



**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

March 2022



ACKNOWLEDGEMENT

I would like to take this opportunity to express my gratitude to the people that have been part of my journey to complete this study.

To my adviser, Dr. Khatalyn E. Mata, for providing guidance on my research. Her insights, patience and encouragement helped complete various requirements of the study.

To the members of the panel, Prof. Edgardo S. Dajao and Dr. Dan Michael A. Cortez, for their constructive comments and suggestions which made this study more relevant and concrete. To panel chairman, Prof. Manuel L. Ocampo, for fostering communications between the researchers and panel board, and his unyielding belief that I can accomplish this feat.

To Ms. Lydia de Leon and Ms. Rose Lopez, the staff of CET-GP, Director Joseph Berlin P. Juanzon and Dean Juan C. Tallara for their support to assist my research needs.

To Dr. Ryan Joseph Magtibay and Mr. Cedric Villapando, for helping in my information gathering activities and sharing their technical knowledge.

To my parents and extended family, for expressing their support and believing that I can finish this endeavor.

To my partner, Joane Marie Llamera, for pushing me to the limits but always keeping me grounded when pressure starts to settle. It was your steadfast confidence in me that kept me motivated throughout the entire study.

Lastly, I dedicate this study to my late grandmother, Geralda Lineses. Your memories never fail to make me smile. I hope I made you proud.

Jennifer L. Fadriquela



ABSTRACT

Filipinos were compelled to obtain vaccination certificates as evidence of their protection from coronavirus disease (COVID-19). The current processes to handle this varies per Local Government but majority are still manual, prone to tampering and time-consuming for medical workers. This study aims to develop a web application for vaccine certificate management. It utilized concepts of Blockchain and Merkle Tree to secure transactions and files generated from the process. The transactions were stored on a Proof-of-Authority blockchain and files were managed using InterPlanetary File Storage. These two frameworks were combined to increase data security given their native implementations of hashing and immutability. Also, it employed QR Code technology to increase ease-of-use when collecting and validating vaccine data. Face-to-face interviews with medical personnel were conducted to gather pertinent data. The application's security was assessed by using various tests from Smart Contract Weakness Classification Registry and National Institute of Standards and Technology - Cybersecurity Framework. The study's conclusions include limitations and suggestions for further research.



TABLE OF CONTENTS

TABLE OF CONTENTS	i
LIST OF FIGURES	ii
LIST OF TABLES	iv
INTRODUCTION	1
1.1 Background of the Study	1
1.2 Statement Problem	2
1.3 Objectives of the Study	3
1.4 Scope and Limitations	3
1.5 Significance of the Study	4
1.6 Definition of Terms	4
REVIEW OF RELATED LITERATURE	8
2.1 Literature Mapping.....	8
2.2 Related Literature.....	10
THEORETICAL FRAMEWORK	23
METHODOLOGY	34
4.1 Requirements Modeling	34
4.2 Quick Design.....	36
RESULTS AND DISCUSSION	54
5.1 Functionalities of the System	54
5.2 Implementation of Proof of Authority and Keccak Hash	60
5.3 Implementation of Merkle DAG	69
5.4 Data Tampering Results	73
5.5 Security Assessment.....	75
5.6 Performance Test.....	81
CONCLUSIONS AND RECOMMENDATIONS	83
REFERENCES.....	85
APPENDICES	89



LIST OF FIGURES

Figure 1 - Literature Map	8
Figure 2 - Hashing Process	23
Figure 3 - Generic Blockchain Transactions.....	25
Figure 4 - Clique PoA Block Creation Process.....	26
Figure 5 - Merkle Tree Implementation using Hashes.....	28
Figure 6 - DAG Illustration.....	29
Figure 7 - Merkle DAG Implemented on a File System.....	30
Figure 8 - Diagram of Proposed Solution	32
Figure 9 - Conceptual Framework	32
Figure 10 - Prototype Model Phases and Process	34
Figure 11 - Context Diagram	37
Figure 12 - Data Flow Diagram	38
Figure 13 - Use Case Diagram	39
Figure 14 - Patient Registration Flowchart	40
Figure 15 - Vaccine Record Creation Flowchart	41
Figure 16 - Verify Vaccine Certificate File	42
Figure 17 - Verify Vaccine QR Code	43
Figure 18 - High-Level System Design Diagram	45
Figure 19 - System Architecture Diagram	46
Figure 20 - Blockchain Data and Function Mapping.....	47
Figure 21 - Relational Database Diagram.....	48
Figure 22 - Transactional Operation Diagram	49
Figure 23 - File Storage in Blockchain and IPFS Flowchart	50
Figure 24 - File Retrieval Flowchart.....	51
Figure 25 - Validate File from Blockchain Flowchart.....	52
Figure 26 - Validate QR Code from Blockchain Flowchart	53
Figure 27 - Patient Registration Screen	55
Figure 28 - Patient Login Screen	55
Figure 29 - Patient Home Screen (Upper Section)	56



Figure 30 - Patient Home Screen (Lower Section)	57
Figure 31 - Vaccine Record Creation Screen.....	58
Figure 32 - File Validation (Successful)	58
Figure 33 - File Validation (Failure).....	59
Figure 34 - Scan Summary QR Code Screen.....	59
Figure 35 - Node Blockchain Logs for File Upload	64
Figure 36 - Detected Metamask Address of Log Sender	64
Figure 37 - Metamask account details of vaccine file uploader.....	64
Figure 38 - Metamask Notification of File Upload Transaction.....	65
Figure 39 - Merkle DAG representing sample records.....	70
Figure 40 - Encrypted file in IPFS	71
Figure 41 - Content Identifier Breakdown using IPFS CID Explorer	72
Figure 42 - Merkle DAG Illustration using IPFS DAG Visualizer	73
Figure 43 - UI Screenshot for invalid Summary QR Code.....	74
Figure 44 - Original File for Tamper Test	74
Figure 45 - Tampered File for Tamper Test.....	74
Figure 46 - UI Screenshot for Invalid File	75



LIST OF TABLES

Table 1 - Keccak Parameters Per Length	31
Table 2 - Frontend Specifications	44
Table 3 - Backend Specifications	44
Table 4 - Blockchain Specifications	44
Table 5 - Functionality of the System	54
Table 6 - Generated Hash Value for Sample Record #1	70
Table 7 - Generated Hash Value for Sample Record #2	70
Table 8 - Securify Scan Result Details	76
Table 9 - Securify Results Summary	77
Table 10 - Slither Scan Result Details	78
Table 11 - Slither Results Summary	78
Table 12 - NIST-CF Maintenance Area Matrix	79
Table 13 - NIST-CF Data Security Area Matrix	79
Table 14 - NIST-CF Protective Technology Area Matrix	80
Table 15 - Create Vaccine Record Load Test Result	81
Table 16 - Validate Vaccine QR Code Load Test Result	81
Table 17 - Validate Vaccine File Load Test Result	82



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 of Tsai and Bond (2007), the authors 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.

The work of Khalifa (2008) pointed out six ways Electronic Medical Records (EMR) 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. He 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. The study of 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. The work of Gesulga et al. (2017) 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 the paper of Ebardo and Tuazon (2019), the authors 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 the 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 in 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 to obtain vaccination certificates, it is worth noting that most of the province and remote 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. 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 that some people have tampered certificates for personal gains. A regional news from DILG dated December 3, 2021 warns the public on using fake vaccination documents (<https://dilg.gov.ph/regional-news/DILG-R10-warns-public-Dont-fake-vaccination-cards-or-face-raps/NR-2021-1192>).



1.3 Objectives of the Study

This study aims to design and develop an application that integrates blockchain and InterPlanetary File System (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 assess 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 was focused on developing an application for management of COVID-related records. Since there are privacy regulations concerning health information, the researcher used dummy data and instead probed more on the processes on how these records were archived or managed.

The study excluded vaccine management such as scheduling. Thus, it was focused on the results or outputs of these processes. The study assumed that information from the vaccine transaction were ready to be encoded in the system.

The study was only concerned on Vaccine Certificates. The researcher concentrated 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 benefits the following stakeholders:

Patients. Above all, patients 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 solves the woes of patients in terms on ease of access and portability of their records. They also have full autonomy of said records.

Medical Personnel. The application helps 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 (DAG) - It consists of vertices and edges (also called arcs), with each edge directed from one vertex to another, such that following those directions 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 – A 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

2.1 Literature Mapping

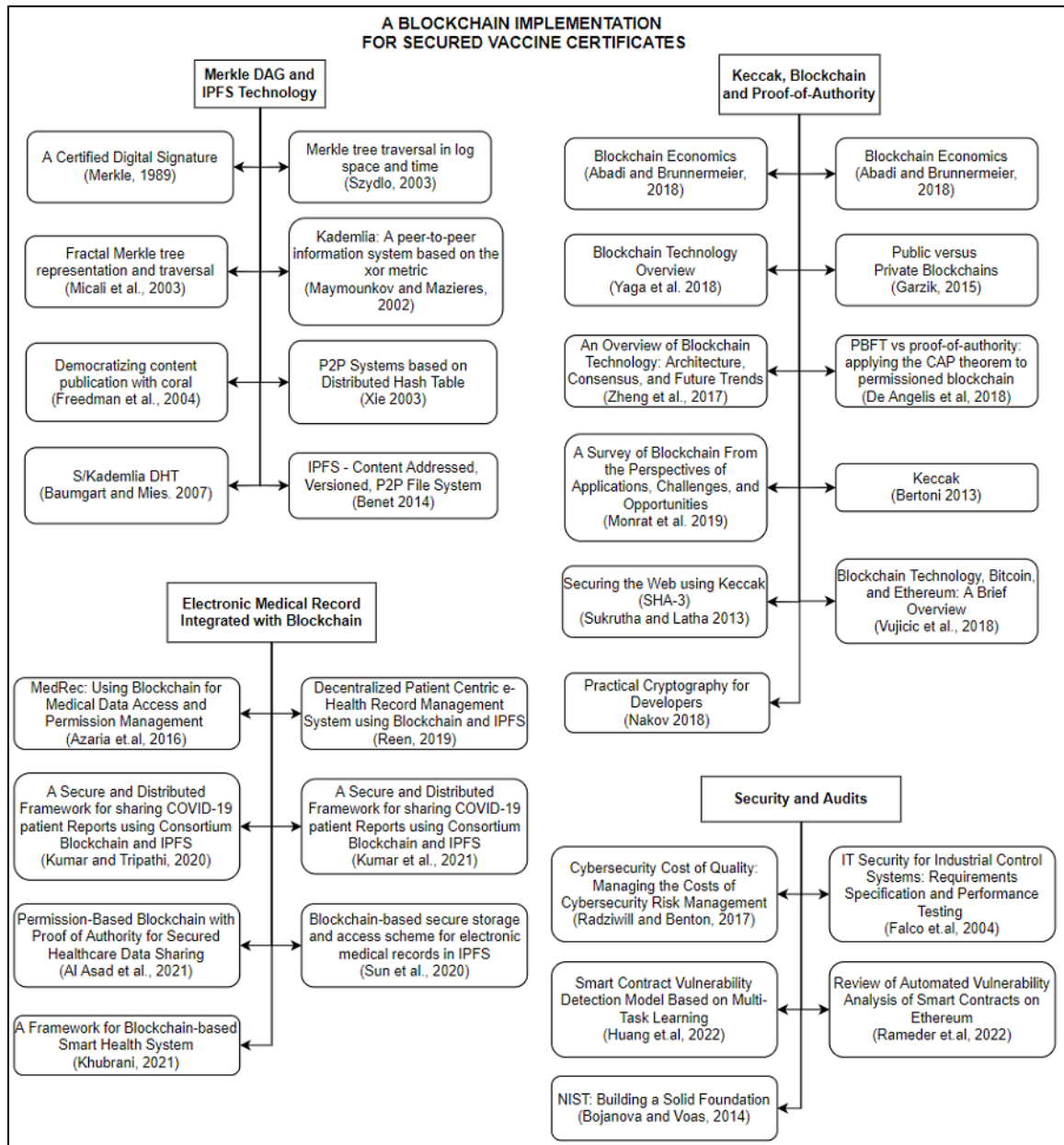


Figure 1 - Literature Map



Figure 1 illustrates how related literature were grouped and mapped. The researcher clustered the items into 4 main divisions:

1. Merkle DAG and IPFS
2. Keccak, Blockchain and Proof-of-Authority
3. Electronic Medical Record Integrated with Blockchain
4. Security and Audits

Group 1 starts with studies focused on theoretical discussion of Merkle Tree algorithm as proposed by Ralph Merkle. Next are variations and iterations of the tree such as Fractal Merkle Tree. It then mentions studies about its applications to real-life problems such as decentralized web applications, peer-to-peer systems and file storage.

Group 2 deals with Blockchain. Its concept is based upon theories of hashing and immutability. With this, the researcher included literature and studies about Keccak as this was the hashing algorithm used by the consensus used on the proposed application. Studies about comparison of blockchain consensus and performance were listed as basis in choosing the consensus. Lastly, Proof-of-Authority literature were included to discuss its theoretical and technical aspects.

Group 3 is about the various studies about integration of blockchain to Electronic Medical Records. The researcher gathered different studies that used various combination of technologies such as centralized database, cloud and blockchain. It highlights the strengths and weaknesses of each integration. The researcher also noted the author's suggestion based from their studies.

Group 4 discusses the security audits and frameworks used to assess the proposed application. It mainly dwells on two frameworks: Smart Contract Weakness Classification Registry and National Institute of Standards and Technology - Cybersecurity Framework.



2.2 Related Literature

Merkle Tree

In 1989, Ralph Merkle introduced the Merkle tree in his paper. It 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 h_m , and the tree owns 2^{h_m} data blocks x_i and $y_i = \text{hash}(x_i), i \in [0, 2^{h_m} - 1]$, where y_i is a leaf node value of the Merkle tree. Each value of the parent node is the hash of the concatenation of its children, $y_{\text{parent}} = \text{hash}(y_{\text{left}} | y_{\text{right}})$, where $|$ refers to concatenation. Below is a pseudocode format of the Classic Merkle Tree Traversal algorithm:

1. Set $\text{leaf} = 0$.
2. Output:
 - Compute and output leaf with $\text{LEAFCALC}(\text{leaf})$
 - For each $h \in [0, H - 1]$ output $\{\text{auth}_h\}$.
3. Refresh Auth Nodes:
 - For h such that 2^h divides $\text{leaf} + 1$:
 - Set auth_h be the sole node value in stack_h .
 - Set $\text{startnode} = (\text{leaf} + 1 + 2^h) \oplus 2^h$.
 - $\text{stack}_h.\text{initialize}(\text{startnode}, h)$.
4. Build Stacks:
 - For all $h \in [0, H - 1]$:
 - $\text{stack}_h.\text{update}(2)$.
5. Loop
 - Set $\text{leaf} = \text{leaf} + 1$.
 - If $\text{leaf} < 2^H$ 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 $h \in [0, H - 1]$ output $\{auth_h\}$.

3. Refresh Auth Nodes:

For h such that 2^h divides $leaf + 1$:

- Set $auth_h$ be the sole node value in $stack_h$.
- Set $startnode = (leaf + 1 + 2^h) \oplus 2^h$.
- $stack_h.initialize(startnode, h)$.

4. Build Stacks:

Repeat the following $2H - 1$ times:

- Let l_{min} be the minimum of $\{stack_h.low\}$ for all $h = 0, \dots, H - 1$.
- Let focus be the least h so that $stack_h.low = l_{min}$.
- $Stack_{focus}.update(1)$.

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 $j \in [0, H - 1]$ output $\{auth_j\}$.



3. Next Subtree:

For each i for which $Exist_i$ is no longer needed, i.e., for $i \in \{1, 2, \dots, L\}$ with $leaf = 1 \pmod{2^{h_i}}$:

- Set $Exist_i = Desire_i$.
- Create new empty $Desire_i$ (if $leaf + 2^{h_i} < 2^H$).

4. Grow Subtrees

For each $i \in \{1, 2, \dots, h\}$: Grow tree $Desire_i$ by applying 2 units to modified treehash (unless $Desire_i$ 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 $\log_2(n)$ 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 the study of Freedman et al. (2004), the authors 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. 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 returns 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

In a paper by Benet (2014), the author introduced the InterPlanetary File System (IPFS). It is a peer-to-peer distributed file system that seeks to connect all computing devices with the same system of files. In some ways, IPFS is similar to the Web, but IPFS could be seen as a single BitTorrent swarm, exchanging objects within one Git repository. In other words, IPFS provides a high throughput content-addressed block storage model, with content-addressed hyper links. This forms a generalized Merkle DAG, a data structure upon which one can build versioned file systems, blockchains, and even a Permanent Web.



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 by Yaga et al. (2018), the authors 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 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) is 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), the authors 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 decides 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. (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 $l = s \bmod N$. Clique is the PoA algorithm implemented in Geth, the GoLang-based Ethereum client. It tolerates up to half of the validators failing. QBFT and IBFT 2.0 networks require greater than or equal to two-thirds of validators to be operating to create blocks. The probability of a fork increases as the number of validators increases. 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 a 160 bit hash function which is 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". Sponge construction is based on a wide random function or random permutation and allows inputting ("absorbing" in sponge terminology) any amount of data, and outputting ("squeezing") any amount of data, while acting as a pseudorandom function with regard to all previous inputs. This leads to great flexibility. 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 et.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 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 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 is integrated in the system.

