID-19 Vaccination cards.

## **1.3 Objectives of the Study**

This study aims to design and develop an application that will integrate blockchain and IPFS to ensure the integrity of vaccination data.

Specifically, the study seeks to address the following objectives:

1. To apply Proof of Authority (PoA) blockchain and Keccak Hash Algorithm in maintaining transactional records.
2. To apply hash tree concept of Merkle DAG for data storage.
3. To validate security aspects of the proposed application by using Solidity Security Audits and NIST-CSF (National Institute of Standards and Technology - Cybersecurity Framework)

## **1.4 Scope and Limitations**

The study will be focusing on developing an application for management of COVID-related records. Since there are privacy regulations concerning health information, the researcher will use dummy data and instead will probe more on the processes on how these records are archived or managed.

The study will exclude vaccine management such as scheduling. Thus, it will be focused on the results or outputs of these processes. The study assumes that information from the vaccine transaction are ready to be encoded in the system.

The study will only be concerned on Vaccine Certificates. The researcher will concentrate on developing an alternative storage system and accessibility strategy for medical units, patients and other verifying party.

## **1.5 Significance of the Study**

Results obtained from the study will benefit the following stakeholders:

**Patients**. Above all, patients will greatly benefit on this application. Various regulations and laws have been implemented to ensure people are not spreaders or vaccinated. Currently, there are no unified way in getting and presenting these records are proof. More so, bad actors are using this pandemic to make money out of tampering records. The application will help solve the woes of patients in terms on ease of access and portability of their records. They will also have full autonomy of said records.

**Medical Personnel**. The application will help medical workers to focus on their medical line of duty and alleviating various admin jobs.

**Third Party Validators.** As mentioned above, records tampering has become rampant. Businesses or employers requiring such records can now be protected of this illegal activity.

## **1.6 Definition of Terms**

Block - A place in a blockchain where information is stored and encrypted

Blockchain - A digital ledger of transactions that is duplicated and distributed across the entire network of computer systems

Blockchain network - A technical infrastructure that provides ledger and smart contract (chaincode) services to applications.

Cipher text - A series of randomized letters and numbers which humans cannot make any sense of.

Clique - PoA protocol implemented in Geth

Consensus Mechanism - A fault-tolerant mechanism that is used in computer and blockchain systems to achieve the necessary agreement on a single data value or a single state of the network among distributed processes or multi-agent systems, such as with cryptocurrencies.

Consensus protocol - Forms the backbone of blockchain by helping all the nodes in the network verify the transactions.

Content addressing - A way to find data in a network using its content rather than its location.

Content Identifier (CID) - A label used to point to material in IPFS. It doesn't indicate where the content is stored, but it forms a kind of address based on the content itself. CIDs are short, regardless of the size of their underlying content.

Cryptographic hash function - Process that converts data of arbitrary size (commonly referred to as the "message") into a fixed-size bit array ("hash value", "hash", or "message digest")

Cryptography - Science of secret writing with the intention of keeping the data secret.

Digital Envelope - A secure electronic data container that is used to protect a message through encryption and data authentication.

Digital Signature - A cryptographic value that is calculated from the data and a secret key known only by the signer.

Directed Acyclic Graph - It consists of vertices and edges (also called arcs), with each edge directed from one vertex to another, such that following those directions will never form a closed loop.

Distributed Hash Table (DHT) - A decentralized data store that looks up data based on key-value pairs.

Ethereum - A decentralized, open-source blockchain with smart contract functionality.

Genesis block - First block of a block chain.

Geth (Go Ethereum) - A command line interface for running Ethereum node implemented in Go Language.

Hash Digest - Output of the hash function.

Hash Table - A type of data structure that stores key-value pairs. The key is sent to a hash function that performs arithmetic operations on it.

Hashing - Process that calculates a fixed-size bit string value from a file.

InterPlanetary File System (IPFS) - A protocol and peer-to-peer network for storing and sharing data in a distributed file system.

Keccak - A family of hash functions that is based on the sponge construction, and hence is a sponge function family. Also known as SHA-3.

Merkle DAG - DAG where each node has an identifier, and this is the result of hashing the node's contents.

Merkle root - Created by hashing all the transaction hashes together in pairs — producing a unique hash for all the transactions in a block.

Merkle tree - A tree in which every "leaf" (node) is labelled with the cryptographic hash of a data block, and every node that is not a leaf (called a branch, inner node, or inode) is labelled with the cryptographic hash of the labels of its child nodes.

MetaMask - A software cryptocurrency wallet used to interact with the Ethereum blockchain.

Peer-to-Peer (P2P) Network - A group of computers are linked together with equal permissions and responsibilities for processing data.

Plain Text - Clear, basic unencrypted string of text.

Private Key - Used to decrypt cipher text to plain text and only available to its owner.

Proof of authority (PoA) - An algorithm used with blockchains that delivers comparatively fast transactions through a consensus mechanism based on identity as a stake.

Proof of work (PoW) - Describes a system that requires a not-insignificant but feasible amount of effort in order to deter frivolous or malicious uses of computing power, such as sending spam emails or launching denial of service attacks.

Proof-of-Stake (PoS) - A cryptocurrency consensus mechanism that requires you to stake coins, or set them aside, to be randomly selected as a validator.

Public Key - Used to encrypt plain text to cipher text and available to anyone accessing the application.

QR Code - A two-dimensional version of the barcode, typically made up of black and white pixel patterns.

Signer nodes - Authorized nodes in the blockchain to propose a block

Smart Contract Weakness Classification (SWC) - A smart contract specific software weakness classification scheme for developers, tool vendors and security practitioners.

Smart contracts - Programs stored on a blockchain that run when predetermined conditions are met

Solidity - An object-oriented programming language for implementing smart contracts on various blockchain platforms, most notably, Ethereum.

Sponge function - Any of a class of algorithms with finite internal state that take an input bit stream of any length and produce an output bit stream of any desired length.

**Chapter Two**

# **REVIEW OF RELATED LITERATURE**

This chapter covers studies and other literatures carried out by foreign and domestic researchers that have a significant impact on the variables investigated in this study. These studies focus on several factors that will help with the research's development. Literatures mentioned here will be of different sources: books, journals, articles, electronic materials such as PDF or E-Book, and other existing thesis and dissertations, foreign and local. Their inclusion will be considered supplemental in developing the proposed solution of this study.

**Merkle Tree**

In 1989, Ralph Merkle introduced the Merkle tree in his paper “A Certified Digital Signature”. The Merkle tree is a tree constructed bottom-up. More precisely, the tree discussed in this paper is a full binary tree and constructed from the bottom-up. Assume that the height of the tree is *hm*, and the tree owns 2hm data blocks *xi* and *yi=hash(xi),i∈[0,2hm−1]*, where *yi* is a leaf node value of the Merkle tree. Each value of the parent node is the hash of the concatenation of its children, *yparent=hash(yleft|yright)*, where | refers to concatenation. Below is a pseudocode format of the Classic Merkle Tree Traversal algorithm:

1. Set *leaf* = 0.

2. Output:

• Compute and output *leaf* with *LEAFCALC(leaf)*

• For each *h* ∈ [0,*H* − 1] output {*authh*}.

3. Refresh Auth Nodes:

For h such that 2h divides leaf + 1:

• Set authh be the sole node value in stackh.

• Set *startnode* = (*leaf* + 1 + 2h) ⊕ 2h.

• *stackh.initialize(startnode,h)*.

4. Build Stacks:

For all *h* ∈ [0,H − 1]:

• stackh.update(2).

5. Loop

• Set *leaf* = *leaf* + 1.

• If *leaf* < 2*H* go to Step 2

A Logarithmic Merkle Tree Traversal was proposed by M. Szydlo (2003). The main idea of the improved algorithm is, to reduce the memory requirements, by reducing the number of active treehash instances during the signature generation.. Here is the pseudocode:

1. Set *leaf* = 0.

2. Output:

• Compute and output leaf with *LEAFCALC*(*leaf*)

• For each *h* ∈ [0,*H* − 1] output {*authh*}.

3. Refresh Auth Nodes:

For *h* such that 2*h* divides *leaf* + 1:

• Set *authh* be the sole node value in *stackh*.

• Set *startnode* = (*leaf* + 1 + 2*h*) ⊕ 2*h*.

• *stackh.initialize(startnode,h).*

4. Build Stacks:

Repeat the following 2*H* − 1 times:

• Let *lmin* be the minimum of {*stackh.low*} for all *h* = 0,...,*H* − 1.

• Let focus be the least *h* so that *stackh.low* = *lmin*.

• *Stackfocus.update*(1).

5. Loop

• Set *leaf* = *leaf* + 1.

• If *leaf* < 2*H* go to Step 2.

In Fractal merkle tree representation (Micali et al., 2003) and traversal, the goal is to divide the merkle tree in subtrees and to preserve and compute these subtrees, instead of single nodes. Below is the pseudocode:

1. Set *leaf* = 0.

2. Output:

• Compute and output leaf with *LEAFCALC*(*leaf*)

• For each *j* ∈ [0,*H* − 1] output {*authj*}.

3. Next Subtree:

For each *i* for which *Existi* is no longer needed, i.e., for *i* ∈ {1, 2,...,*L*} with *leaf* = 1(*mod*2*hi*):

• Set *Existi* = *Desirei*.

• Create new empty *Desirei* (if *leaf* + 2*ih* < 2*H*).

4. Grow Subtrees

For each *i* ∈ {1, 2,...,*h*}: Grow tree *Desirei* by applying 2 units to modified treehash (unless *Desirei* is completed)

5. Increase *leaf* and return back to step 2 (while *leaf* < 2*H*).

**Distributed Hash Tables**

Distributed Hash Tables (DHTs) are widely utilized to manage metadata for peer-to-peer systems. For example, the BitTorrent MainlineDHT monitors sets of peers’ part of a torrent swarm. Kademlia was introduced in a paper titled “Kademlia: A peer-to-peer information system based on the xor metric” (Maymounkov and Mazieres, 2002). It is a DHT which provides:

1. Efficient lookup through massive networks: queries on average contact dlog2(n)e nodes.

2. Low coordination overhead: it optimizes the number of control messages it sends to other nodes.

3. Resistance to various attacks by preferring long-lived nodes.

4. Wide usage in peer-to-peer applications, including Gnutella and BitTorrent, forming networks of over 20 million nodes.

