**Chapter Three**

# **THEORETICAL FRAMEWORK**

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

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

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

**Proposed Documents Storage Structure**

**Figure 3.0.1: Diagram of Proposed Solution**

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

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

**Cryptographic Hash Functions**

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

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

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

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

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

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

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



**Figure 3.0.2: Hashing Process**

The majority of cryptographic hash functions accept any length string as input and return a fixed-length hash value.

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

* Pre-image resistance

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

* Second pre-image resistance

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

* Collision resistance

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

**Blockchain**

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

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

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

1. Block Header

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

1. Block Data

* Actual data (text, files)

**Figure 3.0.3: Generic Blockchain Transactions**

Figure 3.0.3 shows how blockchain works given we have a simple data of text. The initial block is referred to the genesis block and is automatically generated upon the chain’s creation. This genesis block will be the seed and considered as reference of all blocks going forward. Blocks are linked through each block containing the hash value of the previous block’s header, thus creating the chain. In case a previously published block was changed, it will have a different hash. This will create a domino effect on all subsequent blocks to also have a different hash because they contain the hash of the altered block.

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

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

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

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

****

**Figure 3.0.4: 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:
  + Name: the name of the Link.
  + Hash: the hash of the linked IPFS object.
  + Size: the cumulative size of the linked IPFS object, including following its links.

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

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



**Figure 3.0.5: 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 3.0.6: 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 would affect all ascendants in the DAG, effectively resulting in the creation of a new DAG.
* Merkle DAGs are like Merkle trees, but they don't have to be balanced, and each node can have a payload. Many branches can re-converge in DAGs, or, to put it another way, a node can have multiple parents.

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

**Figure 3.0.7: 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 Keccak-f[b], where b∈{25,50,100,200,400,800,1600} is the width of the permutation. The width of the permutation is also the width of the state in the sponge construction.

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

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

The parameters defining the standard instances are given in the table below.

Table: the 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.

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

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

Diagram

Description automatically generated

**Figure 3.0.9: Conceptual Framework**

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

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

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