Security and Audits

The National Institute of Standards and Technology is defining and applying standard information security requirements and developing laboratory and field test methods for information security products and approaches to secure industrial control systems while maintaining their critical operational requirements. NIST has developed a laboratory scale testbed comprised of several implementations of typical industrial control and networking equipment as well as relevant sensors and actuators. This testbed is being used to develop performance test methods that can be applied to process control security products to determine if time-sensitive requirements can be met (Falco et.al, 2004). NIST is the National Institute of Standards and Technology at the U.S. Department of Commerce. The NIST Cybersecurity Framework helps businesses of all sizes better understand, manage, and reduce their cybersecurity risk and protect their networks and



data. The Framework is voluntary. It gives your business an outline of best practices to help you decide where to focus your time and money for cybersecurity protection. (Bojanova and Voas, 2014)

Below are its main areas:

1. Identify

- List of all equipment, software, and data you use, including laptops, smartphones, tablets, and point-of-sale devices.
- Roles and responsibilities for employees, vendors, and anyone else with access to sensitive data.
- Steps to take to protect against an attack and limit the damage if one occurs.

2. Protect

- Control who logs on to your network and uses your computers and other devices.
- Use security software to protect data.
- Encrypt sensitive data, at rest and in transit.
- Conduct regular backups of data.

3. Detect

- Monitor your computers for unauthorized personnel access, devices (like USB drives), and software.
- Investigate any unusual activities on your network or by your staff.
- Check your network for unauthorized users or connections.

4. Respond

- Notifying customers, employees, and others whose data may be at risk.
- Keeping business operations up and running.
- Reporting the attack to law enforcement and other authorities.
- Investigating and containing an attack.



5. Recover

- Repair and restore the equipment and parts of your network that were affected.
- Keep employees and customers informed of your response and recovery activities.

The value of reporting costs of cybersecurity in terms of quality costs lies less in the levels themselves, and more in how the values relate to one another, change over time, and change in response to changes in strategy, organization, or cybersecurity investments. The primary limitation of this study is that the practical applicability of the model can only be assessed through future work on a broad scale: implementation at different organizations, case studies, and empirical research (Radziwill and Benton, 2017).

The Smart Contract Weakness Classification Registry relates smart contract vulnerabilities to the Common Weakness Enumeration (CWE) typology and collects test cases. Currently, the registry holds 36 vulnerabilities, with descriptions, references, suggestions for remediation and sample Solidity contracts (Rameder et.al, 2022). Considering the economic loss and harm caused by contract vulnerabilities to the real world, three classic and common vulnerabilities for detection have been selected, namely arithmetic vulnerability, reentrancy, and the contract contains unknown address. (Huang et.al, 2022). They are described as follows.

Arithmetic vulnerability: This type of vulnerability is also known as integer overflow or underflow, arithmetic problems, and so forth.

Reentrancy: The ability to call external contract codes is one of the features of smart contracts, and contracts can send digital currency to external user addresses for transactions.

Contract contains unknown address: Smart contracts are P2P computer transaction protocols. Thus, when a contract contains an unknown address, this address is likely to be used for some malicious activities.



Chapter Three

THEORETICAL FRAMEWORK

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

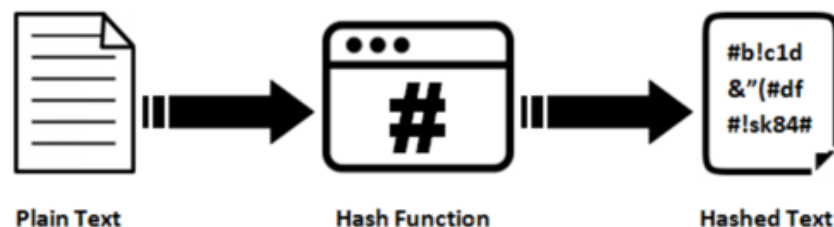


Figure 2 - Hashing Process



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 $h = \text{hash}(m)$. 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 m_1 , it should be hard to determine a different input m_2 such that $\text{hash}(m_1) = \text{hash}(m_2)$. 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 m_1 and m_2 such that $\text{hash}(m_1) = \text{hash}(m_2)$. 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

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 then chooses a publishing node to update the pending transaction. Once this is done, transaction is 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.

2. Block Data

- Actual data (text, files)

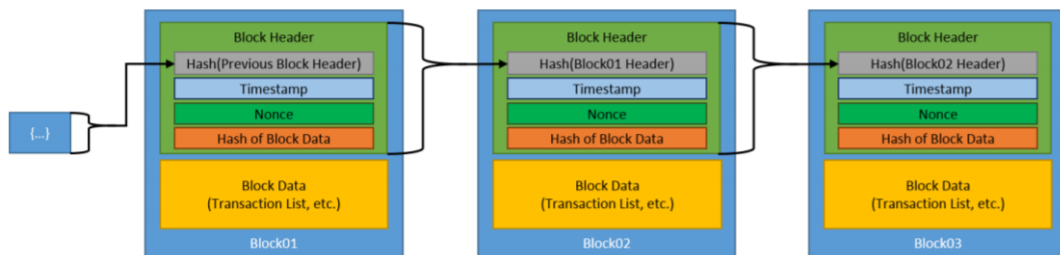


Figure 3 - Generic Blockchain Transactions

Figure 3 shows how blockchain works given 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 is 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



generates a different hash. This creates 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 publishes 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 publishes 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 are referenced 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.

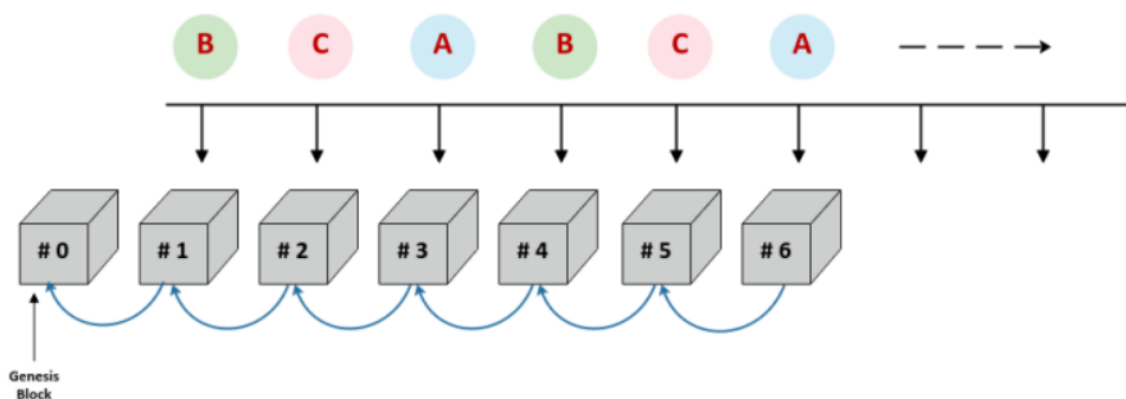


Figure 4 - Clique PoA Block Creation Process



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:
 - o Name: the name of the Link.
 - o Hash: the hash of the linked IPFS object.
 - o 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 is repeated once to establish an even number of leaf nodes if the number of transactions is odd.

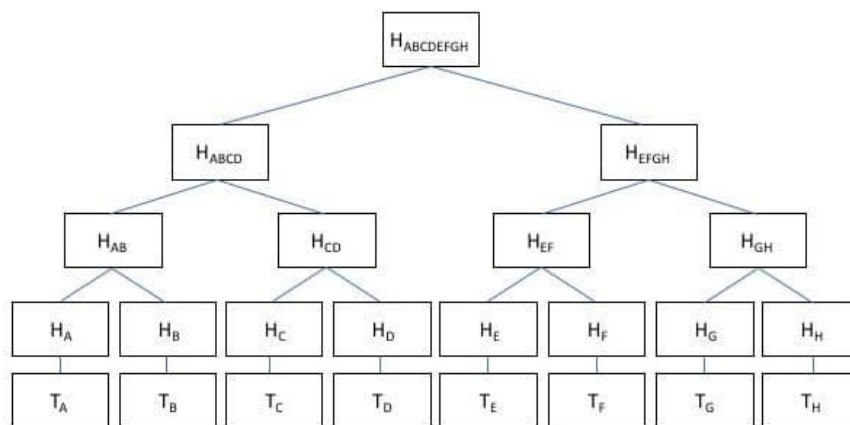


Figure 5 - Merkle Tree Implementation using Hashes

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.

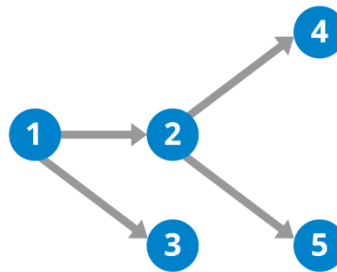


Figure 6 - DAG Illustration

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 affects 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.

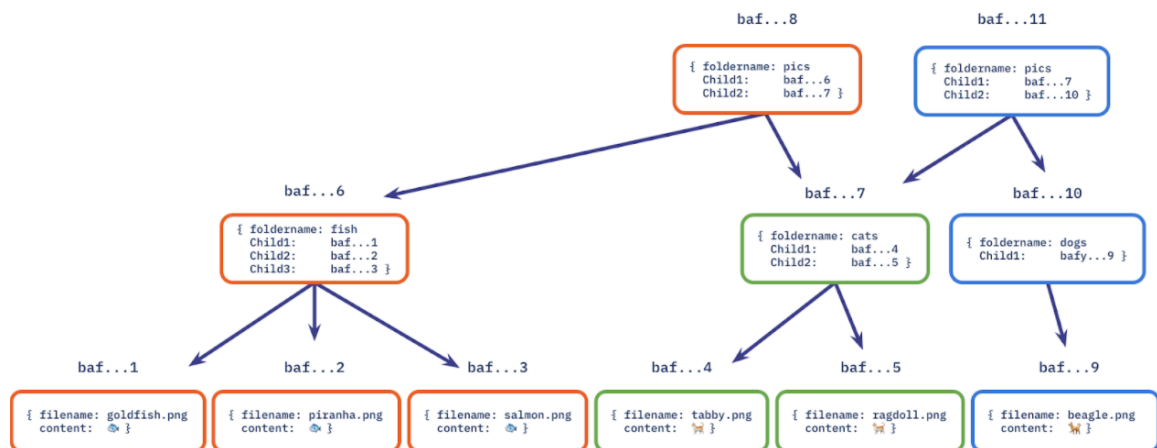


Figure 7 - Merkle DAG Implemented on a File System

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 $\text{Keccak-f}[b]$, where $b \in \{25, 50, 100, 200, 400, 800, 1600\}$ 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 $w \in \{1, 2, 4, 8, 16, 32, 64\}$ and $b = 25w$. When implemented on a 64-bit processor, a lane of $\text{Keccak-f}[1600]$ can be represented as a 64-bit CPU word.

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



Table 1 - Keccak Parameters Per Length

	<i>r</i>	<i>c</i>	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

Table 1 shows the parameters defining the standard instances. 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.

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-separated instances give unrelated outputs and act as independent functions.

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 are used on different areas of the Philippines. The usual proposition to solve this is to create a unified website hosted in a central server.



Proposed Documents Storage Structure

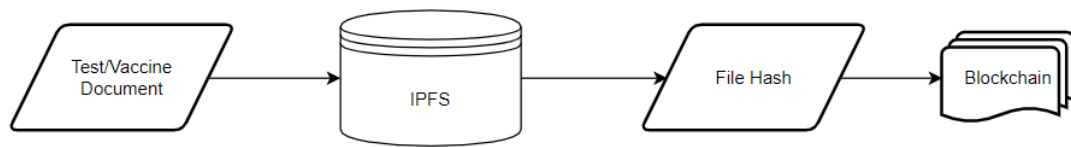


Figure 8 - Diagram of Proposed Solution

Figure 8 illustrates the summarized approach in solving the problem in document storage. The main components of this application are IPFS for file storage and blockchain for logging records of transaction in the system.

Conceptual Framework

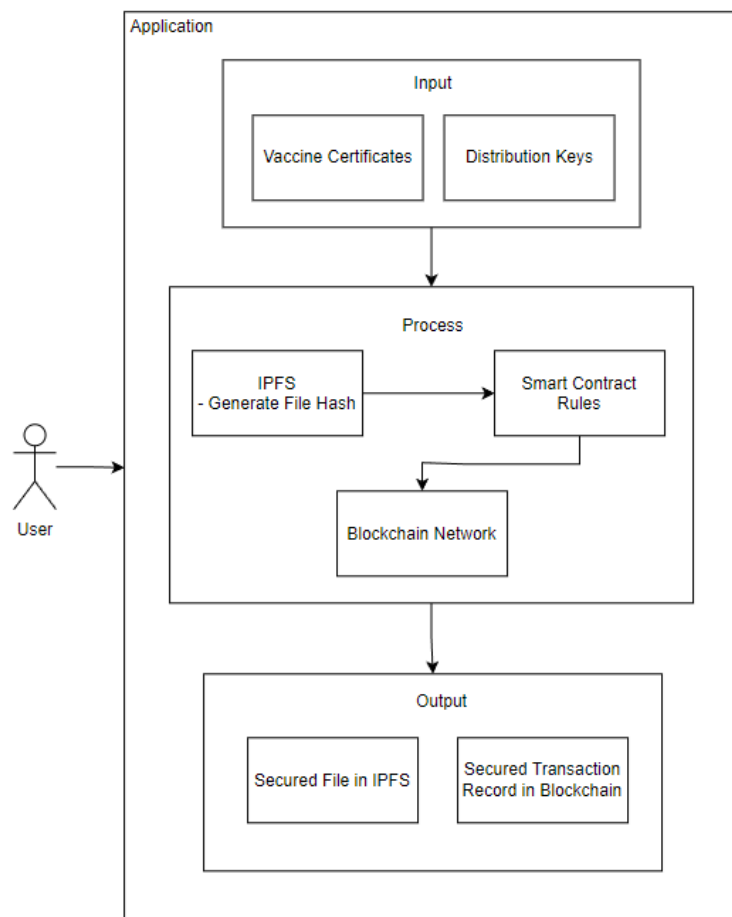


Figure 9 - Conceptual Framework



Figure 9 depicts the conceptual framework. The users of the proposed application are patients, medical workers or other third-party requiring the patient to present a COVID-19 Vaccine Certificate. The users access the same application but with different levels of access depending on their role.

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

Once all required inputs are provided, the file goes 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 is stored in the blockchain after going thru smart contracts. Once the blockchain successfully updated the network, provided file becomes an immutable component of both IPFS and blockchain network.

The researcher conducted interviews and probing to gather necessary information in building the proposed application. Also, differentiating each users on their level of access and scope of functionality are included on the specifications.



Chapter Four

METHODOLOGY

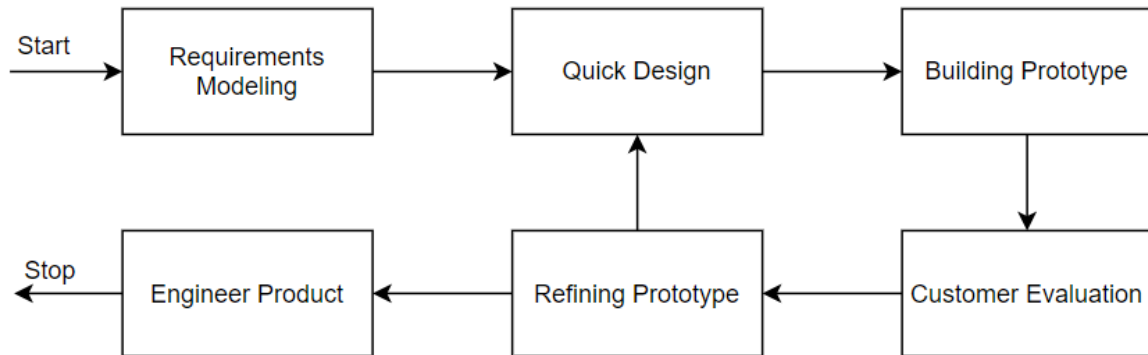


Figure 10 - Prototype Model Phases and Process

Figure 10 illustrates Prototype Model used by the researcher in developing the proposed study 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.

4.1 Requirements Modeling

The Prototype Model starts with outlining the requirements. The researcher conducted 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 this investigation, the researcher was able to piece out a picture of transactions involved and analyzed them against the proposed solution. This



information also enabled 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 ensures a scheduled slot on a specified date. On the day of vaccination, patient is 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, authorized medical unit signs a vaccination card while tagging the patient in their system as fully vaccinated.

Mocked test data were used for the application prototype. 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.

Below are requirements grouped by specific role:

Patient

- Register and Login – register to gain access to the system
 - o Upon registration, system creates 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 gives a bird's eye view to the client of the application.

4.2.1 Functional Specifications

Application Users

Patients

- Provides data which in turn converted to information needed to generate vaccine certificates.

Medical Unit

- Creates vaccine record for recognized patients by the system.

Verifying 3rd Party

- Has the capability to verify patient's info via uploading vaccine certificates or scanning vaccine QR code.

Application Responsibilities

The primary responsibilities of the application are:

- provide patient easy access to their vaccine information
- allow external parties (private companies, mall establishments) to verify vaccine records
- automatic vaccine certificate generation once necessary information is available
- ease of use for medical personnel in the upkeep of vaccine records
- provide other means of uniqueness such as QR code to verify authenticity
- maintain data integrity by storing information in a blockchain setup



General Requirements

- The application shall allow patients to download vaccine certificates and view their vaccine record
- The application shall provide QR codes for patient information and vaccine information
- The application shall store transactions in a blockchain network
- The application shall store encrypted files in IPFS
- The application shall allow 3rd parties to verify vaccine information authenticity either via uploading a file or scanning a QR code
- The application shall provide a facility for medical unit to input vaccination transaction and records
- The application shall generate QR code and vaccine certificates once patient vaccine information is available

4.2.1.1 Context Diagram

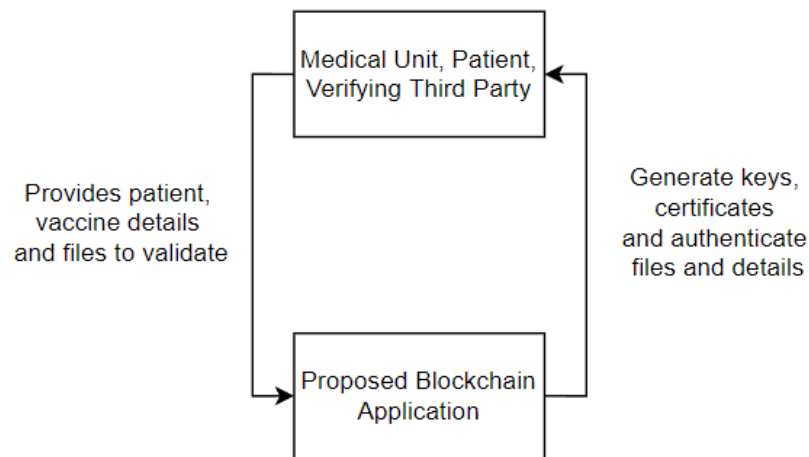


Figure 11 - Context Diagram

The context diagram shown in Figure 11 summarizes the application on inputs and outputs of the system and targeted users. On general, users of the application are required to provide public/private keys and vaccine details. Application generates vaccine certificate and summary details based off these details. After file generation, it triggers and executes various processes to upload, encrypt/decrypt, or release files. Note that this is a general illustration of inputs and outputs.