In “Democratizing content publication with coral” (Freedman et al., 2004), it examined Coral DSHT as an extension of Kademlia in three particularly important ways:

1. Kademlia stores values in nodes whose ids are “nearest” (using XOR-distance) to the key.

2. Coral relaxes the DHT API from get\_value(key) to get\_any\_values(key) (the “sloppy” in DSHT).

3. Additionally, Coral organizes a hierarchy of separate DSHTs called clusters depending on region and size.

Another approach, S/Kademlia DHT (Baumgart and Mies. 2007) extends Kademlia to protect against malicious attacks in two particularly important ways:

1. S/Kademlia provides schemes to secure NodeId generation, and prevent Sybill attacks

2. S/Kademlia nodes lookup values over disjoint paths, in order to ensure honest nodes can connect to each other in the presence of a large fraction of adversaries in the network.

Xie (2003) discussed how DHTs are implemented in P2P systems in his paper “P2P Systems based on Distributed Hash Table”. Files are connected with keys (which are generated by hashing the file name); each node in the system is responsible for storing a specific range of keys and handles a fraction of the hash space. The system will return the identity (e.g., the IP address) of the node storing the object with that key after a lookup for that key. The DHT capability allows nodes to put and get files based on their key and has shown to be a viable substrate for large distributed systems, with a number of projects proposing to overlay Internet-scale services on top of DHTs. Each node in a DHT is in charge of a specific key range and a portion of the hash space. Routing is a distributed lookup that is location-deterministic. Deterministic locating and load balance are the most significant improvements.

• No global knowledge

• Absence of single point of failures

**Blockchain**

Blockchains are a sort of decentralized distributed ledger and usually anonymous groups of agents rather than known centralized parties. This novel method of recordkeeping has introduced two economic innovations that overcome the two limitations of competition among centralized ledgers. The entry of record-keepers is unrestricted: any agent may write on the ledger as long as they follow a set of regulations. Furthermore, information on an existing blockchain is portable to a competing one. A software developer can propose to “fork off” an existing blockchain to establish one with different policies while retaining all the information contained in the original blockchain. Fork competition eliminates the inefficiencies arising from switching costs in centralized record-keeping systems (Abadi and Brunnermeier, 2018).

On an article “Blockchain Technology Overview” (Yaga et al. 2018), they mentioned four key characteristics of this technology:

• Ledger – the technology uses an append only ledger to provide full transactional history. A blockchain, unlike traditional databases, does not allow transactions and values to be overwritten.

• Secure – blockchains are cryptographically secure, ensuring that the data in the ledger has not been changed with and that the data is attestable.

• Shared – multiple participants will share the ledger. This provides transparency across the node participants in the blockchain network.

• Distributed – the blockchain can be distributed. This lets a blockchain network's number of nodes to be scaled up to make it more resilient to bad actors' attacks. By expanding the number of nodes, a bad actor's capacity to influence the blockchain's consensus procedure is lessened.

Like a traditional public ledger, blockchain is a series of blocks that carry a comprehensive list of transaction data. A block has just one parent block if the block header contains a preceding block hash. It's worth mentioning that hashes for uncle blocks (children of the block's ancestors) would be saved as well. The ﬁrst block of a blockchain is called genesis block which has no parent block (Zheng et al., 2017).

In the article of Monrat et al. (2019) titled “A Survey of Blockchain From the Perspectives of Applications, Challenges, and Opportunities”, they identified six comparison perspectives when comparing blockchain networks:

1. Consensus Determination - All the nodes can participate in the consensus process in the public blockchain such as Bitcoin, while only a few selected set of nodes are being responsible for confirming a block in the consortium blockchain. In the private blockchain, a central authority will decide the delegates who could determine the validated block.

2. Read Permission - Public blockchain allows read permission to the users, where the private and consortium can make restricted access to the distributed ledger. Therefore, the organization or consortium can decide whether the stored information needs to be kept public for all or not.

3) Immutability - In the decentralized blockchain network, transactions are stored in a distributed ledger and validated by all the peers, which makes it nearly impossible to modify in the public Blockchain. In contrast, the consortium and private Blockchain ledger can be tampered by the desire of the dominant authority.

4) Efficiency - In the public blockchain, any node can join or leave the network which makes it highly scalable. However, with the increasing complexity for the mining process and the flexible access of new nodes to the network, it results in limited throughput and higher latency. However, with fewer validators and elective consensus protocols, private and consortium blockchain can facilitate better performance and energy efficiency.

5) Centralized - The significant difference among these three types of Blockchain is that the public blockchain is decentralized, while the consortium is partially centralized and private blockchain is controlled by a centralized authority.

**Proof-of-Authority**

Proof of Authority (PoA) is a group of permissioned blockchain consensus algorithms that have gained popularity due to improved performance over traditional BFT algorithms due to fewer message exchanges. PoA was first proposed as part of the Ethereum ecosystem for private networks, and it was implemented in the Aura and Clique clients. The authorities are a group of N trusted nodes that PoA algorithms rely on. Each authority is identifiable by a unique id, and a majority of them, precisely at least N/2 + 1, is believed to be trustworthy. To execute the transactions issued by clients, the authorities run a consensus. The mining rotation schema, a commonly used way to fairly spread the burden of block creation across authority, is used to achieve consensus in PoA algorithms. Time is split into steps, each of which has a mining leader elected by the nodes. (Bitfury Group and Garzik, 2015).

There are two main PoA algorithms currently: AuRa and Clique. Aura (Authority Round) is the PoA algorithm implemented in Parity, the Rust-based Ethereum client. It is expected that the network is synchronous and all authorities to be synchronized within the same UNIX time t. The index s of each step is deterministically computed by each authority as s = t/step\_duration, where step\_duration is a constant determining the duration of a step. The leader of a step s is the authority identified by the id l = s mod N. Clique is the PoA algorithm implemented in Geth, the GoLang-based Ethereum client. The algorithm proceeds in epochs which are identified by a prefixed sequence of committed blocks. When a new epoch starts, a special transition block is broadcasted. It specifies the set of authorities (i.e., their ids) and can be used as snapshot of the current blockchain by new authorities needing to synchronize (De Angelis et al., 2018).

**Keccak**

The National Institute of Standards and Technology (NIST) has published a family of cryptographic hash functions called the secure hash algorithm which is recognized as a U.S. Federal Information Processing Standard (Sukrutha and Latha 2013).

* SHA-0: The original version of 160 bit hash algorithm published in 1993 called SHA was withdrawn right after it was released because of an undisclosed problem. SHA -1 was released which was a modified version of SHA-0.
* SHA-1: It was designed by the National Security Agency (NSA) to be part of digital signature algorithm. It is a160 bit hash function which similar to MD5 algorithm. Nevertheless, its standard was no longer approved for most of the cryptographic uses after 2010 because of its weaknesses.
* SHA-2: It has two similar hash functions called SHA -256 and SHA-512.They have different block sizes and also different word sizes.SHA-256 uses 32-bit words whereas SHA-512 uses 64-bit words. There are also modified versions of both the above algorithms called SHA-224 and SHa-384 which were also designed by the NSA.
* SHA-3: It is also known as Keccak. It was chosen in 2012 from a competition among nonNSA designers. It uses same hash length as SHA-2 and the internal structure of Keccak differs from the SHA family.

In October 2012, the American National Institute of Standards and Technology (NIST) announced the selection of Keccak as the winner of the SHA-3 Cryptographic Hash Algorithm Competition. At the core of Keccak is a set of seven permutations called Keccak-f, with width b chosen between 25 and 1600 by multiplicative steps of 2. Depending on b, the resulting function ranges from a toy cipher to a wide function. The instances proposed for SHA-3 use exclusively Keccak-f for all security levels, whereas lightweight alternatives can use for instance Keccak-f or Keccak-f, leaving Keccak-f as an intermediate choice. Inside Keccak-f, the state to process is organized in 5 × 5 lanes of b/25 bits each, or alternatively as b/25 slices of 25 bits each. The round function processes the state using a non-linear layer of algebraic degree two (χ), a linear mixing layer (θ), inter- and intra-slice dispersion steps (ρ, π) and the addition of round constants (ι). The choice of operations in Keccak-f makes it very different from the SHA-2 family or even Rijndael (AES). On the implementation side, these operations are efficiently translated into bitwise Boolean operations and circular shifts, they lead to short critical paths in hardware implementations, and they are well suited for protections against side-channel attacks (Bertoni 2013).

SHA-3 (and its variants SHA3-224, SHA3-256, SHA3-384, SHA3-512), is considered more secure than SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512) for the same hash length. For example, SHA3-256 provides more cryptographic strength than SHA-256 for the same hash length (256 bits). The SHA-3 family of functions are representatives of the "Keccak" hashes family, which are based on the cryptographic concept "sponge construction". Keccak is the winner of the SHA-3 NIST competition. Unlike SHA-2, the SHA-3 family of cryptographic hash functions are not vulnerable to the "length extension attack". SHA-3 is considered highly secure and is published as official recommended crypto standard in the United States (Nakov 2018).

Keccak, specifically Keccak-256, hash algorithm is used in Ethereum. The block header in the Ethereum blockchain contains the Keccak 256-bit hash of the parent block's header, the mining fee recipient's address, hashes of the roots of state, transaction, and receipts tries, the difficulty, the current block gas limit, a number representing total gas used in the block transactions, timestamp, nonce, and several extra hashes for verification purposes (Vujicic et al., 2018).

**Decentralized Storage, Blockchain and Medical Records**

MedRec, a system proposed by Azaria el.al (2016) shows how principles of decentralization might be applied to largescale data management in an EMR system by using blockchain technology. It utilized Proof-of-Work consensus in mining transaction blocks. Patient data are stored in centralized SQL server while transaction logs of updating patient records are in the Ethereum blockchain. A study by Sharma et al. (2020) did a similar EMR model but introduced cloud storage as an alternative to a centralized on-premise server. These two studies posed limitations on storing files. Though Sharma attempted to solve this by putting a cloud application layer, Cloud providers will have autonomy to data stored in their servers.

Kumar and Tripathi (2020) presented a distributed framework handling COVID-19 patient reports. It utilized Proof-of-Work blockchain and IPFS to decentralize data storage. However, the system has no patient access interface and only shares data for provider use only. Wu and Du (2019) also added IPFS on their Delegated Proof-of-Stake blockchain implementation of EMR. They also used data-masking to protect patient data once uploaded on the network and specified Digital Imaging and Communications in Medicine (.dcm) image format of files to be uploaded. Like Kumar and Tripathi, system did not provide data access to patients.

Sun et al. (2020) proposed attribute-based encryption for EMRs with IPFS and blockchain implementation. The scheme provides good access control for the electronic medical records using attribute-based encryption technology so that people who are not related to the patient cannot see the private data of the patient without authorized. Khubrani (2021) proposed a proposed a theoretical blockchain-based framework via blockchain, IPFS and asymmetric encryption but did not mention technical specifications on how these technologies will integrate with one another.

At this point, related studies mentioned above either used Proof-of-Work (PoW) or Proof-of-Stake (PoS) as their consensus scheme for EMRs. A comparative study of existing literature for EMR system based from blockchain and IPFS was presented by Kumar et al. (2021). It compared different metrics such as Technology used, Cost-effectiveness, Complexity and Shortcomings. Most of the shortcomings were implementation-related such as lack of data formatting and workflow for data sharing, but the authors gave emphasis on the need of a cost-effective way to deploy blockchain as an immutable ledger since most of the studies were using Proof-of-Work as a consensus scheme.

On a paper by Al Asad et al. (2021), they proposed a theoretical blockchain-based framework with Proof-of-Authority (PoA) as the consensus scheme. It cited comparisons among other consensus (Proof-of-Work and Proof-of-Stake) and shown why PoA is a better alternative for EMRs. However, this paper only examined the feasibility of PoA consensus implementation and did not dwell on strategies for decentralized file storage and encryption. Reen (2019) on an earlier study, also mentioned PoA as an excellent choice for medical records. He made a conceptual model on IPFS as a decentralized file storage but did not provide technical specification about PoA and how it will be integrated in the system.

**Chapter Three**

# **THEORETICAL FRAMEWORK**

**Present State of COVID-19 Vaccine Certificate Storage**

Documents and certificates given out by various units (private and public) for COVID-19 are still on paper-form. There are some units that store the results in their server and can be accessed online thru their website. Same is true with giving out vaccine certificates.

Primary providers of vaccines are Local Government Units (LGUs) and they vary in implementation. Some only give out physical copies (certificates, cards) and others have virtual copies on their websites stored on their servers. There is a disconnect on a unified tracking of all these documents and might result to issues when these documents will be used on different areas of the Philippines. The usual proposition to solve this is to create a unified website that will be hosted in a central server.

**Proposed Documents Storage Structure**

Figure : Diagram of Proposed Solution

Above is a summarized approach in solving the problem in document storage (as shown in Figure 1). The main components of this application will be the IPFS for file storage and blockchain to record the logs of transaction being done in the system.

The next sections will discuss the different algorithms and frameworks to be used in order to achieve the proposed solution.

**Cryptographic Hash Functions**

A cryptographic hash function is a process that converts data of arbitrary size (commonly referred to as the "message") into a fixed-size bit array ("hash value", "hash", or "message digest"). A one-way function, which means that inverting or reversing the computation is almost impossible. The only way to identify a message that generates a particular hash is to try a brute-force search of all potential inputs to see whether any of them create a match, or to use a rainbow table of matched hashes. Cryptographic hash functions are a primary instrument of modern cryptography.

The following are the major characteristics of an ideal cryptographic hash function:

- it is deterministic, meaning that the same message always results in the same hash

- it is quick to compute the hash value for any given message

- it is impossible to generate a message that produces a given hash value

- it is infeasible to find two different messages with the same hash value

- a small change to a message should alter the hash value in such a way that a new hash value appears to be unrelated to the old hash value



Figure : Hashing Process

The majority of cryptographic hash functions accept any length string as input and return a fixed-length hash value. Figure 2 depicts the general process of a hash function.

A cryptographic hash function must be cryptanalytically resistant to all known types of attacks. The security level of a cryptographic hash function has been determined using the following properties in theoretical cryptography:

* Pre-image resistance

Given a hash value h, it should be hard to determine any message m such that h = hash(m). This concept is connected to that of a one-way function. Functions that do not have this property are susceptible to preimage attacks.

* Second pre-image resistance

Given an input m1, it should be hard to determine a different input m2 such that hash(m1) = hash(m2). This property is occasionally stated to as weak collision resistance. Functions that do not have this attribute are susceptible to second-preimage attacks.

* Collision resistance

It should be hard to determine two different messages m1 and m2 such that hash(m1) = hash(m2). Such a pair is referred as cryptographic hash collision. This attribute is occasionally called as strong collision resistance. It needs a hash value at least twice as long as that required for pre-image resistance; or else collisions may be identified by a birthday attack.

**Blockchain**

In 2008, Satoshi Nakamoto released a whitepaper titled “Bitcoin: A peer-to-peer electronic cash system”. This paper proposed a system for electronic transactions which uses a peer-to-peer network. Participating nodes in the network utilize Proof-of-Work to record public history of transactions.

At its most basic level, blockchain technology permits a network of computers to have a consensus on the true status of a distributed ledger at regular intervals. Blockchain network users submit potential transactions to participating nodes. The network will then choose a publishing node to update the pending transaction. Once this is done, transaction will be propagated to non-publishing nodes. Transactions are logged chronologically – with information being passed from the first transaction (or blocks) up to the last. This repetitive process forms an immutable chain on which all blocks are interconnected with each other.

Transactions are inserted to the blockchain when a publishing node creates a block. A block may represent various types of data from simple texts to complicated ones such as digital rights or intellectual property. It is divided into two parts, header and body. Header contains metadata and body is for the actual data being persisted in the blockchain. Below is a typical specification of these 2 parts:

1. Block Header

* Previous block header’s hash value
* Hash representation of block data
* Timestamp
* Size of the block
* Nonce value. In Bitcoin and other Proof-of-Work blockchains, this is a number manipulated by the publishing node to solve the hash puzzle.

1. Block Data

* Actual data (text, files)

Figure : Generic Blockchain Transactions

Figure 3 shows how blockchain works given we have a simple data of text. The initial block is referred to the genesis block and is automatically generated upon the chain’s creation. This genesis block will be the seed and considered as reference of all blocks going forward. Blocks are linked through each block containing the hash value of the previous block’s header, thus creating the chain. In case a previously published block was changed, it will have a different hash. This will create a domino effect on all subsequent blocks to also have a different hash because they contain the hash of the altered block.

An essential part of the blockchain is identifying which user will publish the next block or become the next publishing node. This is solved by implementing a consensus model. The common model used is to compete on who will publish it and winning an incentive in doing so.

Once a user joins a blockchain network, they agree to the preliminary state of the system. This is documented in the only pre-configured block or the genesis block. Each blockchain network have a genesis block on to which all subsequent blocks would reference to. Each block must be valid and can be validated independently by each blockchain network user.

**Proof of Authority (POA) - Clique**

In a Proof of Authority (PoA) consensus algorithm, a set of trusted nodes called Authorities, each recognized by their unique identifier, are responsible for mining and validating the blocks in the blockchain. Clique is a PoA protocol implemented in Geth. Figure 4 depicts the block creation process for this algorithm.

****

Figure : Clique PoA Block Creation Process

The Clique consensus protocol adheres to the following rules:

- Set of trusted authorities are referred to as the "Signers"

- Process of mining a block is referred to as "Sealing a block"

- WHEN the next block is identified by BLOCK\_NUMBER and the number of signers is identified by SIGNER\_COUNT

AND the signers are lexicographically sorted by their unique identifiers in a list

THEN the next block is sealed by the signer located at the index

BLOCK\_NUMBER % SIGNER\_COUNT, where % is the modulus operator

The signers compile and execute network transactions into a block, updating the world state. At the fixed interval referred to as the BLOCK\_PERIOD, the next signer in the list (identified by BLOCK\_NUMBER % SIGNER\_COUNT) calculates the hash of the block and then signs the block using its private key (sealing the block). The sealed block is then broadcast to all nodes in the network.

**InterPlanetary File Storage (IPFS)**

IPFS is a distributed platform for storing and retrieving files, websites, applications and data. It has rules that regulate in what manner data and content move around on the network. These rules are similar to Kademlia, the peer-to-peer distributed hash table (DHT) popularized by its use in the BitTorrent protocol.

IPFS is essentially a peer-to-peer system for getting and sharing IPFS objects. An IPFS object is a data structure have two fields:

* Data: a blob of unstructured binary data of size < 256 kB.
* Links: an array of Link structures. These are links to other IPFS objects. Links have 3 sub-parts:
  + Name: the name of the Link.
  + Hash: the hash of the linked IPFS object.
  + Size: the cumulative size of the linked IPFS object, including following its links.

IPFS builds a Merkle DAG, a blend of a Merkle Tree and a Directed Acyclic Graph (DAG).

A Merkle tree summarizes all of the transactions in a block by generating a digital fingerprint of the complete collection of transactions, allowing a user to check whether or not a transaction is included in the block. Merkle trees are made by hashing pairs of nodes repeatedly until only one hash remains (this hash is called the Root Hash, or the Merkle Root). They are built from the ground up, utilizing individual transaction hashes (known as Transaction IDs). Each non-leaf node is a hash of its previous hashes, while each leaf node is a hash of transactional data. Merkle trees are binary, hence an even number of leaf nodes is required. The last hash will be repeated once to establish an even number of leaf nodes if the number of transactions is odd.



Figure : Merkle Tree Implementation using Hashes

A directed acyclic graph (DAG) is a visual representation of a sequence of events. A graph depicting the order of the activities is visually portrayed as a group of circles, each representing an activity, some of which are connected by lines, which represent the flow from one action to the next. Each circle is referred to as a "vertex," and each line is referred to as a "edge". "Directed" signifies that each edge has a specific direction, implying that each edge reflects a single directional flow from one vertex to the next. The term "acyclic" refers to a network that contains no loops (or "cycles"), meaning that if you follow an edge connecting one vertex to another, there is no way to return to the original vertex.



Figure : DAG Illustration

A Merkle DAG is a DAG in which each node has an identification that is generated by hashing the content of the node — any opaque payload carried by the node, as well as a list of its children's identifiers — by utilizing a cryptographic hash function like SHA256. This brings some important considerations:

* Merkle DAGs can only be built from the leaves, or nodes that have no offspring. Parents come after children because the identifiers for the children must be computed ahead of time in order to link them. Every node in a Merkle DAG is the root of a (sub)Merkle DAG, and the parent DAG contains this subgraph.
* Merkle DAG nodes cannot be changed. Any change to a node's identity would affect all ascendants in the DAG, effectively resulting in the creation of a new DAG.
* Merkle DAGs are like Merkle trees, but they don't have to be balanced, and each node can have a payload. Many branches can re-converge in DAGs, or, to put it another way, a node can have multiple parents.

 Content addressing is the process of identifying a data object (such as a Merkle DAG node) based on the value of its hash. As a result, the node identifier is referred to as the Content Identifier, or CID.

Figure : Merkle DAG Implemented on a File System

**Keccak**

The hash algorithm used in Clique is Keccak, since Clique is based off Ethereum.

Keccak is a family of hash functions that is based on the sponge construction, and hence is a sponge function family. In Keccak, the underlying function is a permutation chosen in a set of seven Keccak-f permutations, denoted Keccak-f[b], where b∈{25,50,100,200,400,800,1600} is the width of the permutation. The width of the permutation is also the width of the state in the sponge construction.