4.2.1.2 Data Flow Diagram

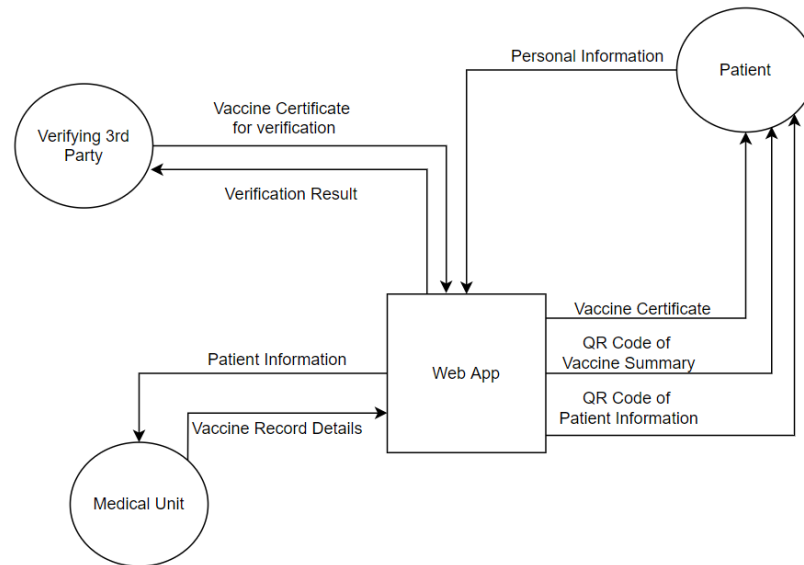


Figure 12 - Data Flow Diagram

Figure 12 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 Patients are the crucial part of data flow since most of the essential information comes from them. Both Medical Unit and Verifying 3rd Party have auxiliary processes that consumes Patient data.

The web application serves as the gateway for all types of users to access vital functionalities such as record creation and validation. Within the application, the researcher will implement role access for it to recognize the user and delegate the appropriate task or functionality assigned to its type. In this way, data will be protected and ensured of its access.

It is important for the application to implement security measures for the Patient type user as it will involve disclosure of personal information.



4.2.1.3 Use Case Diagram

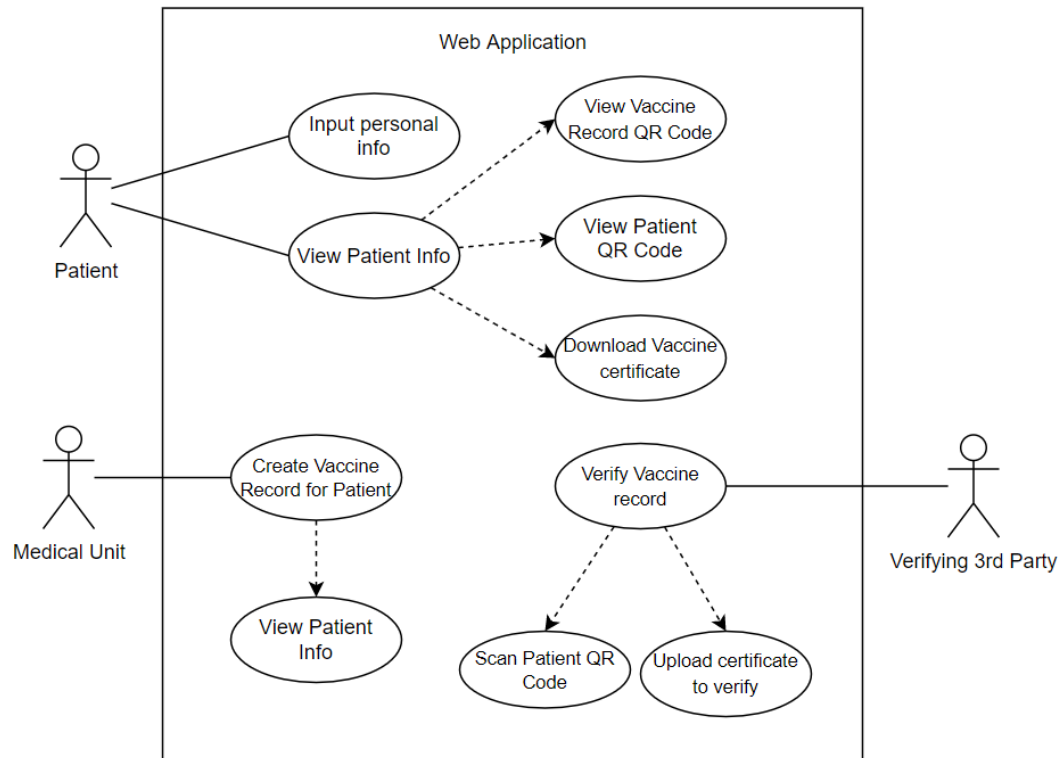


Figure 13 - Use Case Diagram

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 are demonstrated in Figure 13 and utilized a Use Case Diagram. The patient, being the central user of this system provides 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. For the 3rd party validators, it is expected that they have codes and files generated from the web application for it to have a successful transaction. In the event the item they want to be validated is not from the system, it will result in an error notification.



4.2.1.4 Functional System Flowcharts of the Proposed Application

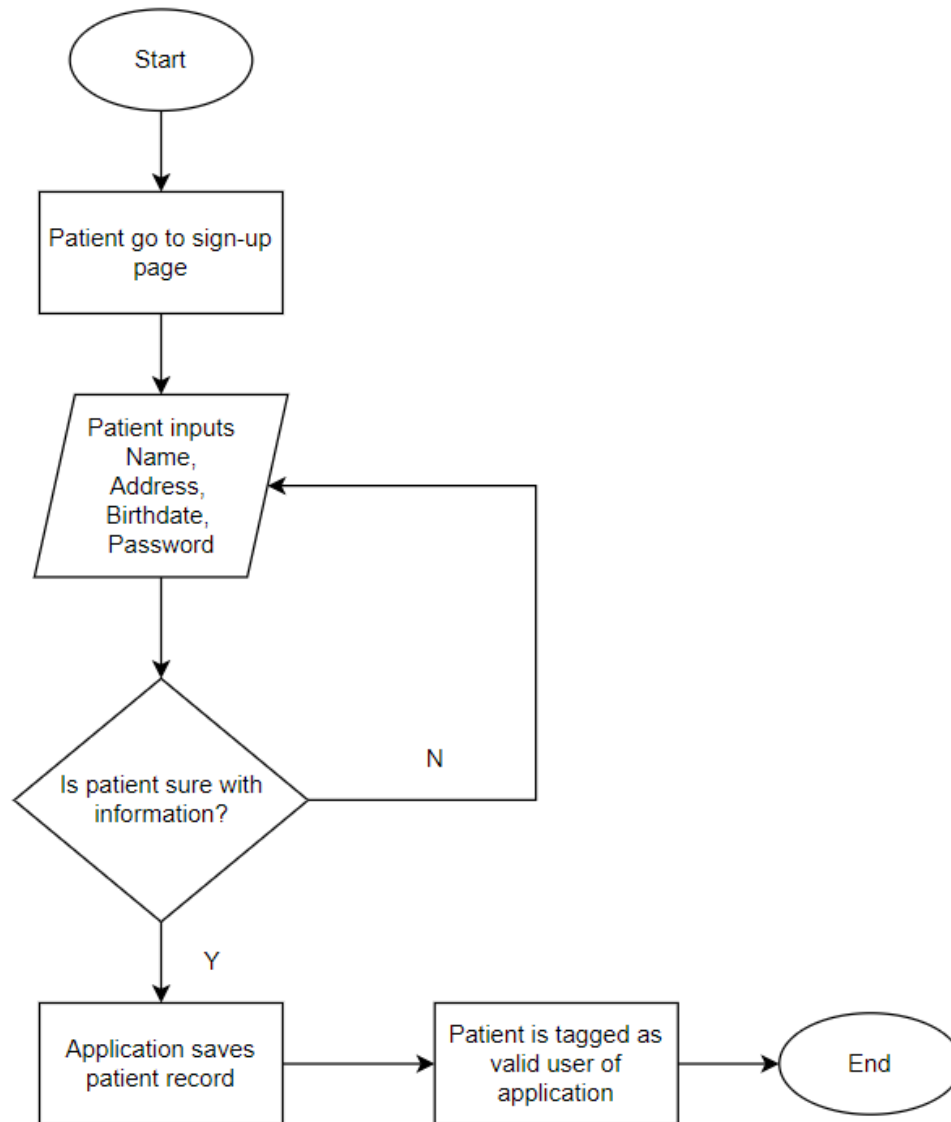


Figure 14 - Patient Registration Flowchart

Figure 14 is about Patient Registration. Once the patient has navigated to the sign-up page, he is required to input personal information. Once he has finalized the information, the application saves the record and triggers the process for the user to be tagged as a valid user of the application.

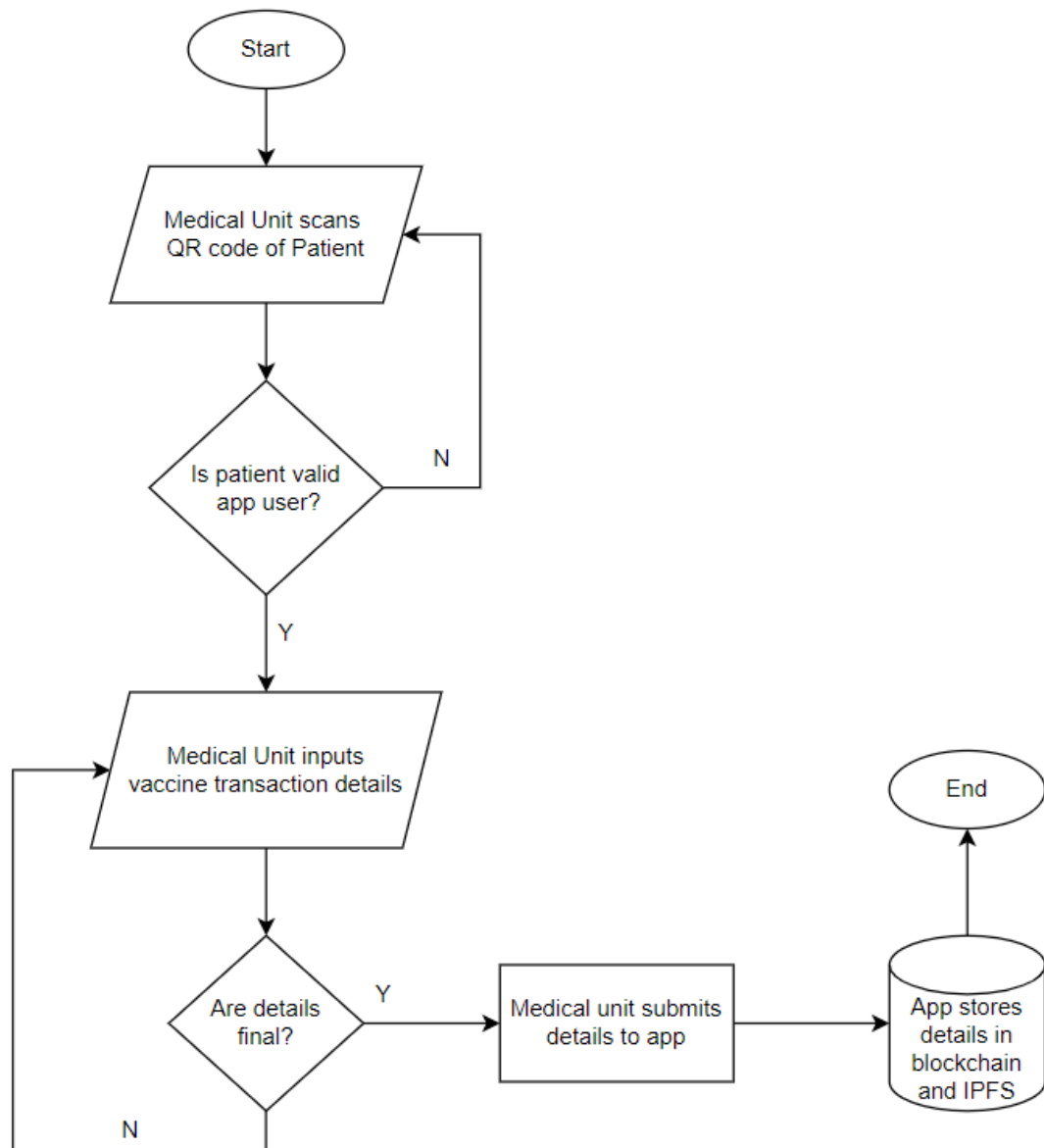


Figure 15 - Vaccine Record Creation Flowchart

Figure 15 illustrates vaccine record creation. Upon scanning a valid patient QR Code, Medical Unit is required to input vaccine details such as vaccine brand, lot number and vaccination date. Once information is finalized, it is submitted to the application and stored in IPFS and blockchain. If the scanned patient QR code is invalid, Medical Units are not allowed to enter vaccine information.

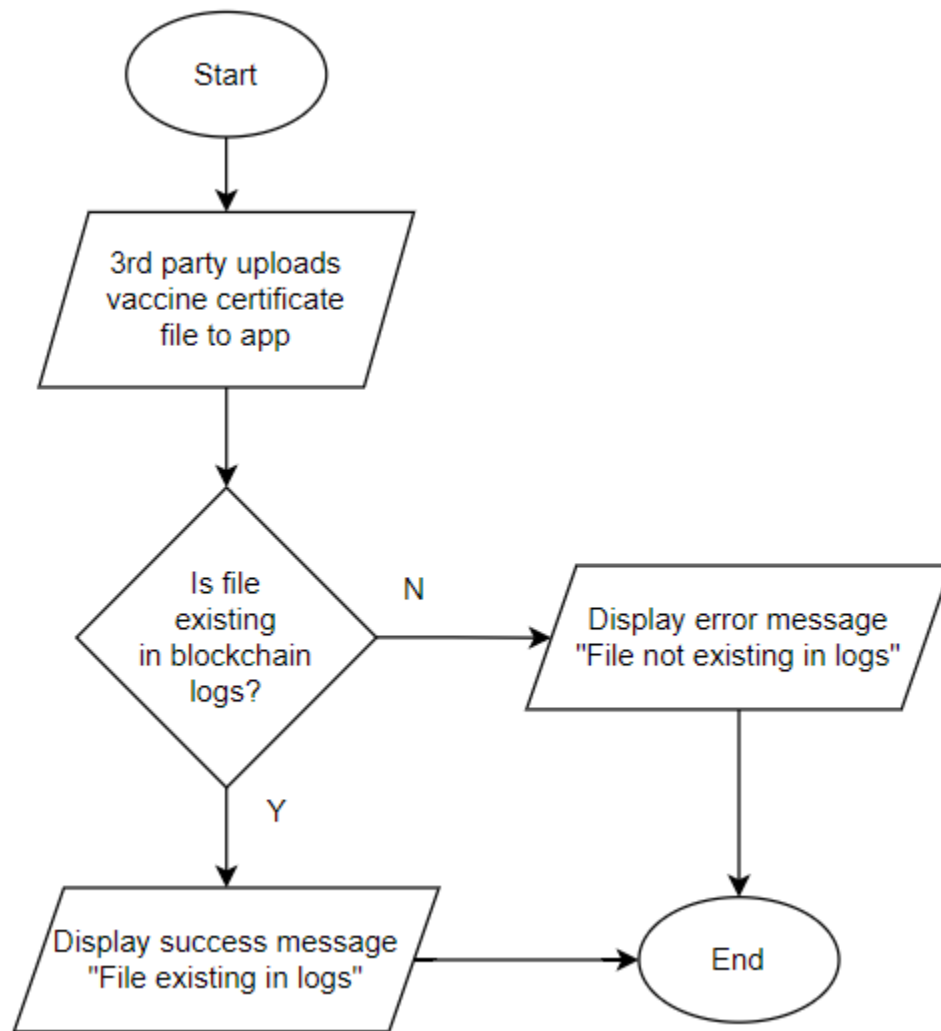


Figure 16 - Verify Vaccine Certificate File

Figure 16 tackles vaccine certificate file verification. 3rd party validators upload the vaccine certificate file into the application. It generates a file hash for the uploaded file then tries to find said hash from the blockchain logs. If the hash exists, the file is deemed as valid and a success message is displayed. But if the hash does not exist, it displays an error message.

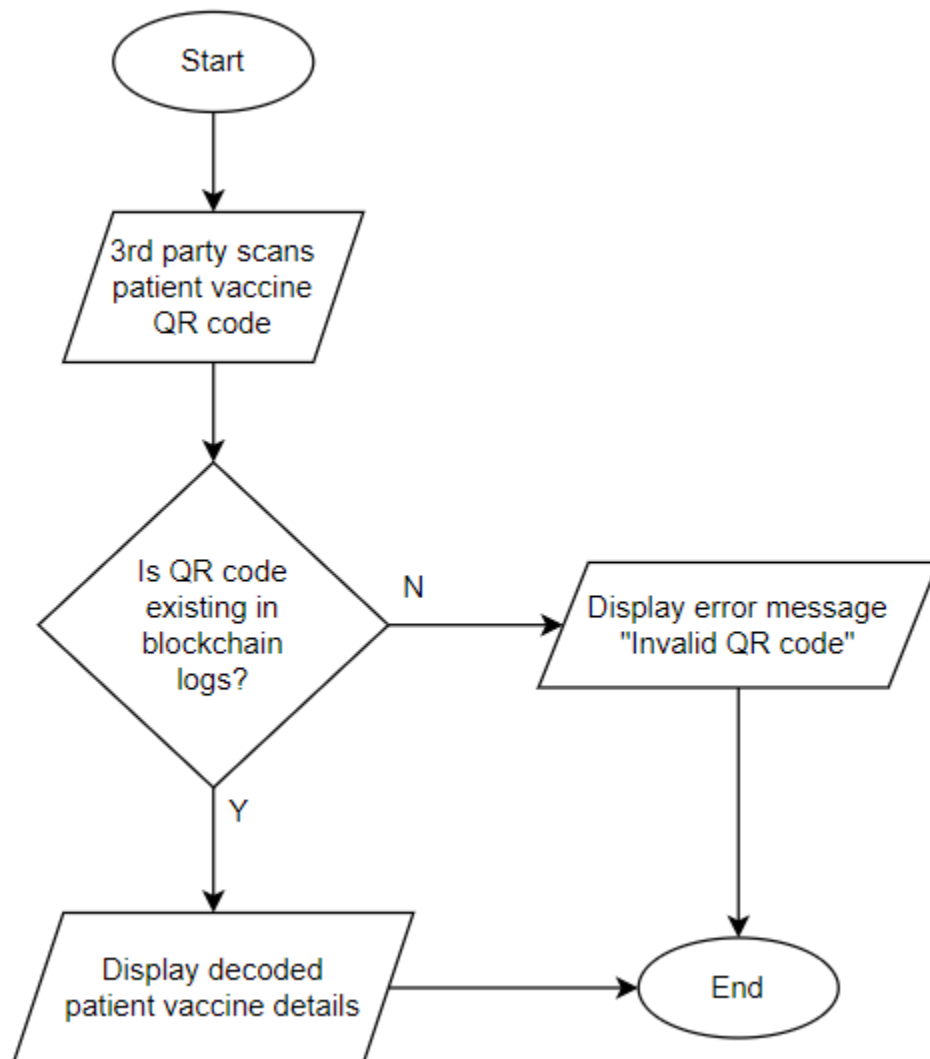


Figure 17 - Verify Vaccine QR Code

Figure 17 is about vaccine QR code verification. 3rd party validators scan a vaccine QR code from a Patient. Application extracts summary hash from the QR code. If the hash exists, the QR code is deemed as valid and a success message is displayed. But if the hash does not exist, it displays an error message.



4.2.2 Technical Specifications

4.2.2.1 General Software Requirements

Frontend Specifications

Table 2 - Frontend Specifications

Framework	Responsive Web Application
Language	ReactJS, Javascript
Other plugins (downloaded via Node Package Manager (NPM))	axios bootstrap js-file-download

Backend Specifications

Table 3 - Backend Specifications

Framework	Web API
Language	NodeJS
Database	MSSQL Server 2019
Other Plugins (downloaded via Node Package Manager (NPM))	sequelize html-pdf ipfs-http-client openpgp qrcode

Blockchain

Table 4 - Blockchain Specifications

Language	Solidity
Blockchain Protocol	Ethereum - Go Eth (GETH)
Consensus Protocol	GETH Clique PoA
Client-Wallet Interface	Web3
CryptoCurrency Wallet	MetaMask



4.2.2.2 High-Level System Design Diagram

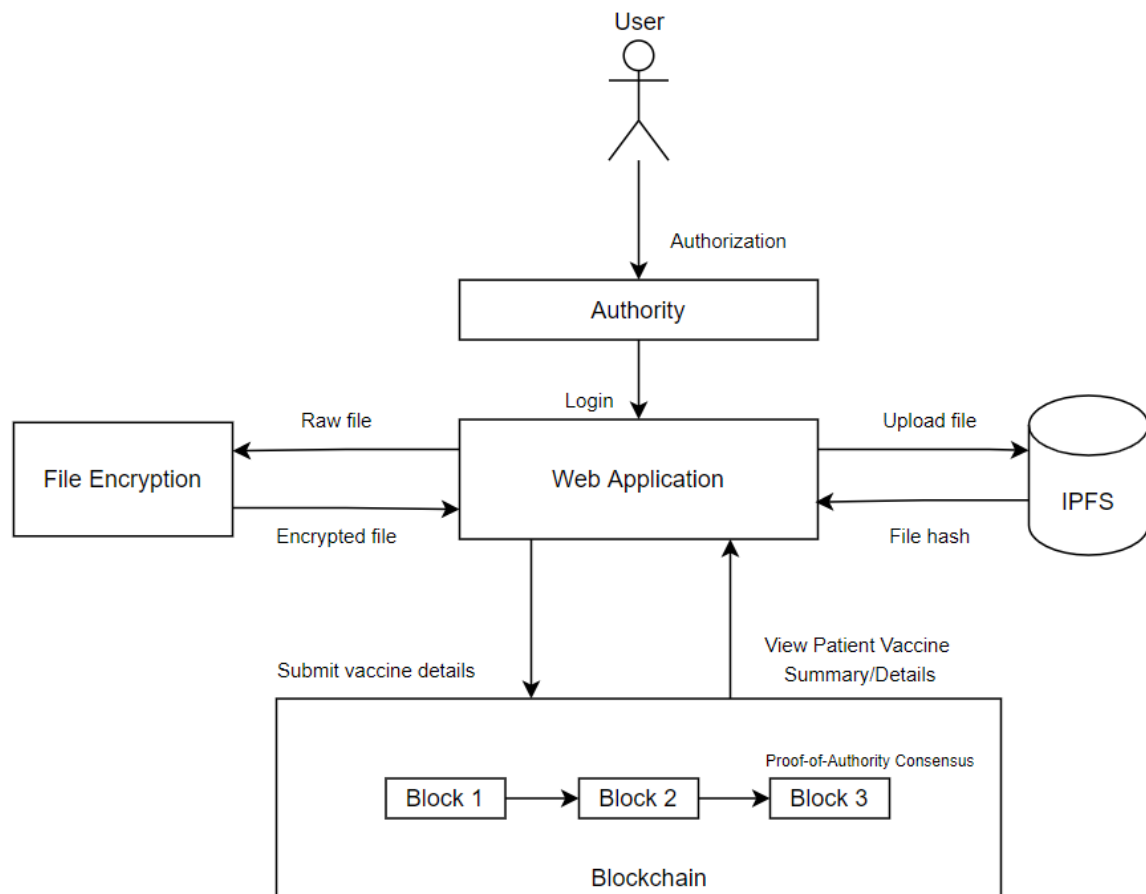


Figure 18 - High-Level System Design Diagram

The diagram shows four core components of the application. The web application is the user's gateway for interaction and access its various functionalities. From the web app, file encryption is then called for data masking feature. It is worth noting that data masking is a prerequisite process for a file to be uploaded in IPFS. Any file transaction done within IPFS are logged to the blockchain. Other transaction logged within the blockchain is summary details of a vaccine event. In this diagram, the web application is highlighted as the mediator between IPFS, File Encryption and blockchain. Main function of the web app is to orchestrate the http requests and responses for these components.



4.2.2.3 System Architecture Diagram

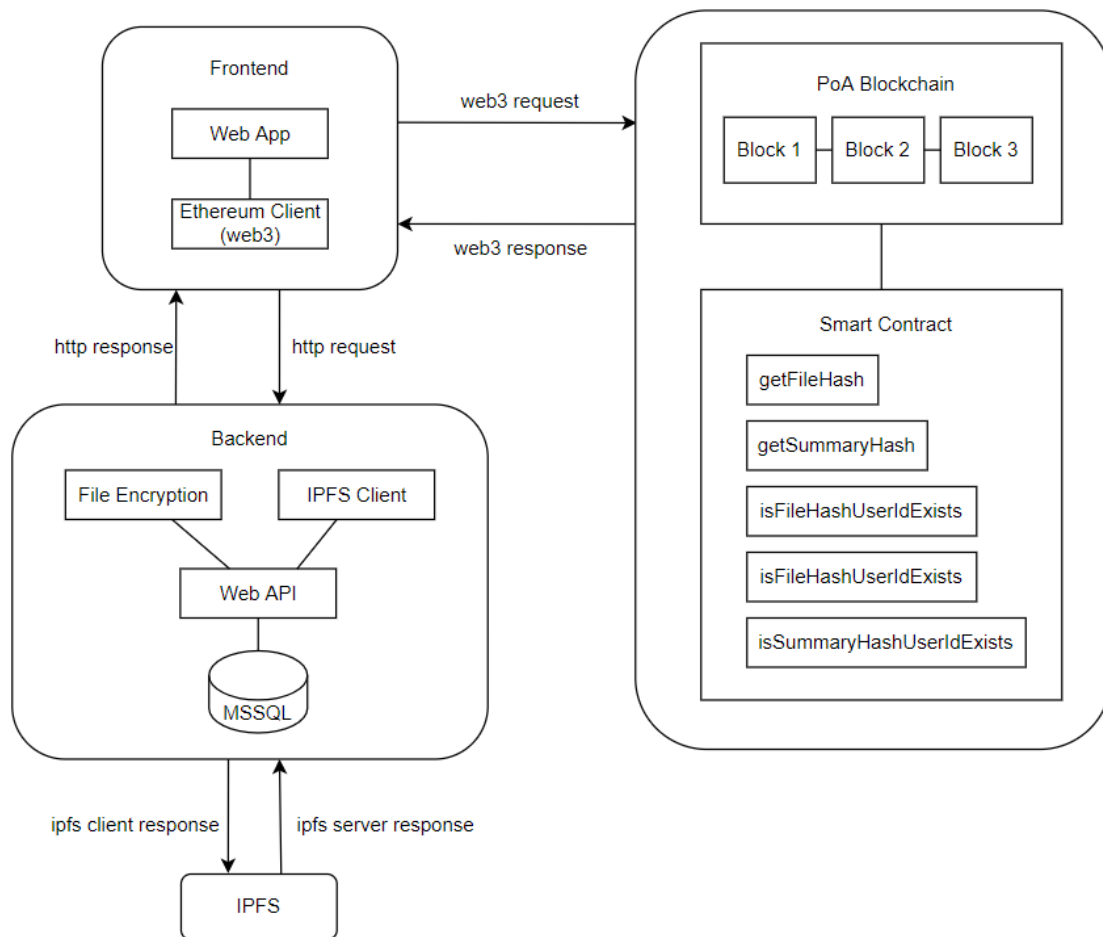


Figure 19 - System Architecture Diagram

The frontend sector is composed of the web application and Ethereum Client. As discussed with the High-Level System Design, this is the gateway for the user to trigger events for transactions. In this application, web3 was used which is a frontend Ethereum client and is pluggable with ReactJS. This is the reason why blockchain interactions are triggered from the web application.

The backend sector is composed of the Web API, File Encryption, Database and IPFS Client. From here, the frontend makes http requests to process file encryption/decryption and IPFS uploads and downloads. The IPFS client used was only pluggable with NodeJS which is why all IPFS interactions are only done in the backend.



CRUD (create, read, update, delete) transactions in the database are also initiated in the backend.

The blockchain sector is composed of the blockchain itself and smart contracts. As specified above, it is only the frontend that can trigger requests to smart contracts. It is worth noting that for file transactions, it is first processed from the backend to generate a file hash which is then passed to the frontend and finally, the frontend making a request to the blockchain to process file hash. As per the functional specifications from previous area of this chapter, it is in this sector that the final processing of transaction logs from the vaccine record creation happens. Once a log is made from the blockchain, the application deems the vaccine details final and valid

4.2.2.5 Data Structure Diagrams

4.2.2.5.1 Blockchain Data and Function Mapping

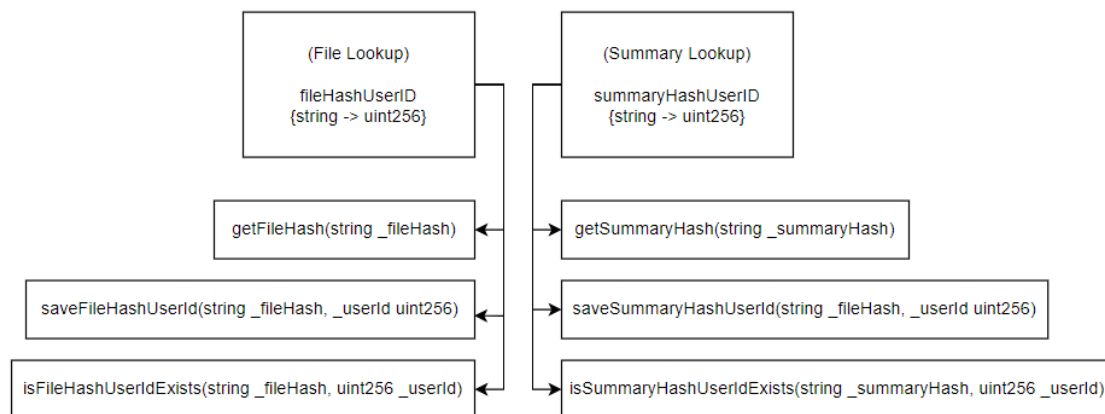


Figure 20 - Blockchain Data and Function Mapping

The blockchain is validated by two lookups: file and summary. Both lookups are of keyvalue pair with typings of {string (key), uint256 (value)}. This maps out patients with one summary or file hash.

Both lookups have separate functions for CRUD as this follows with the Solidity Coding Standard of separation of concerns and updating of values inside the blockchain.



4.2.2.5.2 Relational Database Diagram

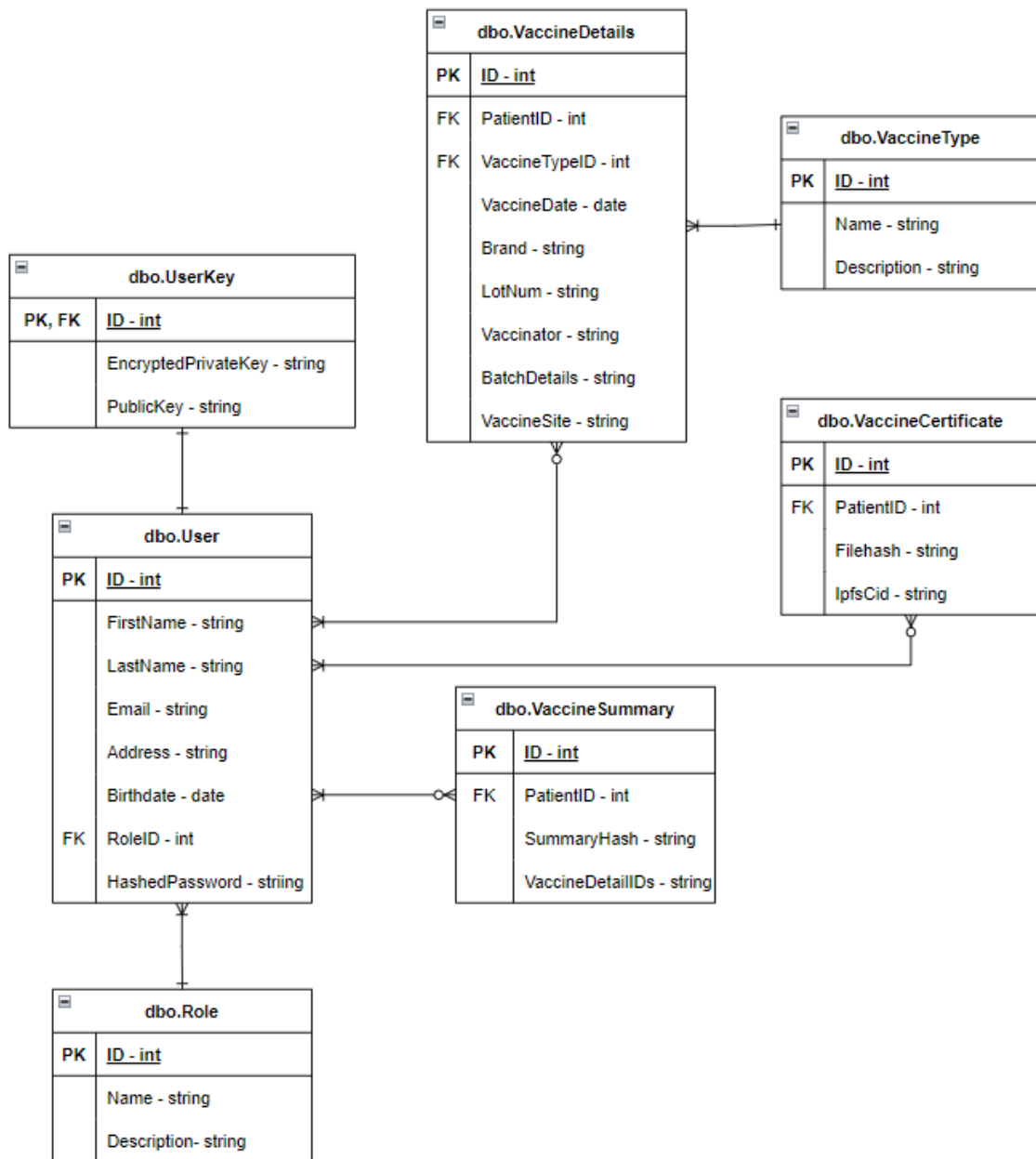


Figure 21 - Relational Database Diagram

It is important to note the segregation of VaccineSummary and VaccineCertificate. This allows for better handling of information when being transmitted and retrieved from the blockchain. Also, a crucial entity of the schema is the User table as this is being referenced by other tables via its primary key.