The state is organized as an array of 5×5 lanes, each of length w∈{1,2,4,8,16,32,64} and b=25w. When implemented on a 64-bit processor, a lane of Keccak-f[1600] can be represented as a 64-bit CPU word.

We obtain the Keccak[r,c] sponge function, with parameters capacity c and bitrate r, if we apply the sponge construction to Keccak-f[r+c] and by applying a specific padding to the message input.

The parameters defining the standard instances are given in the table below (Table 1). Parameters of the standard FIPS 202 and SP 800-185 instances. The values of Mbits and d assume that the input to these functions is made of bytes.

Table : Keccak Parameters Per Length

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***r*** | ***c*** | **Output length (bits)** | **Security level (bits)** | **Mbits** | **d** |
| **SHAKE128** | 1344 | 256 | unlimited | 128 | 1111 | 0x1F |
| **SHAKE256** | 1088 | 512 | unlimited | 256 | 1111 | 0x1F |
| **SHA3-224** | 1152 | 448 | 224 | 112 | 1 | 0x06 |
| **SHA3-256** | 1088 | 512 | 256 | 128 | 1 | 0x06 |
| **SHA3-384** | 832 | 768 | 384 | 192 | 1 | 0x06 |
| **SHA3-512** | 576 | 1024 | 512 | 256 | 1 | 0x06 |
| **cSHAKE128** | 1344 | 256 | unlimited | 128 | 0 | 0x04 |
| **cSHAKE256** | 1088 | 512 | unlimited | 256 | 0 | 0x04 |

The value of the capacity c and of the suffix Mbits jointly provide domain separation between the different instances. Because their input to Keccak never collide, domain-seprated instances will give unrelated outputs and act as independent functions.

**Conceptual Framework**

Diagram

Description automatically generated

Figure : Conceptual Framework

The users of the proposed application will be patients, medical workers or other third-party requiring the patient to present a COVID-19 Vaccine Certificate. The users will access the same application but with different levels of access depending on their role.

The input are the medical documents and distribution key. There will be different types of keys which will be discussed on Chapter 4. These keys will be used to authenticate and unlock or lock the files.

Once all required inputs are provided, the file will now go thru the necessary steps to access it. Depending on the type of transaction (insert a new file or retrieval), the keys provided should have enough privilege for it to succeed. The file hash will be then stored in the blockchain after going thru smart contracts. Once the blockchain successfully updated the network, provided file will now become an immutable component of both IPFS and blockchain network.

**Chapter Four**

# **METHODOLOGY**

Diagram

Description automatically generatedThis chapter provides an overview of the strategies used to attain the study's goals. It describes the study's respondents as well as the research instruments that were used. It then goes on to explain the data collection strategies that contributed in the completion of the study endeavor.

Figure : Prototype Model Phases and Process

Figure 9 illustrates Prototype Model used by the researcher in developing the proposed study entitled “A Blockchain Implementation for Secured Vaccine Certificates” which is under the family of System Development Life Cycle (SDLC). Prototyping was used to ensure faster turnaround time on each phase while addressing client’s requirements and feedbacks. This model also enables the researcher and client to have discussions in between development cycles.

The next sections of this chapter will discuss the phases of the used model.

## **4.1 Requirements Modeling**

The Prototype Model starts with outlining the requirements. The researcher will conduct an initial investigation to determine the purpose and utilization of the application coupled with the nature and scope of the study. It is also in this stage that the researcher requested permission from medical unit authorities and other parties to conduct the study and all relevant data and information were examined.

Fact-finding was used via interviews and probing of processes to build a logical model of the application. With these investigation, the researcher was able to piece out a picture of transactions involved and analyzed them against the proposed solution. This information will also enable the researcher to identify critical decisions geared toward implementing the application.

For vaccination records, citizens are encouraged to register online via the web portal. This will ensure a scheduled slot on a specified date. On the day of vaccination, patient will be checked up by a physician to ensure he is fit for vaccination. The physician’s findings are logged on the system. Upon issuing a go signal, patient can now be vaccinated. After vaccination, vaccination site will sign a vaccination card while tagging the patient in their system as fully vaccinated.

The study will be focused on Vaccine Certificates. Mocked test data will be used and will only be for the purpose of this research. This is due to various privacy regulation such as Health Insurance Portability and Accountability Act (HIPAA). This is a United States created health law adopted by medical facilities in the Philippines.

**Qr code

Description automatically generated**

Figure : Vaccine Certificate File

Below are requirements grouped by specific role:

Patient

* Register and Login – register to gain access to the system
  + Upon registration, system will create private and public keys to be used for data encryption
* Download Vaccine Certificate
* View QR Code for Vaccine Record Summary
* View Record Summary Details

Verifying Third Party

* Publicly Available
* Validate Vaccine Certificate if existing in system

Physician/Medical Unit

* Register and Login – register to gain access to the system
* Create vaccine record for patient

## **4.2 Quick Design**

After identifying the requirements, a design of the proposed application is created. This is not a detailed design with complete technical specifications but a simplified one with critical aspects of the solution. This phase will give a bird’s eye view to the client of the application.

## **4.2.1 Context Diagram**

**Diagram

Description automatically generated**

Figure : Context Diagram

The context diagram shown in Figure 4.0.6 summarizes the application on inputs and outputs of the system and targeted users. On general, users of the application will be required to provide public/private keys and vaccine details. Application will generate vaccine certificate and summary details based off these details.

Then it will trigger and execute various processes to upload, encrypt/decrypt, or release files. Note that this is a general illustration of inputs and outputs. Next sections of this chapter will discuss the mentioned processes on this diagram.

## **4.2.2 Data Flow Diagram**

**Diagram

Description automatically generated**

Figure : Data Flow Diagram

Figure 4.0.7 illustrates how various types of users receives and provides information to the application and how the application provides and receives data from users. This also mentions the executing process to generate the data.

It is important to note that authorized medical personnel are the only ones allowed to upload files. Patients will have to generate private and public keys for their files to be uploaded or requested. These keys are crucial for a patient file to be encrypted or decrypted. Third parties can request for patient files and will be granted access to view decrypted files.

## **4.2.3 Use Case DiagramDiagram Description automatically generated**

Figure : Use Case Diagram

The suggested application's development is not solely dependent on the system's functionality. It also depends on the workflow procedure that needs to be identified, implemented, and followed. The components of the proposed application “A Blockchain Implementation for Secured Vaccine Certificates”, is demonstrated in Figure 4.0.8 and utilized a Use Case Diagram. The patient, being the central user of this system will provide appropriate keys with reference to the executing process. These in turn can trigger uploading or granting of view access to either medical unit or a third party.

## **4.2.4 Transactional Operation Diagram**

Table

Description automatically generated with low confidence

Figure : Transactional Operation Diagram

Figure 4.0.9 illustrates the operations that exist in the proposed application. It is divided according to the users triggering the process. The crucial process of generating the private and public keys will be prompted by the patient. Without these keys, medical personnel cannot upload files which in turn, the third parties will not be able to request any file validation.

## **4.2.5 System Flowchart of the Proposed Application**

Diagram

Description automatically generated

Figure : Patient Registration Flowchart

Diagram

Description automatically generated

Figure : Vaccine Record Creation Flowchart

Diagram

Description automatically generated

Figure : File storage via IPFS and blockchain Flowchart

Diagram

Description automatically generated

Figure : File Validation Flowchart

Diagram

Description automatically generated

Figure : File Retrieval Flowchart

Figure 15 shows patient registration process. Once the patient successfully register, public and private keys will be generated based off patient’s details. For confidentiality, private key will be masked by user’s password. User’s password will then be hashed before saved to the database.

Figure 16 details the process on creating vaccine record. Users will be required to scan a QR code with valid patient details. If this is not met, user is not allowed to enter vaccine details. If met, user will be required to input details such as Dose Number, Vaccine Brand etc. Once submitted, app will generate vaccine file based from these details and be stored with the app’s storage mechanism.

Figure 17 illustrates how data will be stored. First, the patient public key will be checked and if exists, will proceed to storage process. App will first generate a file hash from the raw file then proceeds on encrypting the file using the public key. Next, encrypted file will be sent to IPFS and IPFS will return a Content Identifier. Both filehash and Content Identifier will be sent to the blockchain.

Figure 18 describes the method for file validation. User will be allowed to upload the raw file. App will generate a hash from the file and said hash will be checked against existing blockchain logs. User will receive either a successful or error message for the transaction.

Figure 19 demonstrates file retrieval process. Patient’s metadata is automatically generated once logged in. Once metadata is validated from the blockchain, app will request encrypted file from IPFS and decrypt it using patient’s private key then will be available to the patient via browser download.

**Chapter Five**

# **RESULTS AND DISCUSSION**

This section presents, analyzes, and interprets the results of the study in developing a Blockchain Implementation for Secured Vaccine Certificates. It also presents the results from Solidity Security Scans.

## **5.1 Functionalities of the System**

This section will tackle the various functionalities included in the application and will also show screenshots of the development.

Graphical user interface, application

Description automatically generated**Patient Registration**

Figure : Patient Registration Screen

This is the catalyst for a patient to be signed-up in the system. It will trigger generation of public and private keys in the backend which is crucial for encryption and uploading to IPFS.

**Patient Login**

Graphical user interface, application

Description automatically generated

Figure : Patient Login Screen

The patient will be required to input email (as username) and password to be able to access related vaccine records and details.

**Patient Home Screen**

Qr code

Description automatically generated

Figure : Patient Home Screen (Upper Section)

Upper part of the home screen is the patient’s profile data: Full name, profile photo, address and patient code. Patient QR code will also be displayed. This code will be used later if patient decides to get vaccinated. It will signify that the patient is currently registered to the system.

Qr code

Description automatically generated

Figure : Patient Home Screen (Lower Section)

Lower part of the home screen is for vaccination details. If the patient already got vaccinated, this section will be displayed. It will show a summary of vaccine doses and a QR code for these details. This code can be used for validation of third party such as establishments to validated if patient was indeed vaccinated.

The ‘Download Certificate’ button will download the vaccine certificate file as follows:

Qr code

Description automatically generated

Figure : Auto-Generated Vaccine Certificate

Graphical user interface, application

Description automatically generated**Vaccine Record Creation**

Figure : Vaccine Record Creation Screen

For medical personnel doing the vaccination, this page will be available for them. A valid Patient QR code is required before the system allows encoding of vaccination dose detail. Once details are confirmed, a MetaMask (blockchain plugin) will popup to confirm the transaction. This will log the transaction to the blockchain

**Vaccine Certificate Validation**

Graphical user interface, text, application

Description automatically generated

Figure : File Validation (Successful)

Graphical user interface, application

Description automatically generated

Figure : File Validation (Failure)

For third-party validators that would want to authenticate a Vaccine Certificate file, this page will be available. It will require the user to upload the file and will display a prompt that would tell if file is valid or not (within the blockchain logs context)

Graphical user interface

Description automatically generated**Scan Summary QR Code**

Figure : Scan Summary QR Code Screen

Another way to validate is via summary QR code. For third-party validators that would want quick details of patient’s vaccine records, it will require a summary QR code from a patient and will display related details. QR code will be invalid if app founds out it’s not existing within the blockchain logs.

## **5.2 Implementation of Proof of Authority and Keccak Hash**

The main purpose of using blockchain is to log and validate user transactions. Transactions we want to monitor and be authenticated are summary details and vaccine certificate creation. Apart from file and summary hashes, the app will include supporting details such as userId and Content Identifier from IPFS.

Assuming we have obtained hashes and Content Identifiers, we will create a blockchain of transactions given the files were already uploaded to IPFS and CIDs are generated. JSON Objects will be used as format of the payload. Summary details of vaccine records will also be stored in the blockchain

Both file and summary hash lookup have the same structure:

<hash>: <userId>

Sample:

[QmZkJLp7PJGMc3mMSxTeLtyQCRqZ5CudGdjPB3jjTSFaoX](https://cid.ipfs.io/#QmZkJLp7PJGMc3mMSxTeLtyQCRqZ5CudGdjPB3jjTSFaoX): 1001

Below is the Solidity code to manage hash lookups for vaccine certificate files and summary details:

// SPDX-License-Identifier: GPL-3.0

**pragma** solidity >=0.5.0 <0.9.0;

**contract** Certificate {

**mapping** (**string** => **uint256**) fileHashUserId;

**mapping** (**string** => **uint256**) summaryHashUserId;

**function** isFileHashUserIdExists(**string** **memory** \_fileHash, **uint256** \_userId)

**public** **view** **returns**(**bool**)

{

**if** (fileHashUserId[\_fileHash] == \_userId) {

**return** **true**;

}

**return** **false**;

}

**function** saveUserIdHashes(**string** **memory** \_fileHash, **string** **memory** \_summaryHash, **uint256** \_userId)

**public**

{

summaryHashUserId[\_summaryHash] = \_userId;

fileHashUserId[\_fileHash] = \_userId;

}

**function** isSummaryHashUserIdExists(**string** **memory** \_summaryHash, **uint256** \_userId)

**public** **view** **returns**(**bool**)

{

**if** (summaryHashUserId[\_summaryHash] == \_userId) {

**return** **true**;

}

**return** **false**;

}

}

**Clique Proof-of-Authority**

Proposed application will use Clique Proof of Authority consensus. Below is a simulation of various test cases:

*// block represents a single block signed by a parcitular account, where*

*// the account may or may not have cast a Clique vote.*

**type** block **struct** {

signer **string** *// Account that signed this particular block*

voted **string** *// Optional value if the signer voted on adding/removing someone*

auth **bool** *// Whether the vote was to authorize (or deauthorize)*

checkpoint []**string** *// List of authorized signers if this is an epoch block*

}

*// Define the various voting scenarios to test*

tests **:=** []**struct** {

epoch **uint64** *// Number of blocks in an epoch (unset = 30000)*

signers []**string** *// Initial list of authorized signers in the genesis*

blocks []block *// Chain of signed blocks, potentially influencing auths*

results []**string** *// Final list of authorized signers after all blocks*

failure **error** *// Failure if some block is invalid according to the rules*

}{

{

*// Single signer, no votes cast*

signers**:** []**string**{"A"},

blocks**:** []block{

{signer**:** "A"}

},

results**:** []**string**{"A"},

}, {

*// Single signer, voting to add two others (only accept first, second needs 2 votes)*

signers**:** []**string**{"A"},

blocks**:** []block{

{signer**:** "A", voted**:** "B", auth**:** true},

{signer**:** "B"},

{signer**:** "A", voted**:** "C", auth**:** true},

},

results**:** []**string**{"A", "B"},

}, {

*// Two signers, voting to add three others (only accept first two, third needs 3 votes already)*

signers**:** []**string**{"A", "B"},

blocks**:** []block{

{signer**:** "A", voted**:** "C", auth**:** true},

{signer**:** "B", voted**:** "C", auth**:** true},

{signer**:** "A", voted**:** "D", auth**:** true},

{signer**:** "B", voted**:** "D", auth**:** true},

{signer**:** "C"},

{signer**:** "A", voted**:** "E", auth**:** true},

{signer**:** "B", voted**:** "E", auth**:** true},

},

results**:** []**string**{"A", "B", "C", "D"},

}, {

*// Single signer, dropping itself (weird, but one less cornercase by explicitly allowing this)*

signers**:** []**string**{"A"},

blocks**:** []block{

{signer**:** "A", voted**:** "A", auth**:** false},

},

results**:** []**string**{},

}, {

*// Two signers, actually needing mutual consent to drop either of them (not fulfilled)*

signers**:** []**string**{"A", "B"},

blocks**:** []block{

{signer**:** "A", voted**:** "B", auth**:** false},

},

results**:** []**string**{"A", "B"},

}, {

*// Two signers, actually needing mutual consent to drop either of them (fulfilled)*

signers**:** []**string**{"A", "B"},

blocks**:** []block{

{signer**:** "A", voted**:** "B", auth**:** false},

{signer**:** "B", voted**:** "B", auth**:** false},

},

results**:** []**string**{"A"},

}, {

*// Three signers, two of them deciding to drop the third*

signers**:** []**string**{"A", "B", "C"},

blocks**:** []block{

{signer**:** "A", voted**:** "C", auth**:** false},

{signer**:** "B", voted**:** "C", auth**:** false},

},

results**:** []**string**{"A", "B"},

}, {

*// Four signers, consensus of two not being enough to drop anyone*

signers**:** []**string**{"A", "B", "C", "D"},

blocks**:** []block{

{signer**:** "A", voted**:** "C", auth**:** false},

{signer**:** "B", voted**:** "C", auth**:** false},

},

results**:** []**string**{"A", "B", "C", "D"},

}, {

*// Four signers, consensus of three already being enough to drop someone*

signers**:** []**string**{"A", "B", "C", "D"},

blocks**:** []block{

{signer**:** "A", voted**:** "D", auth**:** false},

{signer**:** "B", voted**:** "D", auth**:** false},

{signer**:** "C", voted**:** "D", auth**:** false},

},

results**:** []**string**{"A", "B", "C"},

}, {

*// Authorizations are counted once per signer per target*

signers**:** []**string**{"A", "B"},

blocks**:** []block{

{signer**:** "A", voted**:** "C", auth**:** true},

{signer**:** "B"},

{signer**:** "A", voted**:** "C", auth**:** true},

{signer**:** "B"},

{signer**:** "A", voted**:** "C", auth**:** true},

},

results**:** []**string**{"A", "B"},

}, {

*// Authorizing multiple accounts concurrently is permitted*

signers**:** []**string**{"A", "B"},

blocks**:** []block{

{signer**:** "A", voted**:** "C", auth**:** true},

{signer**:** "B"},

{signer**:** "A", voted**:** "D", auth**:** true},

{signer**:** "B"},

{signer**:** "A"},

{signer**:** "B", voted**:** "D", auth**:** true},

{signer**:** "A"},

{signer**:** "B", voted**:** "C", auth**:** true},

},

results**:** []**string**{"A", "B", "C", "D"},

}, {

*// Deauthorizations are counted once per signer per target*

signers**:** []**string**{"A", "B"},

blocks**:** []block{

{signer**:** "A", voted**:** "B", auth**:** false},

{signer**:** "B"},

{signer**:** "A", voted**:** "B", auth**:** false},

{signer**:** "B"},

{signer**:** "A", voted**:** "B", auth**:** false},

},

results**:** []**string**{"A", "B"},

}, {

*// Deauthorizing multiple accounts concurrently is permitted*

signers**:** []**string**{"A", "B", "C", "D"},

blocks**:** []block{

{signer**:** "A", voted**:** "C", auth**:** false},

{signer**:** "B"},

{signer**:** "C"},

{signer**:** "A", voted**:** "D", auth**:** false},

{signer**:** "B"},

{signer**:** "C"},

{signer**:** "A"},

{signer**:** "B", voted**:** "D", auth**:** false},

{signer**:** "C", voted**:** "D", auth**:** false},

{signer**:** "A"},

{signer**:** "B", voted**:** "C", auth**:** false},

},

results**:** []**string**{"A", "B"},

}, {

*// Votes from deauthorized signers are discarded immediately (deauth votes)*

signers**:** []**string**{"A", "B", "C"},

blocks**:** []block{

{signer**:** "C", voted**:** "B", auth**:** false},

{signer**:** "A", voted**:** "C", auth**:** false},

{signer**:** "B", voted**:** "C", auth**:** false},

{signer**:** "A", voted**:** "B", auth**:** false},

},

results**:** []**string**{"A", "B"},

}, {

*// Votes from deauthorized signers are discarded immediately (auth votes)*

signers**:** []**string**{"A", "B", "C"},

blocks**:** []block{

{signer**:** "C", voted**:** "D", auth**:** true},

{signer**:** "A", voted**:** "C", auth**:** false},

{signer**:** "B", voted**:** "C", auth**:** false},

{signer**:** "A", voted**:** "D", auth**:** true},

},

results**:** []**string**{"A", "B"},

}, {

*// Cascading changes are not allowed, only the account being voted on may change*

signers**:** []**string**{"A", "B", "C", "D"},

blocks**:** []block{

{signer**:** "A", voted**:** "C", auth**:** false},

{signer**:** "B"},

{signer**:** "C"},

{signer**:** "A", voted**:** "D", auth**:** false},

{signer**:** "B", voted**:** "C", auth**:** false},

{signer**:** "C"},

{signer**:** "A"},

{signer**:** "B", voted**:** "D", auth**:** false},

{signer**:** "C", voted**:** "D", auth**:** false},

},

results**:** []**string**{"A", "B", "C"},

}, {

*// Changes reaching consensus out of bounds (via a deauth) execute on touch*

signers**:** []**string**{"A", "B", "C", "D"},

blocks**:** []block{

{signer**:** "A", voted**:** "C", auth**:** false},

{signer**:** "B"},

{signer**:** "C"},

{signer**:** "A", voted**:** "D", auth**:** false},

{signer**:** "B", voted**:** "C", auth**:** false},

{signer**:** "C"},

{signer**:** "A"},

{signer**:** "B", voted**:** "D", auth**:** false},

{signer**:** "C", voted**:** "D", auth**:** false},

{signer**:** "A"},

{signer**:** "C", voted**:** "C", auth**:** true},

},

results**:** []**string**{"A", "B"},

}, {

*// Changes reaching consensus out of bounds (via a deauth) may go out of consensus on first touch*

signers**:** []**string**{"A", "B", "C", "D"},

blocks**:** []block{

{signer**:** "A", voted**:** "C", auth**:** false},

{signer**:** "B"},

{signer**:** "C"},

{signer**:** "A", voted**:** "D", auth**:** false},

{signer**:** "B", voted**:** "C", auth**:** false},

{signer**:** "C"},

{signer**:** "A"},

{signer**:** "B", voted**:** "D", auth**:** false},

{signer**:** "C", voted**:** "D", auth**:** false},

{signer**:** "A"},

{signer**:** "B", voted**:** "C", auth**:** true},

},

results**:** []**string**{"A", "B", "C"},

}, {

*// Ensure that pending votes don't survive authorization status changes. This*

*// corner case can only appear if a signer is quickly added, removed and then*

*// readded (or the inverse), while one of the original voters dropped. If a*

*// past vote is left cached in the system somewhere, this will interfere with*

*// the final signer outcome.*

signers**:** []**string**{"A", "B", "C", "D", "E"},

blocks**:** []block{

{signer**:** "A", voted**:** "F", auth**:** true}, *// Authorize F, 3 votes needed*