4.2.2.6 Transactional Operation Diagram

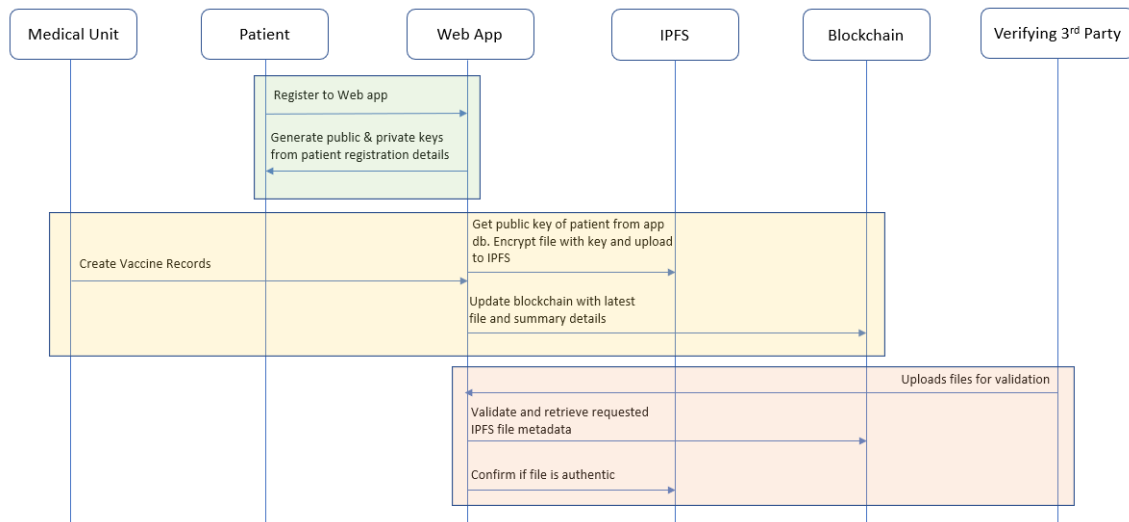


Figure 22 - Transactional Operation Diagram

Figure 22 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 is prompted by the patient. Without these keys, medical personnel cannot upload files which in turn, the third parties are not able to request any file validation.

The diagram attempts to highlight the overlaps of processes between the application's entities. It is worth noting that blockchain and IPFS both interlaps with vaccine record creation and validation processes. This is because both entities are crucial to maintain the immutable nature of the data.

Also, medical units are responsible in most of the record modification as both patient and 3rd party validators are just concerned with data acquisition.



4.2.2.7 Technical Flowcharts

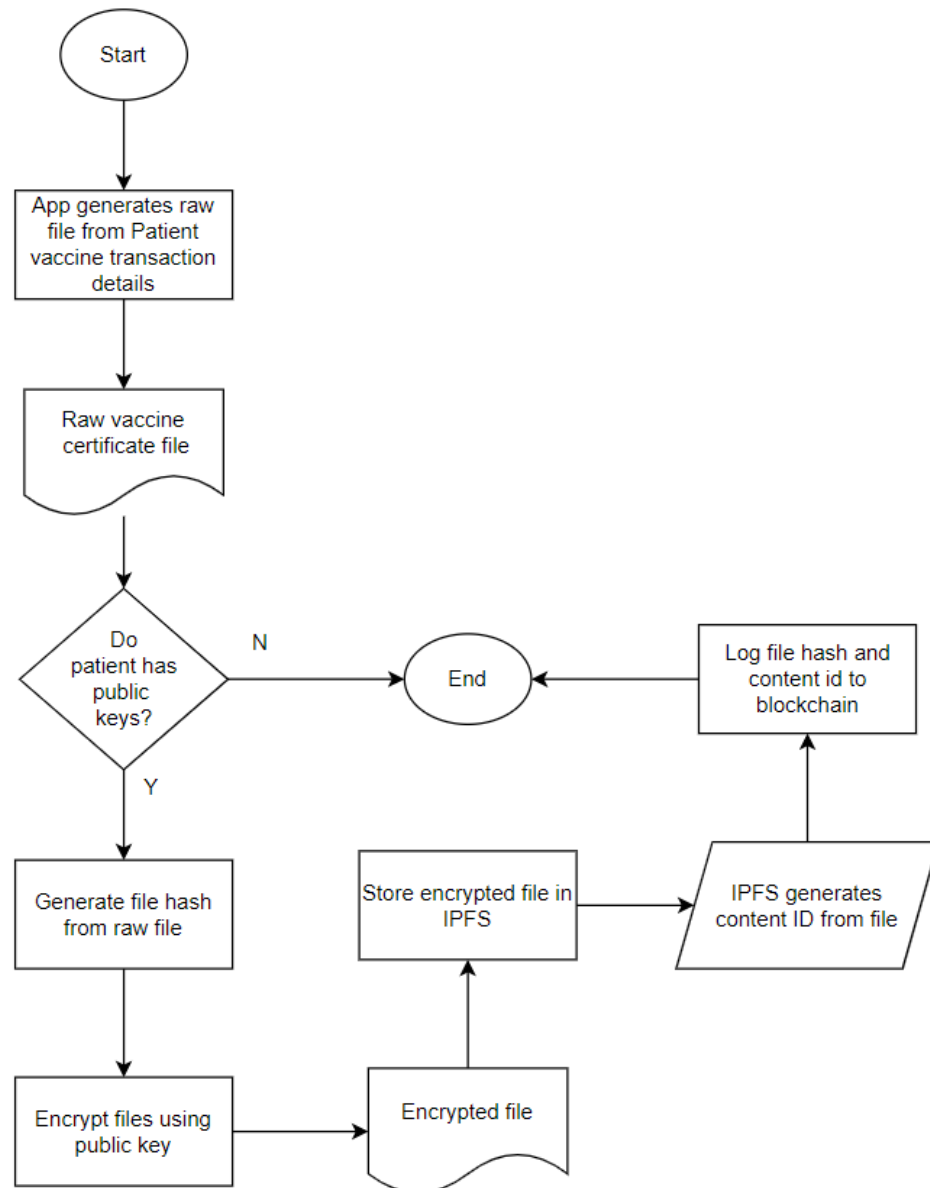


Figure 23 - File Storage in Blockchain and IPFS Flowchart

Figure 23 dwells with file storage in blockchain and IPFS. Given vaccine record is created, the application generates a raw .PDF file from the details. It then retrieves patient's public key. If patient has no public key, process is aborted. If public key is existing, application proceeds to generate a file hash then encrypt the file using the public key. Encrypted file is then stored in IPFS. IPFs generates a Content Identifier (CID) from this transaction. File hash and CID are be logged onto the blockchain.

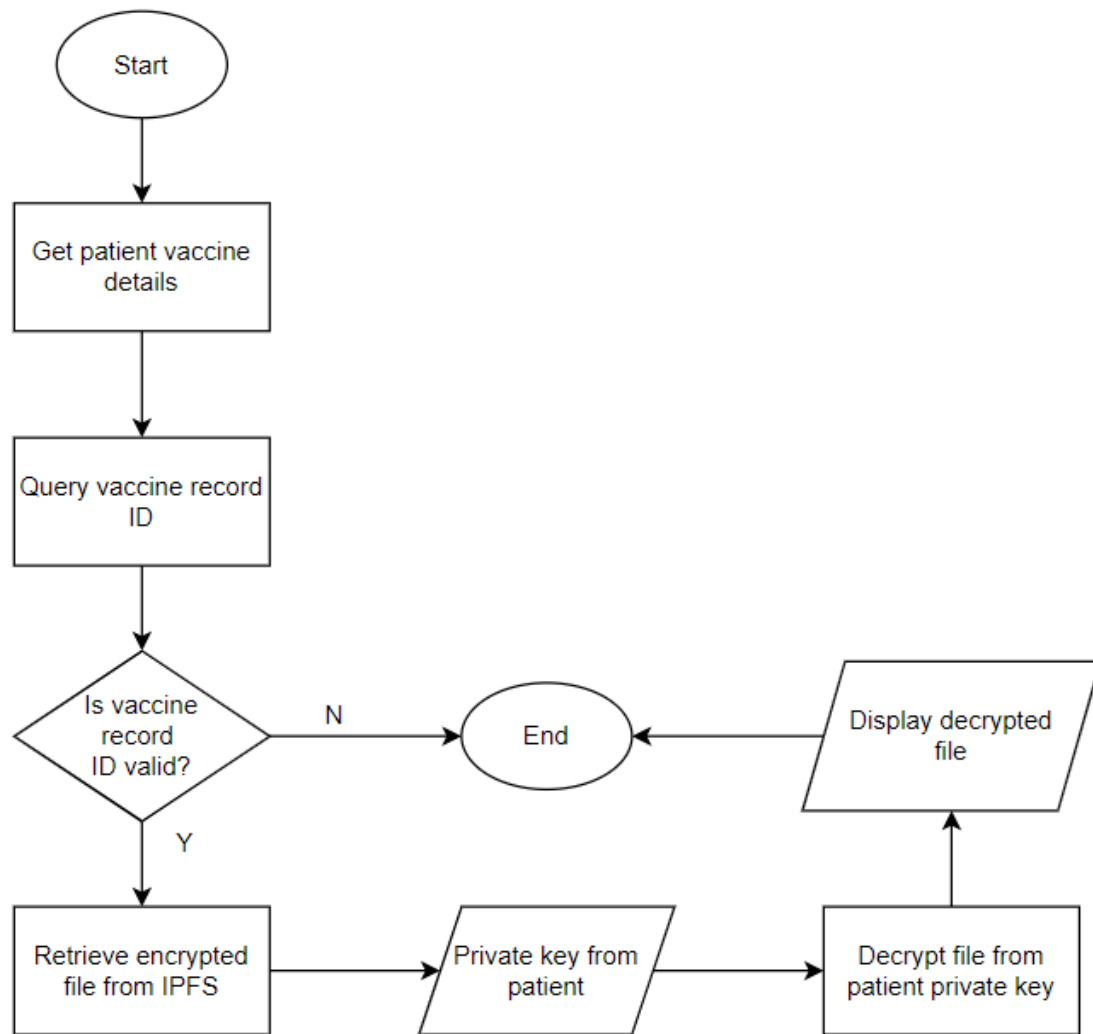


Figure 24 - File Retrieval Flowchart

Figure 24 tackles file retrieval. Application first retrieves patient details and identify patient's unique identifier. If vaccine record exists for the patient, application proceeds to query patient's file hash from the blockchain. If hash exists, it proceeds to retrieve the encrypted file from IPFS. After this, patient's private key is used to decrypt the vaccine certificate file. Decrypted file is now available for download and displayed in the application.

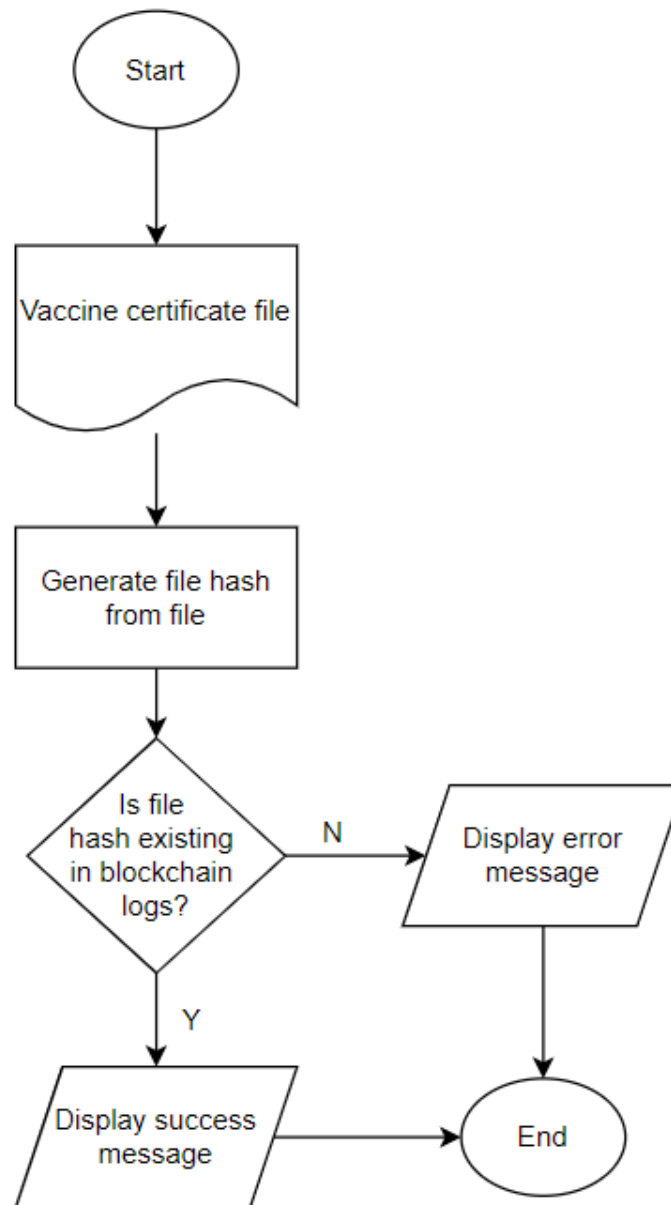


Figure 25 - Validate File from Blockchain Flowchart

Figure 25 discusses file validation from the blockchain. 3rd party uploads a raw file to the application. Application generates a file hash from the raw file. It then queries the file hash if it exists in the blockchain logs. If exists, it displays a successful verification message. And if not, it displays an error message.

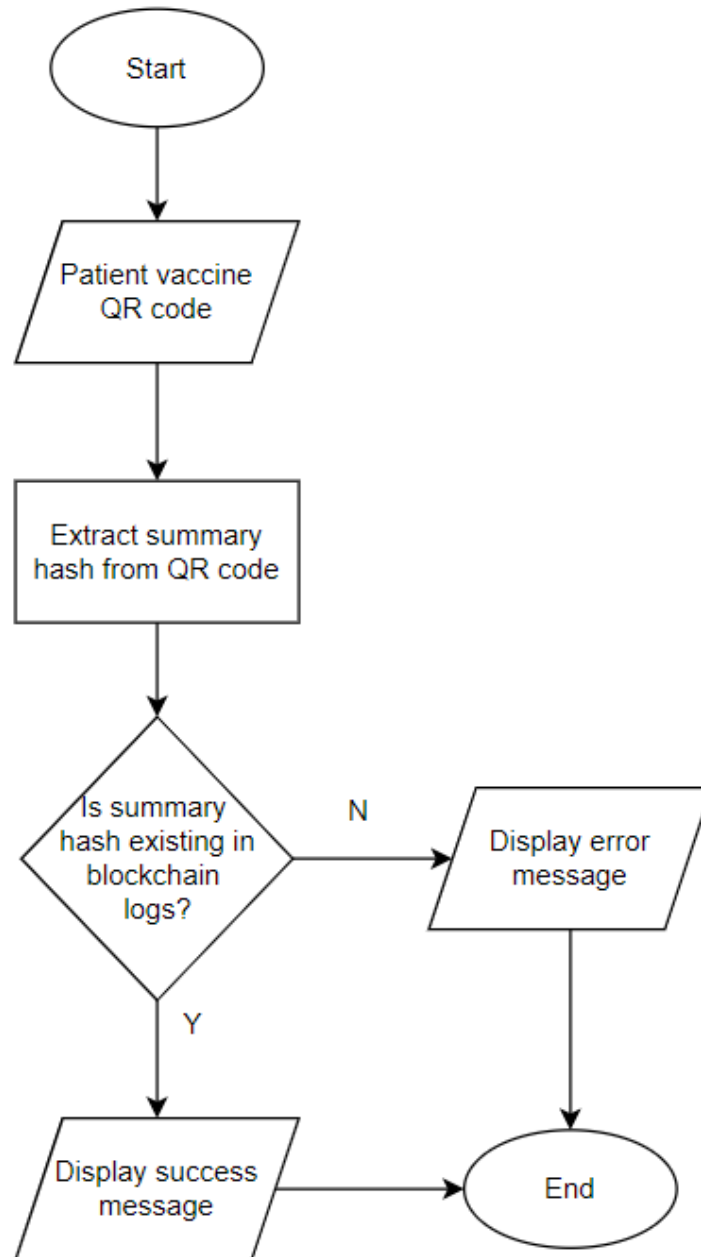


Figure 26 - Validate QR Code from Blockchain Flowchart

Figure 26 illustrates QR code validation from the blockchain. 3rd party scans a patient vaccine QR code. Application extracts summary hash from QR Code. If summary hash exists, it displays a successful verification message. And if not, it displays an error message.



Chapter Five

RESULTS AND DISCUSSION

5.1 Functionalities of the System

This section tackles the various functionalities included in the application and shows screenshots of the development.

Table 5 - Functionality of the System

Functionality	Intended User/s	Description
Patient Registration	Patient	Allows unregistered users make an account
Patient Login	Patient	Entry point for patients using email and password
Patient Home Screen	Patient	Displays overview of patient details
Vaccine Record Creation	Medical Unit	Allows authorized medical personnel to input vaccination details to the system
Vaccine Certificate Validation	Third Party Validators	Allows validators to upload and verify a vaccine certificate file
Scan Summary QR Code	Third Party Validators	Allows validators to verify patient details via QR code

Table 5 shows the various functionalities built into the system. It maps out to the intended users and description. It is important to note that most of the functionality is patient-centric and user experience was made robust to their needs. Medical units have the crucial role of creating vaccine records while patient and 3rd party validators are particularly concerned in viewing and gaining access to the information.



Patient Registration

VaxMon

Register

First Name
Jennifer

Last Name
Fadriquela

Email
jennifer@test.com

Password
.....

Address
Paco, Manila

Birthdate (mm/dd/yyyy)
10/02/1991

Save

Figure 27 - Patient Registration Screen

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

Patient Login

VaxMon

Login

Email

Password

Log In

Figure 28 - Patient Login Screen

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



Patient Home Screen

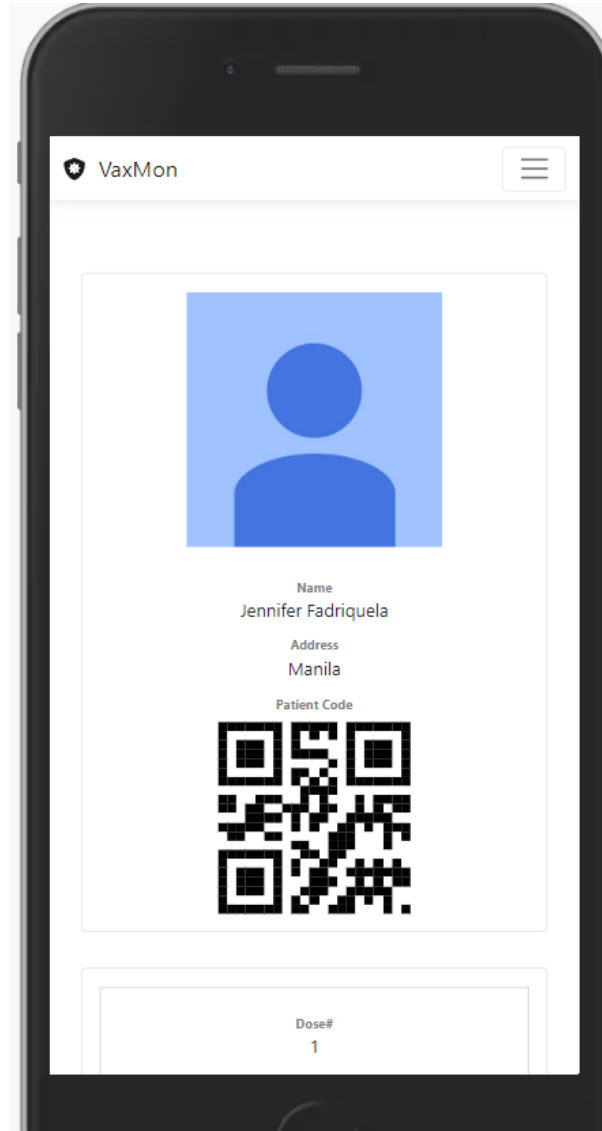


Figure 29 - Patient Home Screen (Upper Section)

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

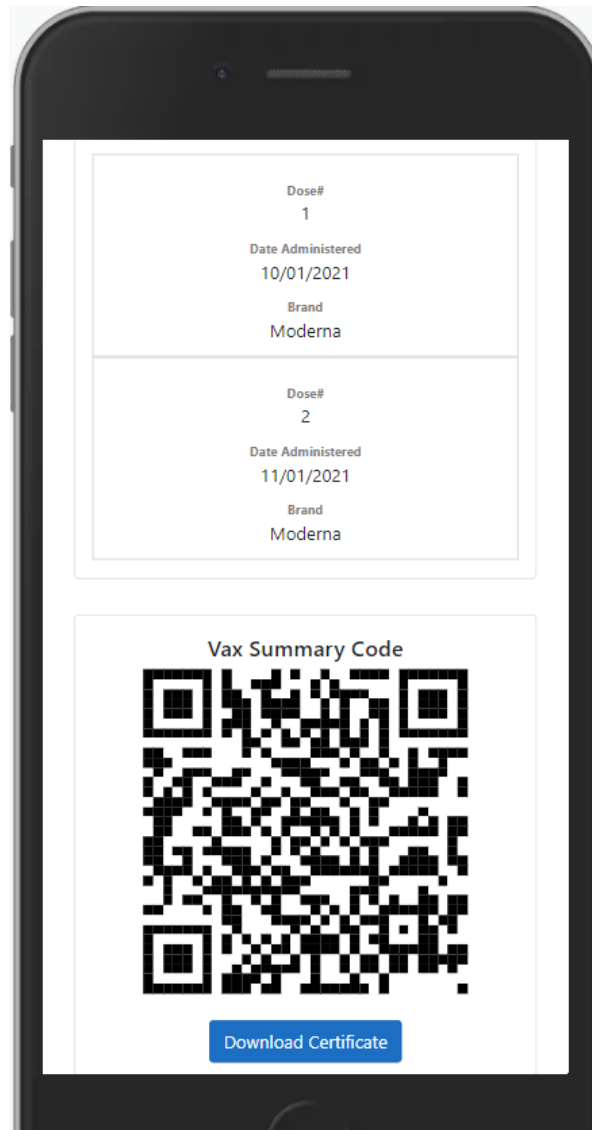


Figure 30 - Patient Home Screen (Lower Section)

Lower part of the home screen is for vaccination details. If the patient already got vaccinated, this section is displayed. It shows 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 downloads the vaccine certificate file



Vaccine Record Creation

Figure 31 - Vaccine Record Creation Screen

For medical personnel doing the vaccination, this page is 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) pops up to confirm the transaction. This logs the transaction to the blockchain

Vaccine Certificate Validation

Figure 32 - File Validation (Successful)



The screenshot shows the VaxMon interface for file validation. At the top, there is a 'Choose File' button and the filename 'patient-2.pdf'. Below this is a blue 'Validate' button. A red error message 'Invalid File' is displayed, followed by the text 'File is not existing in blockchain logs'.

Figure 33 - File Validation (Failure)

Figure 32 and 33 depicts file validation. For third-party validators that wants to authenticate a Vaccine Certificate file, this page is available. It requires the user to upload the file and displays a prompt that tells if file is valid or not (within the blockchain logs context)

Scan Summary QR Code

The screenshot shows the VaxMon interface for scanning a QR code. On the left, there is a 'Scan Summary QR Code' section with a QR code image. To the right, there is a user profile section with a blue silhouette icon, the name 'Jennifer Fadriquela', and the address 'Manila'. On the far right, there is a 'Dose#' section with two entries: Dose# 1, Date Administered 07/01/2021, Brand Moderna; and Dose# 2, Date Administered 08/01/2021, Brand Moderna.

Figure 34 - Scan Summary QR Code Screen



In figure 34, it illustrates validation via summary QR code. For third-party validators that wants quick details of patient's vaccine records, it requires a summary QR code from a patient and displays related details. QR code is invalid if app finds 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 includes supporting details such as userId and Content Identifier from IPFS.

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

Both file and summary hash lookup have the same structure:

<hash>: <userId>

Generated CID Sample:

QmZkJLp7PJGMc3mMSxTeLtyQCRqZ5CudGdjPB3jjTSFaoX: 1001

Above sample is comprised of CID which has the encoded identifier of the file uploaded in IPFS. This identifier makes the file unique and used as look up when retrieving files from the file storage network.



Clique Proof-of-Authority

The application used Clique Proof of Authority consensus. Below is a simulation of various test cases:

```
// block represents a single block signed by a particular
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
    // corner case 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"},
}
```



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

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
INFO [01-09|18:07:57.049] ⚡ mined potential block
INFO [01-09|18:07:57.174] Setting new local account
INFO [01-09|18:07:57.186] Submitted transaction
D905a283Ca27c7c07b11d2A1D6189 nonce=0 recipient=0xE76AE0227943de207E1D47197080Ea883108e711 value=0
INFO [01-09|18:07:57.843] Looking for peers
INFO [01-09|18:07:59.000] Successfully sealed new block
INFO [01-09|18:07:59.001] block reached canonical chain
INFO [01-09|18:07:59.016] Commit new mining work
INFO [01-09|18:07:59.016] ⚡ mined potential block
INFO [01-09|18:07:59.036] Commit new mining work
INFO [01-09|18:08:01.001] Successfully sealed new block
INFO [01-09|18:08:01.002] block reached canonical chain
INFO [01-09|18:08:01.020] ⚡ mined potential block
INFO [01-09|18:08:01.020] Commit new mining work
INFO [01-09|18:08:03.001] Successfully sealed new block
INFO [01-09|18:08:03.001] block reached canonical chain
INFO [01-09|18:08:03.068] Commit new mining work
INFO [01-09|18:08:03.068] ⚡ mined potential block
INFO [01-09|18:08:05.000] Successfully sealed new block
INFO [01-09|18:08:05.001] block reached canonical chain
INFO [01-09|18:08:05.019] Commit new mining work
INFO [01-09|18:08:05.019] ⚡ mined potential block
INFO [01-09|18:08:07.000] Successfully sealed new block

number=463 hash=fbc2c3..7128
address=0x6785221e588D9D5a283Ca27c7c07b11d2A1D6189
hash=0xe2935511e2ca4ea34d9e9a332959762f0735acfbdae485adabc51d8d83990f68 from=0x6785221e588
D905a283Ca27c7c07b11d2A1D6189 nonce=0 recipient=0xE76AE0227943de207E1D47197080Ea883108e711 value=0
peercount=1 tried=0 static=0
number=464 sealhash=21fcfc..2fa21d hash=b74917..786ce4 elapsed=1.980s
number=457 hash=866c5c..4fec
number=465 sealhash=13d8e9..53e555 uncles=0 txs=0 gas=0 fees=0 elapsed=15.659ms
number=464 hash=b74017..786c
number=465 sealhash=91a7e6..81e902 uncles=0 txs=1 gas=65762 fees=0 elapsed=36.060ms
number=465 sealhash=91a7e6..81e902 hash=ecac8c..21544a elapsed=1.971s
number=458 hash=655d5b..e8ea
number=465 hash=ecac8c..21544a
number=466 sealhash=b4efb0..0171f5 uncles=0 txs=0 gas=0 fees=0 elapsed=18.596ms
number=466 sealhash=b4efb0..0171f5 hash=a8dd07..9edbcd elapsed=1.998s
number=459 hash=e0eeb4..5650
number=467 sealhash=5b75fe..345822 uncles=0 txs=0 gas=0 fees=0 elapsed=67.807ms
number=466 hash=a8dd07..9edb
number=467 sealhash=5b75fe..345822 hash=923471..0e2cca elapsed=1.999s
number=460 hash=606ed4..c05e
number=468 sealhash=8a5426..bc4884 uncles=0 txs=0 gas=0 fees=0 elapsed=19.151ms
number=467 hash=923471..0e2c
number=468 sealhash=8a5426..bc4884 hash=2255de..b88f17 elapsed=1.999s
```

Figure 35 - Node Blockchain Logs for File Upload

```
INFO [01-09|18:07:57.049] ⚡ mined potential block
INFO [01-09|18:07:57.174] Setting new local account
INFO [01-09|18:07:57.186] Submitted transaction
D905a283Ca27c7c07b11d2A1D6189 nonce=0 recipient=0xE76AE0227943de207E1D47197080Ea883108e711 value=0
INFO [01-09|18:07:57.843] Looking for peers
INFO [01-09|18:07:59.000] Successfully sealed new block

number=463 hash=fbc2c3..7128
address=0x6785221e588D9D5a283Ca27c7c07b11d2A1D6189
hash=0xe2935511e2ca4ea34d9e9a332959762f0735acfbdae485adabc51d8d83990f68 from=0x6785221e588
D905a283Ca27c7c07b11d2A1D6189 nonce=0 recipient=0xE76AE0227943de207E1D47197080Ea883108e711 value=0
peercount=1 tried=0 static=0
number=464 sealhash=21fcfc..2fa21d hash=b74917..786ce4 elapsed=1.980s
```

Figure 36 - Detected Metamask Address of Log Sender

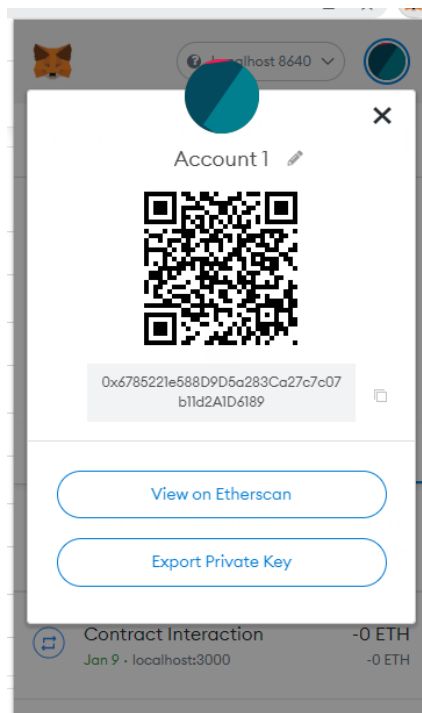


Figure 37 - Metamask account details of vaccine file uploader

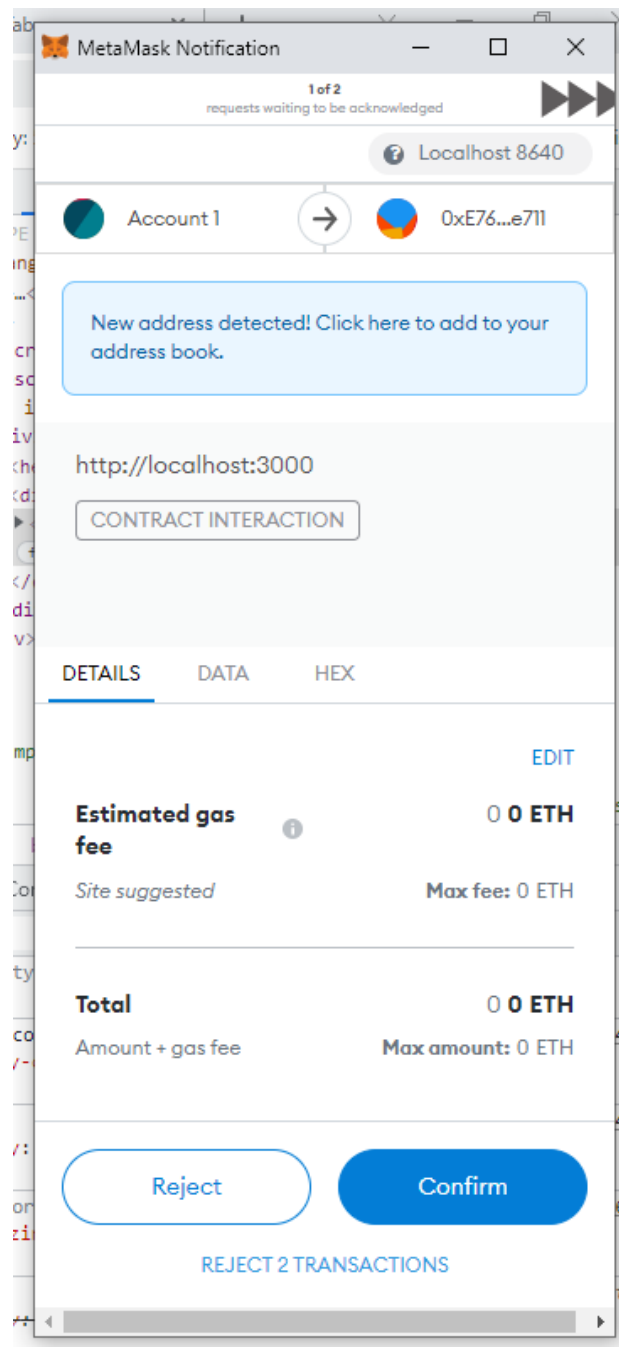


Figure 38 - Metamask Notification of File Upload Transaction



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 shift to accumulator then XOR with b

return accumulator



#example

```
#input_bytes = [104, 101, 108, 108, 111, 32, 119, 111]
```

each iteration results to (consecutively)

0

28416

7304960

1870077952

478739984128

122557435964416

31374703606918144

8031924123371070720

8031924123371070824

```
#final value is 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

[illegible]



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, 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]
```

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 is:

```
[[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
```



```
1  
= int(log(lanew, 2))  
= 6
```

```
number_of_rounds = 12 + 2 * 1  
= 24
```

5.3 Implementation of Merkle DAG

The algorithm used in IPFS to manage content and assets is Merkle DAG. For the below example, a small size text file is used to better illustrate the process. The default chunk size of IPFS is 256Kb but in this example it is reduced to 32Kb to have appropriate representation using small sample files.

File 1

Name: cert_allen_smith.txt

Size: 86 bytes

Content:

```
this is a sample certificate given to Allen Smith  
and only used for research purposes
```

File 2

Name: cert_john_doe.txt.txt

Size: 83 bytes

Content:

```
this is a sample certificate given to John Doe  
and only used for research purposes
```



Generated Details for cert_allen_smith.txt:

Table 6 - Generated Hash Value for Sample Record #1

Node Type	Size (Bytes)	Hash
Root	0	QmZkJLp7PJGMc3mMSxTeLtyQCRqZ5CudGdjPB3jjTSFaoX
Links	32	QmdsyzBk5nWmC7a92gaAuRHxWTQu6e4wwyv2bVmZtF7mcq
Links	32	QmPsFk9hcP4WmN96r8mXYjV5rKCNNb94c95jfqLBNZvigT
Links	22	QmVVrfBPAnF5DC1DXDZH2yftW6MEoCSKXEQEbY5LKfFzAt

Generated Details for cert_john_doe.txt:

Table 7 - Generated Hash Value for Sample Record #2

Node Type	Size (Bytes)	Hash
Root	0	QmanmTVLostTheeLiz8vr99QDWmVbmbd53rSA2iFoDcmXu
Links	32	QmdsyzBk5nWmC7a92gaAuRHxWTQu6e4wwyv2bVmZtF7mcq
Links	32	QmTfDsTDe3nVu7b3hij43R3mBzyhJZgVm9eFBewVb5FfKV
Links	32	QmRF3DNTkA43a7AG26uva4n7pgR22ctz6PjZW4KMuN5Cvu

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

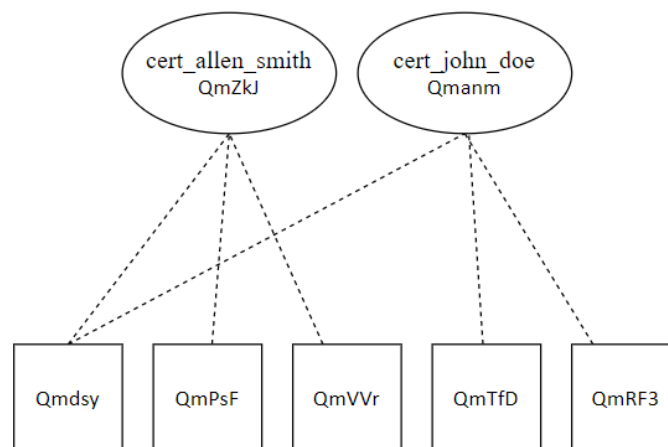


Figure 39 - Merkle DAG representing sample records



Actual IPFS DATA