{signer**:** "B", voted**:** "F", auth**:** true},

{signer**:** "C", voted**:** "F", auth**:** true},

{signer**:** "D", voted**:** "F", auth**:** false}, *// Deauthorize F, 4 votes needed (leave A's previous vote "unchanged")*

{signer**:** "E", voted**:** "F", auth**:** false},

{signer**:** "B", voted**:** "F", auth**:** false},

{signer**:** "C", voted**:** "F", auth**:** false},

{signer**:** "D", voted**:** "F", auth**:** true}, *// Almost authorize F, 2/3 votes needed*

{signer**:** "E", voted**:** "F", auth**:** true},

{signer**:** "B", voted**:** "A", auth**:** false}, *// Deauthorize A, 3 votes needed*

{signer**:** "C", voted**:** "A", auth**:** false},

{signer**:** "D", voted**:** "A", auth**:** false},

{signer**:** "B", voted**:** "F", auth**:** true}, *// Finish authorizing F, 3/3 votes needed*

},

results**:** []**string**{"B", "C", "D", "E", "F"},

}, {

*// Epoch transitions reset all votes to allow chain checkpointing*

epoch**:** 3,

signers**:** []**string**{"A", "B"},

blocks**:** []block{

{signer**:** "A", voted**:** "C", auth**:** true},

{signer**:** "B"},

{signer**:** "A", checkpoint**:** []**string**{"A", "B"}},

{signer**:** "B", voted**:** "C", auth**:** true},

},

results**:** []**string**{"A", "B"},

}, {

*// An unauthorized signer should not be able to sign blocks*

signers**:** []**string**{"A"},

blocks**:** []block{

{signer**:** "B"},

},

failure**:** errUnauthorizedSigner,

}, {

*// An authorized signer that signed recenty should not be able to sign again*

signers**:** []**string**{"A", "B"},

blocks []block{

{signer**:** "A"},

{signer**:** "A"},

},

failure**:** errRecentlySigned,

}, {

*// Recent signatures should not reset on checkpoint blocks imported in a batch*

epoch**:** 3,

signers**:** []**string**{"A", "B", "C"},

blocks**:** []block{

{signer**:** "A"},

{signer**:** "B"},

{signer**:** "A", checkpoint**:** []**string**{"A", "B", "C"}},

{signer**:** "A"},

},

failure**:** errRecentlySigned,

},,

}

When someone uploads a certificate to the system, the application sends the transaction log to the blockchain.

Figure : Node Blockchain Logs for File Upload



Figure : Detected Metamask Address of Log Sender



Figure : Metamask account details of vaccine file uploader



Figure : Metamask Notification of File Upload Transaction

**Keccak-256**

The hashing algorithm used by Ethereum (implemented in Clique POA) is Keccak-256. Below is a simulation of the algorithm using a simple input string:

Keccak256 presets:

bitrate\_bits = 1088

capacity\_bits = 512

output\_bits = 256

bitrate\_bytes = 136 -- convert bitrate\_bits to bytes

multirate\_padding(used\_bytes, align\_bytes)

padlength = align\_bytes - used\_bytes

zero\_elements = [0] \* padlength - 2

padding = [1] + zero\_elements + [128]

return padding

#example

#if used\_bytes = 130, align\_bytes = 136

#padlength = 136 - 130 = 6

#zero\_elements = [0, 0, 0, 0]

#padding = [1, 0, 0, 0, 0 128]

bytesToLane(input\_bytes)

accumulator = 0

for b in reversed(input\_bytes)

accumulator = ( accumulator << 8 ) | b

#apply 8 bitwise left shit to accumulator then XOR with b

return accumulator

#example

#input\_bytes = [104, 101, 108, 108, 111, 32, 119, 111]

each iteration will result to (consecutively)

0

28416

7304960

1870077952

478739984128

122557435964416

31374703606918144

8031924123371070720

8031924123371070824

#final value will be 8031924123371070824

-----------------------------------------------------------

input\_text = "hello world"

1. Get byte array (input\_byte\_array) equivalent of input\_text

input\_byte\_array = [104, 101, 108, 108, 111, 32, 119, 111, 114, 108, 100]

2. Pad input\_byte\_array using multirate\_padding

used\_bytes = input\_byte\_array.length = 11 (count number of elements inside array)

align\_bytes = presets.bitrate\_bytes = 136

padded\_bytes = [104, 101, 108, 108, 111, 32, 119, 111, 114, 108, 100, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 128]

3. Append another batch of zero elements to padded\_bytes

zero\_count = convertToBytes ( (presets.bitrate\_bits + presets.capacity\_bits) - presets.bitrate\_bits )

= convertToBytes((1088 + 512) - 1088)

= 64

zero\_elements = [0] \* 64

padded\_bytes += zero\_elements

= [104, 101, 108, 108, 111, 32, 119, 111, 114, 108, 100, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

4. Convert padded\_bytes to array of lanes (lane\_array) and put then in a 5x5 2D array.

- get a batch of 8 elements from padded\_bytes

- convertedBatch1ToLane = bytesToLane(batch)

- result will be:

[[8031924123371070824L, 0, 0, 0, 0], [23358578, 0, 0, 9223372036854775808L, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]

5. Process lane\_array to permutation\_rounds (greek alphabet methods - theta, rho and phi, chi, iota)

lanew

= (presets.bitrate\_bits + presets.capacity\_bits) // 25 #the floor division // rounds the result down to the nearest whole number

= 64

l

= int(log(lanew, 2))

= 6

number\_of\_rounds = 12 + 2 \* l

= 24

## **5.3 Implementation of Merkle DAG**

The algorithm used in IPFS to manage content and assets is Merkle DAG. Suppose we want to upload 2 vaccine certificates. For brevity, we will use a small size text file to better illustrate the process. The default chunk size of IPFS is 256Kb but in this example we will reduce it to 32Kb to have appropriate representation using small sample files.

File 1

Name: cert\_allen\_smith.txt

Size: 86 bytes

Content:

A picture containing text

Description automatically generated

File 2

Name: cert\_john\_doe.txt.txt

Size: 83 bytes

Content:

A picture containing text

Description automatically generated

Generated Details for cert\_allen\_smith.txt:

Table : Generated Hash Value for Sample Record #1

|  |  |  |
| --- | --- | --- |
| Node Type | Size (Bytes) | Hash |
| Root | 0 | [QmZkJLp7PJGMc3mMSxTeLtyQCRqZ5CudGdjPB3jjTSFaoX](https://cid.ipfs.io/#QmZkJLp7PJGMc3mMSxTeLtyQCRqZ5CudGdjPB3jjTSFaoX) |
| Links | 32 | [QmdsyzBk5nWmC7a92gaAuRHxWTQu6e4wwyv2bVmZtF7mcq](https://cid.ipfs.io/#QmdsyzBk5nWmC7a92gaAuRHxWTQu6e4wwyv2bVmZtF7mcq) |
| Links | 32 | [QmPsFk9hcP4WmN96r8mXYjV5rKCNNb94c95jfqLBNZvigT](https://cid.ipfs.io/#QmPsFk9hcP4WmN96r8mXYjV5rKCNNb94c95jfqLBNZvigT) |
| Links | 22 | [QmVVrfBPAnF5DC1DXDZH2yftW6MEoCSKXEQEbY5LKfFzAt](https://cid.ipfs.io/#QmVVrfBPAnF5DC1DXDZH2yftW6MEoCSKXEQEbY5LKfFzAt) |

Generated Details for cert\_john\_doe.txt:

Table : Generated Hash Value for Sample Record #2

|  |  |  |
| --- | --- | --- |
| Node Type | Size (Bytes) | Hash |
| Root | 0 | [QmanmTVLostTHeeLiz8vr99QDWmVbmbd53rSA2iFoDcmXu](https://cid.ipfs.io/#QmanmTVLostTHeeLiz8vr99QDWmVbmbd53rSA2iFoDcmXu) |
| Links | 32 | [QmdsyzBk5nWmC7a92gaAuRHxWTQu6e4wwyv2bVmZtF7mcq](https://cid.ipfs.io/#QmdsyzBk5nWmC7a92gaAuRHxWTQu6e4wwyv2bVmZtF7mcq) |
| Links | 32 | [QmTfDsTDe3nVu7b3hij43R3mBzyhJZgVm9eFBewVb5FfKV](https://cid.ipfs.io/#QmTfDsTDe3nVu7b3hij43R3mBzyhJZgVm9eFBewVb5FfKV) |
| Links | 32 | [QmRF3DNTkA43a7AG26uva4n7pgR22ctz6PjZW4KMuN5Cvu](https://cid.ipfs.io/#QmRF3DNTkA43a7AG26uva4n7pgR22ctz6PjZW4KMuN5Cvu) |

We can now map out the links with their respective roots. Notice that link “Qmdsy” is referenced by both root objects.

Diagram

Description automatically generated

Figure : Merkle DAG representing sample records

## **5.4 Security Scans**

The researcher employed two scans to audit the application’s functionality. This will ensure that the application complies with existing standard for Solidity and Open Web Applications. Because the application size is fairly small, we will only focus the scans on the blockchain and web application aspects.