```

-----BEGIN PGP MESSAGE-----
Version: OpenPGP.js v4.10.10
Comment: https://openpgpjs.org

wV4DA5AqweADDBUSAQdAp2uzFP/qva6I00lMmdHBpKpVqrhxL6/NhA7ddbrQ
WCswd/1aoTGUL3WK4UrsuBbFiTmuLVdVr/vXpCCceZX8Ftsjmm2uABUmolV2
rP/1Btng0v8AAQAEaK4iwyDyDtIJA8Q3T8MbXS/s4fYgyFG6aEvZ5Fwpq
dkRQGKdrT8jpmUsZKlruKuZ+TPEvxZLUHDKT1d1ymxpHPPx0PZxjyu/zn6q/
tRldhTddQM1RaSmzWAIGHWm9HTF41oNcLbalQfj2XHxttccrLPiAu79x17Xo
sy+0Skbq0PE5gO2I6gjr5nXvJC4J4ThChWfZikpkPk2ZFU1gZxwh0wSA0T2k
pkDOgNaHJsK2pxMjATGILg4j6GVbroDYMqpUwb/shMKuw5ZqjDDNLI8f6pE
E/hOLwOKow2JHbD08VFkp0GzJusBx/JXtotp8LHZgkTzRJXH3Jv+1fbL6wKS
qrhd25fU7RMA7tA7rpT7tDxaWoSDXk175y+yS5vGouMq0weCF4vYMB6Q8gyw
+TfDZPKQNYUS0adhHawp2D+E9VDG9VWKSfQpOkdVV15NBj7SuHORzCYc9zk
O2g1wc5Ar4WaVtE9o3SLNKMDnMibPBo4A5kitgum8kNjIj0QLrYzg97N6bQ
oOrs3xvKxYQW3BxtVliY0I1r0i0Xj8ini2btUF9LFr+a/8khsWxv5XGdoFt4
ta3R0kAUioXln/55Q6xYJFoebKwnuyLXTZFC+4m011LYFGclGB0osKtN4DP
Ocoo3UPPjqjEiaopPp6klDvv8I3Ww79S4iRbKIRrXXCWZYM08C8WY3MwxgU3
HZSSFLwOcm01Wxt/rCkz69tvrnwmYtKcwzUYw/GHK0c0/kKz7bDfYqqdSX7Q
11Gytq7wZztTxubnvWo3lPKKJu0i3uP6MIIZOLiYpGF6NnQOMkj1pc9MxWby
IUUJERax6sanQvfcmeY3gBgBKgpMo6QFzX/yMOoSUVrBzjDxhA7uSBW63/6l
wbzMPwqtD1nRmJAj9JoSXpv1AJUrUKDZKR4q9Rw9nXZN3SKEBaXOFm+gMgRS
h8mUgjKk8CM09e86HXChSoFX5ZVfkiEE6gJCvYrN7jqep72KZnpXauH73sA
jgOor2B79Nv7VNAkONlctgf68C0Z7tyTy5yEMVzZArYDyXZm6MiSqhZi0S8M
R4vKUuccXHznuYIpeVtLYPYKPiUYA8vqo2te7f4UNM1I1ut9kD86Eb+1MqIG
UOQ6bYot3DqBUR7o0Ib3qDTiKVKhurkuAGLApzBkDGJHYRp93Bd+nnWQew
4y0u1dTQ+RDs/+zUhkHRGrJ/C7GASpm0oLUaNNq8al9JshrtPUX0cIufoUYo
70j6CPzSn07vQXLwHCqH41VfywVJG3veVCsV3rVVuFY+94rUJmMp4CZyGytL
ylG6lX1j7wo+TB+9ZApZgLQegYQV+AIn59ce0fzkUDnbt8p3IJC+zm/OKkdm
chaDKXjKlTP78bKhuppd/LcCIs/6BK1eu7zf3z2XUDNME0AnP/K1UYcz1ffj
nr8TgPTLTAGlYdoaYtKRgHuzGt+OLtz45+Wia0iw8K6asoCi7n+lm28aeVlc
XDWjfldfHc7o8MJZUZHw8qlQ20u18G4lht6f0S1L1cQ6apBMn+58zw/jX0C3
kw3grncgKboS2Iwazg0vSL3IAAXjCRnKJF87RC0SHzhNxxmCklBf5g0JXk9f
5Khos3d7Jmee+bM2yvilTpyecQlwcmqwit16Amn3DRYMGkQe178KjBB1wKO+
fRDRXR0rTA+oW07h51low6yT/RJTWzGNv4L+fdtAQ0e8ZsHVJSf1N2N7Sv+y
ADw5F5Yv23DgAHuL17556u3N57X1eTadbcVU07c0V0UYe5YgA70mW1g1cH

```

Figure 40 - Encrypted file in IPFS

Since the application wants to prevent unauthorized access to patient's information, the files stored in IPFS is encrypted and is displayed if access directly in IPFS' server.



Content Identifier Explorer

The screenshot shows the IPFS CID Explorer interface. At the top, there's a browser window with the address bar showing the URL: `https://cid.ipfs.io/#bafybeidunpaqyupfg3rgrahytcf2kth4ybusjzanboosxzhqs6f6by2zm`. Below the browser window, the IPFS logo is visible. The main content area displays the CID: `bafybeidunpaqyupfg3rgrahytcf2kth4ybusjzanboosxzhqs6f6by2zm`. Below this, there are links for [Docs](#), [Spec](#), and [Tutorial](#). The section titled **HUMAN READABLE CID** shows the breakdown: `base32 - cidv1 - dag-pb - (sha2-256 : 256 : 746BC10C51E536E26880F8988BA54CFCC06924E40D0B9CE95F2784BC5F071ACB)`. Below this, the **MULTIBASE** section shows: `PREFIX: b` and `NAME: base32`. The **MULTICODEC** section shows: `CODE: 0x70` and `NAME: dag-pb`. The **MULTIHASH** section shows: `CODE: 0x12`, `NAME: sha2-256`, `BITS: 256`, and `DIGEST (HEX): 746BC10C51E536E26880F8988BA54CFCC06924E40D0B9CE95F2784BC5F071ACB`. At the bottom, the **CIDV1 (BASE32)** section shows the full CID: `bafybeidunpaqyupfg3rgrahytcf2kth4ybusjzanboosxzhqs6f6by2zm`.

Figure 41 - Content Identifier Breakdown using IPFS CID Explorer

Figure 41 is an illustration of the breakdown of the generated Content Identifier of the uploaded encrypted file from IPFS. This contains details concerned on how the identifier was assembled and what type of hashing algorithm was used. It also has human readable breakdown for easier investigation of its component.



DAG Visualization

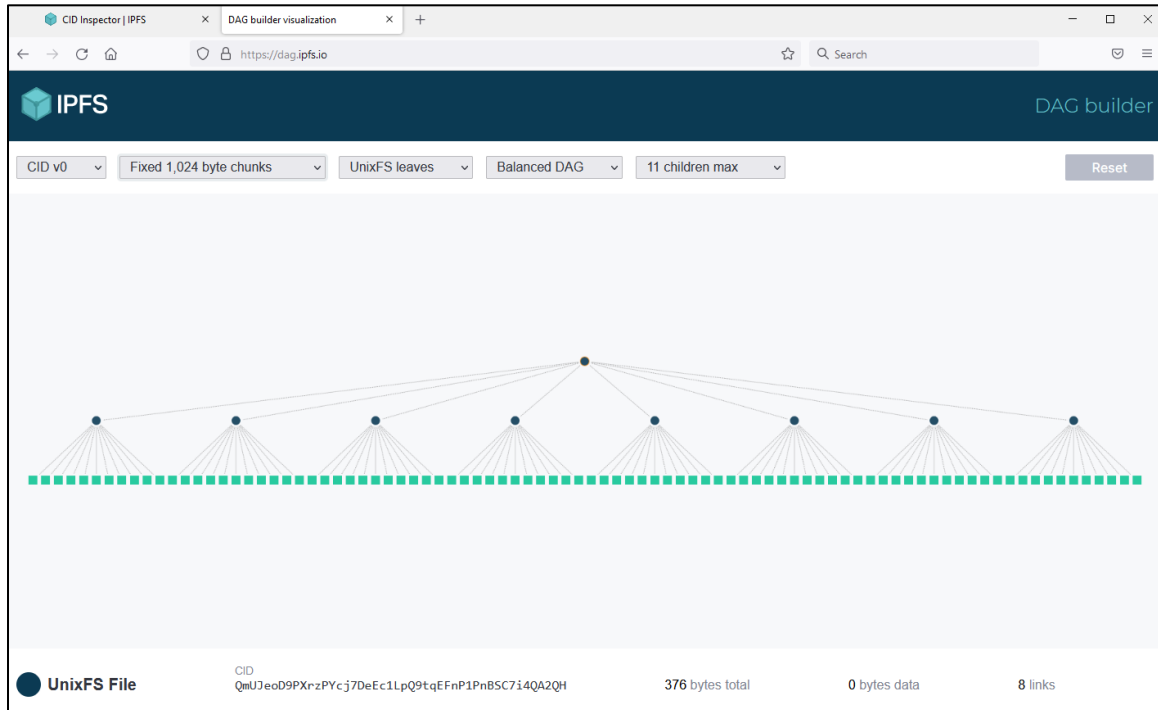


Figure 42 - Merkle DAG Illustration using IPFS DAG Visualizer

Figure 42 shows the generated Merkle DAG of a sample uploaded file. It is chunked in 1,024 (over 1KB) and all nodes has two levels of ancestry.

5.4 Data Tampering Results

1. Tamper summary hash in database
 - Update summary hash of a patient ID via database update script
 - Scan patient Vaccine QR Code
 - Expected Result: App flags the QR code as Invalid.

Update Script:

```
-- original SummaryHash = 3806adeb70cea6eb9fd1dfdcb081945ae169a09f086e5781d3aae1cb730b370
UPDATE dbo.VaccineSummary
SET SummaryHash = '1806adeb70cea6eb9fd1dfdcb081945ae169a09f086e5781d3aae1cb730b370'
WHERE SummaryHash = '3806adeb70cea6eb9fd1dfdcb081945ae169a09f086e5781d3aae1cb730b370'
```




UI Result:

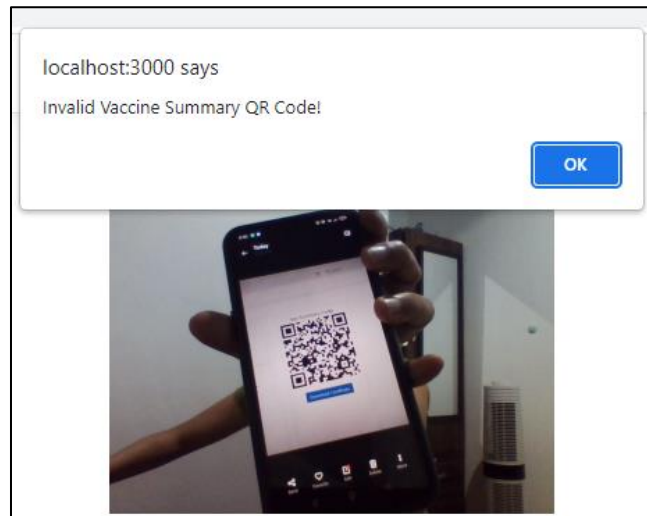


Figure 43 - UI Screenshot for invalid Summary QR Code

2. Tamper file hash in database

- Generate a tampered vaccine certificate pdf from an existing file
- Get file hash of tampered data and put it in the database via database update script
- Validate file via “Validate File Upload”
- Expected Result: App flags the uploaded file as Invalid.

Original File

Address: 309
Tondo

Protect the QR code from marks and damage

VACCINATION EVENT				
Vaccine or prophylaxis			Disease or Agent	
XM68M6 (COVID-19)			RA01.0 (COVID-19)	
Date of Vaccination	Dose Number	Vaccine Brand	Country of Vaccination	Age
Oct 1, 2021	1	Sinovac	PH	18-59
Oct 1, 2021	2	Sinovac	PH	18-59

This certificate was issued on Apr 25, 2022
purpose this may serve best.

Figure 44 - Original File for Tamper Test

Tampered File

Address: 309
Tondo

Protect the QR code from marks and damage

VACCINATION EVENT				
Vaccine or prophylaxis			Disease or Agent	
XM68M6 (COVID-19)			RA01.0 (COVID-19)	
Date of Vaccination	Dose Number	Vaccine Brand	Country of Vaccination	Age
Oct 1, 2021	1	Pfizer	PH	18-59
Oct 1, 2021	2	Pfizer	PH	18-59

This certificate was issued on Apr 25, 2022
purpose this may serve best.

Figure 45 - Tampered File for Tamper Test



Update Script

```
-- original FileHash = 205b583ca3cdf8ad2e5c5cb0695d3938fb77d16c62ab60581734ba963d6cf2da  
UPDATE dbo.VaccineCertificate  
SET FileHash = 'a31c6d1020cc91de3a7f7837cb7131d5148ae927ba8e60172e1faf3034132665'  
WHERE FileHash = '205b583ca3cdf8ad2e5c5cb0695d3938fb77d16c62ab60581734ba963d6cf2da'
```

UI Result:

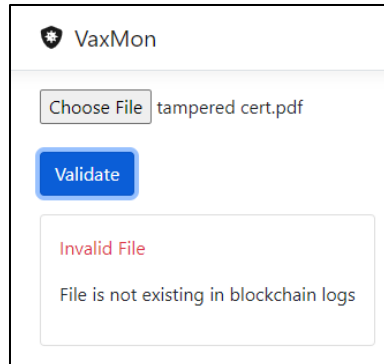


Figure 46 - UI Screenshot for Invalid File

The application was able to handle above tampering scenarios which both used database intervention from the attacker.

5.5 Security Assessment

The researcher employed two Solidity scans and a Cybersecurity Assessment to audit the application's functionality. This ensures that the application complies with existing standard for Solidity and Open Web Applications. Because the application size is small, the scans were focused on the blockchain and web application aspects.

5.5.1 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. It is loosely aligned to the terminologies and structure used in the Common Weakness Enumeration



(CWE) 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

5.5.1.1 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.

Scan Result

Table 8 - Securify Scan Result Details

Item #	Severity	Pattern	Description
1	Medium	Missing Input Validation	Method arguments must be sanitized before they are used in computations.
2	Medium	Missing Input Validation	Method arguments must be sanitized before they are used in computations.
3	Medium	Missing Input Validation	Method arguments must be sanitized before they are used in computations.
4	Low	Solidity pragma directives	Avoid complex solidity version pragma statements.



Table 9 - Securify Results Summary

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

Table 8 shows a list of all the scan vulnerabilities encountered. There are three redundant entries which were found on different parts of the smart contract code. Table 9 shows a tally of the vulnerabilities based on their severity. All 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 under 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.

5.5.1.2 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.



Scan Result

Table 10 - Slither Scan Result Details

Item #	Severity	Pattern	Description
1	Informational	Pragma version is too complex	Solidity versions mismatch

Table 11 - Slither Results Summary

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

Table 10 shows the audit items found in the scan. 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. Table 11 summarizes the scan by severity. It is worth noting that Slither and Securify have audit findings that have the same specifications and criteria such as the item mentioned here. “Pragma version is too complex” is comparable to Securify’s “Solidity pragma directives”.

5.5.2 NIST – Cybersecurity Framework

The NIST CSF enables providers to assess their cybersecurity environment, regardless of size, degree of risk, or experience. It also offers voluntary guidance to providers to understand, select, and implement cybersecurity controls.

The researcher focused the assessment on the Protect Core function because this research is specifically concerned with the security aspects of the application. This function provides the framework develop and implement the appropriate safeguards to limit or contain the potential impact of a cybersecurity event.



Table 12 - NIST-CF Maintenance Area Matrix

Maintenance (PR.MA): Maintenance and repairs are performed for all industrial controls and information system components.					
PR.MA	Implemented a process to:	Readiness			
		Informal	Developing	Established	N/A
1	Use approved and controlled tools to timely perform, repair, and log maintenance and repairs	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2	Approve, log, and perform all remote maintenance of organizational assets to prevent unauthorized access	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

As shown on table 12, both items were established. Item 1 was implemented by using Solidity audit scans on deployed smart contracts. This enables the continuous maintenance of the app if Solidity standards are updated. Item 2 was implemented by using a source control (GIT) to monitor all assets being pushed in the application repository.

Table 13 - NIST-CF Data Security Area Matrix

Data Security (PR.DS): Appropriately manage all information and records at the organization in accordance with the organization's risk strategy to protect the confidentiality, integrity, and availability of information.					
PR.DS	Appropriate safeguards to:	Readiness			
		Informal	Developing	Established	N/A
1	Protect data-at-rest	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2	Protect data-in-transit	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3	Formally manage assets during removal, transfer, and disposition	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
4	Ensure adequate capacity to maintain data availability	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5	Protect against data leaks	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
6	Verify software, firmware, and information integrity	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
7	Maintain separation between the development and testing environment(s), and the production environment	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>



As seen on table 13, items pertaining to Confidentiality of the Information (protect data-at-rest, protect data-in-transit, ensure adequate capacity, protect against data leaks and verify software) were established. These were implemented by various measures such as file encryption/decryption, appropriate package upgrades using plugin managers and usage of Proof-of-Authority to manage downtime and performance. However, item 3 is not applicable to the system because of the prototyping nature of the application. Item 7 is informally implemented because the environment of the application prototype persists on virtual sandboxes. This type of environment is used for testing purposes only.

Table 14 - NIST-CF Protective Technology Area Matrix

Protective Technology (PR.PT): <i>The security and resilience of systems and assets are managed through the use of technology security solutions that are consistent with related policies, procedures, and agreements.</i>					
PR.PT	Implemented a process to:	Readiness			
		Informal	Developing	Established	N/A
1	Create, document, implement, and review audit/log records	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2	Protect and restrict use of removable media	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
3	Limit access systems and assets to the minimal level necessary to maintain normal functioning	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4	Protect communications and control networks	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

On table 14, items 1 and 4 are established. Item 1 was implemented by logging all errors in the database and organizing them to be easily sorted and recognized. This also includes application and event logs on server side. Item 4 was executed by the segregated architecture of the application to separate communications between frontend, business logic and data layer. This enables decoupling of responsibilities among these components. Item 3 is developing because of the limited scope of the application. This requires more information to the level of data users can access within their persona. Finally, item 2 is not applicable to the system because it does not have media components



5.6 Performance Test

The researcher used JMeter tool to conduct the load tests. A sample of 20 concurrent users were identified as test data for the 3 main functionality of the application

a. Create Vaccine Record

Table 15 - Create Vaccine Record Load Test Result

Run #	No. of Concurrent Users	Response Time (seconds)	No. of PoA Nodes
1	20	121	5
2	20	113	5
3	20	99	5
4	20	134	5
5	20	118	5
	Average	117	

- This transaction involves file creation, IPFS upload and writing to the blockchain. It is important to note that writing to the blockchain is slow. This is because it has to sync-up with all the validator nodes and update all the logs within those nodes.

b. Validate Vaccine QR Code

Table 16 - Validate Vaccine QR Code Load Test Result

Run #	No. of Concurrent Users	Response Time (seconds)	No. of PoA Nodes
1	20	7	5
2	20	7	5
3	20	8	5
4	20	5	5
5	20	9	5
	Average	7.2	



- This transaction includes scanning of QR code and reading from the blockchain. As compared with item a, reading from the blockchain is fast. It only needs to read from the first node that gives a response to the web3 request.

c. Validate Vaccine File

Table 17 - Validate Vaccine File Load Test Result

Run #	No. of Concurrent Users	Response Time (seconds)	No. of PoA Nodes
1	20	10	5
2	20	8	5
3	20	9	5
4	20	10	5
5	20	10	5
	Average	9.4	

- Same as item b, this also reads from the blockchain. The additional process is generating a filehash from the uploaded file which concurred additional response time.



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. Application was able to implement Proof-of-Authority blockchain (which natively uses Keccak Hash Algorithm to process transactional blocks) to store vaccine records. Other blockchain-related technologies were used such as: Solidity (smart contract), web3 and Geth.
2. Application was able to implement Merkle DAG by using native functionalities of IPFS. Data storage was illustrated by various IPFS tools (CID Explorer, IPFS file server). Specifically, the generated hash tree for a sample vaccine certificate file can be reproduced by IPFS DAG Visualizer.
3. The researcher was able to assess the vulnerabilities flagged by the scans and made efforts to eliminate and lessen the weaknesses for blockchain transactions via Smart Contract Weakness Classification Registry (SWC Registry). Accomplishment of NIST-CF Protect Core Function, specifically Maintenance, Data Security and Protective Technology, was also done to ensure the system complies with existing standards for Cyber Security. The researcher listed out areas already handled by the native functionality of the system and excluded items that are not applicable given the nature of the application.



Recommendations

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

1. Blockchain-based electronic health records gives 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. 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 is a good addition to create a dedicated mobile application to patients to enhance their user experience.



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*
- Bojanova, Irena and Voas, Jeffrey (2014). *NIST: Building a Solid Foundation*
- De Angelis, Stefano; Aniello, Leonardo; Baldoni, Roberto; Lombardi, Federico; Margheri, Andrea; Sassone, Vladimiro (2018). *PBFT vs proof-of-authority: applying the CAP theorem to permissioned blockchain*
- 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*
- Falco, Joseph; Gilsinn, James; Stouffer, Keith (2004). *IT Security for Industrial Control Systems: Requirements Specification and Performance Testing*



- 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*
- Garzik, Jeff (2015). *Public versus Private Blockchains*
- 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*
- Huang, Jing; Zhou, Kuo; Xiong, Ao; Li Dongmeng, 2022. *Smart Contract Vulnerability Detection Model Based on Multi-Task Learning*
- 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)*
- Rameder, Heidelinde; di Angelo, Monika; Salzer, Gernot (2022). *Review of Automated Vulnerability Analysis of Smart Contracts on Ethereum*
- Radziwill, Nicole and Benton, Morgan (2017). *Cybersecurity Cost of Quality: Managing the Costs of Cybersecurity Risk Management*
- 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*
- Sukrutha, 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; Randić, 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*




APPENDICES

Generated Vaccine Certificate


COVID-19 Vaccination Certificate


Proof of VaccinationIssued by PHVersion 1


Protect the QR code from marks and damage

Name of holder: Joane Llamera
Date of Birth: May 4, 1990
Address: Bacoor, Cavite

VACCINATION EVENT						
Vaccine or prophylaxis			Disease or Agent Targeted			
XM68M6 (COVID-19)			RA01.0 (COVID-19)			
Date of Vaccination	Dose Number	Vaccine Brand	Country of Vaccination	Administering Center	Vaccinator	Vaccine Batch Number
Oct 22, 2021	1	4	PH	4	4	{{batchNumber}}
Oct 22, 2021	2	4	PH	4	4	{{batchNumber}}

This certificate was issued on Dec 8, 2021 for whatever legal purpose this may serve best.


Jaime Santos, MD
Authorized Health Officer



Solidity Code to Manage Hash Lookups

```
// 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;
    }
}
```



Source Code

```
import { useState, useEffect } from "react";
import getWeb3 from "../getWeb3";
import truffleContract from "truffle-contract";
import detectEthereumProvider from '@metamask/detect-provider'
import CertificateContract from "../contracts/Certificate.json"
import VaxForm from "./VaxForm";
import ScanPatientId from "./ScanPatientId";

const CertificateUpload = () => {
  const [vaxDetails, setVaxDetails] = useState({
    patientId: null,
    firstName: '',
    lastName: '',
    email: '',
    address: '',
    birthdate: '',
    doses: {}
  });

  const {patientId} = vaxDetails;

  const [ethState, setEthState] = useState({
    web3: null,
    accounts: null,
    contract: null,
  });

  useEffect(() => {
    const initWeb3 = async () => {
      try {
        const web3 = await getWeb3();
        const accounts = await web3.eth.getAccounts();

        const provider = await detectEthereumProvider()
```



```
    if (!provider) {
      console.log('Please install metamask!');
      return;
    }

    const contract = truffleContract(CertificateContract);
    contract.setProvider(provider);
    const instance = await contract.deployed();
    setEthState({ ...ethState, accounts: accounts, contract:
instance });

    } catch (error) {
      alert(`Failed to load web3, accounts, or contract. Check
console for details.`);
      console.log(error);
    }
  }
  initWeb3();

}, []));

const sendToBlockchain = async (fileHash, summaryHash) => {
  const contract = ethState.contract;
  const account = ethState.accounts[0];
  await contract.saveUserIdHashes(fileHash, summaryHash, patientId, {
from: account });
  alert('data sent to blockchain');
}

const manualSendToBc = async () => {
  const
                                fileHash
                                =
'3a206c85ccf6a92931276f1eb10882069e868413aff3694761a8ee675dd50034'
  const
                                summaryHash
                                =
'1060832ae769ce5d899a5b1ecda4f9304d80d49b81f6d4f3e22fdcca5e9c040e'
  await sendToBlockchain(fileHash, summaryHash);
}
```



```
const checkFileHash = async () => {
  const fileHash =
    '3a206c85ccf6a92931276f1eb10882069e868413aff3694761a8ee675dd50034';
  const contract = ethState.contract;
  const account = ethState.accounts[0];
  const isExists = await contract.isFileHashUserIdExists(fileHash,
patientId, { from: account });
  alert(`fileHash - ${fileHash} - ${isExists}`);
}

const sendDataToParent = (doseType, data) => {
  const doses = JSON.parse(JSON.stringify(vaxDetails.doses));
  doses[doseType] = data;
  setVaxDetails({ ...vaxDetails, doses: doses });
};

const handlePatientDetails = (details) => {
  setVaxDetails({
    ...vaxDetails,
    patientId: details.patientId,
    firstName: details.firstName,
    lastName: details.lastName,
    email: details.email,
    address: details.address,
    birthdate: details.birthdate
  });
};

const submitRecord = async (e) => {
  e.preventDefault();
  e.target.reset();

  const requestOptions = {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(vaxDetails)
  };
};
```



```
const response = await fetch('api/create-vaccine-record',
requestOptions);
const data = await response.json();
const fileHash = data.fileHash;
const summaryHash = data.summaryHash;
console.log('data', data);
await sendToBlockchain(fileHash, summaryHash);
}

if (!ethState.contract) {
  return <div>Loading Web3, accounts, and contract...</div>;
}

return (
  <div className="d-flex justify-content-around">
    <div>
      <ScanPatientId
handlePatientDetails={handlePatientDetails}></ScanPatientId>
    </div>
    <form onSubmit={submitRecord} className="w-50 d-flex flex-column"
style={{maxWidth: '500px'}}>
      {patientId == null && <p className="text-danger">Please scan a
Patient QR Code first.</p>}
      <VaxForm title={'Dose 1'} doseType={'dose1'}
sendDataToParent={sendDataToParent}></VaxForm>
      <VaxForm title={'Dose 2'} doseType={'dose2'}
sendDataToParent={sendDataToParent}></VaxForm>
      <br/>
      <div>
        <button disabled={patientId == null} type="submit"
className="btn btn-primary w-50">Submit</button>
      </div>
    </form>
  </div>
);
}
```



```
export default CertificateUpload;

import React from 'react';
import { useState, useEffect } from "react";
import SummaryView from "../SummaryView";
import SummaryDoses from "../SummaryDoses";
import getWeb3 from "../../getWeb3";
import truffleContract from "truffle-contract";
import detectEthereumProvider from '@metamask/detect-provider'
import CertificateContract from "../../contracts/Certificate.json"
import QrReader from 'react-qr-scanner'

const SummaryValidate = () => {

  const previewStyle = {
    height: 240,
    width: 320,
  }

  const delay = 100;
  const [ethState, setEthState] = useState({
    web3: null,
    accounts: null,
    contract: null,
  });

  const [isValid, setIsValidate] = useState(false);

  const [details, setDetails] = useState({
    firstName: "",
    lastName: "",
    address: "",
    summary: null,
    userId: null
  });
}
```



```
const {summary} = details;

useEffect(() => {
  if (window.location.href.indexOf("validate/summary-code") !== -1) {
    setIsValidate(true);
  }

  const initWeb3 = async () => {
    try {
      const web3 = await getWeb3();
      const accounts = await web3.eth.getAccounts();

      const provider = await detectEthereumProvider()
      if (!provider) {
        console.log('Please install metamask!');
        return;
      }

      const contract = truffleContract(CertificateContract);
      contract.setProvider(provider);
      const instance = await contract.deployed();
      setEthState({ ...ethState, accounts: accounts, contract:
instance });

    } catch (error) {
      alert(`Failed to load web3, accounts, or contract. Check
console for details.`);
      console.log(error);
    }
  }

  initWeb3();
}, []);

const handleScan = async (data) => {
  if (!data) return;
```



```
let summaryHash = null;

try {
  summaryHash = data['text'];
} catch {
  alert('Invalid QR Code');
  return;
}

const summaryData = await fetchData(summaryHash);

if (!summaryData)
  return;

console.log('data', summaryData);
const isExistInBc = await checkSummaryHash(summaryHash,
summaryData.UserId);

if (isExistInBc) {
  console.log('exists in bc')
  setDetails({
    ...details,
    firstName: summaryData.FirstName,
    lastName: summaryData.LastName,
    address: summaryData.Address,
    summary: !!summaryData ? JSON.parse(summaryData.Summary) : null
  });
}

const handleError = (err) => {
  console.log('qr scan error', err);
}

const fetchData = async (summaryHash) => {
  const url = `/api/summary/${summaryHash}`;
```




```
const response = await fetch(url);
const data = await response.json();
return data;
}

const checkSummaryHash = async (summaryHash, userId) => {
  const contract = ethState.contract;
  const account = ethState.accounts[0];
  const isExists = await contract.isSummaryHashUserIdExists(summaryHash,
userId, { from: account });
  return isExists;
}

if (!ethState.contract) {
  return <div>Loading Web3, accounts, and contract...</div>;
}

return (
  <div class="d-flex justify-content-center">
    <div>
      <h5>Scan Summary QR Code</h5>
      <QrReader
        delay={delay}
        style={previewStyle}
        onError={handleError}
        onScan={handleScan}
      />
    </div>
    {
      !!summary &&
      <div class="d-flex">
        <div className="card mx-5">
          <div className="card-body">
            <SummaryView
              isValid={isValid}
              details={details}></SummaryView>
          </div>
        </div>
      </div>
    }
  </div>
)
```



```
        <div className="card m-3">
          <div className="card-body">
            <SummaryDoses summary={summary}></SummaryDoses>
          </div>
        </div>
      </div>
    }
  </div>
);
}

export default SummaryValidate;
import { useState, useEffect } from "react";
import getWeb3 from "../getWeb3";
import truffleContract from "truffle-contract";
import detectEthereumProvider from '@metamask/detect-provider'
import CertificateContract from "../contracts/Certificate.json"
import VaxForm from "../VaxForm";
import ScanPatientId from "../ScanPatientId";

const CertificateUpload = () => {
  const [vaxDetails, setVaxDetails] = useState({
    patientId: null,
    firstName: '',
    lastName: '',
    email: '',
    address: '',
    birthdate: '',
    doses: {}
  });

  const {patientId} = vaxDetails;

  const [ethState, setEthState] = useState({
    web3: null,
    accounts: null,
    contract: null,
```



```
});

useEffect(() => {
  const initWeb3 = async () => {
    try {
      const web3 = await getWeb3();
      const accounts = await web3.eth.getAccounts();

      const provider = await detectEthereumProvider()
      if (!provider) {
        console.log('Please install metamask!');
        return;
      }

      const contract = truffleContract(CertificateContract);
      contract.setProvider(provider);
      const instance = await contract.deployed();
      setEthState({ ...ethState, accounts: accounts, contract:
instance });

    } catch (error) {
      alert(`Failed to load web3, accounts, or contract. Check
console for details.`);
      console.log(error);
    }
  }
  initWeb3();

}, []);

const sendToBlockchain = async (fileHash, summaryHash) => {
  const contract = ethState.contract;
  const account = ethState.accounts[0];
  await contract.saveUserIdHashes(fileHash, summaryHash, patientId, {
from: account });
  alert('data sent to blockchain');
}
```



```
const manualSendToBc = async () => {
  const fileHash =
'3a206c85ccf6a92931276f1eb10882069e868413aff3694761a8ee675dd50034'
  const summaryHash =
'1060832ae769ce5d899a5b1ecda4f9304d80d49b81f6d4f3e22fdcca5e9c040e'
  await sendToBlockchain(fileHash, summaryHash);
}

const checkFileHash = async () => {
  const fileHash =
'3a206c85ccf6a92931276f1eb10882069e868413aff3694761a8ee675dd50034';
  const contract = ethState.contract;
  const account = ethState.accounts[0];
  const isExists = await contract.isFileHashUserIdExists(fileHash,
patientId, { from: account });
  alert(`fileHash - ${fileHash} - ${isExists}`);
}

const sendDataToParent = (doseType, data) => {
  const doses = JSON.parse(JSON.stringify(vaxDetails.doses));
  doses[doseType] = data;
  setVaxDetails({ ...vaxDetails, doses: doses });
};

const handlePatientDetails = (details) => {
  setVaxDetails({
    ...vaxDetails,
    patientId: details.patientId,
    firstName: details.firstName,
    lastName: details.lastName,
    email: details.email,
    address: details.address,
    birthdate: details.birthdate
  });
};
```



```
const submitRecord = async (e) => {
  e.preventDefault();
  e.target.reset();

  const requestOptions = {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(vaxDetails)
  };

  const response = await fetch('api/create-vaccine-record',
requestOptions);
  const data = await response.json();
  const fileHash = data.fileHash;
  const summaryHash = data.summaryHash;
  console.log('data', data);
  await sendToBlockchain(fileHash, summaryHash);
}

if (!ethState.contract) {
  return <div>Loading Web3, accounts, and contract...</div>;
}

return (
  <div className="d-flex justify-content-around">
    <div>
      <ScanPatientId
handlePatientDetails={handlePatientDetails}></ScanPatientId>
    </div>
    <form onSubmit={submitRecord} className="w-50 d-flex flex-column"
style={{maxWidth: '500px'}}>
      {patientId == null && <p className="text-danger">Please scan a
Patient QR Code first.</p>}
      <VaxForm title={'Dose 1'} doseType={'dose1'}
sendDataToParent={sendDataToParent}></VaxForm>
      <VaxForm title={'Dose 2'} doseType={'dose2'}
sendDataToParent={sendDataToParent}></VaxForm>
```



```
<br/>
<div>
    <button disabled={patientId == null} type="submit"
className="btn btn-primary w-50">Submit</button>
</div>
</form>
</div>
);
}

export default CertificateUpload;
```