**Smart Contract Weakness Classification Registry**

The Smart Contract Weakness Classification Registry (SWC Registry) is an implementation of the weakness classification scheme proposed in [EIP-1470](https://github.com/ethereum/EIPs/issues/1469). It is loosely aligned to the terminologies and structure used in the Common Weakness Enumeration ([CWE](https://cwe.mitre.org/)) while overlaying a wide range of weakness variants that are specific to smart contracts.

The goals of this project are as follows:

* Provide a straightforward way to classify security issues in smart contract systems.
* Define a common language for describing security issues in smart contract systems' architecture, design, or code.
* Serve as a way to train and increase performance for smart contract security analysis tools.

The chosen scans, Securify and Slither, based their supported vulnerabilities from SWC Registry

**Securify**

The Securify tool is a static analyzer tool for Ethereum Solidity contracts. This tool scans the contract code and finds the security vulnerability patterns in the code. After scanning, it generates a report along with descriptions of each vulnerability it has found and provides an idea of how to solve each vulnerability.

**Supported Vulnerabilities**

Below are the supported vulnerabilities (table 4) included in Securify’s scans.

Table : Securify Supported Vulnerabilities

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Pattern name** | **Severity** | **Slither ID** | **SWC ID** |
| 1 | TODAmount | Critical | - | [SWC-114](https://swcregistry.io/docs/SWC-114) |
| 2 | TODReceiver | Critical | - | [SWC-114](https://swcregistry.io/docs/SWC-114) |
| 3 | TODTransfer | Critical | - | [SWC-114](https://swcregistry.io/docs/SWC-114) |
| 4 | UnrestrictedWrite | Critical | - | [SWC-124](https://swcregistry.io/docs/SWC-124) |
| 5 | RightToLeftOverride | High | rtlo | [SWC-130](https://swcregistry.io/docs/SWC-130) |
| 6 | ShadowedStateVariable | High | shadowing-state, shadowing-abstract | [SWC-119](https://swcregistry.io/docs/SWC-119) |
| 7 | UnrestrictedSelfdestruct | High | suicidal | [SWC-106](https://swcregistry.io/docs/SWC-106) |
| 8 | UninitializedStateVariable | High | uninitialized-state | [SWC-109](https://swcregistry.io/docs/SWC-109) |
| 9 | UninitializedStorage | High | uninitialized-storage | [SWC-109](https://swcregistry.io/docs/SWC-109) |
| 10 | UnrestrictedDelegateCall | High | controlled-delegatecall | [SWC-112](https://swcregistry.io/docs/SWC-112) |
| 11 | DAO | High | reentrancy-eth | [SWC-107](https://swcregistry.io/docs/SWC-107) |
| 12 | ERC20Interface | Medium | erc20-interface | - |
| 13 | ERC721Interface | Medium | erc721-interface | - |
| 14 | IncorrectEquality | Medium | incorrect-equality | [SWC-132](https://swcregistry.io/docs/SWC-132) |
| 15 | LockedEther | Medium | locked-ether | - |
| 16 | ReentrancyNoETH | Medium | reentrancy-no-eth | [SWC-107](https://swcregistry.io/docs/SWC-107) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Pattern name** | **Severity** | **Slither ID** | **SWC ID** |
| 17 | TxOrigin | Medium | tx-origin | [SWC-115](https://swcregistry.io/docs/SWC-115) |
| 18 | UnhandledException | Medium | unchecked-lowlevel | - |
| 19 | UnrestrictedEtherFlow | Medium | unchecked-send | [SWC-105](https://swcregistry.io/docs/SWC-105) |
| 20 | UninitializedLocal | Medium | uninitialized-local | [SWC-109](https://swcregistry.io/docs/SWC-109) |
| 21 | UnusedReturn | Medium | unused-return | [SWC-104](https://swcregistry.io/docs/SWC-104) |
| 22 | ShadowedBuiltin | Low | shadowing-builtin | - |
| 23 | ShadowedLocalVariable | Low | shadowing-local | - |
| 24 | CallToDefaultConstructor? | Low | void-cst | - |
| 25 | CallInLoop | Low | calls-loop | [SWC-104](https://swcregistry.io/docs/SWC-104) |
| 26 | ReentrancyBenign | Low | reentrancy-benign | [SWC-107](https://swcregistry.io/docs/SWC-107) |
| 27 | Timestamp | Low | timestamp | [SWC-116](https://swcregistry.io/docs/SWC-116) |
| 28 | AssemblyUsage | Info | assembly | - |
| 29 | ERC20Indexed | Info | erc20-indexed | - |
| 30 | LowLevelCalls | Info | low-level-calls | - |
| 31 | NamingConvention | Info | naming-convention | - |
| 32 | SolcVersion | Info | solc-version | [SWC-103](https://swcregistry.io/docs/SWC-103) |
| 33 | UnusedStateVariable | Info | unused-state | - |
| 34 | TooManyDigits | Info | too-many-digits | - |
| 35 | ConstableStates | Info | constable-states | - |
| 36 | ExternalFunctions | Info | external-function | - |
| 37 | StateVariablesDefaultVisibility | Info | - | [SWC-108](https://swcregistry.io/docs/SWC-108) |

**Scan Result:**

#1

Severity: MEDIUM

Pattern: Missing Input Validation

Description: Method arguments must be sanitized before they are used

in computations.

Type: Violation

Contract: Certificate

Line: 19

Source:

>

> function isFileHashUserIdExists(string calldata \_fileHash, uint256 \_userId)

> ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

> external

#2

Severity: MEDIUM

Pattern: Missing Input Validation

Description: Method arguments must be sanitized before they are used

in computations.

Type: Violation

Contract: Certificate

Line: 33

Source:

>

> function saveUserIdHashes(

> ^^^^^^^^^^^^^^^^^^^^^^^^^^

> string calldata \_fileHash,

#3

Severity: MEDIUM

Pattern: Missing Input Validation

Description: Method arguments must be sanitized before they are used

in computations.

Type: Violation

Contract: Certificate

Line: 45

Source:

>

> function isSummaryHashUserIdExists(

> ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

> string calldata \_summaryHash,

#4

Severity: LOW

Pattern: Solidity pragma directives

Description: Avoid complex solidity version pragma statements.

Type: Violation

Contract: None

Line: 3

Source:

>

> pragma solidity >=0.5.0 <0.9.0;

> ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Reported Vulnerabilities:

Table : Securify Results Summary

|  |  |
| --- | --- |
| Critical | 0 |
| High | 0 |
| Medium | 3 |
| Low | 1 |
| Informational | 0 |
| Total | 4 |

All the three Medium vulnerabilities fall under “Missing Input Validation - Method arguments must be sanitized before they are used in computation”. The researcher chose to bypass this vulnerability as sanitation is already done on the web browser by using a front-end framework. The update/create transaction in Solidity is also guarded by an authorization modifier, thus only letting known entity to the blockchain make a successful transaction.

The single Low vulnerability is categorized on “Solidity pragma directives - Avoid complex solidity version pragma statements”. The researcher also skipped this vulnerability as this is not a security threat and more of a namespace convention or best practice. Since the application is still on Proof-of-Concept stage, the pragma versions used was a range to keep an open option when porting the testing from Remix (web) and local blockchain network.

**Slither**

Slither is a Solidity static analysis framework written in Python 3. It runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to easily write custom analyses. Slither enables developers to find vulnerabilities, enhance their code comprehension, and quickly prototype custom analyses.

**Supported Vulnerabilities**

Below are the supported vulnerabilities (table 6) included in Slither’s scans.

Table : Slither Supported Vulnerabilities

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Num** | **Detector** | **What it Detects** | **Impact** | **Confidence** |
| 1 | abiencoderv2-array | Storage abiencoderv2 array | High | High |
| 2 | array-by-reference | Modifying storage array by value | High | High |
| 3 | incorrect-shift | The order of parameters in a shift instruction is incorrect. | High | High |
| 4 | multiple-constructors | Multiple constructor schemes | High | High |
| 5 | name-reused | Contract's name reused | High | High |
| 6 | public-mappings-nested | Public mappings with nested variables | High | High |
| 7 | rtlo | Right-To-Left-Override control character is used | High | High |
| 8 | shadowing-state | State variables shadowing | High | High |
| 9 | suicidal | Functions allowing anyone to destruct the contract | High | High |
| 10 | uninitialized-state | Uninitialized state variables | High | High |
| 11 | uninitialized-storage | Uninitialized storage variables | High | High |
| 12 | unprotected-upgrade | Unprotected upgradeable contract | High | High |
| 13 | arbitrary-send | Functions that send Ether to arbitrary destinations | High | Medium |
| 14 | controlled-array-length | Tainted array length assignment | High | Medium |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Num** | **Detector** | **What it Detects** | **Impact** | **Confidence** |
| 15 | controlled-delegatecall | Controlled delegatecall destination | High | Medium |
| 16 | delegatecall-loop | Payable functions using delegatecall inside a loop | High | Medium |
| 17 | msg-value-loop | msg.value inside a loop | High | Medium |
| 18 | reentrancy-eth | Reentrancy vulnerabilities (theft of ethers) | High | Medium |
| 19 | storage-array | Signed storage integer array compiler bug | High | Medium |
| 20 | unchecked-transfer | Unchecked tokens transfer | High | Medium |
| 21 | weak-prng | Weak PRNG | High | Medium |
| 22 | enum-conversion | Detect dangerous enum conversion | Medium | High |
| 23 | erc20-interface | Incorrect ERC20 interfaces | Medium | High |
| 24 | erc721-interface | Incorrect ERC721 interfaces | Medium | High |
| 25 | incorrect-equality | Dangerous strict equalities | Medium | High |
| 26 | locked-ether | Contracts that lock ether | Medium | High |
| 27 | mapping-deletion | Deletion on mapping containing a structure | Medium | High |
| 28 | shadowing-abstract | State variables shadowing from abstract contracts | Medium | High |
| 29 | tautology | Tautology or contradiction | Medium | High |
| 30 | write-after-write | Unused write | Medium | High |
| 31 | boolean-cst | Misuse of Boolean constant | Medium | Medium |
| 32 | constant-function-asm | Constant functions using assembly code | Medium | Medium |
| 33 | constant-function-state | Constant functions changing the state | Medium | Medium |
| 34 | divide-before-multiply | Imprecise arithmetic operations order | Medium | Medium |
| 35 | reentrancy-no-eth | Reentrancy vulnerabilities (no theft of ethers) | Medium | Medium |
| 36 | reused-constructor | Reused base constructor | Medium | Medium |
| 37 | tx-origin | Dangerous usage of tx.origin | Medium | Medium |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Num** | **Detector** | **What it Detects** | **Impact** | **Confidence** |
| 38 | unchecked-lowlevel | Unchecked low-level calls | Medium | Medium |
| 39 | unchecked-send | Unchecked send | Medium | Medium |
| 40 | uninitialized-local | Uninitialized local variables | Medium | Medium |
| 41 | unused-return | Unused return values | Medium | Medium |
| 42 | incorrect-modifier | Modifiers that can return the default value | Low | High |
| 43 | shadowing-builtin | Built-in symbol shadowing | Low | High |
| 44 | shadowing-local | Local variables shadowing | Low | High |
| 45 | uninitialized-fptr-cst | Uninitialized function pointer calls in constructors | Low | High |
| 46 | variable-scope | Local variables used prior their declaration | Low | High |
| 47 | void-cst | Constructor called not implemented | Low | High |
| 48 | calls-loop | Multiple calls in a loop | Low | Medium |
| 49 | events-access | Missing Events Access Control | Low | Medium |
| 50 | events-maths | Missing Events Arithmetic | Low | Medium |
| 51 | incorrect-unary | Dangerous unary expressions | Low | Medium |
| 52 | missing-zero-check | Missing Zero Address Validation | Low | Medium |
| 53 | reentrancy-benign | Benign reentrancy vulnerabilities | Low | Medium |
| 54 | reentrancy-events | Reentrancy vulnerabilities leading to out-of-order Events | Low | Medium |
| 55 | timestamp | Dangerous usage of block.timestamp | Low | Medium |
| 56 | assembly | Assembly usage | Informational | High |
| 57 | assert-state-change | Assert state change | Informational | High |
| 58 | boolean-equal | Comparison to boolean constant | Informational | High |
| 59 | deprecated-standards | Deprecated Solidity Standards | Informational | High |
| 60 | erc20-indexed | Un-indexed ERC20 event parameters | Informational | High |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Num** | **Detector** | **What it Detects** | **Impact** | **Confidence** |
| 61 | function-init-state | Function initializing state variables | Informational | High |
| 62 | low-level-calls | Low level calls | Informational | High |
| 63 | missing-inheritance | Missing inheritance | Informational | High |
| 64 | naming-convention | Conformity to Solidity naming conventions | Informational | High |
| 65 | pragma | If different pragma directives are used | Informational | High |
| 66 | redundant-statements | Redundant statements | Informational | High |
| 67 | solc-version | Incorrect Solidity version | Informational | High |
| 68 | unimplemented-functions | Unimplemented functions | Informational | High |
| 69 | unused-state | Unused state variables | Informational | High |
| 70 | costly-loop | Costly operations in a loop | Informational | Medium |
| 71 | dead-code | Functions that are not used | Informational | Medium |
| 72 | reentrancy-unlimited-gas | Reentrancy vulnerabilities through send and transfer | Informational | Medium |
| 73 | similar-names | Variable names are too similar | Informational | Medium |
| 74 | too-many-digits | Conformance to numeric notation best practices | Informational | Medium |
| 75 | constable-states | State variables that could be declared constant | Optimization | High |
| 76 | external-function | Public function that could be declared external | Optimization | High |

**Scan Result:**

Pragma version>=0.5.0<0.9.0 (Certificate2.sol#3) is too complex

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Certificate2.sol analyzed (1 contracts with 75 detectors), 2 result(s) found

Reported Vulnerabilities:

Table : Slither Results Summary

|  |  |
| --- | --- |
| Critical | 0 |
| High | 0 |
| Medium | 0 |
| Low | 0 |
| Informational | 1 |
| Total | 1 |

The single vulnerability found is “Pragma version is too complex” and is categorized under “Informational”. This is the same finding as with Securify scan pertaining to pragma version.

**Chapter Six**

# **CONCLUSIONS AND RECOMMENDATIONS**

This section contains the discussions of the conclusions and researcher recommendations based on the results of the study.

**Conclusions**

Based on the stated objectives of the study, the researcher concludes the following:

1. Blockchain was instrumental in maintaining a reliable source of transaction that can be accessed by various parties. The immutable nature of the ledger made it possible to add a unique layer of security for the vaccine certificates’ transactions. This is in combination with the usage of IPFS which also maintained an immutable file system. The researcher was able to develop an application that took advantage of these technologies.
2. The usage of hashes as reference for files and patient information was both convenient and secured. It ensures uniqueness and does not explicitly reveal information at first glance. As also shown with the Merkle Tree algorithm, generating a hash was pivotal in locating a file within IPFS’ file system.
3. The security scans used to examine the smart contract of the application ensured that it is less vulnerable to the weaknesses specified in the Smart Contract Weakness Classification Registry (SWC Registry). The researcher was able to assess the vulnerabilities flagged by the scans and made efforts to eliminate and lessen the weaknesses for blockchain transactions.

**Recommendations**

This study recommends the use of blockchain in keeping medical records as it provides the following benefits:

1. Blockchain-based electronic health records would give medical personnel control over the flow of information from a single, trusted platform.
2. Eliminates the need to have multiple system to consolidate other transactions as blockchain can handle and validate different types of transaction logs.
3. When used with IPFS, provides a better file storage implementation for systems that need file upkeeps.

Further development and enhancement of the system is thereby recommended to future researchers, especially to include the following:

1. The domain of this study was on vaccine certificates, future work can look into integrating the other phases of vaccination such as scheduling or vaccine management.
2. The potentiality of processing large quantities of blockchain data makes it promising to use Artificial Intelligence to provide insights and reports.
3. Since the application is web-based and creates a web-responsive viewport to mobile users, it will be a good addition to create a dedicated mobile application to patients to enhance their user experience.

# **LIST OF REFERENCES**

Abadi, Joseph and Brunnermeier, Markus (2018). *Blockchain Economics*

Al Asad, Nafiz; Elahi, Md. Tausif; Al Hasan, Abdullah; Yousuf, Mohammad Abu (2020). *Permission-Based Blockchain with Proof of Authority for Secured Healthcare Data Sharing*

Azaria, Asaph; Ekblaw, Ariel; Vieira, Thiago; Lippman, Andrew (2016). *MedRec: Using Blockchain for Medical Data Access and Permission Management*

Baumgart, Ingmar and Mies, Sebastian (2007). *S/kademlia:* *A practicable approach towards secure key-based routing.*

Bellare, Mihir; Desai Anand; Pointcheval, David; Rogaway, Phillip (1998). *Relations among notions of security for public-key encryption schemes*

Benet, Juan (2014). *IPFS - content addressed, versioned, P2P file system (draft 3)*

Bertoni, Guido; Daemen, Joan; Peeters, Michael; Van Assche, Gilles (2013). *Keccak*

Ebardo, Ryan and Celis, Nelson (2019). *Barriers to the Adoption of Electronic Medical Records in Select Philippine Hospitals: A Case Study Approach*

Ebardo, Ryan and Tuazon, John Byron (2019). *Identifying Healthcare Information Systems Enablers in a Developing Economy*

Freedman, Michael J.; Freudenthal, Eric; Mazieres, David (2004). *Democratizing content publication with Coral*

Fujisaki, Eiichiro and Okamoto, Tatsuaki (2011). *Secure Integration of Asymmetric and Symmetric Encryption Schemes*

Gesulga, Jaillah Mae; Berjame, Almarie; Moquiala, Kristelle Sheen; Galido, Adrian (2017). *Barriers to Electronic Health Record System Implementation and Information Systems Resources: A Structured Review*

Gilbert, Henri and Handschuh, Helena (2004). *Security Analysis of SHA-256 and Sisters*

Goldwasser, Shafi and Micali, Silvio (1984). *Probabilistic encryption*

Khalifa, Mohamed (2018). *Perceived Benefits of Implementing and Using Hospital Information Systems and Electronic Medical Records*

Khubrani, Mousa Mohammed (2021). *A Framework for Blockchain-based Smart Health System*

Kumar, Randhir and Tripathi, Rakesh (2020). *A Secure and Distributed Framework for sharing COVID-19 patient Reports using Consortium Blockchain and IPFS*

Kumar, Shivansh; Bharti, Aman Kumar; Amin, Ruhul (2021). *Decentralized secure storage of medical records using Blockchain and IPFS: A comparative analysis with future directions*

Maymounkov, Petar and Mazieres, David (2002). *Kademlia: A peer-to-peer information system based on the xor metric*

Menezes, Alfred; van Oorschot, Paul; Vanstone, Scott (1997). *Handbook of Applied Cryptography*

Merkle, Ralph (1989). *A Certified Digital Signature*

Micali, Silvio; Jakobsson, Markus; Leighton, Tom; Szydlo, Michael (2003). *Fractal merkle tree representation and traversal.*

Monrat, Ahmed Afif; Schelén, Olov; Andersson, Karl (2019). *A Survey of Blockchain From the Perspectives of Applications, Challenges, and Opportunities*

Nakov, Svetlin (2018). *Practical Cryptography for Developers*

National Institute of Standards and Technology (2002). *FIPS-180-2:* *Secure Hash Standard (SHS)*

Reen, Gaganjeet (2019). *Decentralized Patient Centric e-Health Record Management System using Blockchain and IPFS*

Sharma, Saurabh; Mishra, Ashish; Lala, Ajay; Singhai, Deeksha (2020). *Secure Cloud Storage Architecture for Digital Medical Record in Cloud Environment using Blockchain*

Sklavos, Nicolas and Koufopavlou, Odysseas G. (2003). *On the hardware implementation of the SHA-2 (256, 384, 512) Hash functions*

Sukratha, V. and Latha, Y. Madhavi (2013). *Securing the Web using Keccak (SHA-3)*

Sun, Jin; Yao, Xiaomin; Wang, Shangping; Wu, Ying (2020). *Blockchain-based secure storage and access scheme for electronic medical records in IPFS*

Szabo, Nick (1997). *Smart contracts: formalizing and securing relationships on public networks.*

Szydlo, Michael (2003). *Merkle tree traversal in log space and time*

Tsai, Jack and Bond, Gary (2007). *A comparison of electronic records to paper records in mental health centers*

Vujičić, Dejan; Jagodić, Dijana; Ranđić, Siniša (2018). *Blockchain Technology, Bitcoin, and Ethereum: A Brief Overview*

Wu, Sihua and Du, Jiang (2019). *Electronic medical record security sharing model based on blockchain*

Xie, Ming (2003). *P2P Systems based on Distributed Hash Table*

Yaga, Dylan; Mell, Peter; Roby, Nik; Scarfone, Karen (2018). *Blockchain Technology Overview*

Yoshida, Hirotaka and Biryukov, Alex (2005). *Analysis of a SHA-256 Variant*

Zheng, Zibin; Xie, Shaoan; Dai, Hong-Ning; Chen, Xiangping (2017). *An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends*